REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From – To)*
09-08-2010	Final Report	1 June 2007 - 01-Jul-10

4. TITLE AND SUBTITLE

Agent-based Computing in Distributed Adversarial Planning

5a. CONTRACT NUMBER
FA8655-07-1-3083

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Dr. Michal Pechoucek

5d. PROJECT NUMBER

5d. TASK NUMBER

5e. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Czech Technical University
CTU, FEE-K333
Technicka 2
Prague 6 166 27
Czech Republic

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
Unit 4515 BOX 14
APO AE 09421

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
Grant 07-3083

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report results from a contract tasking Czech Technical University as follows: This project investigated the problem of adversarial reasoning and planning, i.e. goal-oriented decision making in the presence of other adversarial actors. In contrast to existing approaches, our line of research addresses the problem within the context of complex, asymmetric domains with properties similar to those found in real-world conflict situations (a higher number of parties, asymmetry in party's objectives and resources, huge state space etc.). The focus of the previous project was on fully non-cooperative scenarios. The current project took into consideration also (partial) explicit cooperation among individual parties in the scenario. The pursued approach combines theoretical analysis with practical algorithm development with strong emphasis on empirical evaluation using a multi-agent adversarial behavior testbed.

The project delivered the following specific results:

- extended formal framework for adversarial reasoning: The framework refines the concepts defined in our previous work and adds notions related to creating, maintaining and reasoning about coalitions in adversarial and semi-cooperative settings.
- extended adversarial behavior testbed: The existing testbed was adjusted to allow negotiations and explicit coordination among the players. New testing scenarios that require coordination of self-interested parties were developed.
- agent subset adversarial search: A novel algorithmic scheme for substantial speed-up of generic adversarial-search based algorithms has been developed, implemented and extensively evaluated within the testbed.
- sub-game negotiation tree search: The algorithm uses negotiation among self-interested parties to create mutually more beneficial plans. An agent is expected to agree to deviate from its optimal uncoordinated plan only if it improves its position.
- process models for opponent modeling: We have analyzed the suitability of business process models for creating models of opponents with internal states. We experimentally evaluate plausibility of learning such models and discuss their usability in (semi) cooperative environment.
- deception in large adversarial scenarios: We have formally defined the notion of deception for teams of mobile sensing agents and we have developed algorithms that are robust against this form of adversarial behavior.

15. SUBJECT TERMS

EOARD, Modeling & Simulation, Distributed Artificial Intelligence, C4ISR, Agent Based Systems

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18, NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			JAMES LAWTON Ph. D.
UNCLAS	UNCLAS	UNCLAS	UL	107	**19b. TELEPHONE NUMBER** *(Include area code)* +44 (0)1895 616187

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39-18

Final report of the project FA8655-09-1-3060, prepared by the
Czech Technical University, Gerstner Laboratory, Agent Technology Center.

Item 003

Cooperative Adversarial Reasoning and Planning in Complex Environments (Final report)

Michal Pěchouček(PI), Viliam Lisý, Branislav Bošanský, Roman Vaculín

This document provides the final statement of the work as of July 31, 2010 after termination of the month 10 of the project.

The motivation of the FA8655-09-1-3060 research project is to study the problem of adversarial planning in distributed, resource bounded and information limited environments. The goal of the project is to advance the state of the art of adversarial planning by exploiting selected multi-agent techniques in distributed semi-cooperative environments.

Contents

Executive Summary

The presented report summarizes work performed within research project FA8655-09-1-3060 - Cooperative Adversarial Reasoning and Planning in Complex Environments. The aim of the project was to leverage and extend the results of the successfully completed FA8655-07-1-3083 project - Agent-based Computing in Distributed Adversarial Planning.

The projects investigated the problem of adversarial reasoning and planning, i.e. goal-oriented decision making in the presence of other adversarial actors. In contrast to existing approaches, our line of research addresses the problem within the context of complex, asymmetric domains with properties similar to those found in real-world conflict situations (a higher number of parties, asymmetry in party's objectives and resources, huge state space etc.). The focus of the previous project was on fully non-cooperative scenarios. The current project took into consideration also (partial) explicit cooperation among individual parties in the scenario. The pursued approach combines theoretical analysis with practical algorithm development with strong emphasis on empirical evaluation using a multi-agent adversarial behavior testbed.

The project delivered the following specific results:

- **extended formal framework for adversarial reasoning** – The framework refines the concepts defined in our previous work and adds notions related to creating, maintaining and reasoning about coalitions in adversarial and semi-cooperative settings.
- **extended adversarial behavior testbed** – The existing testbed was adjusted to allow negotiations and explicit coordination among the players. New testing scenarios that require coordination of self-interested parties were developed.
- **agent subset adversarial search** – A novel algorithmic scheme for substantial speed-up of generic adversarial-search based algorithms has been developed, implemented and extensively evaluated within the testbed.
- **sub-game negotiation tree search** – The algorithm uses negotiation among self-interested parties to create mutually more beneficial plans. An agent is expected to agree to deviate from its optimal uncoordinated plan only if it improves its position.
- **process models for opponent modeling** – We have analyzed the suitability of business process models for creating models of opponents with internal states. We experimentally evaluate plausibility of learning such models and discuss their usability in (semi) cooperative environment.
- **deception in large adversarial scenarios** – We have formally defined the notion of deception for teams of mobile sensing agents and we have developed algorithms that are robust against this form of adversarial behavior.

The scientific results of this project have been published at a premier conference in the field of Autonomous Agents and Multi-Agent Systems – AAMAS 2010, conference on Computational Intelligence and Games (CIG2010) and they will be shortly submitted to the International Journal of Autonomous Agents and Multi-agent Systems.

Chapter 1

Introduction

When multiple (teams of) agents operate towards their own goals in a shared environment, situations may arise in which the actions and strategies of individual agents interfere. Examples of such scenarios include military operations, traffic control, disaster recovery etc. A characteristic feature of such situations, often termed *games* is that the outcome of an agent's actions and sequences of actions depends on the actions chosen by others. One of the fundamental problems of operating in such game scenarios is deciding on the course of action, understanding that any such course can be affected by the activity of other agents, who autonomously act towards achieving their own objectives in the shared environment.

In many of these scenarios, the individual (teams of) agents are not willing to communicate and coordinate their action choices in advance. Keeping the future actions private leads to the element of surprise, which is generally desirable in adversarial settings. This is the typical case in military conflicts or law enforcement operations and also the main focus in our previous research project (FA8655-07-1-3083 – Agent-based Computing in Distributed Adversarial Planning). In these scenarios, the only way how to adjust agent plans to actions of the other agents is to model its behavior and predict its further actions. The other extreme are the situations with fully cooperative agents that share a common goal and create joint plans to achieve them, i.e. the agents form a single team. Agents in the real world scenarios are typically in between these extremes. They often have their own objectives and motivations, but they are willing to communicate and coordinate as long as it supports achieving their own goals. For example, military and humanitarian organization in a region may have completely different objectives and no motivation to fully cooperate, but they are still willing to coordinate if it is beneficial for both parties. This project studies intelligent planning and decision making in adversarial as well as the described semi-cooperative settings.

1.1 Project Objectives

Besides other results, our previous research in the area of adversarial planning and reasoning has identified the key future research directions in the area of automated planning of actions for agent in complex real world conflict situations. The three of them, which were pointed out in the proposal of this project are:

1. *Complexity* – Unlike classical academic games which are still the subject of most current adversarial planning and search methods, real-world adversarial domains, with asymmetry of the goals, dynamics in the changes in the environment, non-determinism and concurrency of player's actions, entail huge branching factors and enormous search spaces. Scalability of adversarial reasoning methods to such a class of problems is a persistent challenge.
2. *Uncertainty* – Realistic adversarial domains are partially observable and typically contain non-determinism in the outcomes of actions performed by players in the domain. Although reasoning under uncertainty in adversarial scenarios has been studied in the AI research, it generally assumed that the partial information is caused by natural limitation of the agents. Another important cause of uncertainty in conflict situations the intentional information manipulation by the adversary – deception.
3. *Cooperation* – In many realistic adversarial scenarios, individual players can be to various degrees cooperative, willing collaborate or at least coordinate in their decision-making and planning. Inter-player relationships may have a substantial impact on the form and interactions of strategies considered by individual actors. As such, they need to be properly represented and reflected in the reasoning and planning algorithms.

The objective of this project was to investigate selected aspects of the listed challenges. In order to achieve this objective, we have defined four research tasks.

1. *RT1: Formal Framework and Testbed for Cooperative Adversarial Reasoning* Define a new collaborative adversarial planning framework for adversarial reasoning scenarios with possible coalitions. Define and implement a testbed for simulating complex adversarial scenarios where players can form into coalitions.
2. *RT2: Scalable Adversarial Planning and Search* Define and implement improvements of Goal-Based Game-Tree Search algorithm in order to support complex scenarios suitable for collaborative adversarial search and planning techniques.
3. *RT3: Cooperative Opponent Modeling* Creating representations of various kinds of information about other agents behavior with focus on universality and cooperative and semi-cooperative environments. Namely focusing on suitability for sharing and negotiating about (parts of) the models and describing coalition bindings between the agents.
4. *RT4: Cooperative Adversarial Search* Designing algorithms and protocols that enable coalitions of agents to efficiently create common strategies that are more favorable for the agents then strategies created without coordination. Moreover, these strategies will allow saving computational resources by sharing results of computations that would have to be performed by each of the agents without the coalition.

1.2 Overview of Project Results

In this final report, we deliver the following six main results.

1. Extended Formal Framework for Adversarial Reasoning
2. Extended Adversarial Behavior Testbed
3. Agent Subset Adversarial Search (ASAS)
4. Sub-game Negotiation Tree Search (SGNTS)
5. Process Models for Opponent Modeling
6. Deception in Teams of Mobile Sensing Agents

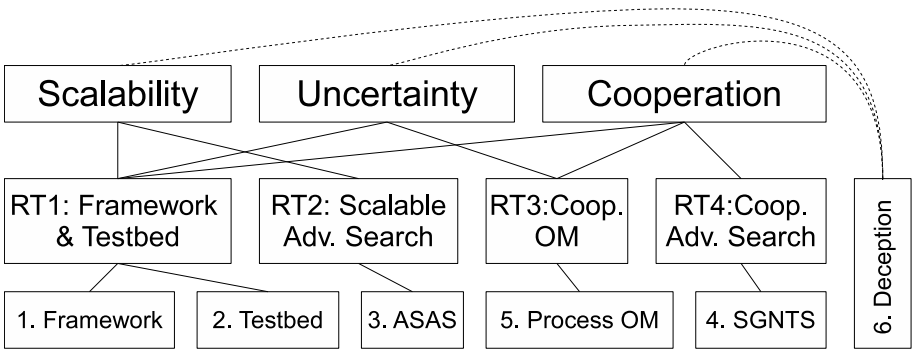

Figure 1.1: The overview of the research directions defined in the proposal, resulting research tasks and the achieved results.

The relation of these results to the individual research tasks and objectives of the project is depicted in Figure 1.1. The result 6 was not originally included in the plan for the project and it does not directly support any of the research tasks. However, it supports all three main research directions defined in the project proposal.

In the following, we briefly overview the components and refer to the respective chapters where the results are described in detail.

1.2.1 Extended Formal Framework for Adversarial Reasoning

The team of this project focuses on the research of adversarial behavior in multi-agent systems already for a longer period of time. During this research, we continuously develop and refine a formal map of the most fundamental concepts related to adversarial reasoning in multi agent systems, their relations and properties. Our former result in this line of research include the tri-base acquaintance model (3bA) [1], the model of interaction stance of the agents [2], or the conceptual framework of adversarial planning [3]. In this report, we utilize the experience gained during our work on this project to further refine some aspect of the previously introduced models. We relate the different kinds of knowledge describing agent's behavior to the popular BDI framework to make them more accessible for researchers in the field of autonomous agents and multi-agent systems. After that, we investigate how the 3bA model needs to be modified in order to appropriately describe the social reasoning of an agent in presence of adversaries. Our results in this tasks are reported in Chapter 3.

1.2.2 Extended Adversarial Behavior Testbed

Most of the experimental results presented in this report were obtained from the adversarial reasoning testbed implemented for the previous project and described in [3]. However, the testbed needed several adjustments in order to support the research tasks of this project. These are presented in Chapter 3 of this report.

The improvements fit into three categories. First, we have implemented ASAS and SGNTS algorithms within the testbed. Now it provides even stronger players that can be used in development

of other adversarial reasoning techniques. A decent player that can handle reasonably large settings in the testbed is useful as a comparison for any methods developed in future. Moreover, it can be used as the adversary in development of other game playing techniques and as the object of opponent modeling.

The second category of changes in the testbed are the new testing scenarios. In order to demonstrate higher scalability of the methods developed in this project, we had to increase the size of the main testing scenarios. A new testing scenario was also developed to clearly show the benefits of cooperation among self-interested agents in complex asymmetric games.

The third improvement is implementation of a software component capable of logging the events in the game in the standard MXML file format. This format is used as the input to many available tools for further analysis of the logs.

1.2.3 Agent Subset Adversarial Search

Agent Subset Adversarial Search is a novel algorithmic scheme that is the main result of our effort on RT2. This scheme allows asymptotic reduction of the computational complexity of multi-agent adversarial search. We describe the method in Chapter 4. The main idea of the approach is to decompose the game into a set of smaller overlapping sub-games featuring only a subset of the players, solve each sub-game separately, and then combine the results into a global solution. We present two variants of the sub-results merging algorithms with different applicability trade-offs. One of them reduces the exponential dependence of the adversarial search complexity on the number of agents to polynomial, the other cannot guarantee as strong reduction in the computational effort, but it achieved better game playing performance. Still, the proposed method is targeted only to the domains with sparse agents' interactions.

The proposed algorithmic scheme can be used in combination with various adversarial search algorithms. We have implement it on top of the goal-based game tree search (GB-GTS) developed in the previous project. The experimental evaluation was conducted in our adversarial behavior testbed. The results show that the ASAS heuristic often finds the same solutions as the complete GB-GTS search, but the search efficiency is significantly improved.

1.2.4 Sub-game Negotiation Tree Search

As a result of our work on RT4, we deliver analysis of cooperation of self-interested agents with respect to disclosure of their plans. We focus on a fully distributed approach without the presence of any third-party mediator. Our approach models the situation as a non-zero-sum n-player game in extensive form. Our results can be divided into two parts: (1) the analysis of the maximal possible improvement of the utility value that an agent can gain by cooperation, and (2) the novel negotiation-based algorithm which enables the agents to find a better solution without the necessity to reveal complete plans to each other. The experimental evaluation is performed both on synthetic games abstracting real world scenarios and in our adversarial behavior testbed. First we analyze the options for finding generally more preferable plans based on domain parameters. Then the utility achieved by agents using the proposed cooperation method is compared to the utility value obtained without cooperation. We investigate the dependence of these values on the search depth, the number of players in the game, correlation of the players utility functions, etc.

The analysis on the synthetic game allowed us to easily vary the domain parameters and investigate the properties of the algorithm. The experiments from the adversarial behavior testbed confirm that the abstract game is a good model of cooperation in complex asymmetric game. We were able to observe the cooperation that leads to improvement of the utility of both negotiating agents also in this scenario. The detailed description of the proposed method and the experimental analysis is provided in Chapter 5.

1.2.5 Process Models for Opponent Modeling

While working on RT3, we have performed initial literature survey and analysis of requirements of an opponent model representation suitable for (semi)cooperative settings. In these setting, agents are assumed to collaborate on modeling common adversaries, share/trade the acquired opponent models and use the predictions of their private models within their temporary coalitions.

The results related to this problem are presented in Chapter 6. We have identified the formalisms for capturing and analyzing business processes (e.g. Petri nets) to be a promising language for this general task. The process description formalisms are more general then Finite State Machines often used for modeling intelligent behavior in research as well as industry. In addition to the power of FSM, they allow capturing team behavior, pursuit of multiple goals and other forms of concurrent behavior.

We have performed several proof-of-concept experiments concerning using processes to model behavior of intelligent agents in our adversarial behavior testbed. We have implemented an agent that logs selected events in the system into a general XML format usable by various process analysis frameworks. We have identified a publicly available software framework for automatic extraction of process models from such XML logs of events in a system. Using the standard methods available in the software, we were able to extract useful models of agents behavior. This supports our assumption, that the process description formalisms and the related tools are powerful enough to describe and extract models of agents in complex asymmetric games. We have also identified several limitations of the existing methods and suggested developing more specific methods that could overcome these limitations.

We did not continue any further in this research direction. We have not implemented the collaborative methods on top of process models of other agents' behavior in this project. We have decided to use most of the effort planned for this task to support collaborative effort on a slightly different topic, closely related to the objectives of this project. See Section 1.2.7 for further details.

1.2.6 Deception in Large Adversarial Scenarios

We study deception created by the adversary in order to minimize the efficiency of a team of mobile sensing agents. To our best knowledge, this is the first work that deals with deception in this setting. We employ a game theoretic model to analyze the expected strategy of the adversary and find the best response. More specifically, we consider that the adversary deceptively changes the importance that agents give to targets in the area. The opponent is expected to use camouflage and decoys in order to create confusion among the sensors regarding the importance of targets, and reduce the team's efficiency in target coverage.

We formally represent the Mobile Sensor Team problem based on the Distributed Constraint Op-

timization Problem (DCOP) framework. We extend the formulation of mobile senor team problem previously developed at CMU by possible deception from the adversary. We propose a method for selecting the optimal target to be covered by a single agent facing a deceptive adversary. This method serves as a heuristic for agents to select their position in the full scale problem with multiple agents in a large area. Our empirical study featuring teams of over a hundred agents demonstrates the success of our model compared with baseline solutions as well as existing models in the presence of deceptions.

In addition to the more applied results concerning deception, we have identified the fundamental problem of using stochastic local strategies in local search methods for solving DCOP. We provide the first local search algorithms for solving this variant of DCOP.

This work was a result of collaborative effort and some of the co-authors are not authors of this report. That is why these results are reported in form of a journal article attached to this report in Appendix A.

1.2.7 Discussion

The study of deception does not directly support any of the research task listed above. It broadens the formal conceptual framework from RT1 by formally defining the notion of deception and it partially supports RT4 by creating algorithms for explicit cooperation of large teams of agents against a common adversary. The focus on large cooperative settings in adversarial environment extends our research of explicit cooperation and scalability in adversarial reasoning. Moreover, it directly addresses also the third key challenge of adversarial planning identified in the project proposal – *uncertainty* in agents' observations.

We have decided to reallocate most of the research effort planned for cooperative opponent modeling to study of deception, because of its high relevance for the project and a unique collaboration opportunity. We managed to obtain a Czech government grant intended to co-fund a collaborative research effort between ATG and Intelligent Software Agent's Lab of Katia Sycara at Carnegie Mellon University. The main objective of the collaboration is to leverage the state of the art in the field of adversarial planning, which is well aligned with the objectives of this project. The additional funding allowed two three months long visits of a research team member and two one week long visits of the principal investigator at CMU.

This intensive collaboration allowed us to combine the expertise of our research groups and efficiently address more challenging problem than we originally intended. Together with our partner, we were able to deliver results of high quality and significance for the project. The main results of the first three months of this collaboration were published as a full paper on a premiere conference in the field of Autonomous Agents and Multi-Agent Systems – AAMAS 2010. This work was further extended and an advanced draft of a journal version of the paper is included as Appendix A of this report.

1.3 List of Publication Supported by the Project

The scientific results achieved during the progress of this project are part of the following publications:

– V. Lisý, R. Zivan, K. Sycara, M. Pěchouček: Deception in Networks of Mobile Sensing Agents. In Proc. of 9th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2010)
– V. Lisý: Adversarial Planning for Large Multi-agent Simulations. In Proc. of 9th Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2010)

The following paper is accepted for publication:

– V. Lisý, B. Bošanský, R. Vaculín, M. Pěchouček: Agent Subset Adversarial Search for Complex Non-cooperative Domains. In Proc. of the 2010 IEEE Conference on Computational Intelligence and Games (CIG 2010)

Beside these accepted papers, we have two more publications in the stage of advanced drafts, which will be submitted shortly. Namely, it is an extended version of the paper on deception, which is included in this report as Appendix A. We plan to submit it for the Journal of Autonomous Agents and Multi-Agents Systems in August of 2010. The second paper will be based on the interesting problem and results presented in Chapter 5 of this report.

Chapter 2

Extended Adversarial Behavior Testbed

This chapter describes the extensions to the adversarial behavior testbed originally implemented and presented in [3]. The testbed is used for performing the experiments with adversarial reasoning and planning algorithms in realistic adversarial domains. At first, we remained the basic scenario and rules of the game from [3], and we follow by enhancements made in this project to support the solved research tasks.

2.1 Original Game

In [3] the Tsunami recovery game was introduced. The game is a representative of a complex asymmetric game of n-players modeled after a disaster recovery operation in a politically unstable environment. There are three players in the game: government, non-governmental humanitarian organization, and separatists. Each player controls multiple units that are placed in the game world represented by a planar graph forming a network of roads. Any number of units can be located in each vertex of the graph and each unit can change its position to an adjacent vertex in one game move. Some of the vertices of the graph contain cities, which can take in commodities players use to construct buildings or to produce other commodities.

Some of the cities have been hit by a natural disaster. As the result, these cities lack infrastructure (indicated by the light gray number in the city status) and they do not have government head quarters (a blue GovHQ indicator). This means that the government has no control over the cities. The cities also need to be provided with food (the green number in the city status) by a non-government organization whose goal is to transport food from farms (a green Farm indicator) to cities that need it. The food in the cities is consumed over time. The cities controlled by government have a full-width blue bar shown next to them, the cities that are not controlled by government and have enough food have a half-width green bar, and a city without food has a small red bar.

In order to restore the control, the government has to first transport explosives from explosives factory (an orange Fact indicator) to a quarry (a yellow Quarry indicator) where they are used to

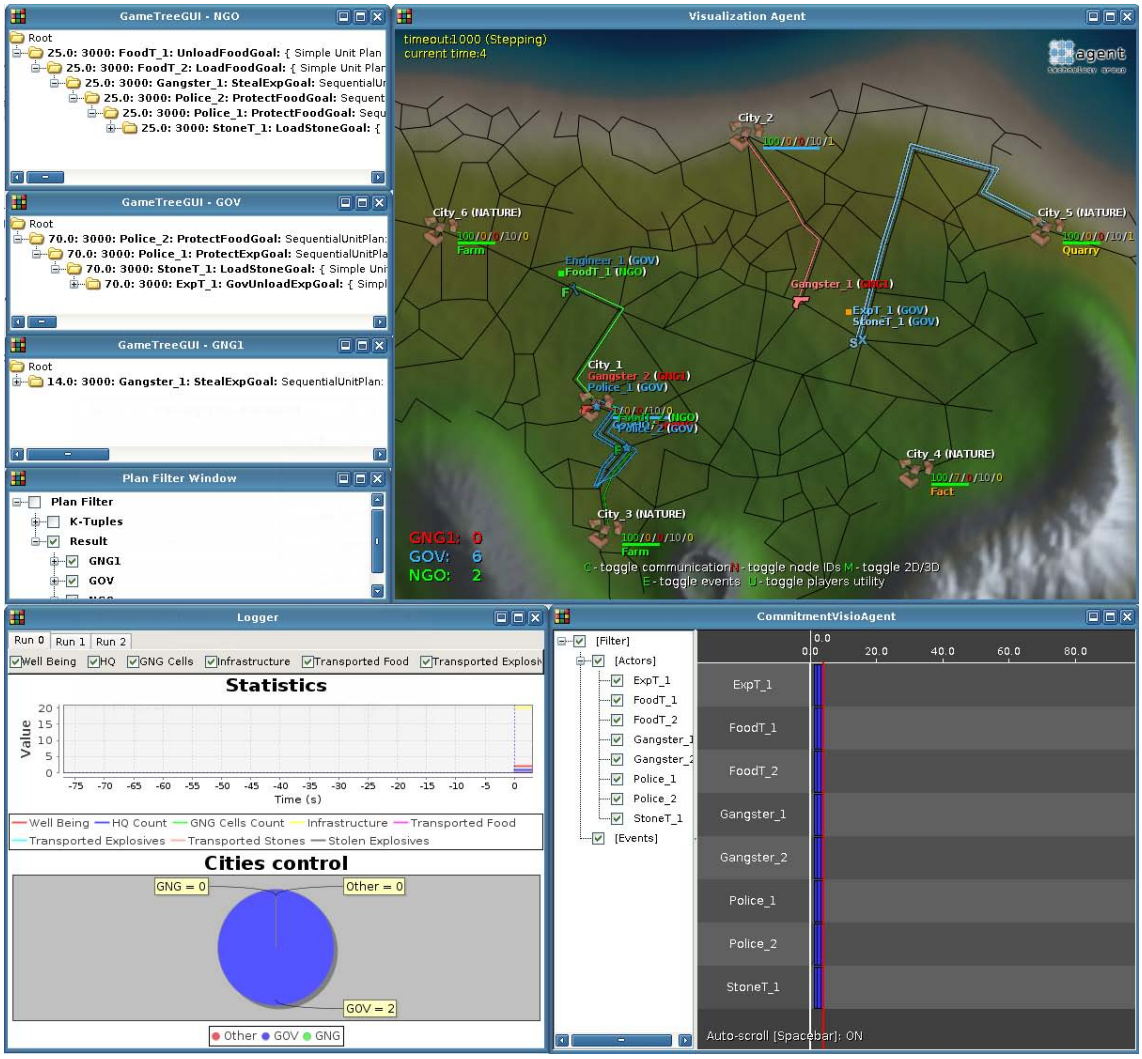

Figure 2.1: The screenshot of the games scenario in the adversarial behavior testbed.

produce stones that are later used in damaged cities by engineer units to repair the infrastructure and build the headquarters. The transportation of stones and explosives is performed by trucks.

However, there are separatists trying to prevent the government from achieving control in the cities. The separatists steal the explosives from the government trucks and store them in their separatist camps (the red GngHQ indicator). If they have enough explosives they carry out a suicide bomb attack that destroys the government headquarters in a chosen city. Moreover, the separatist can destroy the food supplies in food transport causing a riot in the city with food shortage that lead to destroying the government HQ.

The utilities representing the objectives of the players are expressed as weighted sums of components, such as the number of cities with sufficient food supply, or the number of cities under the control of the government.

2.2 Improvements

Several improvements in the testbed has been implemented in order to correctly evaluate the newly developed approaches to game playing and opponent modeling in complex asymmetric games.

2.2.1 Agent Subset Adversarial Search

Besides the implementation of the algorithm itself, there were only minor modifications in the testbed needed to evaluate ASAS. In comparison to the full GB-GTS algorithm, the ASAS method evaluates significantly less nodes in the game tree; hence it can be applied on a larger scenario. More specifically, we used a scenario with 9 operating units and 6 cities (see Section 4.4.1 for more detailed description of the scenario). The most significant change in the testbed was necessary as a part of the implementation of the sub-result merging algorithm of the ASAS method (see Section 4.3.3). The goals defined in the testbed and used by the GB-GTS algorithm were enhanced with explicit representation of the information about their preconditions and effects in terms of variables from the domain. This way we managed to split the domain when possible during the merging search. Note, that adding this type of knowledge into the goals opens a possibility for applying more advanced reasoning about goals into GB-GTS (e.g. as a heuristic function for selecting the first move to evaluate) in the future work.

2.2.2 Sub-Game Negotiation Tree Search

In the case of Sub-Game Negotiation Tree Search, we have implemented multiple enhancements of the testbed. First of all, the scenario needed to be changed in order to offer the possibility for negotiation between the players. The utility values for already existing players (government, non-government organization, and separatists) are mostly negatively correlated and the actions of the players do not offer a good potential for negotiation and cooperation.

Therefore we introduce a new separatist player that has utility values similar to the first separatist player. They both want to destroy the government HQs in the cities, but newly there is a difference

between who has the control over the city. In the original version of the game the city either has been under the government control or not. In the version of the game used for SGNTS evaluation, we differentiate between four states: government controls the city, first separatist controls the city, second separatist controls the city, and no player controls the city. Moreover, several rules of the game had to be updated – (1) for a player in order to gain control in the city by force, the number of its units has to be higher that the number of units in the same city owned by any other player. However, if there are e.g. two gangster units (one of each player) and one cop in one city, then the government looses the control, but no separatist player gains it. The rules concerning stealing and destroying commodities transported by trucks are modified in a similar way – if number of all gangster units of all separatist players is higher than number of cops protecting the truck, the stealing/destroying is successful – the stolen commodity is given to a separatist player with highest number of units.

The separatist players use a sub-game negotiation-based algorithm described in Section 5.3 and communicate with each other in order to improve their strategies in the game. As a result the separatist players can avoid a centipede-like situations in the game (see Section 5.4 for more specific scenario in the Tsunami recovery game, where such a situation can occur) while still preserving their self-interest nature of the players.

By implementing the separatist players willing to communicate and cooperate with each other we created more sophisticated opponents with more realistic behavior, which allow the separatist to take advantage of the asymmetric utility values in the domain.

2.2.3 MXML Logger Agent

Another improvement of the testbed is implementation of `LoggerAgentProM`, its data structures and the related system of logging the events in the game into the standard Mining XML file format. This log currently contains all the actions performed by the agent in the simulation and it can be easily extended to include also the event caused by the dynamics of the game, such as food consumption. Capturing the log in this general format allowed us to use publicly available state of the art framework for analyzing the log and extracting models of behavior of individual agents.

Chapter 3

Extended Formal Framework for Adversarial Reasoning

This chapter refines and extends the framework of concepts, methods and entities involved in adversarial planning introduced in [3]. The framework serves as a common reference and vocabulary for describing and comparing specific (components) of adversarial planning methods. First we introduce the formal definition of the domain, in which the agents operate. Then we focus on the information about opponent agents that is important in the process of adversarial reasoning. Afterwards we set this information into a generalized interaction among the agents as well as among the different kinds of knowledge within an agent.

3.1 Game World

In this Chapter, we use the same formalization of the game world as in our previous works [3, 2]. The domain can be formalized as a tuple $(\mathcal{I}, \mathcal{A}, \mathcal{W}, \mathcal{O}, \tau)$.

- $\mathcal{I}$ is a finite set of agents (players) – indexed by $1, \ldots, n$ – capable of performing actions in the world.
- $\mathcal{A}_i$ is a set of actions an agent $i \in \mathcal{I}$ can perform, and $\mathcal{A} = \mathsf{X}_{i \in I}\, \mathcal{A}_i$ is a set of joint actions where each $\vec{a} \in \mathcal{A}$ ($\vec{a} = \langle a_1, \ldots, a_n \rangle$, $a_i \in A_i$) denotes a joint action.
- $\mathcal{W}$ is the set of possible world states
- $\mathcal{O}_i$ is a set of observations for agent i and $\mathcal{O} = \mathsf{X}_{i \in I}\, \mathcal{O}_i$ is the set of joint observations where $o = \langle o_1, \ldots, o_n \rangle$ denotes a joint observation
- τ: $\mathcal{W} \times \mathcal{A} \to \mathcal{W}$ is the transition function realizing one *move* of the game, where the game world is changed via joint action of agents and the world's own dynamics.

The set of the world states is represented by values of variables $\{x_1, \ldots, x_m\} = D$, i.e., $\mathcal{W}$ is the set of all valuations of these variables. We denote the set of all valuations of a set of variables S by $\bar{S}$ (i.e., $\bar{D} = \mathcal{W}$). Each action $a_i \in A_i$ uses a subset $\mathsf{pre}(a_i) \subseteq D$ of variables in its preconditions and it modifies a subset $\mathsf{eff}(a_i) \subseteq D$ of variables by its execution. Finally, we define a plan of an agent i as a finite sequence of actions, i.e., $p = (a_1, \ldots, a_d)$, $a_j \in A_i$, $j \in \{1, \ldots, d\}$.

The game proceeds in moves. In one move, each player chooses one of its legal actions. Next state of the world and the information the players get about it is chosen according to the transition probabilities T.

3.2 Single Agent Models

In this section, we classify the basic minimal set of different types of knowledge that is necessary do describe behavior of single agent, without considering its interaction with other agents. The most popular paradigm for description and specification of agents behavior is the BDI model. The information about any of these basic concepts of agents' reasoning can be very useful in a multi-agent system.

3.2.1 Believes

The concept of *belives* is usually used for representation of agent knowledge for the purpouse of modeling reasoning and rationality. The agents can create believes about wide variaty of components of the world, such as state of the world, other agents' intentions, and inter-agent relations. Modeling other agents' believes is very useful especially in domains with partial information. For example in the military domain, the agent can gain a considerable advantage if he knows that the opponent is unaware of some of the agents' well positioned resources. However, the knowledge about the opponent's believes can be very useful even in the domains with full information about the game state. In that case we mainly deal with the believes in form of the social knowledge - an object and product of agents' opponent modeling processes. The opponent can create believes about other agents future actions that can be exploited or manipulated.

3.2.2 Desires

Agents' *desires* represent the agents preferences and top level goals. Knowing other agent's desires can help agents to anticipate future course of action of the other agents even without any communication. They can create expectation about whether the other agents will try to prevent them from achieving their goals or support them in their effort. The top level desires also define the potential of the agents to create coalitions.

There are two basic forms of explicit representation of agent's top level desires in agents literature. The one that is more usual in intelligent planning and BDI literature is the logical representation of a *goal* as a formula defining a set of states (or traces) that represent satisfaction of the goal. The satisfaction of such goal is strict. It is either satisfied or not and the notion of partial satisfaction is not well defined.

The second form of top level goals representation is more common in the Game Theory. It is a preference relation over the states (or traces) of the world. Generally, it is a partial ordering representing that the agent prefers one state of the world to another. This preference is often linear and represented by a mapping to the set of real numbers (e.g., utility theory [4]). In this case, the goal of the agent is optimizing its course of action to achieve as high preference value as possible, rather than satisfying a strict criterion.

The second approach is more general. If we define the preference value of the world states with the goal satisfied as 1 and the preference value of the remaining states as 0, we can easily represent the goals. This representation is suitable mainly for descriptive tasks and for that reason, it was also used as a base for the formal framework developed in the previous project [3]. The knowledge about agents desires is referred to as the declarative player model in the report. It assumes a mapping from the set of world state to vectors of real numbers

$$f : \mathcal{W} \to \mathbf{R}^n \tag{3.1}$$

describing several characteristics of the world states that influence how agents prefer individual world states. The actual utility of a state for the player is then defined as a linear combination of these characteristics with player specific weights.

This representation fits well the structure of players desires in the real world problems and allows us to define various interaction stance among the agents [2], such as adversarial behavior of one agent towards another.

Describing agent using (multi-valued) utility is very suitable for descriptive and analytic tasks, but without further refining of the mapping, it hides a lot of information that can be used for creating efficient game playing algorithms. E.g. it disallows using the methods based on heuristic forward search.

Moreover, the representation of the utility function as a task for adversarial planning by human expert would most likely need to have some structure. It is not possible to assign a vector of real values to each of the (possibly infinite) number of states. A reasonable representation would be to create a number of formulas describing various properties of the state and assigning each of them the vector of real values representing the influence of the formula to individual utility factors. For example, capturing a city can mean improvement of the military dominance factor, but it can mean also decrease in the amount of food supplies if the civilians in the city need to be fed. The utility of a world state is then a sum of all the satisfied utility formulas. Formally, we refine the definition of the mapping from formula 3.1 by firstly defining a set of utility formulas $\mathcal{U}$.

Definition 1 *We define the set of utility formulas $\mathcal{U}$, as a set of formulas of a language. Each of the formulas defines a subset of all possible world states in which it satisfied.*

Additionally, we define a mapping representing the additive influence of the satisfaction of the formula to individual utility factors.

Definition 2 *We define the* utility increment *of a formula to be a mapping*

$$v : \mathcal{U} \to \mathbf{R}^n$$

Using the previously defined notions, the function mapping world states to the vectors of real numbers can be refined as

$$f(w) = \sum_{\phi \in \mathcal{U} : w \models \phi} v(\phi) \tag{3.2}$$

The multidimensional representation of the utility is not only intuitive in representation of various measures that may be important for the agents, but it is also very expressive. It allows encoding arbitrary partial (preference) order among the states of the game, which is the most general representation of agents desires.

Lemma 1 *Any partial ordering among the states of the game can be expressed as a multi-dimensional utility function with natural ordering relation*

$$\vec{u} \leq \vec{v} \Leftrightarrow \forall_{i \in \{1,\ldots,n\}} u_i \leq v_i$$

Proof This fact is true even if we allow only binary vectors instead of real vectors. The authors in [5] show that lower semi-lattice can be encoded into a vector of bits with the natural ordering. In order to use this fact, it is sufficient to add an additional (virtual) state that is worse than all the other world states. ∎

The transformation to the multi-dimensional utility does not reduce the expressivity of the framework. However, the final linearization by individual agent preferences already creates a linear order among the world states. This might reduce the expressivity of the framework, because it is sometimes hard to aggregate incomparable measures, such as having something done faster or cheaper. On the other hand, the agent can always choose only from a limited number of actions and it has to choose only one action to perform at the end. Having two incomparable actions to choose, the agent has to come up with another criterion to choose the action to perform, or it really does not care about the ordering and hence it would be equally satisfied with any of the orderings.

The representation of the utility function as a set of formulas corresponding to vector utility values has several advantages. First of all, it is fully compatible with the previously defined notion of declarative player model [3] and player interaction stances [2]. Second, it allows heuristic guidance of the game playing algorithms. The agent can identify the utility formulas it would prefer to be satisfied (unsatisfied) and use their logical structure to generate heuristics that would lead its planning effort in a similar way as in the state of the art single agent planning systems [6]. Third, description of sets of states using logical formulae is fairly compact and hence suitable for sharing with other agents.

3.2.3 Intentions

Intentions in the BDI framework may be seen as agents internal commitment to implement an action [7]. In our opponent modeling framework developed in [3], it corresponds to the procedural player model, describing the probability distribution over players next actions based on the current state of the world and assumed internal state of the opponent.

$$P(A_i|\mathcal{W}, S_i) \tag{3.3}$$

The main question this kind of model answers about the opponents is what will their next actions be in certain state of the game. If other players in the game are cooperative, we can simply ask them a about their next actions and coordinate. In case of non-cooperative agents (opponents), we need to create a predictive model to efficiently react to their actions.

Reactive Agents The situation is simpler if we model purely reactive opponents that choose their next actions solely based on the current state of the game world. Even if the action selection mechanism is stochastic, the model that can fully capture it is a mapping from the current world state to the distribution over the applicable actions of the opponent.

$$P(A_i|\mathcal{W})$$

Complex agents in real world situations are rarely purely reactive. They build internal believes about the world and create longer plans that are executed over multiple time steps. However, even if the modeled agent is not fully reactive, approximating its behavior using a reactive model can certainly be useful. In [8, 9], agent models in form of $P(A)$ and $P(A|W)$ are used as automatically learned domain specific knowledge. This knowledge is used to guide node selection and play-out phases of the UCT [10] (Monte-Carlo based) adversarial search. The reported results show that even using the a priory action probability distribution $P(A)$ not conditioned by anything can focus the search in UCT and lead to a much more successful player. They achieved superior playing performance in simple games, such as checkers and larger versions of tic-tac-toe.

Agents with Internal State Even the simplest forms of models of agents action selection mechanism can be efficient in increasing playing performance and reducing computational afford needed for adversarial reasoning. On the other hand, the more precise and more expressive is the model of the opponents' behavior, the stronger is the reduction in computational effort needed to create good courses of action.

In order to create even more precise models of opponent behavior, we need to add the distinction between opponent's internal states. The opponent can choose different actions if it believes it is winning the game, or based on its current believes about the other agents strategies. This leads us to procedural opponent model in form of equation (3.3).

The specific forms of implementation of the model of intentions of agents with internal states are further discussed in Chapter 6.

3.3 Means of Agent Interactions

This section formally defines the concepts describing various levels of cooperation of (groups) of agents. It refines our previous work [2], defining adversarial, cooperative and self-interested actions solely based on agents utility function and actions they choose among all applicable alternatives. Here we take into account also the aspects that were not part of the earlier model, such as inter-agent communication and trust. We also extends the model of Social Reasoning in Computational Multi-Agent Systems by Pěchouček et al [7] that was designed mainly for cooperative multi-agent systems.

3.3.1 Awareness

One of the most basic requirements for an agent in order to intentionally interact with another agents is its awareness of the other agents existence. That is why [7] defines the total neighborhood of an agent.

Definition 3 *The* total neighborhood *of an agent i is a set of agents $\alpha(i) \subseteq \mathcal{I}$ that the agent is aware of.*

This notion is very important in reasoning about (adversarial) situation. The agent can pursue a completely different course of action if he is not aware of some reserves of the opponent.

Alternatively, its plan can collide even with activities of its (potential) allies if it is not aware of them and cannot coordinate with them.

3.3.2 Communication

Another key element in multi-agent interactions is communication among the agents. It is the main mean of interaction, but it is not always possible or desirable. The simplest example are purely adversarial situation, such as sports games, tactical military missions, or law enforcement operations. In these situations, the agents from different teams do not explicitly communicate and obtain all the information about the world and opponents actions only from their observations. Communication is often not desirable also in cooperative setting. The motivation for that varies from minimizing obtrusiveness of the system that assists/support the user [11], the need to keep radio silence in military operation or time critical (military) situations where explicit communication is too slow and distractive to relay on. In order to capture these situations, we extend the Social Reasoning Model by the notion of communication neighborhood.

Definition 4 *The* communication neighborhood *of an agent i is a set of agents $\kappa(i) \subseteq \alpha(i)$ that the agent is aware of and with whom he is able and willing to communicate.*

Note that as the total neighborhood of an agent, the relation of belonging to an agents communication neighborhood is not necessarily symmetric. The communication can be available only in one direction.

3.3.3 Environmental Coupling

In many (mainly adversarial) domains, the only mean of agent's interaction are their actions in a shared environment. For example in game of chess, the only interaction is via moves on the chessboard. This interaction is often available even in absence of communication. There are two kinds of environmental coupling. The *weak* environmental coupling is if the agents can observe the effects of the other agent's action in the environment and the *strong* environmental coupling is if one agent is capable of negating the preconditions or effects of actions of the other agent.

Definition 5 *The* weak environmental coupling neighborhood *of an agent i is a set of agents $\chi^w(i) \subseteq \alpha(i)$, such that*

$$j \in \chi^w(i) \Leftrightarrow \exists a_j \in \mathcal{A}_j, o_i \in \mathcal{O}_i \quad \mathsf{eff}(a_j) \cap o_i \neq \emptyset$$

Definition 6 *The* strong environmental coupling neighborhood *of an agent i is a set of agents $\chi^s(i) \subseteq \chi^w(i)$, such that*

$$j \in \chi^s(i) \Leftrightarrow \exists a_i \in \mathcal{A}_i, a_j \in \mathcal{A}_j \quad \mathsf{eff}(a_j) \cap (\mathsf{pre}(a_i) \cup \mathsf{eff}(a_i)) \neq \emptyset$$

3.3.4 Modeling Other Agents

In absence of communication and high dependence of the performance of agent's plans on the other agents, an agent can improve its performance by creating a model of other agents' behavior

and using them to predict its actions. We summarize the different kinds of knowledge that can be acquired about opponents in Section 3.2. Here we define the modeling neighborhood of an agent.

Definition 7 *The* modeling neighborhood *of an agent i is a set of agents $mo(i) \subseteq \alpha(i)$, whose models agent i acquires, maintains and/or uses in its reasoning process*

This neighborhood does not have to be disjoint with communication neighborhood of the agents. Even if the agents communicate, they may not be willing to share exact information about their future action.

3.4 Purpose of Interaction

After defining the main capabilities of agents that concerns the other agents, we can focus on the purpose of the agents interaction. In this subsection, we will try to answer the question: Why do the agents interact? Even though explicit interaction in more common in cooperative setting and interaction through modeling the other agents is more common in adversarial setting, the purpose of interaction is in general orthogonal to the means used for interaction.

3.4.1 Future Actions Coordination

When acting in a shared environment with other agents, a special form of non-determinism in outcome of agent's actions occurs. The actual outcome does not only depend on the actions chosen by the agent, but it can be strongly influenced by the actions of other agents. Any information about the future actions of other agents can help the agent to be more efficient.

Coordination among the *cooperative agents* is often necessary to achieve a persistent joined goal [12], which is a basic building block of cooperation among the agents. This notion is used to define the most important cooperative structures, such as notions of alliance, coalition and team [7]. Cooperative agents often need to coordinate, because their capabilities are not sufficient to achieve their individual goals on their own, or simply because they are a part of a team and their objective is to satisfy the team goal. An example of coordination in the interest of team objective allocation of individual agents for various tasks in disaster rescue scenarios [13]. Coordination of agents with different capabilities to achieve goals that would not be achievable by the agents on their own can be seen for example in many planning domains [14].

Coordination of actions is common also among *self-interested agents*. The efforts leading to satisfaction of agents own goals can be beneficial for all the participating agents. A clear example of this phenomenon is Air Traffic Control [15], where avoiding collision is clearly beneficial and the goals of individual airplanes are often disjoint. Rational self-interested agents are always willing to coordinate if it improves the expected utility of their plans. These situations are often studies in the field of Game Theory as pre-play communication. However, this phenomenon was previously studied in the context of simple matrix games. One of the contributions of this project is the analysis in the context of more complex extensive form games presented in Chapter 5.

3.4.2 Knowledge Sharing

A more complex form of agents interaction is knowledge sharing. The hard requirement for knowledge sharing is the ability of explicit communication. Agent i can share knowledge with agent j only if the following holds.

$$j \in \kappa(i)$$

Knowledge sharing is a more general form of agents' interaction then action coordination. The communicated knowledge can include also knowledge about future actions of an agent. The most commonly shared kinds of knowledge are following.

- knowledge about the actual state of the world
- background knowledge about domain properties
- knowledge of algorithms useful in the domain (closely related to reflection [16])
- knowledge about agents intentions and models of their behavior
- knowledge about inter-agent relations

The cooperative agents share their knowledge either as a default design decision or intentionally, in order to promote easier satisfaction of joined goals. Knowledge sharing among cooperative agents has been analyzed in [7]. Knowledge sharing in self-interested or adversarial setting is much less studied, but also not uncommon. For example, several self-interested (competitive) agents facing the same adversary can share knowledge about the model of the adversary in order to make it less efficient. Our group had previously studied this phenomenon on examples of logistics companies facing bandits attacking their transports [17], or a general abstract model learning tasks [18].

Deception

A special form of knowledge sharing is present also among adversarial agents. It is generally used to deceive the adversary to perform actions that are suboptimal. There are two basic form of deception, based on the capabilities of the agents that is being deceived.

Most real world agents are not perfectly rational, which means they have biases that can lead to accommodating false believes because of insufficient modeling of trustworthiness of knowledge sources. In this case, the goal of deception is to provide the other agent information that lead to creating false believes. In order for an agent to accommodate and use this kind of information, it usually has to believe that the information was not provided intentionally by the adversary, e.g. the information is provided by a third party. The notions of intentional and unintentional knowledge disclosure are define in [7].

Another option, why an agent might be willing to use the information provided by its adversary is if the adversary cannot completely prevent the agent from acquiring the information. For example, the agents can partially perceive the information by its sensors. Even these situations enable the use of deception by the adversary. If the adversary can partially influence the knowledge that will be available to the agents (such as importance of a target) it can use this capability to minimize the utility of the information. As a part of this project, we have studied this form of deception in the domain of sensor networks facing decoys and camouflaged targets. This research was conducted in cooperation with Carnegie Mellon University and the results are summarized in a draft of paper attached at the end of this report.

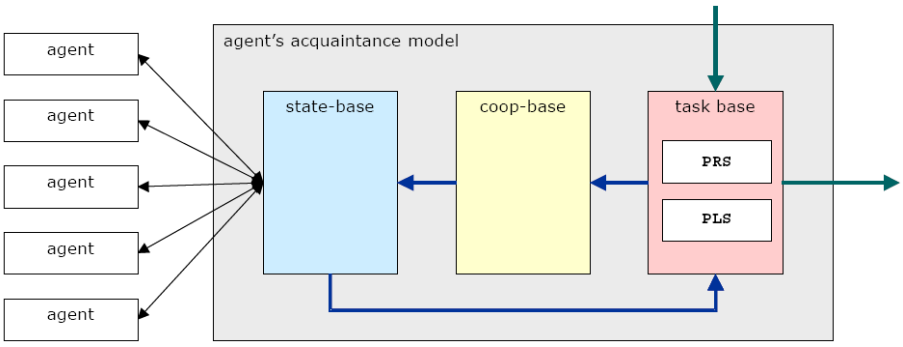

Figure 3.1: The schema of the tri-base acquaintance model.

3.5 Generalized Tri-base Acquaintance Model

The basic BDI model we use in the previous section is mostly used for describing the reasoning process of single intelligent agent without the need for extensive reasoning about other actions, their goals and capabilities. In order to focus on the collaborative aspects of multi-agent systems, more specific models of acquisition, maintenance and use of social knowledge must be used. One of such models is also the Tri-base Acquaintance Model (3bA) developed earlier at CTU. This model assumes mostly cooperative collective of agents and provides a formal framework decomposition, delegation and monitoring of collaborative tasks. We summarize the main ideas of the model in the next section and then we analyze the extensions necessary to use the model in setting with cooperative as well as neutral and adversarial agents.

3.5.1 Original Tri-base Acquaintance Model

The work on the 3bA model is quite extensive and it has found its applications in multiple domains. Here, we sketch only it basic ideas and the details can be found e.g. in [1]. The models divide the components needed for planning collaborative tasks in cooperative multi-agent system into three bases.

- **co-operator base (CB)** – maintains permanent information on cooperating agents (i.e.: their addresses, communication languages, and their predefined responsibilities). This type of knowledge is expected not to be changed very often.
- **task base (TB)** – stores in its *problem section* (PRS) the general problem solving knowledge - (i) information on possible decompositions of the tasks to be coordinates by the agents and (ii) in its *plan section* (PLS) it maintains the actual and most up-to-date plans on how to catty out those tasks
- **state base (SB)** – stores in its *agent section* (AS) all information on the current load of co-operating agents. This part of the state base is updated frequently and informs the agent who is busy and who is available for collaboration. In the *task section* (TS) there is stored information on the status of tasks the agent is currently solving.

The schema of the model is depicted in Figure 3.1. If the agent is planning a new task (top level goal), it chooses a suitable task decomposition rule from PRS. It identifies the optimal collaborators from CB and stores the created plan (e.g. task assignments) to PLS. These plans are then

gradually updated based on current information about the collaborating agents in SB.

3.5.2 3bA with Non-cooperative Agents

The 3bA model was developed mainly to facilitate collaboration among a group of agents in environments, where agents need to combine their capabilities to perform complex task, which would not be achievable by an agent on its own. This means that other agents in the system are assumed to either collaborate with the agent of not to interact with him at all. In many real world multi-agent systems, the interaction stances among the agents are more complex. We have analyzed and formally define all this options as a part of the preceding project [3]. We have identified the basic interaction stance as cooperative, competitive, self-interested, altruistic and adversarial. Agents having any of these stance towards the agent of interest[1] can influence applicability and efficiency of agent's plans. For this reason, the co-operator base needs to be extended to include also the information about the other (non-cooperating) agents. In the generalized tri-base acquaintance model (G3bA), we call the base storing the knowledge about the relevant agents in the system the *other agents base* (OAB).

The names of the other two bases remain unchanged and still represent the same basic kinds of knowledge. The state base stores the information about other agents' current states and the task base still represent the components related to the planning process of the agent of interest.

Other Agents Base

At least some of the other agents are cooperative in most settings. Therefore, OAB contains the same information as the original CB about these agents. However, there is much higher variety of useful information about other agents in the general (not necessarily cooperative) settings. The information in OAB is structured based on the neighborhoods defining the means of interactions with those agents. Each neighborhood implies the need to store a different kind of knowledge about the other agent. If an agent belongs to multiple neighborhoods, the knowledge corresponding to each of them is stored.

- *total neighborhood* – For all the agents in its total neighborhood, the agent stores a subset of data defined in the CB of 3bA

  $$\mathsf{OAB}(i) \supseteq \{< j, \beta(j) >\}_{\forall j \in \alpha(i)}$$

  where $\beta(j)$ represent the capabilities of agent j. The capabilities can be the set of tasks the agent can decompose (as in case of 3bA), but it can include even more general concepts, such as the set of actions that can be performed by the agent ($\mathcal{A}_j$). The capabilities are stored for whole the total neighborhood, because any interaction or reasoning about other agents includes its capability, regardless of its interactions stance of mean of interaction.
- *communication neighborhood* – For the agents in its cooperation neighborhood, the agent stores the remaining data from CB of 3bA

  $$\mathsf{OAB}(i) \supseteq \{< j, \mathsf{Addr}(j), \mathsf{Lang}(j) >\}_{\forall j \in \kappa(i)}$$

  where $\mathsf{Addr}(j)$ is the physical address where the communication to j should be addressed (e.g. IP address) and $\mathsf{Lang}(j)$ is the language in which agent j communicates. Note that the

[1] the agent using the proposed acquaintance model

data stored in these two structures do not have to be changed from the 3bA model, because they do not depend on the exact interaction stance among the agents.

– *modeling neighborhood* – For the other agents that are the object of modeling, the agent has to store their models. We have formally defined the different kinds of models that can be aquired about another agents in [3].

$$\mathsf{OAB}(i) \supseteq \{< j, \mathsf{OM}^{dec}(j), \mathsf{OM}^{proc}(j) >\}_{\forall j \in mo(i)}$$

where OM^{dec} is the declarative opponent model and OM^{proc} is the procedural opponent model. These models can be acquired either by communication with other agents of automatically extracted using the opponent modeling tools.

State Base

This knowledge base still contains the task section that stores the same information about the state of the active tasks of the agent. It contains the information about the progress of the task execution and the believe that the task will be accomplished. In particular, it reflects failure of execution of a task.

The agent section has to include additional information. Instead of more specific current capability and load of the agent from 3bA, this section of the state base includes a general information about the internal states of the other agents and the actual action that have performed

$$\mathsf{AS}(i) \equiv \{< j, \mathsf{State}(j), \mathsf{Action}(j), \mathsf{Trust}(j) >\}_{\forall j \in \chi^s(i)}$$

where the Trust has the same measure of trust in this piece of information as in 3bA. The State is a subset of the set of all internal states of agent j, which agent i believes might be the current internal state of agent j. The $\mathsf{Action}(j)$ is the current action performed by agent i.

Task Base

The task base is composed of the same two sections as in the case of original 3bA.

$$\mathsf{TB}(i) \equiv< \mathsf{PRS}(i), \mathsf{PLS}(i) >$$

The problem section (PRS) of the TB contains the fixed planning knowledge for the domain. It is some compact representation of the set of plans that an agent can use to achieve its goals. It directly corresponds to the domain heuristic (DH) defined in [3]. The plan section has to be more general then in 3bA. It includes the current intentions of agent i.

Dynamics of Reasoning with G3bA

Reasoning of an agent using the G3bA model is composed of three mostly independent processes running between each pair of bases in G3bA. The schema of the dynamics of G3bA model is depicted in Figure 3.2. In the following description, we always refer to the agent using G3bA from whose perspective we explain the processes as agent i.

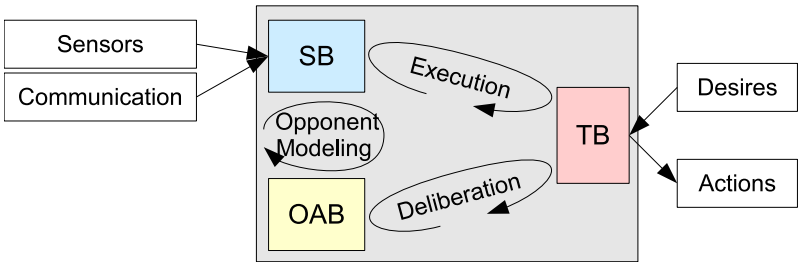

Figure 3.2: The schema of the dynamics in the generalized tri-base acquaintance model.

deliberation If the agent acquires a new high level goal it starts the *deliberation* process in the model. It identifies the set of suitable plans for achieving this goal in PRS section of the task base. These options are matched against OAB to find the most suitable collaborators that could support successful completion of the goal, but at the same time OAB is searched also for the agents that can negatively interact with the plan. Then there are two ways in which the deliberation may proceed. (1) For the agents are in the communication neighborhood of agent i, they can try to negotiate about a coordinated course of actions that would be beneficial for both agent i and the other agent. In case of cooperative agents, they can use any method of multi agent planning (MAP) to make it efficiently. If the agents are self-interested, the methods of MAP are not sufficient, but they can still coordinate using the SGNTS method developed in this project (see Chapter 5). (2) For agents that are in the modeling neighborhood of agent i, it uses their opponent models to reduce the set of possible plans to those that are likely to succeed. In this process, various algorithms developed within this project (such as GB-GTS, IROP, ASAS) can be used. If the most suitable plan is identified, it is stored in PLS of the task base of the agent.

execution The plans stored in PLS can be either executed immediately, if all the collaborators are available and all the potential adversaries are in an acceptable state, or it can remain in PLS until a suitable opportunity arises for execution of the plan execution. When the plan is executed, the progress of the subtasks, commitments of the collaborators and the threats from the adversary are monitored in the state base. It a plan execution fails of is situation changes so that it is not likely that the plan will succeed, the execution process updates the task base. It removes the plan that is no longer applicable from PLS. Moreover, it can produce a new high level goal that will trigger a new deliberation process.

opponent modeling The last process modifying the knowledge bases is opponent modeling. It creates the higher level generalizations of the continually changing information about the other agents. This information is then used for creating predictions about other agent and for consideration of the influence of their actions to the plans of agent i. We describe a specific method for extraction of OM^{dec} and the purely reactive version of OM^{proc} in the previous project [3] and we provide an initial study of acquiring more complex form of OM^{proc} in Chapter 6.

3.6 Language

Our original adversarial planning framework as well as the current extension defines most of the concepts by means of sets, sequences and relations. These definitions are very general, however,

if we want to use them in a specific application or reason about their higher level properties, we usually cannot represent them explicitly by enumeration of all the members of the set. We need a more compact representation with clear unambiguous semantics. Creating sound languages for representing complex phenomena is widely studied in the field of mathematical logics. Here, we summarize the most important requirements on a language, which can be used for representing the defined concepts. After that, we review the existing logics that satisfy these requirements to different levels.

In order to cover most of the concepts and phenomena present in our adversarial planning framework, we need a reasonably expressive language that is capable of compact representation of

- sets of world states
- future states of the world and action of the agents
- knowledge available to individual agents
- capabilities of coalitions of agents

Describing World States The world state is usually composed of a set of variables, which can be assigned values from finite domains. These represent for example the positions of situated agents or the amounts of available resources. The basic sets of the states of the world can be efficiently represented as the states of the world, where certain variable takes a certain value. Other basic propositions about the state of the world are (partial) relations between two or more variables of the world state. These relations are often represented implicitly by an externally computed function, but even explicit enumeration of the tuples that belong to a relation can often be much more efficient than explicit representation of the whole sets of world states. The basic propositions about the world states can then be combined to more complex structures using arbitrary logic.

The choice of the specific logic used in an application depends on the required properties, mostly the tradeoffs between expressivity of the language and the computational complexity (decidability) of reasoning about the language.

Reasoning about Temporal Properties The temporal relations between agents' actions, strategies, and goals achieved are a fundamental aspect of adversarial planning. We often need to express that an agent will perform a specific action in the next time step, that an agent will eventually reach its goal, or that under certain conditions, the adversary will never achieve his goal. All these properties can be expressed using temporal logics, such as CTL [19] and its extensions. The most common basic modal operators in these logics are $\Box$ "always", $\Diamond$ "eventually", $\bigcirc$ "in the next step", and U "until".

Reasoning about Knowledge In domains with partial information, we need to distinguish between the fact that a formula ϕ holds in a state of the world, and the fact that an agent i knows about it. Even in domains with full information about the state of the world, different agents can have different knowledge about internal states (e.g. knowledge) of other agents. None of the logical formalism mentioned above can express this distinction. These properties can be expressed in epistemic logic. A very good introduction to this subfield is [20]. The author describes the properties sketched above using modal operators $\forall i \in \mathcal{I} K_i$ and $\forall G \subseteq \mathcal{I} E_G, C_G, D_G$, with the intuitive meanin of $K_i\phi$ being that "the agent i knows ϕ". $E_G\phi, C_G\phi, D_G\phi$ means that "everyone in group G knows ϕ", "ϕ is common knowledge among the agents in G", and "ϕ is distributed knowledge

among the agents in G". These epistemic operators are then combined with the standard temporal operators creating a very expressive logic that satisfies most of our requirement. They use the logic to proof several fundamental properties of communication protocols, such as unreachability of common knowledge in case of unreliable communication.

This knowledge is powerful enough to express what certain groups of agents know together, but it cannot express what individual groups of agents can achieve in the system.

Reasoning about Coalitions An extension of CTL designed to formalize capabilities of individual groups of agents in a system is called Alternating-time Temporal Logic (ATL) [21]. This logic introduces to CTL an addition path quantifier $<< G >>$ parameterized by a subset of agents in the system. The meaning of the expression $<< G >> \phi$ is that the agents in G can cooperatively assure that formula ϕ holds, whatever the remaining agents ($\mathcal{I} \setminus G$) do. Formula ϕ is a temporal formula, so the language allows expressing many important notions concerning coalition, such as "in agents in G cooperate, all of them will eventually satisfy their goals". ALT has recently become very popular and it has been shown to be able to express many basic game theoretic notions such as Nash equilibrium [22] and it has been shown to be usable for defining and automatic checking of properties of game specifications in Game Description Language used by the General Game Playing Competition [23].

Knowledge and Coalitions The previous two logics were successfully combined in [24], creating Alternating-time Temporal Epistemic Logic (ATEL). This logic satisfies all of our main requirements and keeps model checking of formulas in the language tractable. Hence, we suggest using this language for expressing and proving more complex (derived) properties of domains and algorithms for adversarial planning. The only disadvantage of this language is its inability to express more fuzzy concepts, such as that an agent knows a fact with certain probability, or that a coalition can achieve ϕ with specified probability. However, probabilistic versions of ATL [25] as well as epistemic logics [26] has been studied and can be used for to describe creating specific aspects of the systems.

3.7 Conclusion

This chapter summarizes the conceptual view to the components and processes of planning and decision making in complex multi-agent settings. In general, the agents in these settings can have various interaction stances towards each other (e.g. cooperative, adversarial) and different capabilities. In the previous project, we have formally defined the notion of interaction stance and some of the basic kind of knowledge that describe behavior of individual agents. Here, we have refined the knowledge concepts of the individual agents and adapted the previously developed 3bA model to be applicable in adversarial setting.

Chapter 4

Agent Subset Adversarial Search

4.1 Introduction

One of the key factors prohibiting the application of the search-based methods in large scale multi-agent setting is the consideration of all possible interactions of all the agents involved in a game situation. Surely, the inter-agent interactions in game situations cannot be neglected, because they substantially change the development of the game. On the other hand, in many real-world situations, each agent interacts only with a small number of other agents. Each agent has its own specialization and its capabilities cannot be beneficial or harmful for objectives of all of other agents.

We have therefore developed a method that applies this insight to make search-based game-playing algorithms more efficient with a relatively small decrease of quality of produced strategies. The main idea of the presented approach is to substitute one global search in the search space of all agents with multiple sub-searches in the search spaces of small overlapping *subsets* of the agents, and use the information obtained through the sub-searches to synthesize a global solution. We term this method *Agent Subset Adversarial Search (ASAS)*.

We show that if the size of the subsets is limited, the ASAS method can be polynomial in the number of agents involved. The ASAS method is generic and it can be used with multiple known search-based game playing algorithms, such as max^n [27], or its extensions (e.g., [28], [29]).

The major contribution of the ASAS method is a substantial improvement of the search efficiency with a relatively small decrease of quality of produced plans without requiring any background knowledge about the domain. The method is suitable for domains where the most of significant changes in the state of the game need only relatively small number of agents to interact. These changes can be either of positive or negative nature for an agent, i.e., some agents want to achieve the change which other agents want to prevent. In the ASAS method, the agent subsets do not have to be defined explicitly. Instead, the “correct subsets” leading to high-quality plans emerge from the sub-searches for many different subsets.

In this chapter we firstly introduce preliminaries in Section 4.2 following by presentation of the ASAS method in Section 4.3. Section 4.4 presents an experimental evaluation of the ASAS method. Section 4.5 reviews the related work and Section 4.6 concludes the chapter and outlines

possible future directions of following research.

4.2 Preliminaries

The problem tackled in this project is to determine a suitable course of actions (a *plan*) for an agent in complex multi-agent environments that can be modeled as *n-player non-zero-sum games*. We consider games, in which all agents are self-interested, i.e., each agent maximizes only its own utility. We focus on the environments whose complexity prohibits the application of classical game-playing algorithms based on the exhaustive state space search because of its prohibiting size.

In the following, we use the game domain definition from Chapter 3 with one addition. For simplicity, we assume that the agents' desires can be represented by a real valued utility function and we add to the game definition:

– $\mathcal{U} = (u_1, \ldots, u_n)$ denotes the global utility function, where $u_i : \mathcal{W} \mapsto \mathbb{R}$ is a real-valued utility function for player i on world states $\mathcal{W}$.

We also assume full observability of the game domain; hence we do not use the set of agents' observations $\mathcal{O}$ in this chapter of the report.

We already mentioned that the main idea of the ASAS method is to substitute the global search by multiple sub-searches in small agent subsets. We employ the notion of *plan independence* in splitting the whole set of agents into subsets and, later, for merging the partial solutions into a global solution. For simplicity, assume that we have two agents 1 and 2. Let $P_1 = \{p_1, p_2, \ldots, p_l\}$ denote the set of some plans of the first agent, where $p_i = (a_1^i, \ldots, a_d^i)$, $i = 1, \ldots, l$ are plans composed of actions of the first agent. For such plans of agent 1, we define the sets of variables occurring in the plans preconditions and effects as follows:

$$\mathsf{pre}(p_i) = \bigcup_{j=1}^{d} \mathsf{pre}(a_j^i) \ , \qquad \mathsf{pre}(1) = \bigcup_{i=1}^{l} \mathsf{pre}(p_i) \tag{4.1}$$

$$\mathsf{eff}(p_i) = \bigcup_{j=1}^{d} \mathsf{eff}(a_j^i) \ , \qquad \mathsf{eff}(1) = \bigcup_{i=1}^{l} \mathsf{eff}(p_i) \tag{4.2}$$

For the second agent, P_2, $\mathsf{pre}(2)$ and $\mathsf{eff}(2)$ can be defined in a similar fashion.

Definition 8 *We say that sets of plans P_1 and P_2 of agents 1 and 2 are* independent *iff*

$$(\mathsf{pre}(1) \cup \mathsf{eff}(1)) \cap \mathsf{eff}(2) = \emptyset \ \ \& \ \ (\mathsf{pre}(2) \cup \mathsf{eff}(2)) \cap \mathsf{eff}(1) = \emptyset$$

Agents' plan independence, as illustrated in Figure 4.1, expresses that the plans of agents 1 and 2 cannot interfere. Clearly, if plans of agents 1 and 2 are not independent, the success and the utility of execution of such plans of one agent can depend on the plan that is selected by the other agent. On the other hand, if plans of agents 1 and 2 are independent and if P_1 and P_2 contains *all possible plans* of agents 1 and 2 respectively, we can be sure, that agents 1 and 2 can never

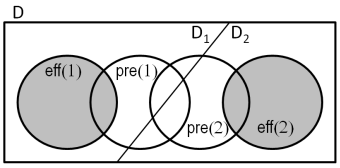

Figure 4.1: The general case of inclusion of the variable sets defining plan independence.

interfere. Therefore, in such a case, the search for an optimal plan for agents 1 and 2 can be performed independently.

If more than two agents are considered, we are interested in dividing the set of agents to disjoint subsets, such that the plans for the agents in each subset can be computed independently from the plans of the agents in any other subset. We use the notion of pairwise agents' plans independence to generalize the agent's plan independence to agent subsets.

Definition 9 *Assume an undirected graph $G = (I, E)$, where the set of nodes corresponds to the individual agents and the edge $\{i, j\}$ is present if the plans of agents i and j are not independent. We call G the* graph of agents' plan dependence*.*

If graph G is not connected, then the subsets of nodes corresponding to individual connected components of the graph create a decomposition of the set of agents to the parts that can be evaluated independently.

The benefits of decomposing the global game into subgames can be expressed in terms of computational complexity of the search process. Let n be the number of agents in the game, b be the number of different actions applicable by an agent in single move (branching factor), and d be the desired look-ahead, i.e., the number of moves in the future, which we want to consider for determining the plan. If we consider a game progressing in time with simultaneous actions of all agents, the size of the search space is approximately

$$b^{n*d} \tag{4.3}$$

If the set of all agents can be decomposed into disjoint independent subsets, such that the optimal plan of the agent we plan for depends on plans of at most $(k-1)$ other agents in the game, the complexity of planning for the agent becomes b^{k*d}.

The major problem of the described notion of agents' plan independence is that in realistic environments we almost never encounter the situation of complete agents' plans independence. In other words, while in a small local neighborhood of an agent the agents' plans independence would be a reasonable criterion for splitting agents into subsets, if a more global planning perspective is needed, the agents' plans independence cannot be used directly.

4.3 ASAS: The Method

The ASAS method relies on the observation that explicit coordination of *all* agents in the game is often not necessary in order to find high-quality plans that reach agents' goals (e.g., in some domains the agents are specialized and interact only with the agents of similar role and/or task

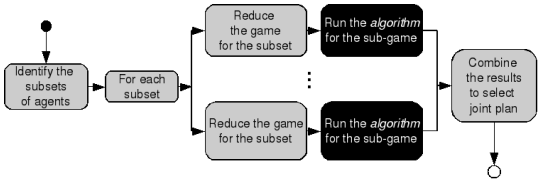

Figure 4.2: The scheme of the ASAS method using a generic search-based algorithm.

group). Therefore, the set of all agents can be decomposed into small subsets, and planning can be performed independently for each subset. We use the term *active agents* for such agents that are members of the subset for which the sub-search is currently performed. The remaining agents, which are not in the subset, are called *inactive agents* (with respect to the specific sub-search).

The ASAS method is designed to be combined with some generic search-based game playing algorithm, such as max^n , and tries to find the *optimal plan*, i.e., the one that would be found by underlying search algorithm without using the ASAS method. Figure 4.2 shows the overall scheme of the method, which in essence consists of the following steps:

1. **Subsets selection**: Create a set of sub-games involving a small number (k) of active agents in each sub-game (Section 4.3.1). These subsets of active agents do not need to be independent and can be overlapping.
2. **Sub-searches**: For each sub-game use the underlying search algorithm to compute the plans which the active agents should pursue in the sub-games. In sub-searches the problem arises of how to reason about the game development without considering the inactive agents (Section 4.3.2)
3. **Merging**: Combine the results from the sub-searches into a single global solution (Section 4.3.3)

An agent uses ASAS in the same way as the underlying adversarial search algorithm. It computes its optimal plan centrally (on its own), without interaction with other agents, based on his model of the game and the utility functions of all agents.

4.3.1 Subsets Selection

There are several possibilities of how to choose a set of subsets in the first step of the algorithm. A rather simple method is to consider all possible subsets of k agents. Even such basic selection method reduces the search complexity substantially. The reason stems from the complexity of computing plans in all sub-games. If all possible subsets of k agents are considered, the number of such subsets is given by the binomial coefficient. Hence, the sum of sizes of all the sub-search spaces can be approximated (with respect to formula 4.3) as

$$\left(\begin{array}{c} n \\ k \end{array} \right) b^{k*d} < (n^k) b^{k*d} \tag{4.4}$$

Note that this number is still polynomial in the number of agents in the game (n) and it form the upper bound on complexity of the method.

Alternatively, different (more sophisticated) approaches could be used for subsets selection, in order to decrease the number of subsets that needs to be explored. For example, the agents'

plan dependence graph as defined in Def. 2 can be employed. Using this graph, such minimal set of subsets of k agents can be selected, for which each clique of size k or smaller is included in at least one of the subsets. This would be less than the number of all subsets of k agents if the graph is not complete.

Although selecting only some of the subsets would further reduce the search space, we use all subsets of specified size as a proof of concept in our experiments, which still allows substantial computation time reduction.

4.3.2 Reasoning about Inactive Agents

Another key issue is what to do with the agents that are not currently active in the given sub-search. We consider the following three options.

(1) Removing inactive agents from the environment can be used in domains where pure presence/absence of an agent does not substantially change the environment state. Such situations may occur, e.g., in domains, where some form of *locality* (e.g., geographical, functional) does not allow many agents to interact with each other even when a long look-ahead is considered. In our domain, as in many adversarial domains, this is not the case. For example, removing a military brigade from an unstable region would change the game completely, and plans optimal for such situation would not make sense in the game with all agents.

(2) Using predefined heuristic behavior. Heuristic behaviors, employing background knowledge of the domain, can represent tasks for inactive agents that are necessary for reasonable consideration of the future development. Conservative heuristic behavior of inactive agents will allow more realistic assessment of future game states during sub-searches. However, creating heuristic behaviors manually for all agents and all situations is non-trivial and time consuming.

(3) Reusing results from the previous search: Instead of creating the heuristics manually, the results from the *previous search* can be used as a heuristic in the current search. Specifically, for every inactive agent we use its best plan resulting from ASAS for the agent in the previous execution of the algorithm (usually in the previous move) as a heuristic approximation of its behavior. Since such plans are sequences of actions (the length corresponds to the size of the planning look-ahead), these plans can be used as a heuristic in the search for several subsequent moves, and such plans will most likely still be reasonable. If the past plan for some inactive agent is not applicable at a certain time because of a major deviation from the past game state development, the inactive agent will move randomly further on. Importantly, random moves have to be used mostly only in the later stages of the search, hence the effect of random moves does not affect the outcome of the sub-search for active agents too much.

For our experimental domain evaluation we follow the third option, which showed to suit best the used domain.

4.3.3 Sub-results Merging

The purpose of merging sub-results is to select the final plan for each agent based on information found in sub-searches. For each subset the sub-search produces:

Input: W: current world state, I: the current set of agents, D_{util}: the part of domain to compute utility, $P[I]$: the set of plans to choose from for each agent
Output: an array of values of the world state (one value for each agent)

Compute the graph of agents' plan dependence G
if G is not connected **then**
  $\vec{V} = \vec{0}$
  for G_i connected component of G **do**
    Let I_i be the agents in component G_i
    Let D_i be the domain partition corresponding to G_i
    $\vec{V} = \vec{V}+$ Merge($W|(D_i \cap \mathsf{pre}(I_i))$,$I_i$,$D_i$, $P[I_i]$)
  end for
  return $\vec{V}$
end if
for agent i in I **do**
  Let $First_i$ be the set of all actions that are first at some plan from $P[i]$
end for
for $\vec{a} = (a_1, a_2, \ldots, a_n) \in \mathsf{X}_{i=1}^{n}\, First_i$ **do**
  $curW$ = $\tau(W, \vec{a})$
  for agent i in I **do**
    $P'[i] = \{p \in P[i] : p$ starts with $a_i\}$
    Remove first action from each plan in $P'[i]$
  end for
  $ActionValues[\vec{a}] =$ Merge($curW, I, D_{util}, P'[I]$)
end for
return ValueBackup($ActionValues$)

Figure 4.3: Merge(W, I, D_{util}, $P[I]$) – the main procedure plan merging algorithm.

- an optimal plan for each agent in the sub-game
- a resulting utility achievable in the sub-game if all active agents apply the plans optimal for the subset
- other characteristics of the resulting plans, such as their resource requirements, preconditions, and effects

If the plans generated in some sub-game are used in the complete game, conflicts between plans originating from different sub-games may occur, and the utility of the resulting game may not be the same as in the sub-game. We developed two merging techniques that address this issue. The *search-based plan selection* achieves better performance, but re-introduces the exponential dependence of computational complexity on the number of agents. Alternatively, the *rule-based plans selection* approach guarantees the polynomial time complexity, but the quality of the selected plans in lower.

Search-based plan selection

In essence, the search-based plan selection tries to find the best possible combination of plans (1 plan for each agent) by exploring all combinations of agents' plans that were found as the best plan in at least one of the sub-searches. The combinations are explored by searching the search space (game tree) that is generated only by the plans of each agent produced by the sub-searches. The algorithm realized by a recursive method showed as the pseudocode in Figure 4.3,

consists of two parts:

1. the *split* part (lines 1–10) partitions the set of agents and their corresponding plans found in sub-searches into independent agent sets (Defs. 1 and 2), for which the best combination of plans can be found separately,
2. the *selection* part (lines 11–22) considers every partition from the split part and explores all plan combinations for the agents of the partition by simulating the plans simultaneously in the shared environment and by computing the utility of every such combination of plans.

The combination of plans with optimal utility values in the sense of the underlying search method equilibrium is returned.

While we discussed the fact that agents' plan independence properties are very unlikely to hold globally for all possible agents' plans, these properties are very useful in the *split* part of the *Merge* procedure. In particular, the set of agents for which we evaluate different combinations of plans, can be split in case their remaining actions in plans are independent (in terms of Def. 2) and evaluated separately. Note, that the *split* part is an optimization of the overall search-based plan selection method that can be omitted.

In order to find the best combination of plans in the *selection* part of the procedure, one of its plans is selected for each agent, and the first actions of such selected plans are executed simultaneously (lines 14, 15). Since some of the plans of each agent can have a shared prefix (i.e., start with the same sequence of actions), as an optimization, the procedure always considers plans with shared action prefixes instead of considering only single plans at a time (lines 17, 18). The *Merge* procedure recursively traverses the tree structure formed by the shared prefixes in depth-first manner. With each descent in the tree, less and less candidate plans share the same prefix with the already executed actions, i.e., the sets $P[i]$, and the variable sets $\mathsf{pre}(i)$ and $\mathsf{eff}(i)$ are getting smaller. Subsequently, in each recursive descent of the procedure the *split* part is more likely to produce an independent partitioning of the agents into smaller subsets.

In the recursive call on line 7, the term $W|S$ means the world state W restricted to the variables in set S. The vector $\vec{V}$ represents utility values of optimal plans for each agent (not just agents in I_i), and the plus sign means vector addition. Function τ on line 15 is the state transition function. The *ValueBackup* function computes the value of the best combination of agents' possible actions (computed in the leaf nodes of the tree traversal), in the same way as the underlying search algorithm of ASAS. The utility values for each action are stored in the $ActionValues$ vector. For the sake of simplicity, the presented algorithm does not include the structures and procedures to store the plans that correspond to the selected solution.

In order to use the *split*, we assume that the utility functions of agents are *decomposable* with respect to the domain D. It means that the value of each utility function u on D can be obtained by computing some derived functions on arbitrary partitions of the domain and then combined at the end, i.e., $\forall D_1, D_2$ s.t. $D_1 \cup D_2 = D$, $D_1 \cap D_2 = \emptyset$ $\exists u_1 : \bar{D}_1 \to \mathbb{R}, u_2 : \bar{D}_2 \to \mathbb{R} \quad (u = u_1 + u_2)$.

The decomposability property of the utility functions is needed in order to allow the utility of the plans to be computed on subdomains of D in the *split* part. Specifically, for independent plans, each agent can evaluate its options independently just on the subset of the domain that is relevant for him $(\mathsf{pre}(i) \cup \mathsf{eff}(i))$. However, if we need to know the utility of performing the plans, working with just the sets pre and eff might not be sufficient. The utility function must be computed on the complete domain. That is why the agents split the domain to two disjoint parts D_1, D_2, so that $D_1 \cup D_2 = D$, such that $\mathsf{eff}(1) \subseteq D_1$ and $\mathsf{eff}(2) \subseteq D_2$ (see Figure 4.1). The agent 1 evaluates its plans on domain $D_1 \cup \mathsf{pre}(1)$ and computes the utility only on D_1. The agent 2 computes the

utility on D_2 in a similar way. The final utility is then just sum of the utilities computed by individual agents thanks to the assumption of decomposable utility.

With respect to the complexity of the *Merge* procedure, in the phase of sub-searches, each agent is active in $\binom{n}{k-1}$ searches. In the worst case, each sub-search can produce a new plan for the agent, i.e., the final search tree would contain at most $\binom{n}{k-1}^n$ leaves. Although this number does not depend on the look-ahead factor (see formula (4.3)), it re-introduces the exponential dependence on the number of agents. However, our experiments show that for an agent the same plan often results from multiple sub-searches, which reduces the size of the search space.

Rule-based plan selection

In order to avoid the possibly exponential time complexity of the search-base plan selection, the rule-based plan selection process introduces simple rules for selecting final plan for each agent out of all best plans found in corresponding sub-searches. The rules can employ various domain-independent factors of plans in order to select a good one, such as:

- **Utility**: the maximal absolute value of the plan utility, or the maximal average utility of the plan (one plan can be found in several sub-searches with different utilities)
- **Occurrences**: the number of occurrences within the candidate plans, i.e., the plan that was returned from most sub-searches for an agent will be selected.

Such rules can be determined automatically, e.g., by using machine learning algorithms on a training set created by running the full search algorithm as well as the sub-searches for all the agent subsets on a set of scenarios. However, our focus in this project is not on the learning methods, therefore we have constructed several rules combining both of these ideas manually and used them in our experimental evaluation to assess the feasibility of this approach (see Section 4.4).

4.4 Experimental Evaluation

The ASAS method can be applied on top of various search algorithms. For the experimental evaluation, we use our goal-based game-tree search (GB-GTS) [29] developed in the previous project as the underlying search method.

The results for the rule-based plan selection method presented in this section were obtained by the heuristic rule that chooses the plan that is maximal in lexicographical order given firstly by the average utility value of the plan and secondly by the number of occurrences of the plan within the set of all candidate plans. This rule was the most successful rule from all the fixed rules we considered.

4.4.1 Experimental Scenario

For the experimental evaluation we used a scenario in the Tsunami Recovery Game. Figure 4.4 depicts the schema with all important details. The scenario includes 6 cities, 2 of which can be

Figure 4.4: One of the sample situations from the experimental scenario. Lines represent the edges in the graph, agents are represented by appropriate symbols in the vertex, and cities are shown as a group of buildings in the vertex.

controlled by the government as well as the separatist player (City 1 and City 2). At the beginning of the scenario, both of these cities are under the control of the government player. Government headquarters (HQ) that help assuring government's control over a city are build only in City 1 and the food reserves in this city are very small.

There are 9 units operating in this scenario – two food trucks delivering food to City 1 from two farms (Cities 3 and 6), a truck transporting explosives from City 4 to City 5 (the explosives are used to mine stone in City 5), a truck transporting stones from City 5 to City 2 and City 1 if they do not contain HQ (in order to build it). Next units are: an engineer that uses the stone to build HQs, two police cars that can escort trucks and gain control in cities, and finally two gangsters that can either gain control for separatists in a city without HQ, destroy food in order to cause riots that lead to the destruction of HQ in the city, or steal explosives and create a suicide bomber in their cell in City 1. The suicide bomber is an agent that can destroy HQ by a suicide attack. There are many possible runs of this scenario. For example the police agents have to deal with several threats (e.g. protecting two food transports, explosives transport, and control of City 2). The gangster agents try to gain control of the cities they do not control and the humanitarian organization team tries to provide supplies efficiently without being robbed.

In our experiments, we compare the ASAS method with the original search method (GB-GTS in our case) without the ASAS being applied (termed *full search*). We measure the complexity (expressed as the number of explored search nodes), and optimality of the found plans. In our basic evaluation scenario, the game is played on a graph that consists of several hundred nodes. Thanks to the background knowledge in GB-GTS, the average branching factor of each agent is approximately 2.5 (minimal branching factor is 1, maximal is equal to 4), and each agent has to decide about it next course of action approximately every 5 moves.

The experiments were performed for different size of subsets $k = 1, \ldots, 4$, different look-ahead values $d = 8, \ldots, 16$, and different number of agents present in the scenario $n = 7, \ldots, 12$. The lower bound of the look-ahead was chosen as a minimal look-ahead that produces reasonable behavior in our scenario. The upper bounds were given by a practical limit of computational resources, especially when comparing to the full search. In each experiment, we ran the game on 42 situations based on the described scenario.

4.4.2 Search Reduction

We evaluate *search space reduction* by comparing the number of search nodes explored by the ASAS method and by the full search. In results regarding the ASAS method, we distinguish between the nodes evaluated in the sub-searches phase and the nodes evaluated during the search-based plan selection. The number of nodes evaluated in the sub-searches is the same for both plan-selection methods, however for the rule-based plan selection, this number is the final one. For the search-based plan selection, the overall complexity equals to the sum of nodes explored in the sub-searches and the nodes explored in the plan selection search. Furthermore, the complexity of computations performed in nodes of sub-searches might differ from computations in plan selection search nodes. In the presented experiments, the complexity of computation of all three kinds of nodes is the same, which allows us to compare these nodes and to depict them in the same graph.

The search space reduction for varying look-ahead values is shown in Figure 4.5. For the ASAS method, the bars show the overall sum of all nodes explored with the search-based plan selection method applied, while the number of nodes evaluated only in sub-searches is highlighted by the

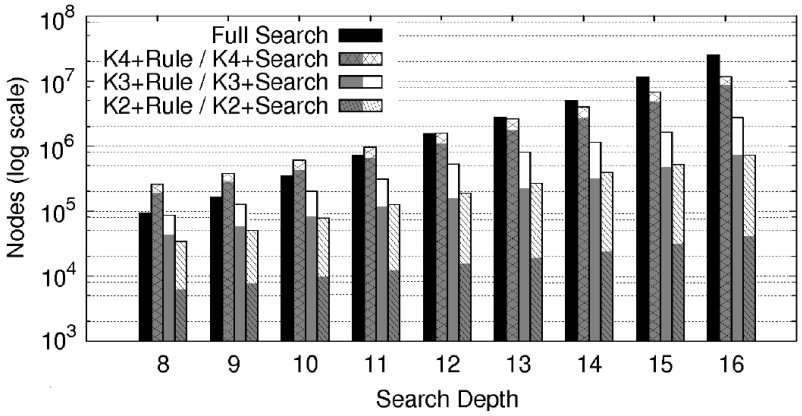

Figure 4.5: The number of nodes explored by the full search compared with the ASAS method for different sizes of agent subsets and various look-ahead values. The nodes axis is in the logarithmic scale.

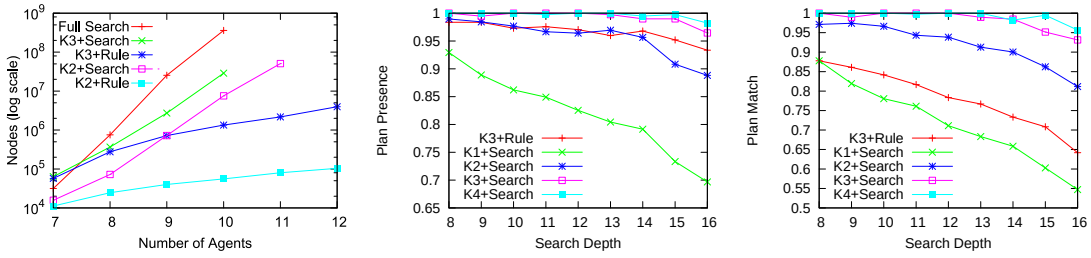

Figure 4.6: (a) The number of nodes (the axis is in logarithmic scale) explored by the full search compared with the ASAS method for different number of agents, fixed look-ahead (16), and different size of subsets $k = 2, 3$; (b) The percentage of optimal plan existence for all agents among the candidate plans resulting from the sub-searches in ASAS for various look-ahead values; (c) The percentage of optimal plans returned by the ASAS method for all agents for various look-ahead values.

gray filling.

As expected, the search space size grows exponentially with the increasing look-ahead in all cases (note that the figure scale is logarithmic). However, the number of nodes explored in sub-searches is a rather small portion of the overall number of explored nodes. For example, for the look-ahead 16 the full search explored $25,429,065$ nodes, while the ASAS for subsets of size $k = 3$ with search-based plan selection method explored $2,728,847$ nodes (i.e., 9.32 times less than the full search) and the ASAS with the rule-based plan selection explored only $722,743$ nodes (i.e., 35.19 times less than the full search).

The search space reduction for varying number of agents (i.e. units in the game scenario) is depicted in Figure 4.6(a). In this experiment we evaluated the dependence of complexity on the number of agents involved in the search. For the look-ahead 16 and two different sizes of subsets ($k = 2, 3$) we measured the number of evaluated nodes. For the full search and the ASAS with search-based plan selection, the number of nodes grows exponentially (the figure scale is again logarithmic). However, the number of nodes evaluated in sub-searches (i.e., if the rule-based plan selection was used) remains polynomial in the number of agents.

4.4.3 Accuracy: Preservation of Optimal Plans

Next, we focus on the accuracy of the ASAS method. We show that in spite of the significant reduction of the search space, the ASAS algorithm finds the same solution as the full search in most cases. We evaluate the accuracy by matching the plans returned by both methods for each agent. Two plans are matching, if they are exactly the same. This criterion is stronger than just comparing the utilities of returned plans – matching plans imply matching utilities. We use plans matching since it captures very well the loss of quality of plans returned by the ASAS method as compared to plans found by the original adversarial search algorithm.

First, we measured the *presence rate*, i.e., how often the optimal plans (the plans found by the original full search algorithm) are present among the candidate plans found by the sub-searches. The presence rate corresponds to the maximal possible precision of the ASAS method in case we had an ideal plan selection strategy (i.e., the one that always selects the optimal plan from sub-search results if it is present). However, in our case the presence rate is also influenced by the used plan selection method, because the results of sub-searches depend on behavior of inactive agents, which in our case are approximated by reusing the plans from previous searches (see Section 4.3.2).

Figure 4.6(b) depicts the presence rate for varying look-ahead and different size of agent subsets k. We emphasize the search-based plan selection method, but for a comparison we performed the experiments with heuristic rule-based plan selection as well. With the increasing look-ahead, the presence rate gets worse, which is due to the exponential expansion of the search space. However, for $k = 2$ the presence rate is almost always more than 90%, and for higher values of k the presence grows towards 100%. Specifically, for $k = 3$ the presence rate is almost always more than 95%. The presence rate for the rule-based plan selection heuristic is slightly worse.

Next, we analyzed the overall accuracy of the ASAS method. For each agent, we compare the plan selected by the particular selection strategy with the plan assigned to the agent by the full search. The results depicted in Figure 4.6(c) show that for subsets size $k = 3$ or $k = 4$ and the search-based plan selection we obtain results almost as good as the in full search. For $k = 3$ the accuracy is always higher than 95% for all evaluated look-ahead values. For the rule-based

plan selection, the results are worse than search-based plan selection, however, still applicable in real game-playing situations. For both $k = 2$ and $k = 3$ we obtained the accuracy of 64% for the look-ahead of 16. Notice, that for $k = 1$, which corresponds to the approach presented in [30] (see Section 4.5), the accuracy dropped to 54%.

The employed plans comparison is quite strict. The plans selected by the two algorithms must be exactly the same. Such precision is often unnecessary, since in most game playing scenarios only the first action or goal of the plan is used in every move. In the next move (or in next couple of moves) the planning is performed again. Therefore, we are also interested in examining the match of only the first goals of the plans returned by the full search and by the ASAS. In this modified experiment, the results improve significantly even for the rule-based plan selection. For $k = 2$ and $k = 3$ the first goal in the plans matched in 86.7% and 85.6% cases respectively with the rule-based selection method.

4.5 Related Work

We focus on the environments whose complexity prohibits the application of classical game-playing algorithms based on complete state space search. Consequently, heuristic game playing and multi-agent planning techniques are predominately applied in these environments and as such they are the closest competitor to the presented approach.

The heuristic techniques used in the search often utilize background knowledge to substantially reduce the search space. Examples can be found in GO [31], where the knowledge is in a form of a hierarchical task network, in card game of bridge [32], that uses domain-specific approach, or in the goal-based game tree search (GB-GTS) algorithm [29], which uses procedural knowledge in a form of higher level goals to reduce the search. These heuristics do not tackle the complexity dependence on the number of agents and they are complementary to the presented ASAS method.

The ASAS method is similar to the work of Mock [30]. Mock developed a version of game tree search, where action choices for only one of the agents are explored at a time and all the other agents behave heuristically. Our approach extends this idea and conducts searchers for subsets of more than one agent. Consequently, in our method a more complex sub-result merging strategy needs to be applied, but quality of the produced solutions is better.

In the field of game theory, the *graphical games* (GG) [33] consider situations where each player interacts only with a small subset of all players in the game (termed agent's *neighborhood*). The algorithms for finding the optimal strategies for graphical games can run with time complexity exponential in the size of the largest neighborhood in the game [34]. However, as we are interested in the multi-stage games, the algorithms for solving GG cannot be applied because they need a fixed structure of agents' interactions, which is unknown in advance in our case as it can possibly change with respect to different game states.

In [35] authors present an approach to multi-agent planning (MAP) that utilizes limited interactions among agents. It is based on a combination of single-agent planning and *constraint satisfaction programming*. It creates a multi-agent plan for fully cooperative agents with time complexity exponential in two sparsity measures, but not the number of agents in the environment. However, as other approaches based on MAP, the work considers only cooperative agents that work together to reach a common goal, which makes the combination of the single-agent plans possible using

conflict-resolving. In adversarial situations, each agent seeks and exploits conflicts in order to get more in specific situations.

The limitation of the MAP is partially solved by Ephrati et. al. in [36], serving as an inspiration for the sub-result merging part of our method. The approach optimizes the utility and considers also the case of agents with conflicting preferences. The solution assumes that the agents agree on certain fairness criterion (e.g., egalitarian or social welfare) and that they fully cooperate to find a global common plan that optimizes it. Such approach is not usable in real-world conflict situations (e.g., military operations), because the parties involved in the conflict would not fully cooperate and they would never trust each other sufficiently.

4.6 Conclusions

Interactions in multi-agent adversarial domains are crucial and cannot be simply neglected while creating plans for an agent. However in many cases, each agent interacts only with a small subset of other agents. Based on this observation, we have developed a method, termed agent subset adversarial search (ASAS), which decomposes the global adversarial search into multiple smaller overlapping sub-searches. In each sub-search only actions of a subset of all agents are searched, while the remaining agents behave heuristically based on the result from the search in the previous time-step. We proposed two different merging methods for construction of the final plan for each agent based on results from the sub-searches. The rule-based selection method, which reduces the computational complexity of the overall algorithm to polynomial in the number of agents, and the search-based method, for which the dependence on the number of the agents is still exponential, but the reduction of the search space is still significant.

The experimental evaluation proved that in spite of the reduced effort, ASAS achieves accuracy similar to the underlying full search. The subsets of three agents turned out to be a reasonable compromise between computational complexity and accuracy for our domain. With $k = 3$, the more precise search-based method produced the exact same plan as the full search in more than 90% and the faster rule-based method in 64% of cases. However, if we compare only the first actions produces by the searches (corresponding to the typical use of a game-playing algorithm), even the rule-based selection method agrees with the full search in over 85% of cases.

Future research will include a more thorough theoretical analysis of which domain properties underlie the applicability of the approach. We also plan to investigate different strategies for selecting the correct agent subsets, which would further reduce the search space. Specifically, we plan to investigate a closer integration of the principle of agents' plans independence into the construction of subsets, allowing different sizes of subsets, or possibly modifying the subsets during the sub-search process.

Chapter 5

Cooperation in Adversarial Search with Confidentiality

Cooperation of self interested agents in complex domains simulating real-world applications is a challenging problem in the multi-agent systems research. Agents in adversarial domains have to reason not only about their own goals, but also about other agents according to the opponents' models (OMs). If the agents use adversarial search for selecting their actions, each agent creates its private game tree that reflects the expectations of the future development of the game. In this tree, it selects a move that gains the maximal utility value for the searching agent according to its expectations. The outcomes of the adversarial search algorithms are often too pessimistic, because they do not assume the possibility of mutually beneficial coordination among the agents. If this coordination is possible, better solutions can be found (i.e. solutions for which each agent can gain a higher utility value).

We tackle this problem using the concept of pre-play communication leading to the cooperation of the agents. After running a search algorithm, an agent can use negotiation about future courses of actions with other agents to improve its anticipated utility value. The subject of negotiation can either be committing to a strategy in the complete agent's game tree, or partial strategies applied only in smaller parts of the tree (we further refer to them as *sub-games* or *sub-trees*). In the first case the agent fully reveals its intentions, which is often undesirable. Hence we follow the latter case and assume that strategies in the smaller sub-games would be used as a subject of the negotiation.

In this chapter we tackle two main goals: (1) If such a pre-play communication (i.e. a negotiation prior to actual playing) is allowed among the adversarial-search-based (self-interested) agents, what is the maximal possible improvement of the utility value that an agent can gain by negotiation about strategies in the sub-games with the other agents? Furthermore, in what way this value depends on parameters of the game, such as number of agents present, number of possible actions, or correlation of agents' utility functions. (2) What algorithm can be used to find the strategy by negotiations about sub-games if agents do not want to negotiate their complete plans?

The answer to the first question describes the negotiation-space of the utility values in synthetic games capturing the basic characteristic of the target domains. It shows how the space depends on the general characteristics of the games proving the usefulness of the concept. The answer to the second question is an algorithm that constitutes a solution for the problem of cooperation of

self-interested agents with confidential plans and without a third-party mediator.

In the next section we analyze the connection between basic game characteristics and the negotiation space in target domains. We do not limit our analysis on the humanitarian relief operation domain, but we aim onto all domains with similar characteristics (further termed as the target domains; including games based on real world scenarios, models of societies, economies, or war-games). We describe the characteristics of the extensive games that can model these domains, together with an example of a situation where the pre-play communication can lead to the improvement of the utility value. We follow with the formal definitions of the game and the utility improvement. We present an algorithm that calculates the maximal utility improvement for the searching agent together with its experimental evaluation. Next section is devoted to the description of a novel negotiation-based algorithm together with the experimental evaluation and application of this algorithm in our Tsunami Recovery Game. In the last sections of this chapter we present related work and give our concluding remarks.

5.1 Problem Definition

We are interested in domains modeled as n-player non-zero-sum extensive-form games with perfect information about the states of the game and number of agents and their actions. We assume that each agent reasons about other agents using opponent models (OMs) in the form of their utility functions. Moreover, we assume that players make their actions in turns with fixed order[1].

Finally, we assume in the presented approach that the players' utility functions have a specific structure. In the games of our interest we expect that players can usually slightly improve or slightly worsen their current situation with most of their actions, but significantly less actions lead to a radical change of the utility value. Therefore, the values of the child nodes of any node form a normal distribution with center in utility value that corresponds to the current situation. Note, that although this assumption holds for the Tsunami Recovery Game, it does not hold for classic games such as chess or checkers.

5.2 Usability of Pre-play Communication in Extensive Games

Solution of the extensive-form game that meets assumptions defined in the previous section can be obtained by a classical game-playing algorithm max^n – the n-player modification of classical minimax algorithm described in [38]. We understand the solution to be a set of strategies (sequences of actions for all players) together with the value of the game state after applying all the actions. Result returned by the max^n algorithm is in the subgame perfect Nash equilibrium. This, however, is not always optimal in the sense that there can exist a combination of actions that would result in higher utility values for all players. To reach a better value than max^n value concept of a different equilibrium has to be used. For example the correlated equilibrium[2] can

[1] In general, the game could have simultaneous moves, but this assumption still holds if the delayed execution as defined in [37] is applied. Also, the assumption that the order needs to be fixed can be relaxed in a way that the order can differ for each of players' game tree.

[2] Correlated equilibrium is defined in [39] as a generalization of the Nash equilibrium – the scenario includes some random variable (external event, or mediator, which can be replaced by using private pairwise message exchanges as mentioned in [40]). Furthermore, the variable has a commonly-known probability distribution and a private signal to each player about the instantiation of the random variable, which can be correlated with some of player's strategies – hence

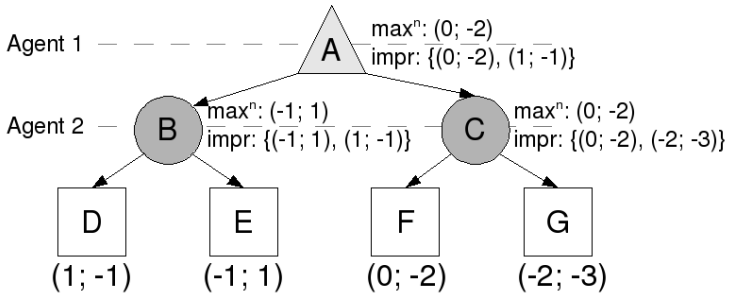

Figure 5.1: An example of the max^n algorithm. Utility values in the leafs are tuples of two elements, where Agent 1 maximizes the first element and Agent 2 the second one. The max^n values next to the nodes *A, B,* and *C* show calculated max^n value. The set $impr$ corresponds to the set of improvements calculated by the algorithm in Figure 5.3

be reached using a pre-play communication between agents and result in higher utility values for all players – we can say it improves the value of the game in comparison to max^n value. The example of such an improvement is depicted in Figure 5.1. The max^n value of the game (the value in the root A coming from node F) is inferior for both agents compared to strategy leading to the leaf D (utility value $(0; -2)$ vs. $(1; -1)$), which can be found using a pre-play communication between agents (e.g., in form of negotiation).

Using a pre-play communication in general does not solve the issue of trust – i.e. the agents can make false promises. Therefore we further assume that agents would execute the strategy that it was agreed upon in negotiation (we address this issue more in detail in the last section of this chapter).

5.2.1 Definitions

Game

Based on the definition of extensive games presented in [39], we define the turn-taking n-player non-zero-sum game in extensive form as a tuple $G = (\mathcal{I}, \mathcal{A}, \mathcal{H}, \mathcal{L}, r, \rho, \sigma, \pi, u)$, where:

- $\mathcal{I}$ is a set of n players indexed $i = 1 \dots n$
- $\mathcal{A}$ is set a of actions, $A = \bigcup_{i \in \mathcal{I}} \mathcal{A}_i$, where $\mathcal{A}_i$ is a set of actions an agent $i \in \mathcal{I}$ can perform
- $\mathcal{H}$ is a set of non-terminal nodes in the game
- $\mathcal{L}$ is a set of terminal nodes (leafs)
- $\rho : \mathcal{H} \mapsto \mathcal{I}$ is the player function which assings to each non-terminal node a player $i \in \mathcal{I}$
- $\sigma : \mathcal{H} \times A_i \mapsto \mathcal{H} \cup \mathcal{L}$ is the transition function realizing one move of agent i in the game
- π is a permutation on $\mathcal{I}$ representing the order of agents assigned to nodes on the path from the root of the tree towards the leafs. This order is repeated until leafs are reached.
- $\overrightarrow{u} = (u_1, \dots, u_n)$ is global utility, where $u_i : \mathcal{L} \mapsto \mathbb{R}$ is a real-valued utility function for player i on leafs $\mathcal{L}$

□

can be interpreted as a recommendation for the player. Now assuming that all players follow their recommendation, the equilibrium condition states that no player must have the incentive to deviate from it.

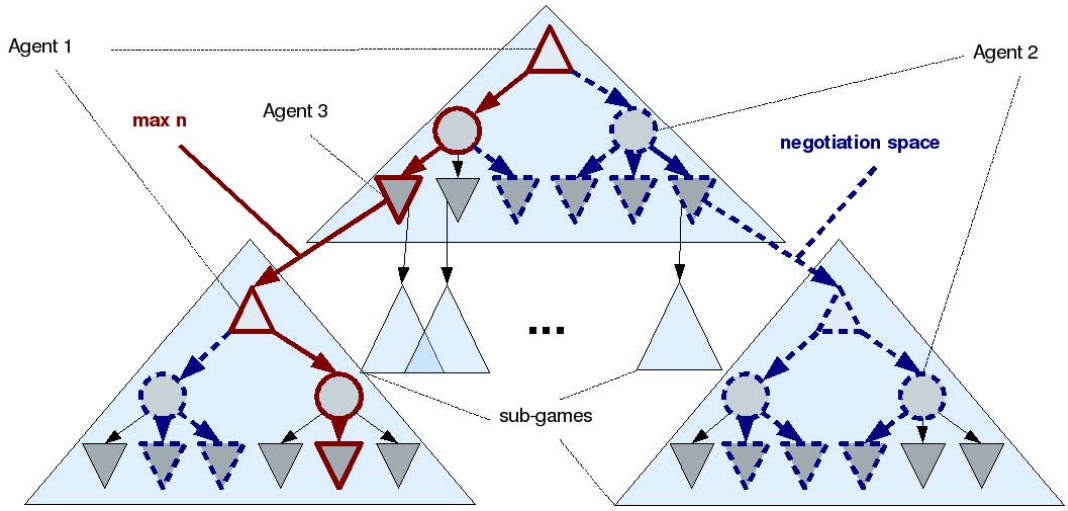

Figure 5.2: Visualization of the negotiation space – the bold solid branches represent the solution obtained by an adversarial algorithm (e.g. max^n), while the dashed branches represent those possible solutions that gain higher utility value for the searching player hence creating that way a space for negotiations with other agents in order to improve the value of the game.

In the following description, we talk about a single game G and we omit some indexing for sake of clarity. Furthermore, we use a notation, where:

- $r \in \mathcal{H}$ is the root of the game tree;
- $\tau(h) = \{j \in \mathcal{H} \cup \mathcal{L}; (\exists a \in A_{\rho(h)} : j = \sigma(h, a))\}$ is the set of the child nodes of a non-terminal node $h \in \mathcal{H}$
- τ can be expanded for a set $\tau(H) = \bigcup_{h \in H} \tau(h)$ where $H \subseteq \mathcal{H}$.
- $d \in \mathbb{N}$ is the depth of the tree calculated as the number of the non-terminal nodes on the path from the root to any leaf divided by n: $\underbrace{\tau(\tau(\dots \tau}_{d \cdot n}(r)) \dots) = \mathcal{L}$

Utility Improvement

We continue by presenting the algorithm for calculation of maximal utility improvement for an agent. We have shown the idea of max^n algorithm in Figure 5.1: for each node $h \in \mathcal{H}$ a max^n value $\overrightarrow{v}^{maxn} \in \mathbb{R}^n$ is calculated, and it corresponds to max^n value of the child node, for which the $v^{maxn}_{\rho(h)}$is maximal. For leaf $l \in \mathcal{L}$, $\overrightarrow{v}^{maxn} = \overrightarrow{u}(l)$. The max^n value $\overrightarrow{v}^{maxn}$ of the root of the tree represent the max^n value of the game, together with the max^n solution of the game which is the path from the root to the leaf with utility value equal to $\overrightarrow{v}^{maxn}$.

The algorithm for the utility improvement calculation is inspired by the *soft-*max^n algorithm described in [41] – a modification of the max^n algorithm. The *soft-*max^n value of the node $h \in \mathcal{H}$ is a set of utility values $\overrightarrow{v}^{sfmaxn} \in \mathbb{R}^n$. The idea is, that in case the maximal value is not strict, all utility values with maximal $\rho(h)$-th index are stored. The main idea is depicted in Figure 5.2 – besides the max^n value and solution (the red bold solid path) the player stores also other branches that form the negotiation space and that could bring him/her possibly better results (the blue dashed branches).

Input: $h \in \mathcal{H}$ current node; $i \in \mathcal{I}$ player performing the search
Output: $(\overrightarrow{v}^{maxn}, V^{impr})$ is a tuple, where $\overrightarrow{v}^{maxn}$ is a max^n utility value of the game, and V^{impr} is a set of utility values containing possible improvements.

1: $\overrightarrow{v}^{\text{maxn}} \leftarrow (-\infty, \ldots, -\infty)$
2: $V^{impr} \leftarrow \emptyset$
3: **if** $h \in \mathcal{L}$ **then**
4: **return** $(u(h), \emptyset)$
5: **end if**
6: **for** $c \in \tau(h)$ **do**
7: $(v', V') \leftarrow$imprMaxN(c, i)
8: **if** $v^{maxn}_{\rho(h)} < v'_{\rho(h)}$ **then**
9: $\overrightarrow{v}^{\text{maxn}} \leftarrow v'$
10: **end if**
11: $V^{impr} \leftarrow V^{impr} \cup V' \cup \{v'\}$
12: **end for**
13: **if** $\rho(h) = i$ **then**
14: **for** $v' \in V^{impr}$ **do**
15: **if** $v^{maxn}_i > v'_i$ **then**
16: $V^{impr} \leftarrow V^{imrp} \setminus v'$
17: **end if**
18: **end for**
19: **end if**
20: **return** $(\overrightarrow{v}^{\text{maxn}}, V^{impr})$

Figure 5.3: imprMaxN(c, i) – algorithm for calculation of the *impr-*max^n value of the game.

In our algorithm, shown in Figure 5.3, we calculate a value we call *impr-*max^n. For each node $h \in \mathcal{H}$ the *impr-*max^n value is a tuple $(\overrightarrow{v}^{maxn}, V^{impr})$, where $\overrightarrow{v}^{maxn}$ is the max^n value of the node (set in line 9), and V^{impr} is a set of utility values of other child nodes (line 11). For the nodes that are assigned to the agent i that runs the algorithm, we remove from the set V^{impr} those values, that are worse for this agent than the max^n value (lines 13-19). This means, that we build a negotiation space in a form of sub-tree of the game (see Figure 5.2). Any of the possibly negotiable solutions (i.e. any of the blue dashed branches in the figure) improves the utility value of the searching player i.

After agent i calculates the *impr-*$max^n = (\overrightarrow{v}^{\text{maxn}}, V^{impr})$ value of the game tree using algorithm in Figure 5.3, we say that there is an improvement possible in the game if there is an improvement option, with which no self-interested agent has an incentive to disagree. I.e.,

$$\exists \overrightarrow{v}' \in V^{impr}, \forall j \in \mathcal{I} : v^{maxn}_j \leq v'_j \tag{5.1}$$

Let $\mathcal{M}$ be a set of all such utility values for which formula 5.1 holds, and

$$v^{impr\_max} = \arg\max_{v' \in \mathcal{M}} v'_i$$

be the improvement with the maximal value for agent i. Using this maximal value from the set $\mathcal{M}$ we can define the amount of the utility improvement, which answers the first question of this chapter. We define this amount as the rate:

$$Impr = \frac{v^{maxn}_i - v^{impr\_max}_i}{2 \cdot 1.96 \cdot w} \tag{5.2}$$

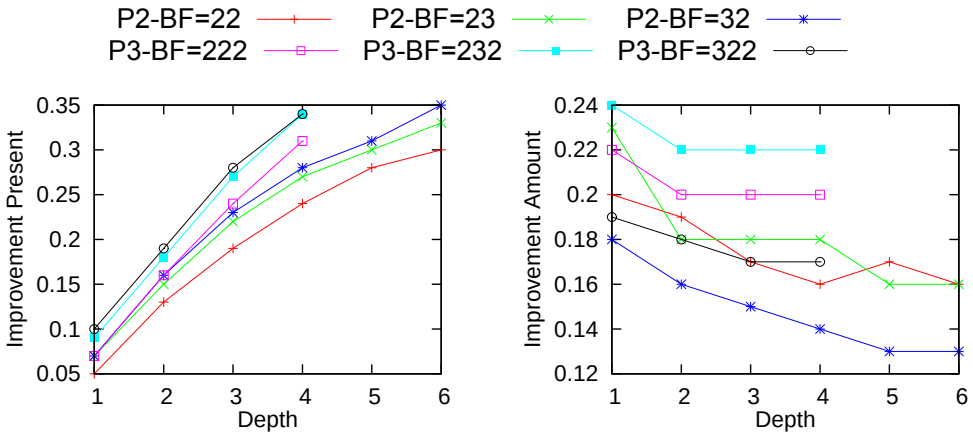

Figure 5.4: The results of the dependence of the improvement presence (the left figure) and the amount of the improvement (the right figure) on the depth (the x-axis). The legend: P2 represents games with 2 agents, P3 represents games with 3 agents; value of BF represents the branching factors for the agents in fixed order (each digit for one agent).

where w is a standard deviation of the probability distribution of all utility values in the leafs. We normalize the improvement of utility difference to the width of the interval that contains 95% of all utility values in leafs. The normalization is transforming the amount of utility value difference from absolute to relative improvement. The reason for this is that the absolute difference is sensitive to actual utility values, which are in general not very informative. For example, we can multiply all the utility values by 10, it does not change the problem anyhow. However, the value of the improvement without the normalization would also rise by the factor of 10. On the other hand, fixed normalization to the interval that includes all the utility values would be easily deformed small number of strategies with extremely high of extremely low payoffs.

5.2.2 Experimental Analysis

In order to practically evaluate the amount of maximal improvement, we performed a set of experiments on synthetic games. We use extensive games created with respect to the definition for 2 or 3 players. The searching agent is the first one ($i = 1$), and we use the fixed order of agents where the π is the identity. We use varying branching factors (number of actions by agents) from the interval 2-3, and the depth $d = 1, \ldots, 6$. The utility values in leafs were generated with respect to the observation presented in the problem description. For a $d = 1$ game we set a value of the game for the root of the tree and generate the utility values using normal distribution centered in this value. For arbitrary higher depth, we use this principle repeatedly – in the root we generate new centers for the non-terminal nodes representing the game states after all agents performed one action – until we reach the leafs.

For each setting of the parameters we created a set of 2000 variants (with different values on the leafs of the tree). We are investigating two aspects: firstly, we want to know how often there is an utility improvement possible in the games (i.e. the number of the games where $V^{impr} \neq \emptyset$ divided by the number of all games; we refer to this aspect as the *improvement presence*). Secondly, we are interested in the amount of the utility improvement $Impr$ as defined in formula 5.2.

In the first experiment setting, we have analyzed the dependence of these two aspects on the

depth of the game tree, the number of the agents involved in the game and the branching factors (BFs) of the agents. The results depicted in Figure 5.4 show that the improvement presence increases with the depth of the tree, while the maximal value of the utility improvement is slightly decreasing. Both of the results are expected: in the first case, there is an increasing opportunity for the occurrence of the situations similar to the one visualized in Figure 5.1 with increasing size of the game tree. For the second case, we argue that the game tree has wider interval of all utility values in the leafs, hence the improvement $Impr$ is slowly decreasing in spite of the fact that there is the higher presence of improvements.

Imprecise Opponent Models

In the previous experiment setting, we assumed that each agent posses a precise models of the opponents. Since this condition is usually not fulfilled in real scenarios, we analyzed how varying levels of noise in the opponent models affect the presence and the amount of the improvement. We simulate the noise by adding a random variable with the normal distribution, with center in zero, and with various standard deviations to the utility values in the leafs that correspond to the opponent(s). The results show that adding the noise has only a minor effect on both the monitored aspects – there is a small decrease of the improvement presence and the improvement amount with the increasing noise, but the differences were not significant.

Utility Correlation

In the next experiment setting, we have analyzed the impact of the correlation of utilities of the players to the monitored aspects. We use the pairwise correlation with the searching agent i – i.e. in 3-agents scenario we analyzed the impact by correlating the utility functions of the first and the second agent, and of the first and the third agent.

The results of the experiment are visualized in Figure 5.5. Because of the small impact of the varying noise in the opponent models, we show the results only for the precise opponent models. The results show an interesting outcome. The negative correlation between the agents increases the improvement presence in the games. For the value of the maximal improvement, the results are as expected – the negative correlation directly negatively affects the amount of the utility improvement, while in case of the positive correlation, there is not much space left for the utility improvement as the agents are driven by the utility to the “implicit cooperation”.

5.3 Sub-game Negotiations in Game-Trees

The previous section described the negotiation space for the utility improvement. Now, we propose a simple negotiation-based algorithm that utilizes the results from the algorithm *impr-*max^n in Figure 5.3.

Before we describe the algorithm, we need make several notation remarks. We work with a game G based on our definition, and i again denotes the agent that executes the search. Now:

– $\gamma : \mathcal{H} \cup \mathcal{L} \mapsto \mathcal{H} \cup \emptyset$ is a function that for each node returns its parent

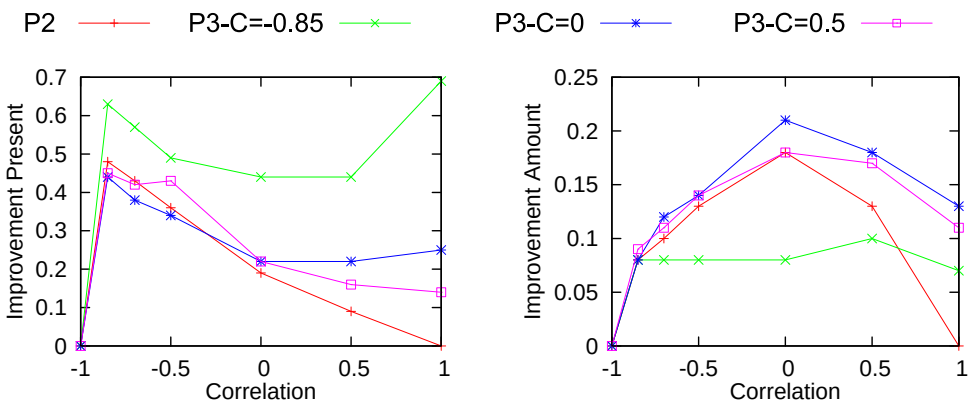

Figure 5.5: The results of the dependence of the improvement presence (the left figure) and the amount of the improvement (the right figure) on the correlation of agents' utility functions (the x-axis). The legend: P2 represents games with 2 agents, P3 represents games with 3 agents; in P3, the value C represents the utility correlation between the first and the third agent; the correlation between the first and the second agent is varying at the x-axis.

$\gamma(h) = \{h' \in \mathcal{H} : (\exists a \in \mathcal{A}_{\rho(h')}(\sigma(h', a) = h))\}$

- $\varphi : \mathcal{H} \mapsto \mathbb{N}^0$ is a function that returns the layer of the node $h \in \mathcal{H}$ in the game tree. The layer definition is similar to the depth – it is the floor of the number of non-terminal nodes on the path from the root to the node h divided by n
- G_h is the sub-game of the game G for some $h \in \mathcal{H}$, where h is the root of the sub-tree G_h
- *impr-*$max^n(h)$ denotes the *impr-*max^n value calculated for the node $h \in \mathcal{H}$ using the algorithm in Figure 5.3

5.3.1 Sub-game Negotiation-based Algorithm

The algorithm consists of two main stages: (1) it selects the sub-games in the game G, and then (2) for each of the sub-game G_h executes the negotiation protocol with the other agents.

The purpose of this algorithm is a proof of concept, hence both of the stages are designed in a straightforward way: (1) We use the sub-games of depth d_{G_h} equal to 1, as the roots of the sub-trees we select the nodes $h \in \mathcal{H}$ for which $\pi(\rho(h)) = 1$ (i.e., the nodes at the top of each layer in the game tree G). (2) We use a simple negotiation protocol, where the negotiation proceeds only between two agents. Moreover, in the negotiation protocol the searching agent i is proposing a strategy for a sub-game (i.e. a strategy for itself, and a strategy for the receiving agent), and the second agent j only replies *accepted* in case the proposing solution is improving agent's j utility and it commits to perform the action, or *not-accepted* if not.

Note, that the sub-games with $d = 1$ can have their leafs identical with non-terminal nodes of the game G. When the agents are negotiating about strategies, the utility values are taken from the $\overrightarrow{v}^{maxn}$ values calculated for the non-terminal nodes. The order of the sub-games G_h used in the negotiation protocol is given by the decreasing value $\varphi(h)$ of the node representing the root. This way, when the strategies in a sub-game are considered, for all sub-games in greater depth the maximal utility value is known (either original max^n value, or one of improving utilities that has already been agreed upon).

Input: $i \in \mathcal{I}$ player performing a search; $j \in \mathcal{I}$ responding agent; R_s set of roots of all sub-trees; d depth of the game G; d_{min} limitation for negotiations
Output: v^{neg}: utility value reached by negotiation for this game

1: **for** $k = d, \ldots, d_{min}$ **do**
2: **for** $r_s \in R_s : \varphi(r_s) = k$ **do**
3: $V^{remove} \leftarrow \emptyset$
4: $(\overrightarrow{v}^{\max n}, V^{impr}) \leftarrow$ *impr-*$max^n(r_s)$
5: sort V^{Impr} by decreasing value at i-th index
6: **while** $V^{Impr} \neq \emptyset$ **do**
7: $\overrightarrow{v}^{prop} \leftarrow$removeFirstUtility($V^{Impr}$)
8: *accepted*$\leftarrow$propose(j, $\overrightarrow{v}^{prop}$)
9: **if** accepted **then**
10: $V^{remove} \leftarrow V^{remove} \cup V^{impr}$
11: $h' = r_s$
12: **repeat**
13: $(\overrightarrow{v'}, V') \leftarrow$ *impr-*$max^n(h')$
14: **if** $v^{prop}_{\rho(h')} > v'_{\rho(h')}$ **then**
15: $\overrightarrow{v'} \leftarrow \overrightarrow{v}^{prop}$
16: **end if**
17: $V' \leftarrow V' \setminus V^{remove}$
18: *impr-*$max^n(h') \leftarrow (\overrightarrow{v'}, V')$
19: $h' \leftarrow \gamma(h')$
20: **until** $h' \neq \emptyset$
21: **break** while loop
22: **end if**
23: $V^{remove} \leftarrow V^{remove} \cup \overrightarrow{v}^{prop}$
24: **end while**
25: **end for**
26: **end for**
27: **return** $\overrightarrow{v}^{\max n}$ for r

Figure 5.6: The negotiation-based algorithm for improving the utility value of the game by the sub-games negotiation.

The negotiation-based algorithm is shown in Figure 5.6. As we described, it traverses through the root nodes of the sub-games from the bottom of the tree (lines 1-2), and for each sub-game it initiates the negotiation protocol about a proposition (line 8). If the proposition is accepted, the *impr-*max^n value is updated along the path from the node r_s to the root r of the game G (line 10-21). If the negotiated utility value is better for the agent assigned to a node on the path compared to the already existing max^n solution, the values are updated (lines 13-16). Also, we need to remove the utility values that agents had not agreed upon (line 23), or would not be further negotiated (line 10) from all the nodes along the path (line 17).

The choice of the value d_{min} is reflecting the confidentiality of agent's plans. If set to higher values, the agents would not negotiate in the upper parts of the tree. This way, even though agents could agree upon some solutions in the greater depth of the game tree, they cannot be sure, whether these nodes would actually be reached in the future development of the game (they can only expect the result at the top of the game tree according to their opponent models).

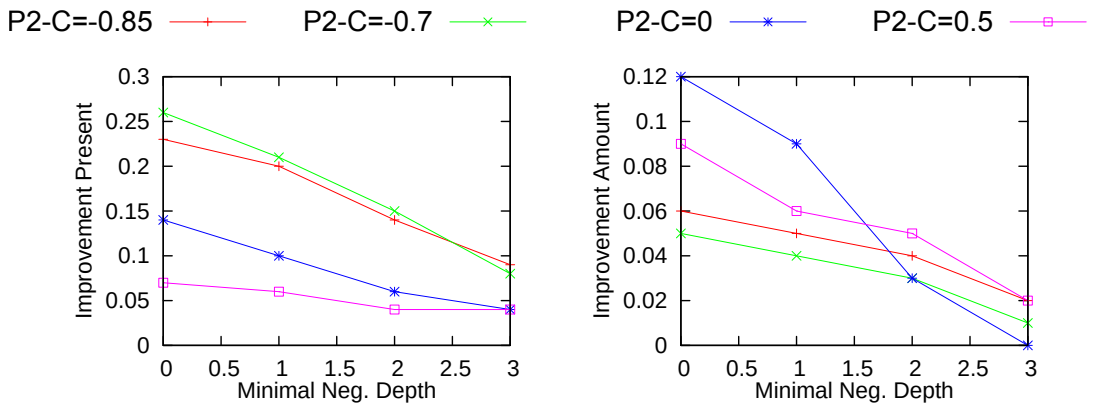

Figure 5.7: The results of the dependence of the improvement presence (the left figure) and the amount of the improvement (the right figure) on the minimal depth for negotiation (the x-axis). The legend: P2 represents games with 2 agents, the value C represents the correlation between their utility functions.

5.3.2 Experimental Analysis

We have practically evaluated the presented negotiation-based algorithm on the same games as in the experiments from the previous section. We investigated the improvement presence – the rate of the games where the negotiation-base algorithm was able to improve the value of the game to the number of all games – and the amount of the improvement – the value returned by the negotiation-based algorithm.

The results depicted in Figure 5.7 visualize the situation in the games with 2 agents and a fixed depth of the game $(d = 4)$. They show that even if we limit d_{min} and allow only the sub-games at the bottom of the tree to be negotiated about, the overall value of the game can still be improved.

5.4 Experiments on Tsunami Recovery Game

We have experimentally evaluated the sub-game negotiation-based algorithm in a scenario of the Tsunami Recovery Game. As described in Section 2.2.2, we added a new separatist player into the scenario. The two separatist players have similar utility value (both wants to destroy Government HQs), however, each of them can gain the control in cities for himself.

Thanks to this modification the separatist players can cooperate on some activities (e.g. destroying the food supplies from food trucks), but can also compete against each other (e.g. when trying to gain control in a city by force). In Figure 5.8 there is a situation depicted in which each of the separatist player has one gangster unit that can either: (1) destroy the food from the food truck, or (2) steal the explosives from the explosives truck, or (3) gain the control in City 2. The situation is depicted in Figure 5.9. We can see that when a variant of the max^n algorithm is used (which GB-GTS/ASAS is) the selected strategies are Ensure Presence Goal in City 2 for both separatist player resulting in utility value $(-1.0; -1.0)$ for the separatists because they cancel each other out and no one gain the control in the city. However, if both of the gangster units would select the destroying the food from the Food Truck 1, they would gain utility values $(2.0; 2.0)$. Note that the presence of negotiation is mandatory, as when the first separatist (GNG1) chooses to destroy the

Figure 5.8: A example situation from the scenario of Tsunami Recovery Game, where each of the gangster units that can either: (1) destroy the food from the food truck, or (2) steal the explosives from the explosives truck, or (3) gain the control in City 2.

food, the maximal variant for the second separatist (GNG2) is to select Ensure Presence Goal in City 2. After applying the sub-game negotiation-based algorithm in the testbed two separatist players agreed on playing different strategies improving this way their utility.

5.5 Related Work

The cooperation of agents is widely studied in the areas of multi-agent planning (MAP), and game theory (GT). However, there are not many works that would combine the concerns about the plan confidentiality together with the cooperation of self-interested agents. Works based on MAP, in general, assume a group of implicitly cooperating agents that want to achieve a common goal. The cooperation is usually addressed by either merging the plans that agents create separately (discussed e.g. in [42]), or by defining specific rules for the game by the methods of the mechanism design (e.g. in [43]). The aspects of the confidentiality of agents' plans are typically based on an independent trusted mediator, to whom the agents reveal their plans. In many domains, the presence of such mediator is impractical or even impossible. Therefore, we focus on the approach that does not need the mediator and enable the agents to limit the amount of the revealed information about their plans.

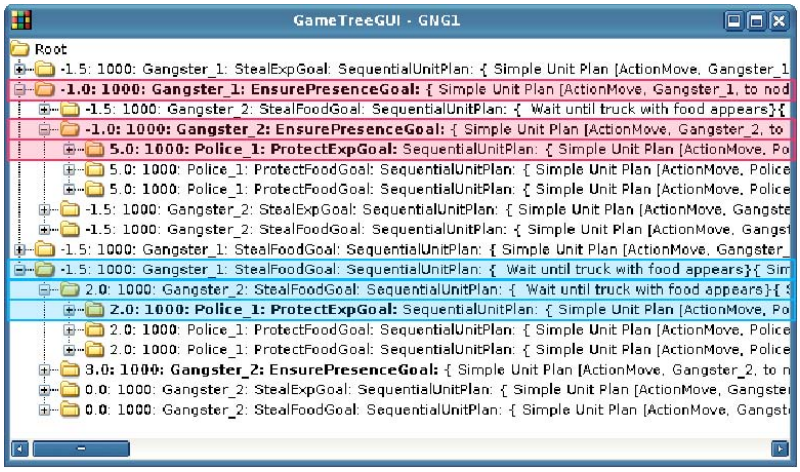

Figure 5.9: The game tree from the scenario showing the result without negotiation (the red items in the game tree) in comparison to a possible solution when negotiations are enabled (the blue items). Each item in the tree represent a node, where first number is the max^n utility value of the player currently making a decision, time-step, name of the unit for which the decision is being made, and selected goal for the unit. If the blue variant is chosen, all separatist players would receive higher utility than when the red variant is chosen.

A distributed approach for cooperation of agents without a mediator is proposed in [44] where the authors use logic framework based on resources constrains. The difference from our work is the limitation of their method to the collaborative agents only while our approach is applicable for more general self-interested agents with arbitrary correlation of utility functions. Works based on the results of game theory better address our goal as they work with self-interested agents, however, the focus is put on finding an appropriate equilibrium reflecting the cooperation (e.g. the formulation of the Correlated Equilibria for extensive games in [40]), and the agents are usually assumed to disclose complete information about their intended plan.

Recently, there have been several works developing new methods for the coordination of the self-interested agents. In [35, 45] the authors present transform the planning and coordination within a set of loosely coupled agents to a constraint satisfaction problem. However, their approach requires agents interaction graph to be acyclic. This is a very strong assumption that is generally not met in complex domains.

The works that are closest to our approach are based on a combination of MAP and GT ([46] and [47]). Both these works model the multi-agent planning problem as a general-sum stochastic game. In the first work the authors use a negotiation in order to approximate the subgame-perfect equilibrium. The second work describe distributed algorithm for finding the Nash Equlibrium in stochastic games, where the communication between the agents is applied. The crucial difference for both of these works is that authors do not consider the issue of plan confidentiality hence the agents negotiate the strategies of the whole game, while in our work only strategies in sub-games are being negotiated.

5.6 Conclusions

We investigated the problem of cooperation of self-interested agents with respect to the confidentiality of their plans. We used the extensive-form games and the concept of pre-play communication to answer two main questions: (1) analysis of the negotiation space of the utility improvement for an agent, and (2) the negotiation-based algorithm improving the utility value of the game by negotiating only sub-game strategies. Both of these goals were reached by experimental evaluation.

The interpretation of the experimental results is twofold. The results answering the first question show, that there is quite a large potential in using the pre-play communication even for domains with self-interested agents based on adversarial search. Particularly interesting is the high improvement presence when the agents' utility functions are correlated negatively. On the other hand, the proposed negotiation-based algorithm is able to utilize only a small part of the potential. Nevertheless, we show that even such simple algorithm can improve the overall utility of the game by negotiating the strategies in sub-games.

We point out the aspect of trust in the problem description. Currently, the agents always keep their promises and use the strategy they agreed upon. However, if the adversarial search in a later stage of the game explores consequences of an agreed strategy to higher depths, the agreement may become unprofitable for both the agents. This problem of horizon can be overcome by allowing cancellation of the agreements under certain conditions.

We plan to address this issue in our future work, together with a new version of negotiation-based algorithm that would be able to reach discovered potential of the utility improvement.

Chapter 6

Collaborative Opponent Modeling

The methods for creating and maintaining models of other agents' behavior were intensively studied in various domains ranging from abstract games like repeated Prisoners Dilemma[48] through Scrabble[49] and Poker[50] to more realistic domains, such as RoboCup soccer[51]. We have also investigated the means of creating opponent models in context of complex asymmetric games in our previous project. The final report of the project [3] presents methods for creating two basic kinds of opponent models: (1) the model of opponent's desires (i.e. the declarative OM) as well as (2) a basic model of agents intentions (i.e. procedural OM). The procedural OM approximated the behavior of the agent only based of the current state of the world, without modeling agent's internal state.

However, our work, as well as majority of other related work, assumes that only one agent models the behavior of one opponent (or a team of collaborating opponents). Less literature can be found on how the requirements to opponent modeling change in case of a coalition of agents model opponent coalitions of other agents, moreover if we allow dynamic changes in the forming the coalitions.

Our plan in this line of research was to (1) identify a suitable formalism that would allow describing procedural models of other agents' behavior (2) evaluate the possibility of automatic acquisition of such models (3) define a set of operation on the models described in the selected formalism that are necessary to support cooperation of opponent modeling agents and investigate properties of these operations.

The definition of the procedural opponent model in Chapter 3 contains agent's actions, states of the world and internal states of the agent. As a result, any formalism suitable for describing these models must contain at least these elements. The most basic formalism for capturing behaviors with internal state is a finite state machine (FSM). FSMs are used for *creating* behavior of agents in many commercial computer games and they were also used for modeling opponents in theoretical research of repeated games (e.g. [39]). They are represented by a set of internal state of an agent and a set of labeled transitions between the states. They assume a world progressing in discrete time steps. In each step, FSM outputs an action based on its internal state and after the agent applies the action, FSM changes its internal state based on the observations available to the agent. This is a very general representation of behavior of an agent and some researchers claim that the representation is sufficient to model even the human brain [52].

FSM is a very suitable formalism for representing behavior of a single agent. On the other hand, it is not well suited for representing dependences among the behaviors of multiple agents in a team, or description of concurrent execution of multiple goals by single agents. However, description and analysis of these phenomena has been widely studies in the field of business process management. The basic formalization often used in these studies are Peti nets [53], which are a generalization of FSM. With the increasing amount of research about business processes and gradual in-leak of its results to other research fields, even more expressive languages for describing processes has appeared. The process representation language whose design requirements are most similar to our needs is probably the Learnable Task Modeling Language (LTML) [54], which uses OWL ontologies and Planning Domain Definition Language (PDDL) to ground the process elements.

Based on the good experience with FSM in the field of artificial intelligence and the game theory and with the higher expressiveness of representation of agent (teams) behavior as processes, we have decided to use the notion of process as a central element of the cooperative opponent modeling.

The basic feasibility of using (and learning) processes for description of behavior of teams of agents has been already proven in [55]. The authors used processes and their automatic acquisition to analyze logs acquired during a Robocop soccer competition.

The first step in our effort in this direction was evaluation of the existing tools for automatic process acquisition in the domain of complex asymmetric games. The progress on this task is reported in the reminder of this chapter. However, as we have already explained in the introduction of this report, this line of research has been discontinued in favor of another interesting topic that was addressed in collaboration with the Carnegie Mellon University. As a result, the actual methods and analysis of collaborative use the process model for modeling behavior of other agents is not part of this report.

6.1 Automatic Opponent Model Acquisition

In this section, we report out results in automatic acquisition of procedural opponent models. During the previous project [3], we have developed and evaluated algorithms for creating reactive opponent models, that predicted agents future actions based solely on the current state of the world. This corresponds to models of reactive agents or the reactive approximation of agent's behavior defined in Chapter 3. In this project, we focused on extending the results to the stronger representation of the procedural opponent model – the model of agent with internal state. Automatic acquisition of this kind of knowledge is important also for easier applicability of one of the important result of our previous project [3], which is the adversarial search algorithm for complex dynamic domains – GB-GTS. The algorithm achieves substantial search reductions while often produces the optimal plans. The key element of this approach is the procedural knowledge describing how the players in the domain usually achieve partial goals and perform routine tasks. This knowledge usually needs to use some permanent internal states of the agents in complex adversarial settings.

Hand-coding this knowledge can be uselessly costly for the agents of the friendly teams and imprecise assumptions about the behavior of the opponents can cause biases and suboptimal courses of actions. Both of these disadvantages can be reduced if the background knowledge can be automatically extracted from the domain. There are two ways how to extract the knowledge.

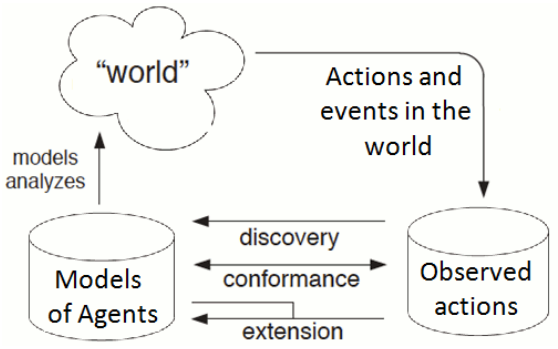

Figure 6.1: Schematic model of the process mining method.

(1) One option is to simulate the execution of the system (e.g. by using all atomic actions and not the goals in our case), analyze the applicability of the behaviors, and identify which of these behaviors produce important changes in the state of the world and then generalize these plans. This approach could find useful knowledge, but the computational effort would be too high. (2) A more applicable way to acquire this information is to analyze logs of past events in similar domains and generalize the plans from this knowledge. Such an analysis require less computational effort and it allows capturing specific strategies and habits of specific players, however, it can be less precise in comparison to the first possibility.

We present our analysis of using currently available state-of-the-art process mining techniques for acquiring procedural knowledge necessary for the GB-GTS algorithm. We identify also the possibility of using similar learning methods for extracting the information about agents' interactions that may be used for a more sophisticated selection of agents' subsets in the ASAS method.

6.1.1 Process Mining

A process is a sequence of actions that actor (or a group of actors) performs in order to achieve a desired outcome. Process Mining (PM) [56] is a computational technique originating from Business Process Management [57]. It is used to analyze logs of events and actions in a complex system (i.e., in general large organizations, but for our case also complex multi-agent system) and extract information about the processes executed by the actors (agents in our case). The main purpose of PM is understanding, reengineering, and optimizing the existing processes.

In the setting of this project, the actions are the ground instances of the action operators that were actually executed in the scenario: a truck has moved to a node on the map, a separatist unit has unloaded a package of explosives in a specific city. The log (that is the input for the PM methods) includes actions form all the agents in the system. The changes of the environment that were not caused by the modeled agents (i.e. events, such as the food consumption in our domain) can be included in the log as actions performed by the "nature" agent.

There are several types of results from applying the method of process mining that all help to describe the process. The first type of the output is a *control flow* – a graph representing which actions are usually performed after each other and what are the alternative sequences of actions that perform the same task. Another part of the output can be discovering *social networking* aspect of the system – i.e. identifying which agents in the system are participating on the same

task, how they work together and what are their goals and dependencies. Finally, a type of the output of the PM may also be the analysis of the *performance* of the system. It allows identifying the bottlenecks of the system and using temporal logic various characteristics can be checked (e.g. the robustness of the plans).

The earliest works in the domain of PM are from mid-nineties and the progress is summarized in [58]. Process mining is in many aspects connected to Machine Learning (ML), but there are several specifics in PM. Mainly, the process models generally form a network of actions (e.g., Petri nets) in contrast to action sequences of probability distributions (e.g., Markov chains) used in ML. Therefore, new issues studied in PM are various forms of concurrency.

PM has been successfully applied for analyzing real-life organizations, e.g., hospitals, banks, municipalities etc. (see [59]). However, the same set of tools was also used for analyzing team behavior in the domain of robotic soccer [55]. The process mining tools were used for two purposes. (1) Firstly for analyzing and tuning own team strategies that emerged from the interaction of the programmed system with the dynamic environment and the opponent activity. (2) But also to analyze the behavior of the adversary. In the adversary analysis, the main application of the techniques was recognizing higher level activities (such as attacking) from the basic observations (such as robot position and velocity). If such an activity was recognized, the team could adjust its strategy to fit better the new situation.

6.1.2 The ProM Framework

Different existing process mining systems and tools usually use different formats for reading/storing log records and present their results in a different fashion. As a result, using and combining results of several different techniques is timely and inefficient. This problem is addressed by the ProM framework (pluggable environment for process mining) [60], developed at Eindhoven Technical University. The goal of ProM is to define independent algorithms for process mining. ProM is an open source tool implemented in Java and distributed under the CPL license.

ProM uses a generic input format – Mining XML (MXML) – for representation and storing of events and allows importing logs from several existing commercial systems. After cleaning the logs with an array of provided filters, it allows processing the data using more than 230 plug-ins and the process mining community is encouraged to contribute more. The exact output of PM depends on the used plug-in, but ProM can visualize and convert between various standard process modeling languages, such as Petri nets, EPCs/EPKs, or YAWL.

The available plug-ins support control-flow mining techniques, staff assignments mining, mining decision rules from the context, but also plug-ins for process analysis and verification.

The applications of the ProM framework include both artificial domains (such as the robotic soccer example mentioned above [55]), or process analysis in real world organizations (e.g., Dutch National Public Works Department [61]).

6.1.3 Process Mining in the Adversarial Behavior Testbed

In our work, we can see three straightforward ways how the process mining can be used for opponent modeling and for learning to support our methods. The first two methods (i.e. creating

control flow and identifying the conditions in branching nodes) help to prune the search space using the procedural knowledge captured in the form of processes and the third method identifies the relevant subsets of agents that interact on sub-tasks in the domain.

Event Log in MXML

In order to apply the PM algorithms available in ProM, we need to have a log of events occurred in the simulation in a compatible format. That is why we have implemented a logger agent that logs the events occurred in the simulation into the Mining XML format. In the current implementation, this log contains the actions performed by the players annotated by the important states of the domain that the actions have changed. A portion of an example log file produced by the logger agent is depicted in Figure 6.2.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <WorkflowLog description="Action log: 2010/04/21 19:38:34">
  - <Process id="0">
    - <ProcessInstance id="0">
      - <AuditTrailEntry>
        - <Data>
            <Attribute name="Position">1764</Attribute>
          </Data>
          <WorkflowModelElement>ActionMove</WorkflowModelElement>
          <EventType>complete</EventType>
          <Timestamp>1970-01-01T01:00:01.000+01:00</Timestamp>
          <Originator>Gangster_32</Originator>
        </AuditTrailEntry>
      - <AuditTrailEntry>
          <WorkflowModelElement>ActionWait</WorkflowModelElement>
          <EventType>complete</EventType>
          <Timestamp>1970-01-01T01:00:01.000+01:00</Timestamp>
          <Originator>Engineer_66</Originator>
        </AuditTrailEntry>
        ...
      - <AuditTrailEntry>
        - <Data>
            <Attribute name="Loaded">true</Attribute>
          </Data>
          <WorkflowModelElement>ActionLoad</WorkflowModelElement>
          <EventType>complete</EventType>
          <Timestamp>1970-01-01T01:00:07.000+01:00</Timestamp>
          <Originator>FoodT_6</Originator>
        </AuditTrailEntry>
        ...
      </ProcessInstance>
    </Process>
</WorkflowLog>
```

Figure 6.2: A portion of a sample event log in the MXML format.

Whole the log of one run of the simulation is captured as one instance of a single process. If we wanted to perceive the log as many instances of small number of processes, we would need some additional (expert) knowledge that would say when one process executed by an agent has finished and another one started. Adding this kind of knowledge can be part of log preprocessing, but we did not do that in our experiments.

jakob2008collaborative

Learning Structures of Agents' Plans

One of the main type of result of PM is the process control-flow that can be very useful for pruning. If we assume that (1) actions (instantiated operators) of the opponent are observable and can be captured in the "log", (2) the ordering of these actions is correct, and (3) we have a reasonable amount of data (which is often not too high), then we can use PM (more specifically the ProM framework) and the existing algorithms (in form of plug-ins for ProM) for learning common sequences of actions (plans or process models) of the opponent. These processes can be represented for example in the form of Petri nets and they would provide information about agents' actions sequences, parallelisms, branches, and loops.

The agents with one of the simplest behavior in our testbed are trucks. In the first experiment we present here, we have applied one of the ProM filters to the log and considered just the actions of single fool truck. We have used the FSM miner to extract a model of the agent's process and then we have used a build-in tool to convert the resulting FSM to a Petri net.

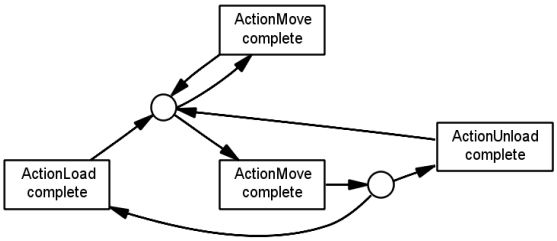

Figure 6.3: The food process control flow model of a food truck automatically extracted using the ProM framework.

The resulting process model is shown in Figure 6.3. This Petri net contain two states. In the left state, agent is just moving and after application of the `move` action, it chooses either stay in the same state or it can transition to the second state. In that state, the agent does not consider moving anymore and it loads or unloads. This process control-flow is very similar to the expected process model shown if the interim report of this project [62]. It represents three of the actions a truck can perform (action `wait` is missing). This process model compactly represents a lot of information included in the log. Based on this model, we can infer that

- Action `load` always follows after a `move` action. I.e. the truck is never loaded at the same place where it was unloaded.
- Action `move` can be executed several times after each other.
- The truck was never idle (action `wait` was not used for the truck).

This kind of process model can be used as the procedural knowledge heuristics for GB-GTS. Only the future developments of the game that are consistent with this process can be explored. Even this simple process model can already cause reduction of the search space.

Learning branching conditions

Even larger computational savings are possible thanks to ProM. Given the process model of the opponent constructed in the previous step, we can try to learn *decision conditions* on the

branchings and loops. The atoms in the decision conditions can have the form of predicates, such as numerical (in-)equalities, or other test referring to data values appearing in the log. A result of applying the decision point analysis plug-in from ProM to our process model is in Figure 6.4.

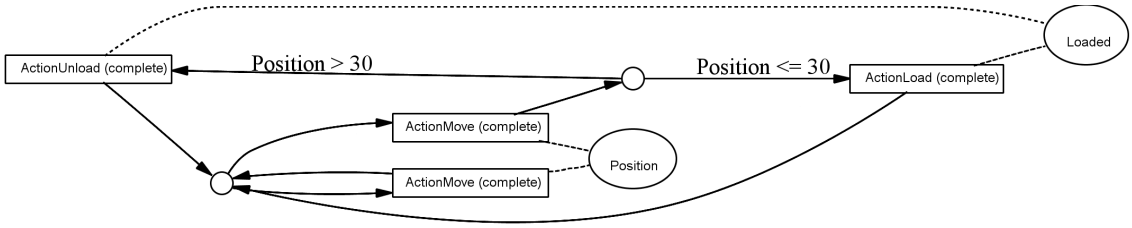

Figure 6.4: Conditions of branching in the process of truck a food truck.

We have analyzed the situations in which the agent chooses to load and in which it chooses to unload the cargo. The decision mining plug-in extracts the data values that were set before the decision point was reached in the process. It then uses these values and the decisions that have been made in the logs to train a decision tree classifier. The circles in the graph in Figure 6.4 represent the data attributes that are set by the individual actions. The decision is to load or unload the truck apparently depends on the position of the food truck. In our domain, the food truck transports the food from the places where it is produced to other places where it is consumed. The city with the food farm is in a node with relatively low, so the decision miner learned a condition that in the truck loads in the cities positioned in the nodes with low ID.

While experimenting with this plug-in, we have learned that its options are quite restricted when using the ProM GUI. For example, it was not possible to disallow the plug-in to use the data stored in the action just after the decision point and still use data from a longer sequence of preceding actions. It had also some problems identifying decisions points with loops (i.e. actions representing a transition to the same point in the Petri net). Furthermore, the plug-in used Weka library [63] for the machine learning tasks, but it allows using only decision trees for classification of the alternative decisions. This machine learning technique seems to be very suitable for this purpose, because it produces human readable rules that can be quickly evaluated. However, in more complex domains, the reasons for the decisions are not straightforward. In our example, we would like the system to learn that the truck is loaded in cities with a farm. In order to learn this kind of knowledge using decision trees, we would have to add to each node of the road network an attribute saying if there is a city and if the city contains a farm. All the “background knowledge” that could be useful to determine the decision has to be explicitly encoded. However, using inductive logic programming (ILP) as the classification model as we did in [64] allows encoding the background knowledge in form of logical formulas. In our example, it would be sufficient to list the nodes in which are the cities and represent a relation saying which cities contain a farm.

Even using the decision trees as the learning method, extracted conditions help further restrict the options that need to be explored for an agent. However, for more practical use of decision mining techniques in our domain, we would have to implement our own methods.

Analyzing agents interactions

As mentioned above, one of the possible outputs of the ProM framework are social networking properties of the processes. In a complex process that involves multiple agents, it can identify which agents participate on which branches of the process, what are their (temporal) depen-

dences, which agents work together, or which agent hand over the work to other agents. Based on this information, it could be possible to learn which opponents can prevent agent from reaching a goal and what friendly agents are necessary for a task – hence derive which agents are not likely to bring new interesting plan candidates in subsets in the ASAS method and therefore we can omit these k-tuples sub-searches (see Section 4.3).

Our initial experiments with the Social Network Miner (SNM) plug-in in ProM have shown that the existing plug-in cannot be directly used on our testbed game. One of the main reasons is that each of the agents usually performs an action every time step, which the SNM interprets as a collaboration of all agents. Therefore further investigation with more configurable algorithm (or using our implementation of such miner) is necessary.

6.2 Conclusions

In this chapter, we have identified business process models as an interesting formalization of behavior of intelligent agents. It is more powerful then FSM commonly used for representing agent behavior in AI and Game Theory. Moreover, a large body of research has focused on various properties of formalisms for describing processes (e.g. Petri nets) and the research community has developed a number of publicly available tools for extracting processes from systems logs. We have found a specific tool (called ProM) that seems to be quite stable and used by the community. Employing this tool, we were able to extract simple process description of the behavior of an agent in our adversarial behavior testbed. Although the results show that the ProM tool cannot be used directly out-of-box due to the lack of configurability of implemented algorithms, they validated plausibility of using process mining in general for opponent model acquisition in complex asymmetric games. The process representation of the agents' behavior can also be used as the background knowledge for goal-based game tree search [3]. Integration of the process mining to a loop of learning and using the background knowledge in the domain could provide the substantial computational savings of GB-GTS without the need to hand-code the background knowledge.

The results reported in this chapter are just the first step towards creating the methods for collaborative construction, refinement and sharing of opponent models, what was the initial objective of our research. Modeling the behavior as processes seems to be suitable for this task, because the formal description of the processes and the available tools for processing these descriptions enable the agents to easily communicate about the their processes and processes of other agents.

The next step of this research would be to develop specific algorithms and protocols for merging (possibly inconsistent) processes from different sources in (semi) cooperative setting. As a part of that, we would have to investigate various quality measures of a process with respect to its capability to predict other agents' behavior. The developed methods could be evaluated in the extended adversarial behavior testbed described in Chapter 2.

However, we have not implemented the collaborative methods on top of process models of other agents' behavior in this project. The reason is that the effort planned for this task was used to address a different problem – deception in teams of mobile sensing agents. Solving this problem was not explicitly listed as a research task in the project proposal, but it is well aligned with the project objectives stated in the proposal and it allowed us to start fruitful collaboration with the Carnegie Mellon University and achieve more significant results thanks to our joined expertise. More details can be found in Section 1.2.6.

Chapter 7

Conclusion

In this project, we are extending the results achieved in Project FA8655-07-1-3083, which has delivered novel results in several key areas of adversarial reasoning: formal framework, adversarial planning and search, and opponent modeling. Besides these results, the project identified the main challenges of adversarial planning to be the high computational *complexity* of the available methods, the need to reason about *uncertainty* caused by the activity of the opponents and the partial knowledge of the agents about the state of the world, and the need for *cooperation* and explicit coordination among individual (self-interested) agents in the game.

All these challenges were successfully addressed in the current project that presents advancements in the state of the art in all three of the problem areas. In order to tackle the computational complexity issues we have developed a novel algorithmic scheme that allows asymptotic reduction of computational complexity of generic adversarial search algorithms. In the area of collaboration of self-interested agents we have studied the concept of a pre-play communication in the context of the extensive form games. We have designed a novel algorithm that allows self-interested agents to negotiate about alternative courses of actions and reach agreements that are beneficial for all the players involved. Moreover, the algorithm allows limiting the amount of information revealed to the other agents during the communication. In the area of reasoning about uncertainty in complex adversarial settings, we present investigation of the notion of deception and the algorithm for deception robust placement of a network of mobile sensing agents.

These results are formally grounded in a unified formal framework describing the main concepts related to adversarial reasoning, their relations and properties. Moreover, the presented algorithms are implemented and experimentally evaluated in the adversarial reasoning behavior testbed developed in our previous projects and further adjusted to meet the specific needs of this project.

Even though we have advanced the state of the art in all three main problem areas of adversarial planning in complex domains, there are various research directions of this field that can be further investigated.

In the direction of computational *complexity*, Monte Carlo techniques, such as UCT [10], are becoming an increasingly popular tool of planning in complex adversarial domains. Therefore evaluating how the properties of the developed methods change if they run on top of this type of algorithms would be an interesting continuation promising further improvement in the speed of

the search algorithm and therefore also further increase in size of the used scenarios. Another approach supporting the research in the direction of computational *complexity* is better utilization of the *cooperation* between the agents. In many domains, including our adversarial behavior testbed, agents could be divided to small number of highly cooperative teams. In these situations, it might be possible to combine the methods developed for fully cooperative multi-agent planning [42] with the adversarial planning methods to further improve the efficiency the adversarial planning process. Finally, in the direction of *uncertainty* we suggest to continue in the started work of cooperative opponent modeling. We have already performed initial experiments showing that the process description formalisms developed in business process management are a promising tool for describing behavior of (teams) of intelligent agents. An obvious future research suggestion is to further investigate the suitability of these formalisms for cooperative tasks related to opponent modeling (such as sharing and merging (parts of) the models.

In order to fully understand the characteristics of the proposed future directions, we suggest further improving the theoretical models as well. The recent research in the field of Graphical Game representations such as the Temporal Action-Game Graphs [65] could be used to formally derive bounds on the quality of the ASAS method in loosely coupled multi-player games. Also, in Chapter 3, we identify the Alternating-time Temporal Epistemic Logic [24] to be a suitable formal language for describing properties of complex adversarial domains. We suggest using this language to formally define the domain properties that make various adversarial planning heuristics efficient. When facing a new domain, an intelligent agent (or his designer) can automatically check this set of properties to identify the most suitable (combination) of adversarial reason techniques that will achieve the best game playing performance with minimal computational effort.

Bibliography

[1] V Marík, M Pechoucek, and O Štepánková. Social knowledge in multi-agent systems. *Multi-Agent Systems and Applications*, pages 211–245, 2006.

[2] Viliam Lisy, Michal Jakob, Jaz Tozicka, and Michal Pechoucek. Utility-based model for classifying adversarial behaviour in multi-agent systems. In *Workshop on Agent Based Computing, International Multiconference on Computer Science and Information Technology*, pages 47–53, 2008.

[3] Michal Pěchouček, Michal Jakob, Viliam Lisý, Eduard Semsch, Branislav Bošanský, and Jan Doubek. Agent-based computing in distrib uted adversaril planning – final report. Technical report, Gerstner Laboratory, Czech Technical University in Prague, Nov 2008.

[4] J. Von Neumann, O. Morgenstern, H.W. Kuhn, and A. Rubinstein. *Theory of games and economic behavior*. Princeton university press Princeton, NJ, 1953.

[5] H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 11(1):115–146, 1989.

[6] P. Bercher and R. Mattmüller. A planning graph heuristic for forward-chaining adversarial planning. In *Proceeding of the 2008 conference on ECAI 2008: 18th European Conference on Artificial Intelligence*, pages 921–922. IOS Press, 2008.

[7] M. Pěchouček. Social reasoning in computational multi-agent systems. Habilitation Thesis, Czech Technical University, 2008.

[8] Hilmar Finnsson and Y Björnsson. Learning Simulation Control in General Game-Playing Agents. In *AAAI*, 2010.

[9] S. Sharma, Z. Kobti, and S. Goodwin. Knowledge generation for improving simulations in UCT for general game playing. *AI 2008: Advances in Artificial Intelligence*, pages 49–55, 2008.

[10] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Machine Learning: ECML 2006*, pages 282–293, 2006.

[11] G. Brajnik, S. Mizzaro, C. Tasso, and F. Venuti. Strategic help in user interfaces for information retrieval. *Journal of the American Society for Information Science and Technology*, 53(5):343–358, 2002.

[12] P.R. Cohen and H.J. Levesque. Confirmations and joint action. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pages 951–957. Citeseer, 1991.

[13] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research. In *Proc. IEEE Int. Conference on Systems Man and Cybernetics*, volume 6, 1999.

[14] M. Helmert. Complexity results for standard benchmark domains in planning. *Artificial Intelligence*, 143(2):219–262, 2003.

[15] M. Pěchouček, D. Šišlák, D. Pavlíček, and M. Uller. Autonomous agents for air-traffic deconfliction. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, page 1505. ACM, 2006.

[16] Martin Rehák, Jan Tožička, Michal Pěchouček, Filip Železný, and Milan Rollo. An abstract architecture for computational reflection in multi-agent systems. In Andrzej Skowron, Jean-Paul A. Barthès, Lakhmi C. Jain, Ron Sun, Pierre Morizet-Mahoudeaux, Jiming Liu, and Ning Zhong, editors, *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 128–131. IEEE Computer Society, 2005.

[17] Jan Tožička, Michal Jakob, and Michal Pěchouček. Market-inspired approach to collaborative learning. In Matthias Klusch, Michael Rovatsos, and Terry R. Payne, editors, *Cooperative Information Agent X*, LNAI 4149, pages 213–227, 2006.

[18] Viliam Lisý, Michal Jakob, Petr Benda, Štěpán Urban, and Michal Pěchouček. Towards cooperative predictive data mining in competitive environments. In *Agents and Data Mining Interaction, 4th International Workshop, ADMI 2009, Revised Selected Papers*, volume 5680/2009 of *Lecture Notes in Computer Science*, pages 95–108, Budapest, Hungary, 2009. Springer Berlin / Heidelberg.

[19] E.A. Emerson and J.Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time* 1. *Journal of computer and system sciences*, 30(1):1–24, 1985.

[20] R. Fagin. *Reasoning about knowledge*. The MIT Press, 2003.

[21] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *Journal of the ACM (JACM)*, 49(5):713, 2002.

[22] Nils Bulling, Wojciech Jamroga, and Jürgen Dix. Reasoning about temporal properties of rational play. *Annals of Mathematics and Artificial Intelligence*, 53(1-4):51–114, January 2009.

[23] J. Ruan, W. van der Hoek, and M. Wooldridge. Verification of Games in the Game Description Language. *Journal of Logic and Computation*, 19(6):1127–1156, August 2009.

[24] W. van der Hoek and M. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Studia Logica*, 75(1):125–157, 2003.

[25] N Bulling and W Jamroga. What agents can probably enforce. *Fundamenta Informaticae*, 93(1):81–96, 2009.

[26] R. Fagin and J.Y. Halpern. Reasoning about knowledge and probability. *Journal of the ACM (JACM)*, 41(2):340–367, 1994.

[27] Carol Luckhardt and Keki B. Irani. An algorithmic solution of N-person games. In T. R. S. Kehler, editor, *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI)*, volume 1, pages 158–162, Philadelphia, Pennsylvania, August 1986. Morgan Kaufmann.

[28] N. Sturtevant, M. Zinkevich, and M. Bowling. Prob-Maxˆ n: Playing N-Player Games with Opponent Models. In *Proc. of the National Conference On Artificial Intelligence*, volume 21, page 1057. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[29] Viliam Lisý, Branislav Bošanský, Michal Jakob, and Michal Pěchouček. Adversarial search with procedural knowledge heuristic. In *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, 2009.

[30] Kenrick J. Mock. Hierarchical heuristic search techniques for empire-based games. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI)*, pages 643–648, 2002.

[31] Steven Willmott, Julian Richardson, Alan Bundy, and John Levine. Applying adversarial planning techniques to Go. *Theoretical Computer Science*, 252(1–2):45–82, 2001.

[32] Stephen J. J. Smith, Dana S. Nau, and Thomas A. Throop. Computer bridge - a big win for AI planning. *AI Magazine*, 19(2):93–106, 1998.

[33] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, pages 253–260. Citeseer, 2001.

[34] C.H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multi-player games. *Journal of the ACM (JACM)*, 55(3):14, 2008.

[35] R.I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In *ICAPS-08, 18th International Conference on Automated Planning and Scheduling, Sydney, Australia*, 2008.

[36] E. Ephrati and J.S. Rosenschein. A heuristic technique for multi-agent planning. *Annals of Mathematics and Artificial Intelligence*, 20(1):13–67, 1997.

[37] Alexander Kovarsky and Michael Buro. Heuristic search applied to abstract combat games. In *Canadian Conference on AI*, pages 66–78, 2005.

[38] C.A. Luckhardt and K.B.Irani. An algorithmic solution of n-person games. In *Proc. of the National Conference on Artificial Intelligence (AAAI-86), Philadelphia, Pa., August*, pages 158–162, 1986.

[39] Yoav Shoham and Kevin Leyton-Brown. Multiagent systems: Algorithmic, game-theoretic, and logical foundations. pages 1–532, February 2009.

[40] Bernhard von Stengel and Françoise Forges. Extensive-form correlated equilibrium: Definition and computational complexity. *Math. Oper. Res.*, 33(4):1002–1022, 2008.

[41] Nathan Sturtevant and Michael Bowling. Robust game play against unknown opponents. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 713–719, New York, NY, USA, 2006. ACM.

[42] Mathijs De Weerdt, Adriaan Ter Mors, and Cees Witteveen. Multi-agent planning: An introduction to planning and coordination. Technical report, In: Handouts of the European Agent Summer, 2005.

[43] Roman van der Krogt, Mathijs de Weerdt, and Yingqian Zhang. Of mechanism design and multiagent planning. In *Proceedings of the Eighteenth European Conference on Artificial Intelligence*, pages 423–427, 2008.

[44] Jiefei Ma, Alessandra Russo, Krysia Broda, and Emil Lupu. Multi-agent planning with confidentiality. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1275–1276, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[45] Ronen Brafman, Carmel Domshlak, Yagil Engel, and Moshe Tennenholtz. Planning Games. In *IJCAI-09, 21st International Joint Conference on Artificial Intelligence*, 2009.

[46] Chris Murray and Geoffrey J. Gordon. Multi-robot negotiation: Approximating the set of subgame perfect equilibria in general-sum stochastic games. In *NIPS*, pages 1001–1008, 2006.

[47] Andriy Burkov and Brahim Chaib-draa. Distributed planning in stochastic games with communication. In *Proceedings of ICAPS'08 Multiagent Planning Workshop (MASPLAN'08)*, Sydney, Australia, September 2008. (This is a preliminary workshop version of the paper. The final version is published in Proceedings of ICMLA'08).

[48] Philip Hingston. Learning versus evolution in iterated prisoners dilemma. In *in Proceedings of the 2004 Congress on Evolutionary Computation (CEC 04). IEEE Publications*, pages 364–372, 2004.

[49] Mark Richards and Eyal Amir. Opponent modeling in scrabble. In *IJCAI*, pages 1482–1487, 2007.

[50] Aaron Davidson, Darse Billings, and Duane Szafron. Opponent modeling in poker. In *In AAAI National Conference*, pages 493–499, 1998.

[51] Patrick Riley and Manuela Veloso. On behavior classification in adversarial environments. In *Distributed Autonomous Robotic Systems 4*, pages 371–380. Springer-Verlag, 2000.

[52] F. Velde. Is the brain an effective Turing machine or a finite-state machine? *Psychological research*, 55(1):71–79, 1993.

[53] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[54] M. Burstein, R.P. Goldman, D.V. McDermott, D. McDonald, J. Beal, and J. Maraist. LTMLa language for representing semantic web service workflow procedures. In *Proceedings ISWC workshop on Semantics for the Rest of Us*, 2009.

[55] A. Rozinat, S. Zickler, M. Veloso, WMP van der Aalst, and C. McMillen. Analyzing Multi-agent Activity Logs Using Process Mining Techniques. *Distributed Autonomous Robotic System 8*, page 251, 2009.

[56] Carlos Pedrinaci and John Domingue. Towards an ontology for process monitoring and mining. In *Workshop: Semantic Business Process and Product Lifecycle Management (SBPM 2007), 4th European Semantic Web Conference (ESWC 2007)*, 2007.

[57] Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In Francis C. M. Lau, Hui Lei, Xiaofeng Meng, and Min Wang, editors, *ICEBE*, pages 535–540. IEEE Computer Society, 2005.

[58] W.M.P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.

[59] Wil M. P. van der Aalst, Hajo A. Reijers, A. J. M. M. Weijters, Boudewijn F. van Dongen, Ana Karla Alves de Medeiros, Minseok Song, and H. M. W. (Eric) Verbeek. Business process mining: An industrial application. *Inf. Syst*, 32(5):713–732, 2007.

[60] Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The proM framework: A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.

[61] WMP Van Der Aalst, HA Reijers, A. Weijters, BF Van Dongen, AK Alves de Medeiros, M. Song, and HMW Verbeek. Business process mining: An industrial application. *Information Systems*, 32(5):713–732, 2007.

[62] Michal Pěchouček, Viliam Lisý, Branislav Bošanský, and Roman Vaculín. Cooperative adversarial reasoning and planning in complex environments – month 7 interim report. Technical report, Gerstner Laboratory, Czech Technical University in Prague, February 2010.

[63] G. Holmes, A. Donkin, and I.H. Witten. Weka: A machine learning workbench. In *Proceedings of the Second Australia and New Zealand Conference on Intelligent Information Systems*, pages 357–361. Citeseer, 1994.

[64] M. Jakob, J. Tožička, and M. Pěchouček. Collaborative learning with logic-based models. *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pages 102–116, 2008.

[65] A.X. Jiang, K. Leyton-Brown, and A. Pfeffer. Temporal action-graph games: A new representation for dynamic games. In *Proc. 25th Conference on Uncertainty in Artificial Intelligence*, pages 268–276.

Appendix A

Draft of the Paper on Deception

Autonomous Agents and Multi-Agent Systems manuscript No.
(will be inserted by the editor)

Local Search Methods for DCOP in Deceptive Environments

Viliam Lisý · Roie Zivan · Katia Sycara · Michal Pěchouček

Received: date / Accepted: date

Abstract Recent studies have investigated how a team of mobile sensors can cope with real world constraints, such as uncertainty in the reward functions, dynamically appearing and disappearing targets, technology failures end changes in the environment conditions.

In this study we consider an additional element, deception by an adversary, which is relevant in many (military) applications. The adversary is expected to use deception to prevent the sensor team from performing its tasks. We employ a game theoretic model to analyze the expected strategy of the adversary and find the best response. More specifically, we consider that the adversary deceptively changes the importance that agents give to targets in the area. The opponent is expected to use camouflage in order to create confusion among the sensors regarding the importance of targets, and reduce the team's efficiency in target coverage. We represent a Mobile Sensor Team problem using the Distributed Constraint Optimization Problem (DCOP) framework. We propose an optimal method for the selection of a position of a single agent facing a deceptive adversary. This method serves as a heuristic for agents to select their position in a full scale problem with multiple agents in a large area. Our empirical study demonstrates the success of our model as compared with existing models in the presence of deceptions.

Keywords Deception · Sensor network · Game theory · Distributed problem solving

V. Lisý, M. Pěchouček
Agent Technology Center, Dept. of Cybernetics,FEE, Czech Technical University, Technická 2, 16627 Prague 6, Czech Republic
Tel.: +420-22435-7581
Fax: +420-22492-3677
E-mail: {lisy, pechoucek}@agents.felk.cvut.cz

R. Zivan, K. Sycara
Robotics Institute, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, i5213, USA
E-mail: zivanr@cs.cmu.edu

1 Introduction

Recent studies have investigated how a team (or a network) of mobile sensing agents can cope with various realistic elements of real world situations. One study (Jain et al, 2009) has considered uncertainty in the reward functions of agents for different alternative combinations of positions. Jain et. al. propose different ways to balance between the exploitation of the information agents acquire during search and the potential of exploring positions which were not visited yet. Another study (Zivan et al, 2009), considered the different dynamic elements of such a problem including new targets, technology failures of sensors and environment conditions which may reduce the quality of agents' reports. (Zivan et al, 2009) proposed a model which captures these dynamic changes, and distributed algorithms for dynamic adjustment of the agents' deployment to the evolving state of the problem.

While some of the relevant applications in which mobile sensor networks (MSN) are expected to be used, as the Rescue scenario (Marecki et al, 2005) or Maximizing Radio Signal (Jain et al, 2009), are "peaceful" applications in which agents are combating the forces of nature in order to solve the problem, other (military) applications include an adversary which is expected to make attempts to prevent the team from performing their task. For such applications a game theoretic model is appropriate, which can analyze the expected actions of the adversary and find the best response to it.

We consider a MSN that needs to "cover" targets, i.e. to allocate groups of sensors to monitor targets. Sensors are allocated as a function of a target's importance with more important targets getting higher value/number of sensors. In addition, we address the possibility of an adversary to use means for deception that will affect the importance that agents give to targets. Opponents in a military application are expected to use camouflage in order to decrease the importance that the MSN will give targets of high importance and on the other hand, attempt to make insignificant targets appear to be of high importance. Such deception would cause the MSN to select a deployment in which precious resources are used for the surveillance of insignificant targets while the targets of high importance are not covered properly.

Following (Zivan et al, 2009), we represent the problem using the Distributed Constraint Optimization Problem for Mobile Sensing agents Team (DCOP_MST) framework and we aim to apply the local search methods developed for this framework. There are two main reasons why we focus on the local (incomplete) search methods instead of using the optimal solutions provided by the complete algorithms. The first is scalability. The complete methods for solving DCOPs (such as ADOPT (Modi et al, 2005) or DPOP (Petcu and Faltings, 2005)) do not scale favorably with the increasing number of agents in the systems and we want to be able to operate networks with hundreds of sensors. The second reason is robustness to dynamic events and technology failures. In a sensor network, especially in the settings where deception can be expected, sensors can be destroyed or their communication interrupted. Local search methods are more robust in such situations.

Our approach, as in (Zivan et al, 2009), consists of finding an optimal strategy for the selection of a position of a single agent and empirically test the success of this optimal strategy when used by agents in a team of agents with a common goal.

The proposed optimal strategy for a single agent considers multiple targets with various degrees of importance, to which an adversary can increase or decrease the reflected importance. This effect on the reflected importance is bounded. The bound

represents the limitations of camouflage for realistic targets (i.e., it is not realistic to consider that a large army base is disguised as a bush or a single car as a brigade).

We applied our local optimal method to problems with multiple agents in which agents have mobility and sensing limitations. In the distributed (local search) algorithm, agents share their positions with their neighbors and select alternative positions using an algorithm based on the locally optimal method.

The local method is stochastic. It gives the guarantee of optimality if it is consistently used over a longer period of time, but the quality of any single change of assignment cannot be evaluated. This property prohibits using the standard versions of the local search algrithms such as the Distributed Stochastic Algorithm (DSA) (Zhang et al, 2005) or the Max Gain Messages algorithm (MGM) (Maheswaran et al, 2004). Therefore, we present modifications of the algorithms that are suitable for this situation as well as a novel local search method designed specifically for this setting.

Our empirical study evaluates the success of both the single agent method and the local search algorithm for the agent team. In the case of the single agent problem, our proposed method is optimal in the worst case scenario, however, if the adversary is very limited or does not use its deception capabilities, a naïve method can produce high quality results. The same phenomenon was consistently found for the multi agent problem. An experimental comparison of the proposed local search algorithm compared to naïve (deception ignoring) local search algorithms, reveals that the success of the proposed local search algorithm is more apparent when the bounds on the deception capabilities of the adversary are less tight.

The rest of this paper is organized as follows: Section 2 presents related work. In Section 3 we introduce deceptions into the standard DCOP_MST model. Section 4 presents the optimal (stochastic) method for selecting the position of a single agent. Section 5 presents generalisation of the approach to multi-agent case using several local search algorithms designed to deal with the stochastic single agent algorithm. Section 6 describes the methods that were used to create deception in our experimental study. The experimental evaluation is presented in Section 7 followed by a discussion of our findings and our conclusions.

2 Related Work

The review of the related work focuses on two main topics. First, we overview the former results concerning deception in multi-agent systems and game theory. Later we explain the relation of this research to the recently introduced Quantified DCOP framework.

2.1 Former attempts to handle deceptions

Deception has been studied in a limited manor in the multi-agent literature. The work that is most similar to the presented research is Hespanha et al (2000). The authors investigate deception in a simple two-player zero-sum game. One of the players is the attacker that decides to attack one of two targets. The defender can distribute 3 units to defend the targets. The more units are defending the attacked target, the lower is the reward for the attacker. Each of the defending units can be observed with one of two probabilities. The lower is the natural chance to observe the unit and

the higher corresponds to the defender intentionally showing the defending unit in order to deceive the attacker. The authors analyze the optimal strategies for a player that by manipulating the information revealed to her opponent, she is rendering the observations of units useless. We perform a similar analysis for the case of mobile sensors, which requires a novel approach.

Deception in the form of lowering the utility of the available information is investigated also in Root et al (2005). The task there is to plan paths for a team of UAVs, which inspect a given path to be used later by a convoy of ground vehicles, in such a way as to give an adversary that would want to ambush the convoy as little information as possible. Besides flying over the desired path, the UAVs would randomly fly over other paths to deceive the observer.

Other analysis of deception in multi-agent systems include the control mechanism for a team of UAVs creating a single "phantom" aircraft by their movements (Waun and Ozguner, 2004) and the study by Michael and Riehle (2001) that suggests using software decoys in network security. These decoys should deceive the attacker to think the attack was successful, make him continue and allow assessing the nature of its attack.

Deception was studied formally also in the field of game theory. A formal deception game was first formulated as an open problem by Spencer (1973). One player is given a vector of three random numbers from uniform distribution on [0,1]. It changes one of the numbers to an arbitrary number from [0,1] and presents the modified vector to the second player. The second player chooses one position in the vector and receives as its reward the number that was originally on that position. The open question stated in the paper is whether there is a better strategy than randomly choosing one of the positions.

The game was solved by Lee (1993), showing that for the case of 3 numbers, the information provided to second player can be made completely useless, but in case of 4 numbers and changing only one of them, the expected gain of the optimal strategy of the second player is more than the mean value guaranteed by random choice.

A generalized form of the game is solved by Fristedt (1997). The game is played with vector of arbitrary length (n). The first player is allowed to *permute* the vector in a way that only up to m numbers change their positions. The second player then selects the position and obtains the reward corresponding to the number at the position in the original vector. The paper shows that if the first player is allowed to change at least half of the positions, the resulting vector would not contain any information useful for selecting a high number by player two.

The model presented in this paper differs from the previous models by allowing the first player to modify each of the numbers, but only by a given amount. This is a more realistic assumption in real-world (i.e. military) setting then permutations of the importance of the targets. The assumption of limited modification of all targets' importances leads to a novel method for creating the selection strategy that is not based on the results from the papers above.

2.2 Quantified Distributed Constraint Optimization Problem

Recently, a model based on DCOPs for solving an optimization problem in the presence of adversarial behavior was proposed: Quantified Distributed Constraint Optimization

Problem (QDCOP) (Matsui et al, 2010). QDCOP can represent adversarial behavior which prevents a team of agents solving a distributed optimization problem from achieving their goal . QDCOP represents the adversarial behavior by adding an additional set of agents. The difference between the behavior of the cooperative agents to the adversarial agents is represented by quantifiers, either $\exists$ for a cooperative agent or $\forall$ for an adversarial agent. The idea behind the use of such quantifiers is that a cooperative agent would search for the best possible assignment therefore the existance of such a value is enough, while an adversarial agent will select the value which will cause maximal damage therefore the $\forall$ quantifier models the worse case scenario.

The solution of a QDCOP is a set of assignments for the existentially quantified variables that optimizes the goal function if all the generally quantified variables take the worst possible value (Matsui et al, 2010).

The semantics of the QDCOP formulation closely relates to our requirements. The optimal solution of QDCOP is an assignment (a position to each sensor in our case), which optimizes the teams benefit when the adversarial team plays optimally.

However, some elements prevent solving of the problems we aim to solve in this study using QDCOP:

1. The solution of the QDCOP is a deterministic assignment, which can be almost arbitrarily bad compared to the *mixed strategies*[1] allowed in our approach. This fact can be demonstrated for example on a zero-sum version of the matching pennies game (Shoham and Leyton-Brown, 2008). Similar situations often arise in our domain.
2. In this work, we focus on *local serach* methods to achieve scalability to large sensor teams and higher robustness of the method. No local search algorithm for QDCOP is known so far. Moreover, in QDCOPs the adversarial agents cooperate in search with the standard agents. The completeness of the algorithms in previous studies of QDCOP ensure that the worst case scenario will be considered. This may not be true if the adversarial agents participate in a local search algorithm.

3 Problem Definition

3.1 Distributed Constrained Optimization Problem

A *DCOP* is a tuple $\langle \mathcal{A}, \mathcal{X}, \mathcal{D}, \mathcal{R} \rangle$. $\mathcal{A}$ is a finite set of agents $A_1, A_2, ..., A_n$. $\mathcal{X}$ is a finite set of variables X_1,X_2,...,X_m. Each variable is held by a single agent (an agent may hold more than one variable). $\mathcal{D}$ is a set of domains D_1, D_2,...,D_m. Each domain D_i contains the finite set of values which can be assigned to variable X_i. $\mathcal{R}$ is a set of relations (constraints). Each constraint $C \in \mathcal{R}$ defines a non-negative *cost* for every possible value combination of a set of variables, and is of the form $C : D_{i_1} \times D_{i_2} \times \ldots \times D_{i_k} \rightarrow \mathbb{R}^+ \cup \{0\}$. An *assignment* (or a label) is a pair including a variable, and a value from that variable's domain. A *full assignment* is a set of assignments that includes all the variables. The *cost of a full assignment* is the sum of all constraints over the assignments in PA. A *solution* is a full assignment of minimal cost.

[1] Probability distribution over agent's actions

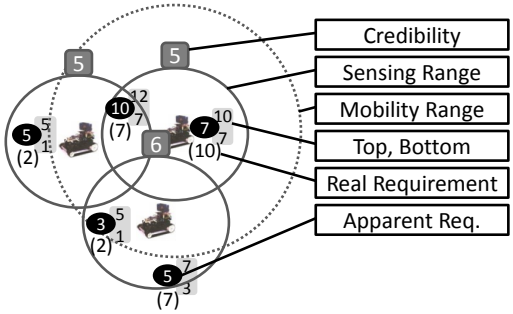

Fig. 1 Schema of the problem. The real environment requirement is not known to the sensors.

3.2 DCOP for Mobile Sensing agents Team (DCOP_MST)

The definition of the problem we aim to solve in this work is similar to the distributed constraint optimization for mobile sensing agents team (DCOP_MST) model as proposed in Zivan et al (2009) with the addition of possible deception. The task in DCOP_MST is to find a deployment for a team of mobile sensing agents, so that they meet the specified requirements for surveillance of individual targets, i.e. points in space. The problem is set to a discretized metric space with a finite set of positions. A network of finite number of agents $A_1, \ldots, A_n$ operates in the space. Each of the agents (A_i) is placed in some position cur_pos_i and it is characterized by three parameters. The *sensing range* (SR_i) is the effective coverage range of the agent, i.e., agent A_i can detect and cover all the targets that are within its sensing range from cur_pos_i. The *mobility range* (MR_i) is the range that an agent can move in a single time step (iteration). And the *credibility* $Cred_i$ is a real positive number representing the quality of the sensor.

DCOP_MST further defines an environmental requirement function ER. This function expresses for each point in the area, the required joint credibility amount (the sum of the credibility variables) of agents that have this point within their sensing range (i.e. are covering it). Function Cur_DIFF calculates for each point in the area the difference between the current value of the ER function and the sum of the credibilities of the agents which are currently covering it. Formally, if we denote the set of agents within their sensing ranges from point p by SR_p then:

$$Cur\_DIFF(p) = ER(p) - \sum_{A_i \in SR_p} Cred_i \tag{1}$$

The global goal of the agents is to cover all the targets according to ER (i.e. to reduce the largest value of Cur_DIFF to zero) in a pre-defined number of time steps. Since this goal cannot always be achieved, we define a more general goal which is to minimize the largest value of the Cur_DIFF function over all targets in the area.

3.3 DCOP_MST with deception

In addition to the original properties of DCOP_MST, we define the properties relevant for deception. We assume that an adversary with limited capabilities has used means

of deception to make the environmental requirements appear to be $v(p)$ for each point p. The capabilities of the adversary are expressed by a pair of functions Δ^+ and Δ^-. For each point in space, the apparent environemnt requirement of the position ($v(p)$) can be any number between $ER(p) - \Delta^-(p)$ and $ER(p) + \Delta^+(p)$.

The sensors do not have the knowledge about the *real* environment requirement of the point ($ER(p)$), but only about the *apparent* environment requirement $v(p)$. Moreover, the sensors have some information about the adversary capabilities, options and effort put into deception. This knowledge is represented as another pair of functions *bottom* and *top*. These functions relate the apparent requirements to the real ones for the individual points in space. The unknown real ER of the position ($ER(p)$) can be any number between $bottom(p)$ and $top(p)$.

The relation between the two pairs of function defining the capabilities of the adversaries is not tight. The only requirement for the mappings is that

$$\forall p \quad ER(p) \in [bottom(p), top(p)] \tag{2}$$

The goal of the sensors remain to cover the *real* requirements given by ER.

Even though our methods were designed for this general formulation, we often use a special form of defining the adversary capability in the ilustrative examples and in the experimental evaluation. It is determined by a limited interval $[0, MaxImp]$ and the constant Δ, which is the same for all points. The ER of any point can be modified by Δ up or down as long as the resulting apparent requirement stays in the interval $[0, MaxImp]$. The same Δ defines also the knowledge available to the sensors. The real ER of a point can be Δ higher or lower then the apparent one. Note that this does not mean that the resulting itervals are the same. Formally, the deception capabilities with fixed Δ are defined as:

$$\begin{aligned} \forall p \quad & \Delta^+(p) = \min(MaxImp - ER(p), \Delta) \\ & \Delta^-(p) = \min(ER(p), \Delta) \\ & top(p) = \min(v(p) + \Delta, MaxInt) \\ & bottom(p) = \max(0, v(p) - \Delta) \end{aligned}$$

Figure 1 illustrates the problem of DCOP_MST with deceptions.

For this paper, we impose a specific restriction on the DCOP_MST framework. We assume that the sensing range of all the sensors is restricted to their position, i.e. the sensors can cover only the targets at its current position.

In the following, we often use for brevity the term *importance* instead of environment requirement and *target* instead of a position with nonzero environment requirement.

4 Formal Deception Game

This section presents one of the main contributions of the paper. It shows the derivation of sensor placement methods that are robust to deception. These methods allow choosing the targets with high importance while keeping the right amount of randomization to prevent the adversary from misleading the sensors via deception.

4.1 Formal Game Definition

We define the problem being solved in this section as a game between a single sensor and an adversary that uses deception (such as decoys or camouflage) to make the sensor less efficient. The adversary starts with n targets of various positive importance values $(y_1, \ldots, y_n)$. It can use deception in order to generate a perceived importance of each target $(v_1, \ldots, v_n)$, but the types of the targets and the effort he can spend on the camouflage for each of the targets does not allow him to modify the importance of the target arbitrarily. The perceived importance for each target is bounded to an interval:

$$v_i \in [y_i - \Delta_i^-, y_i + \Delta_i^+]$$

where Δ_i^+ as well as Δ_i^- are non-negative real numbers and $y_i - \Delta_i^- \geq 0$ for each target i. The sensor can perceive the modified importance values and we assume it knows the boundaries $bottom_i, top_i^-$. Based on this information, it decides to cover a single target, trying to maximize the real importance of the target. We represent the strategy of the sensor as a probability distribution of covering individual targets $(x_1, \ldots, x_n)$. Deterministically selecting single target is a degenerate case of this distribution that is one for one target and zero for all the other targets.

Our goal is to create a sensor placement method that would be robust against deception, so we perform analysis of the worst case scenario. This also relieves us from the need of defining exact relation between the functions Δ^+, Δ^- and $bottom, top$. The natural formalization of the problem would then be maximizing the expected value of the covered target in case of the worst case $\mathbf{y}$ that is consistent with the observations $\mathbf{v}$.

$$\max_{\mathbf{x}} \min_{\mathbf{y}} \sum_{i=1}^{n} x_i y_i$$

However, this optimization is trivial. For any fixed $\mathbf{x}$, the worst case corresponds to setting all $y_i = bottom_i$. It says that the sensor covers less important targets in case that all the targets are generally less important. It constitutes the worst case in problem instance rather than some smart information manipulation by the adversary. In order to focus on adversary deceptive strategies, we use alternative optimization criteria to reduce the influence of the underlying problem.

4.1.1 Relative to the Random Strategy

If we assume that the sensor player is rational (makes optimal decisions), it should be able to figure out when the information provided by the adversary is useless and hence it should always reach at least the payoff of random strategy corresponding to the uniform probability distribution over the targets.

$$\frac{1}{n} \sum_{i=1}^{n} y_i$$

We can reformulate the objectives of the players relative to this assured value. The sensor player tries to gain more than what is provided by the random strategy and the

adversary uses deception to make the information useless and force the opponent to play the random strategy. Formally, the task that the sensor solves is

$$\max_{\mathbf{x}} \min_{\mathbf{y}} \frac{\sum_{i=1}^{n} x_i y_i}{\frac{1}{n}\sum_{j=1}^{n} y_j} \quad \text{s.t.} \quad \begin{array}{l} \mathbf{0} \leq \mathbf{x} \leq \mathbf{1} \\ \sum_{i=1}^{n} x_i = 1 \\ \mathit{bottom} \leq \mathbf{y} \leq \mathit{top} \end{array} \tag{3}$$

4.1.2 Relative to the Optimal Selection

The previous criterion originated from the random strategy that an agent can easily perform. As a result, the solution based on this criterion always achieves at least the quality of the random strategy. However, it is not the only option of reducing the influence of problem instance to the optimization. We can formulate the problem as finding a strategy, that is as close to the (unperformable) optimal strategy as possible. Then the optimization then becomes

$$\max_{\mathbf{x}} \min_{\mathbf{y}} \frac{\sum_{i=1}^{n} x_i y_i}{\max_{j\in\{1\ldots n\}} y_j} \tag{4}$$

subjected to the same constraints as in formula 3. The last option we explore is to substitude the ratio to the comarision strategy with a difference.

$$\max_{\mathbf{x}} \min_{\mathbf{y}} \left[\left(\sum_{i=1}^{n} x_i y_i \right) - \max_{j\in\{1\ldots n\}} y_j \right] \tag{5}$$

All these problem formulations still require finding the strategy that assures the highest expected coverage, but they remove the trivial cases and provide more interesting solutions of the game. The reminder of this section provide the means to solve these optimization problems optimally and the applicability tradeoffs of individual formulations are experimentally evaluated and disscussed in Section 7.

4.2 Worst Case for Fixed Sensor Strategy

The optimal solutions for each game formulation can be found in form of a linear program. We use the standard method often used in game theory to rewrite the *max-min* optimization to *max* optimization for the price of defining additional constraints (e.g. Shoham and Leyton-Brown, 2008). In order to do that efficiently, we need to reduce the set of worst case $\mathbf{y}$ for any $\mathbf{x}$ to a small (polynomial) size. Directly from the problem formulation, the set of possible real importance vectors consistent with the observation is continuum.

4.2.1 Relative to the Random Strategy

In this subsection, we show what is the worst case $\mathbf{y}$ for any fixed $\mathbf{x}$ and for the case of optimization in formula 3.

Lemma 1 *The function* $f(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{n} x_i y_i}{\frac{1}{n}\sum_{j=1}^{n} y_j}$ *is monotonic in any* y_c *if all other* y_i *and* x_i *are fixed.*

Proof For arbitrary $c \in 1..n$, we compute the sign of the partial derivative with respect to y_c. The constant $\frac{1}{n}$ in the denominator can be omitted.

$$\operatorname{sgn}\left(\frac{\partial}{\partial y_c}\left(\frac{\sum_{i=1}^{n} x_i y_i}{\sum_{j=1}^{n} y_j}\right)\right) = \operatorname{sgn}\left(\frac{x_c \sum_{j=1}^{n} y_j - \left(\sum_{i=1}^{n} x_i y_i\right) 1}{\left(\sum_{j=1}^{n} y_j\right)^2}\right) = \operatorname{sgn}\left(\sum_{i=1}^{n} (x_c - x_i) y_i\right) \quad (6)$$

In case of $i = c$ the term $(x_c - x_i) = 0$ and it makes formula (6) independent of the value of y_c. The sign of the derivative is constant in y_c, hence the function is monotonic in y_c.

An immediate corollary of the fact above is that in finding the worst case $\mathbf{y}$, we need to consider only the vectors with all their coordinates set to the extreme values.

Corollary 1 *For all the coordinates in the worst case* $\mathbf{y}$*,* $y_i = bottom_i$*,* $y_i = top_i$ *or the coordinate does not influence the optimized value.*

Proof Let $\mathbf{y}$ be the worst case for a fixed $\mathbf{x}$ and consider an arbitrary coordinate y_i. Lemma 1 says that the sign of the derivative of f according to y_i is constant. If it is (constantly) zero, the optimized value does not depend on the value assigned to y_i. Otherwise f is either strictly increasing or strictly decreasing in y_i. If the sign of the derivative is (constantly) $+1$, any $y_i > bottom_i$ can be decreased to $y_i = bottom_i$ decreasing the optimized function. If it is (constantly) -1, any $y_i < top_i$ can be increased to $y_i = top_i$ decreasing the optimized function.

Lemma 2 *For any fixed* $\mathbf{x}$*, there exists a coordinate* $c \in 1 \ldots n$*, such that the worst case* $\mathbf{y}$ *can be constructed as*

$$\begin{array}{ll} \text{if } x_i \geq x_c & \text{then } y_i = bottom_i \\ & \text{else } y_i = top_i \end{array}$$

Proof The proof is by induction on the number of unset coordinates. First we show that we can always set some coordinates of $\mathbf{y}$ in the global optimum at the beginning. Then we show that after setting any number of coordinates, we can find one more that can be set in the global optimum.

I) Assume the coordinates

$$b = \arg\min_{i \in 1..n} x_i,\ t = \arg\max_{i \in 1..n} x_i$$

Then for any $i \in 1..n$ holds

$$((x_b - x_i) y_i) \leq 0,\ ((x_t - x_i) y_i) \geq 0$$

As a result, for the sign of the partial derivative in formula (6) the following holds:

$$\operatorname{sgn}\left(\sum_{i=1}^{n} (x_b - x_i) y_i\right) \leq 0,\ \operatorname{sgn}\left(\sum_{i=1}^{n} (x_t - x_i) y_i\right) \geq 0$$

for all possible values of other x_i and y_i. That means that in the globally optimal $\mathbf{y}$, we can set

$$y_b = top_b,\ y_t = bottom_t$$

II) Assume that some of the values y_i are already set. Without loss of generality rename the coordinates so that the set coordinates are $1, \ldots, k$. Then continuing from formula 6

$$S_c = \operatorname{sgn}\left(\underbrace{\sum_{i=1}^{k}(x_c - x_i)y_i}_{A_c} + \sum_{i=k+1}^{n}(x_c - x_i)y_i\right)$$

and the term A_c is a fixed constant for each c. Using the same argument as in part I)

(a) if $c_t = \arg\max_{i \in k+1..n} x_i$ & $A_{c_t} \geq 0$ then $S_{c_t} \geq 0$ for all x_i, y_i and we can set $y_{c_t} = bottom_{c_t}$
(b) if $c_b = \arg\min_{i \in k+1..n} x_i$ & $A_{c_b} \leq 0$ then $S_{c_b} \leq 0$ for all x_i, y_i and we can set $y_{c_b} = top_{c_b}$

In order to finish the proof, we have to show that at least one of the conditions in (a),(b) always holds. Clearly $x_{c_t} \geq x_{c_b}$. Assume that (b) does not hold

$$A_{c_b} > 0 \Leftrightarrow \sum_{i=1}^{k}(x_{c_b} - x_i)y_i > 0 \xrightarrow{x_{c_t} \geq x_{c_b}} \sum_{i=1}^{k}(x_{c_t} - x_i)y_i > 0 \Leftrightarrow A_{c_t} > 0 \Rightarrow (a) \text{ holds}$$

The induction always sets the y_i for the unset target with minimal x_i to top_i or the y_i corresponding to the unset target with maximal x_i to $bottom_i$. As a result, if all of the coordinates are set, the last set coordinates define the x_c from the proposition.

4.2.2 Relative to the Optimal Selection

The worst case in the formulation relative to the optimal target selection has a simpler form.

Lemma 3 *For any fixed* $\mathbf{x}$*, there exists a coordinate* $c \in 1 \ldots n$*, such that the worst case* $\mathbf{y}$ *in formula 4 can be constructed as*

$$\begin{array}{ll} \text{if } i = c & \text{then } y_i = top_i \\ & \text{else } y_i = bottom_i \end{array}$$

Proof For arbitrary $\mathbf{x}$ and $\mathbf{y}$, let $c = \arg\max_j y_j$. Then the optimized function from formula 4 is

$$\frac{x_c y_c + \sum_{i \in \{1 \ldots n\} \setminus \{c\}} x_i y_i}{y_c}$$

This function is always non-increasing with increasing y_c, because of $0 \leq x_c \leq 1$; hence the worst case can always have the most important target set to its upper bound $y_c = top_c$.

The function is non-increasing with decrease of any other y_i, $i \neq c$ and that is why all the other targets can in the worst case be decoys, i.e., $y_i = bottom_i$.

The same lemma with a very similar proof holds also for the optimization in formula 5.

Lemma 4 *For any fixed* $\mathbf{x}$*, there exists a coordinate* $c \in 1 \ldots n$*, such that the worst case* $\mathbf{y}$ *in formula 5 can be constructed as*

$$\begin{array}{ll} \text{if } i = c & \text{then } y_i = top_i \\ & \text{else } y_i = bottom_i \end{array}$$

4.3 Optimal Strategies for the Sensor

In the previous section, we have managed to restrict the set of all situations consistent with the observations that constitutes the worst case for any x to a finite set for all of the suggested optimization criteria. In this section, we present the corresponding linear programs for finding the optimal strategy x for each case. The case of strategy relative to the optimal selection is much simpler and it is presented at the end of this chapter. We start with the optimization relative to the random strategy, for which the set of possible worst cases is based on the previous section still exponential in the number of targets. When a *max-min* optimization is transformed to the corresponding *max* optimization (e.g., Shoham and Leyton-Brown, 2008), a constraint must be added for each strategy $\mathbf{y}$ that can possibly be the worst case for a valid strategy $\mathbf{x}$ of the maximizing player.

4.3.1 Relative to the Random Strategy

For the optimization relative to the random strategy, Corollary 1 assures that we need to add a finite, but exponential number of constraints. Each target could have been camouflaged all the way to the higher or all the way to the lower importance.

Corollary 2 *If we denote the set of all targets T, the optimal strategy for the sensor in the deception game can be computed by the linear program*

$$\begin{array}{lll} & \mathbf{1} \geq \mathbf{x} \geq \mathbf{0} \\ \max_{\mathbf{x},z} z \quad s.t. & \sum_{i=1}^{n} x_i = 1 \\ & \forall A \subseteq T \ \frac{\sum_{i \in A} x_i bottom_i + \sum_{i \in (T \setminus A)} x_i top_i}{\sum_{i \in A} bottom_i + \sum_{i \in (T \setminus A)} top_i} \geq z \end{array} \tag{7}$$

The rest of this subsection shows, how the number of constraints can be further reduced. First, we need to define the notion of partial ordering of the targets and present a technical lemma.

Definition 1 We define the relation $t_i \trianglerighteq t_j$ on the targets that have both the upper and lower bounds of their intervals ordered.

$$bottom_i \geq bottom_j \ \& \ top_i \geq top_j$$

Note, that this relation is transitive, because of the transitivity of the ordering of the bounds.

Lemma 5 *If $t_k \trianglerighteq t_l$ then there is an optimal strategy for the sensor for which $x_k \geq x_l$.*

Proof Remember that the worst case for any sensor strategy is modifying the importance of the targets to the bound of the interval. If both of the inequalities in the definition of $\trianglerighteq$ hold as equalities, the ordering of the probability of covering them (x_k, x_l) cannot make difference for the sensor player.

Next we assume both inequalities defining $t_k \trianglerighteq t_l$ are strict. We consider a modified game, where the players are presented the real importance of the targets and they simultaneously decide on which target to cover on the sensor side and how to modify the real importance of the targets in the available bounds on the adversary side. The optimal strategy for the sensor in this game corresponds to the deception-robust play

in the original game. The formulation of the alternative game as *max-min* optimization is exactly as in formula 3. The game is a two player full information zero-sum game with finite number of strategies on each side and hence it has a Nash equilibrium with a unique value (e.g. Shoham and Leyton-Brown (2008)). Let $(\mathbf{x}, \mathbf{y})$ be a pair of the equilibrium strategies.

With coordinates renamed such that $\forall i \in 2..n\ x_{i-1} \geq x_i$, Lemma 2 says that there is a coordinate c, such that $\mathbf{x}$ optimizes

$$\frac{n}{\sum_{j=1}^{n} y_j} \left(\sum_{i=1}^{c-1} x_i bottom_i + \sum_{i=c}^{n} x_i top_i \right)$$

In order to prove that in this case $x_k \geq x_l$, we show that if $x_k < x_l$ then swapping values of x_k and x_l improves the strategy of the sensor player for the fixed $\mathbf{y}$ which would be a contradiction with $(\mathbf{x}, \mathbf{y})$ being the Nash equilibrium. The sensor player would have an incentive to change its strategy if the adversary keeps its strategy unchanged. The fixed strategy of the adversary implies that the denominator of the optimized function stays the same.
For $l < c \leq k$ the strategy is improved if

$$\begin{gathered} x_l bottom_l + x_k top_k < x_k bottom_l + x_l top_k \\ (x_l - x_k) bottom_l + (x_k - x_l) top_k < 0 \\ \underbrace{(x_l - x_k)}_{>0} (bottom_l - top_k) < 0 \\ bottom_l < top_k \end{gathered}$$

The last inequality holds from strict $t_k \trianglerighteq t_l$, because

$$bottom_l \leq top_l < top_k$$

For $l < k < c$ the strategy is improved if

$$\begin{gathered} x_l bottom_l + x_k bottom_k < x_k bottom_l + x_l bottom_k \\ bottom_l < bottom_k \end{gathered}$$

For $c \leq l < k$ the strategy is improved if

$$\begin{gathered} x_l top_l + x_k top_k < x_k top_l + x_l top_k \\ top_l < top_k \end{gathered}$$

All the final inequalities trivially hold from $t_k \trianglerighteq t_l$.

The last two cases to finish the proof are if just one of the inequalities holds strictly. If $bottom_k = bottom_l$ and $top_k > top_l$ then for cases besides $l < k < c$ the argument about improving the value for the sensor player above holds. In this case, the pay-off of the sensor player does not change, but we show that if the sensor player switches the probabilities of covering targets t_k and t_l, the adversary does not have an incentive to deviate from its original strategy. This means that the original strategy of the adversary and the modified strategy of the sensors form a Nash equilibrium.

Remember the construction of the worst response of the adversary in the proof of Lemma 2. If the adversary modifies the real importance of the targets one by one, it arrives first to x_l as before. All the targets that are camouflaged so far are set on the same values as for the unmodified sensor strategy. The direction in which the target t_k will be camouflaged does not depend on the bounds on y_k. It depends only on x_l and

the already camouflaged targets. As a result the target t_k, currently corresponding to the value x_l will be camouflaged in the same direction as t_l before, setting it to the same value. If construction of the optimal strategy for the adversary continues and encounters x_k, all the data used to compute the direction of deceptively altering the target t_l are the same as with the original sensor strategy and it is altered to the same value as t_k before. That is why the worst response to the modified sensor strategy is the same as to the original sensor strategy. The proof of the last case where the second condition holds as equality is symmetric.

Using the previous lemma, we can reduce the number of constraints needed in the linear program (7).

Theorem 1 *The optimal solution for the sensor player in optimization relative to the random strategy can be computed by the linear program*

$$\begin{array}{lll} & \mathbf{1} \geq \mathbf{x} \geq \mathbf{0} \\ & \sum_{i=1}^{n} x_i = 1 \\ \max_{\mathbf{x},z} z \quad s.t. & \forall t_i \trianglerighteq t_j, \nexists t_k \; t_i \trianglerighteq t_k \trianglerighteq t_j : \quad x_i \geq x_j \\ & \forall A \in T, \nexists i \in A, j \in (T \setminus A) t_j \trianglerighteq t_i : \\ & \quad \frac{\sum_{i \in A} x_i bottom_i + \sum_{i \in (T \setminus A)} x_i top_i}{\sum_{i \in A} bottom_i + \sum_{i \in (T \setminus A)} top_i} \geq z \end{array} \tag{8}$$

Proof From Lemma 5, we know that introducing the constraints on the ordering of x_i will not prevent us from finding the optimal solution. The linear program finds the correct solution, if for any strategy $\mathbf{x}$ that respects the ordering, a constraint representing the worst response to the strategy is present. From Lemma 2, we do not have to consider that the adversary decreases the importance of a target that is covered with higher probability than another target for which it increased the importance.

Now consider a basic setting, in which a single sensor can cover one of n targets and the sensor knows that the importance of each of the targets could have been modified by a fixed Δ up or down. Even if the resulting apparent importance is bounded to a predefined interval [0,MAX_IMP], the relation $\trianglerighteq$ creates a full ordering. It is the same ordering as the ordering of the apparent importance. For this kind of settings, the linear program computing the strategy for the sensor has only small number of constraints that is linear in the number of targets.

Corollary 3 *If the ordering induced by the relation $\trianglerighteq$ is full and if we rename the targets so that*

$$\forall i \in 2..n \;\; t_{i-1} \trianglerighteq t_i$$

then the optimal strategy for the sensor can be computed by the linear program

$$\begin{array}{lll} & \mathbf{1} \geq \mathbf{x} \geq \mathbf{0} \\ & \sum_{i=1}^{n} x_i = 1 \\ \max_{\mathbf{x},z} z \quad s.t. & \forall i \in 2..n \;\; x_{i-1} \geq x_i \\ & \forall j \; \frac{\sum_{i=1}^{j} x_i bottom_i + \sum_{i=j+1}^{n} x_i top_i}{\sum_{i=1}^{j} bottom_i + \sum_{i=j+1}^{n} top_i} \geq z \end{array} \tag{9}$$

4.3.2 Relative to the Optimal Selection

The number of possible worst case $\mathbf{y}$ for any $\mathbf{x}$ in the case of optimization relative to the optimal selection is linear in the number of targets directly from Lemma 3 for the ratio case as well as from Lemma 4 for the differece case.

Theorem 2 *For optimizing the ratio of the importance of the covered target to the importance of the best possible selection (formula 4), the optimal strategy of the sensor is the result of*

$$\max_{\mathbf{x},z} z \quad s.t. \quad \begin{array}{l} \mathbf{1} \geq \mathbf{x} \geq \mathbf{0} \\ \sum_{i=1}^{n} x_i = 1 \\ \forall j \ \frac{x_j top_j + \sum_{i \neq j} x_i bottom_i}{top_j} \geq z \end{array} \tag{10}$$

And similarly in case of difference from the optimal selection.

Theorem 3 *For optimizing the difference between the importance of the covered target to the importance of the best possible selection (formula 5), the optimal strategy of the sensor is the result of*

$$\max_{\mathbf{x},z} z \quad s.t. \quad \begin{array}{l} \mathbf{1} \geq \mathbf{x} \geq \mathbf{0} \\ \sum_{i=1}^{n} x_i = 1 \\ \forall j \ \left(\sum_{i=1}^{n} x_i bottom_i\right) - top_j \geq z \end{array} \tag{11}$$

All the methods derived in this section give a mixed strategy (distribution) over alternative sensor positions. The applicability tradeoffs of different formulation are shown and discussed in Section 7.

5 Multiple Coordinated Sensors

In this section, we propose local search algorithms for a team of agents aiming to cover multiple targets in a given area, which use the solution for the formal deception game described in the previous section as a deception-robust heuristic. We start by presenting local search methods for standard DCOP_MST and then present the adjustments required to convert them into deception aware methods.

5.1 Local search for DCOP_MST

We assume that the ER function used by the team of agents is accurate. As in standard DCOP, the local search method which we present perform in a synchronous manner. The neighborhood of an agent is defined as all the other agents that can possibly cover the same target as the agent, after a single iteration of the algorithm. In each synchronous step of the algorithm all neighboring agents exchange messages. The main building block which all algorithms use is the method for a single agent to decide what is the best alternative position within its mobility range. An immediate trivial choice in the domains with sensing range that allows covering only single position is the point with the highest Cur_DIFF value. However, for the case that this point can be

covered from multiple positions, an efficient method is proposed in Zivan et al (2009) for selecting the optimal position among them.

The local search methods proposed by Zivan et al (2009) are based on the following two algorithms:

1. MGM_MST: as in standard DCOPs, after finding the best alternative position as described above, an agent sends to its neighbors the utility improvement (or in our case, reduction in the Cur_DIFF function) it expects to get by moving to this position. The agents, whose best expected reduction is higher than their neighbors', perform the move. Ties are broken according to agent's indexes.
2. DSA_MST: An agent may move to the best alternative position only if the reduction in the Cur_DIFF value is positive. In that case, it performs a random choice to move with probability p or stay at its current position otherwise. In (Zivan et al, 2009), p was set to 0.6.

Zivan et al (2009) propose novel exploration methods that improve the performance of these algorithms, when the technology, e.g., sensing and mobility ranges, is limited.

5.2 Deception aware Local Search methods

Introducing deception to the DCOP_MST model adds one important challenge which is not faced by standard local search methods for DCOP. As shown in the previous section, the locally optimal strategy is not deterministic as in the standard model. The result of the methods proposed in Section 4 is a probability distribution over the alternative positions an agent has for covering individual targets. This fact requires modifications of the mentioned local search algorithms, which build upon the ability of an agent to detect her single best alternative and quantify its quality.

5.2.1 Deception aware DSA

In each synchronous step in the DSA algorithm, agents are required to compute their optimal alternative assignment (position with the largest reduction of the Cur_DIFF function in our case) under the assumption that all the other agents keep their assignments unchanged. Using the deception aware local methods presented in Section 4, many local reductions have incomparable quality. It is often not possible to say that one position is strictly better than another. The benefit of using the proposed algorithm demonstrates only statistically, if the agents follow the probability distribution given by the method. In other words, the basic building block of finding the best alternative, on which local search methods are based on, is not possible in many cases.

If the upper bound on the possible real importance of the currently covered target is lower than the lower bound on the possible real importance of a target covered from an alternative position, then covering the alternative target is clearly better than covering the current one. However if this does not hold, any ordering of the real importance of the positions is possible.

If we want to apply DSA in local search with a stochastic local improvement function, we need to use a variant of DSA proposed by Zhang et al (2005) (DSA-C), which allows the sensors to move not only in cases of positive local reduction, but also if the best new position is not worse than the current one. If we disallowed moving to the

incomparable states, the deception-aware algorithm would have very restricted exploration capabilities.

We use this version of DSA for the deception ignoring variant as well. It allows additional exploration in the DCOP_MST problem. It is often beneficial for an agent that cannot see any more important targets from its current location to move to an equivalently good position, because it changes the sets of its possible assignments in the next steps of the algorithm.

5.2.2 Deception aware MGM

When we try to incorporate the stochastic local method in MGM, we encounter similar problems as explained above for DSA. It is not possible to define a strict ordering between the alternative assignments and for the same reason, it is not possible to enumerate the actual gain of a change in assignment by a single agent.

However, the main motivation of the MGM algorithm is clear. If two neighboring agents change their assignment simultaneously to what they consider best if all the others do not change their assignments, the resulting combination of assignments can be arbitrarily bad. The main benefit of MGM is that only one agent in each neighborhood moves in a single step. The second benefit of MGM is that the agents that make more difference tend to move first; hence speeding up the convergence.

In the deception aware variant of MGM, we keep just the first benefit. If we decide on the agents replacing their assignment solely according to the indices, some agents would tend to change their assignments much more often than others, which would lead to a very limited exploration of possible assignments. If we want to explore the assignments for individual agents evenly, we should ensure that all the agents will perform their local reductions as equally often as possible. That is why in the deception aware variant of MGM, each agent counts how many times it has already changed location and shares this information with its neighbors in each step. Only the agent that moved the least times in its neighborhood moves. The ties are again broken by agents' indices.

5.3 Problem Specific Heuristic

Besides the modificatons of the standard DCOP local search methods, we propose a problem specific local search heuristic. Our goal is to create a heuristic that would satisfy three requirements.

1. It should minimize the chance of placing redundant sensors on a target. A sensor is redundant on a target (t), if even after removing the sensor from the target, the sum of the credibility of the remaining sensors covering it is higher than $top(t)$.
2. Increase the probabilities that targets will be covered.
3. Each agent should keep the ratios between the probability of covering each of the targets in its range by any of the agents as similar to the probability distribution given by its local method as possible. This property ensures that the chance for this agent to be deceived is minimized.

A simple solution is that in each step, each agent computes its distribution over its assignments (possible targets to cover) and sends it to its neighbors together with a random number for each target. After obtaining all the distributions from all neighbors,

Input: $neighbors$ – set of neighbors; $dist : A \to (T \to [0,1])$ – received distribution; $rnd : A \to (T \to [0,1])$–received random numbers

```
1: myDist := dist(this)
2: myTargets := keys(myDist)
3: for t ∈ myTargets do
4:    higher = 0
5:    for a ∈ neighbors do
6:       if myDist(t) < dist(a)(t)
           or (myDist(t) = dist(a)(t) and rnd(this)(t) < rnd(a)(t)) then
7:          higher := higher + Cred_a
8:       end if
9:    end for
10:   if higher ≥ top(t) then
11:      myDist(t) := 0
12:   end if
13: end for
14: normalize(myDist)
15: return select target according to myDist
```

Fig. 2 Method executed by an agent in the $maxStay$ heuristic after receiving messages from its neighbors.

the agent sets the probability of covering some targets where it might be redundant to zero.

The pseudo-code of this process is presented in Figure 2. For each target the agent considers, it computes the sum of credibility of the neighbors whose distributions include higher probability to cover the target than its own. If the sum is higher than the maximal possible real requirement of the target, it sets its probability of covering the target to zero. After analyzing all targets, the distribution on the remaining targets is normalized and the agent chooses its assignment according to the modified distribution. If no targets remain in the distribution, the agent moves randomly.

Ties are broken using random numbers agents pick for each target. The reason for using the random numbers instead of indices for breaking ties ensures uniform assignment of the targets. If the algorithm preferred coverage by the agents with higher indices, they would end up covering more targets than the agents with lower indices. In general, the method results in agents reducing the probability that they will cover some target. In the extreme case, an agent can reduce the probability for all targets which it had positive probability to cover in the original distribution. In order to prevent a completely uniform selection of position in such a case, we add a small constant probability for all targets within range. Thus, if an agents did reduce the probability of all targets in its original distribution, she would still prefer covering a target than avoiding any coverage.

This heuristic has several favorable properties. First, all the agents move in each iteration, which allows fast convergence. Second, the algorithm never creates redundant agents (e.g., in case of agents with credibility which allows them to fully cover the requirement of any target, two agents never go for the same target). However, in a single step, a target can be under-covered if the agents that did not ignore the target did not want to cover it with probability 1. Note, that in case a target has not been covered after a single iteration, it might be covered after the next iteration. In fact, the set of agents that consider it for coverage might have changed due to agents' movements. Third, the probability for any target to be covered is never lower than the

Input: T – set of targets; $ER : T \to \mathbb{R}^+$ – real importance; $\Delta^+, \Delta^- : T \to \mathbb{R}^+$ – the boundaries on the apparent importance
Output: $masked : T \to \mathbb{R}^+$ – apparent importance of the targets
1: $S := \{t_1 \in T : \forall t_2 \in T \;\; ER(t_2) - \Delta^-(t_2) < ER(t_1) + \Delta^+(t_1)\}$
2: $d := \arg\min_{t \in S} ER(t)$
3: **for** $t \in T$ **do**
4: **if** $ER(t) > ER(d)$ **then**
5: $masked(t) := ER(t) - \Delta^-(t)$
6: **else**
7: $masked(t) := ER(t) + \Delta^+(t)$
8: **end if**
9: **end for**
10: **return** $masked$

Fig. 3 $DeceiveMax$ – a method for optimally deceiving the naïve selection of the apparently maximal target.

original probability calculated by any of the agents using the local deception aware algorithm.

6 Deception Algorithms

The methods presented in Section 4 were developed with the worst case deception in mind. However, creating the optimal deception is a nontrivial problem of its own. For the general problem with multiple agents and multiple targets, there is no straight forward way to compute the optimal deception and the solutions are not unique. That is why in this section we describe the (heuristic) algorithms we use for creating deception in our experimental study. Generally, we were trying to create methods that would minimize the efficiency of the naïve (deception ignoring) methods as much as possible. All the algorithms are designed for sensors with sufficient credibility to fully cover the requirement of any target, but some of them are efficient even in the more general setting.

6.1 Single Sensor case

We start with the single sensor formal deception game (i.e. one sensor covering one of the targets without any mobility restrictions). In this case, the optimal deception will minimize the real importance of the target that will appear to be the most important target after the deception is applied.

One option for creating the optimal deception in this scenario is to follow the algorithm presented in Figure 3. The apparent importance of all targets are modified to the boundary of their possible importance values as suggested by Corollary 1. The algorithm finds the optimal target d (as decoy) that would be covered by the naïve algorithm. It is the least important target that may be modified to appear as the most important target. The apparent importance of d is set to the maximal value and the apparent importances of all the targets that are more important than d are decreased. As a result, the target d appears to be the most important. This is regardless of the apparent importance of all targets with real importance lower then the importance of d.

The performance of the naïve algorithm does not depend on the apparent importance of the targets with importance below d. However, they can influence the performance of the deception aware algorithms. In order to evaluate this effect, we use two versions of the algorithm. The version in Figure 3 is inspired by the worst case described by Lemma 2. It increases the importance of the targets making the stochastic algorithms more likely to pick the less important targets. The other variant we use for comparison is to decrease the apparent importance of all the targets less important than d. This variant should allow the deception-robust algorithms to perform better and we denote it by $DeceiveMax^{+}$.

6.2 Multi-agent Case

Generating the optimal deception in the more general scenario, which includes multiple agents with limited mobility, is a much more complicated task and therefore the algorithms we use are heuristic. The main reason why creating a good deception in this case is hard are the different individual views of the agents:

- Two agents consider covering different, partially overlapping, subsets of targets.
- Different subsets of targets are considered from consecutive positions of a sensor.
- Part of the targets can be (partially) covered by other agents in the previous steps.

On the other hand, there is only one fixed deception that should deceive as many sensors as possible. A situation, in which the local view of the sensor can influence the optimal deception is demonstrated in the following example.

Example 1 Assume that a sensor can cover targets with real importances $(9, 8, 7)$ from a single position in the area. If the deception capability is constantly 2 ($\Delta = 2$), the optimal deception without considering mobility restrictions based on the algorithm in Figure 3 would be $(7, 6, 9)$. However, if the agent considers the situation from a position, where the third target is not in its mobility range, based on the remaining targets $(7, 6)$, the naïve algorithm easily selects the most important target and receives the reward of 9. In this case, a more efficient deception against the naïve sensors would be $(7, 8, 9)$. The agent in the alternative position would cover the target with apparent importance 8 and obtain the reward 8.

Based on the motivation from the above example, the heuristic used for creating the deception in cases with limited mobility ranges tries to make the gradient of the apparent importance opposite to the gradient of the real importance on as large subsets of targets as possible. It maximizes the chance that two randomly selected targets will have the ordering of their apparent importance oposite to the ordering of their real importance. We have developed two algorithms based on this idea. The first just uses the idea on the whole set of targets without considering their spacial relations (Figure 4) and the second uses additional information about the sensors' starting positions as well as the information about the exact positions of the targets (Figure 6). The later algorithm uses the earlier as a sub-method.

The algorithm *GlobalSteps* in presented in Figure 4 ignores the mobility ranges of the sensors and positions of the targets. It repetitively takes a large as possible set of the most important targets, and makes them appear to have reverse ordering of importance. The algorithm starts, in a very similar way to the algorithm *DeceiveMax*, by finding the least important target d that can be modified to appear the most important. However,

Input: T – set of targets; $ER : T \to \mathbb{R}^+$ – real importance; $\Delta^+, \Delta^- : T \to \mathbb{R}^+$ – the boundaries in which the adversary chooses the apparent importance
Output: $masked : T \to \mathbb{R}^+$ – apparent importance of the targets; $decoys \subset T$ – the targets turned to strongest decoys

while $T \neq \emptyset$ **do**
  $MAX := \arg\max(ER(T))$
  $S := \{t_1 \in T : \forall t_2 \in T \;\; ER(t_2) - \Delta^-(t_2) \leq ER(t_1) + \Delta^+(t_1)\}$
  $d := \arg\min_{t \in S} ER(t)$
  $Between := \{t \in T : [ER(t) - \Delta^-(t), ER(t) + \Delta^+(t)] \cap (ER(MAX) - \Delta^-(MAX), ER(d) + \Delta^+(d)) \neq \emptyset\}$
  $\epsilon := (masked(d) - masked(MAX))/(|Between| + 1)$
  $val := ER(d) + \Delta^+(d) - \epsilon$
  for $t \in Between$ in increasing order **do**
    if $ER(t) + \Delta^+(t) \geq val \geq ER(t) - \Delta^-(t)$ **then**
      masked(t) := val
    else
      masked(t) := $ER(t) - \Delta^-(t)$
    end if
    $val := val - \epsilon$
  end for
  $T := T \setminus Between$
  $Decoys := Decoys \cup \{d\}$
end while
return $masked, Decoys$

Fig. 4 *GlobalSteps* – a method for creating deception for multi-agent case that ignores the spacial relation of the targets.

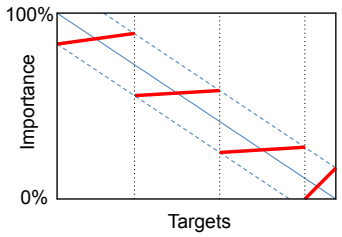

Fig. 5 Illustration of the deception created by the *GlobalSteps* algorithm.

instead of modifying all the more important targets down to the boundaries of their importance, it attempts to modify them to have apparent ordering opposite to their real ordering. It further ignores all the targets that have been already modified and performs the same change in ordering with the remaining targets. The algorithm ends when all targets are modified.

The result of the algorithm is illustrated in Figure 5. The x axes depicts all the targets in the problem sorted by their real importance and the y axes is a scale of importance. In this example, we assume a high number of targets; hence we do not depict each of them individually. For the case of importances drawn from a uniform distribution, the real importances of targets create the solid diagonal line (from top left to bottom right). The boundaries in which the adversary can select the apparent importance for individual targets are marked by the dashed diagonal lines. Algorithm *GlobalSteps* starts from the left-most targets and group them to several levels of importance. In each level, it orders their apparent importances opposite to the order of their

Input: $SensorPos$ – the positions of the sensors; T – set of targets; $ER : T \to \mathbb{R}^+$ – real importance; $\Delta^+, \Delta^- : T \to \mathbb{R}^+$ – the boundaries in which the adversary chooses the apparent importance
Output: $masked : T \to \mathbb{R}^+$ – apparent importance of the targets

queue $Positions := SensorPos$
$Processed := \emptyset$
while $Positions \neq \emptyset$ **do**
  $pos := pop(Positions)$
  $CloseT :=$ targets in T a sensor can cover from pos
  if $pos \in Processed \ \vee \ |CloseT| \leq 1$ **then**
    continue
  end if
  $T := T \setminus CloseT$
  $(locMax, Decoys) = GlobalSteps(CloseT, ER, \Delta^+, \Delta^-)$
  $masked(CloseT) := locMax$
  $Positions := Positions \cup Decoys$
  $Processed := Processed \cap \{pos\}$
end while
return $masked$

Fig. 6 *LocalSteps* – a method for creating deception in multi-agent case that considers the spacial relations of the targets.

real importances. The apparent importance of the targets is marked by the sequence of thick lines.

GlobalSteps does not use any information about the positions of the targets. Generally, it may lead to less efficient deception (see the following example).

Example 2 If we apply algorithm *GlobalSteps* to a setting which includes $(9, 6, 4)$ and $\Delta = 2$ it creates deception $(7, 8, 6)$. A sensor choosing among all the targets or among the first two targets is deceived, but a sensor choosing among the second and third target is not. However, if the first target is too far from the other two targets and all of them cannot be in mobility range of any sensor at once, we can use this information to create a more efficient deception $(7, 4, 6)$. The sensor with the first target in its mobility range will pick it anyway (there is no other alternative), but if the sensor has the other two targets in its mobility range, it will pick the less important (third) target.

This observation inspires the algorithm *LocalSteps* in Figure 6. This algorithm assumes it knows the initial positions of the sensors. It gradually uses algorithm *GlobalSteps* to decide about the apparent values of subsets of targets that are in the mobility range from these positions. After each call of *GlobalSteps*, it removes the targets it considered and adds the positions that were used as decoys (are likely to be covered by the naïve algorithm) to the queue of positions to process.

We expected this algorithm to be more efficient in the case of sparse scenarios where the local deceptions prepared for individual agents do not interact. In the setting of Example 2 and two agents at the mentioned positions, the algorithm *LocalSteps* indeed creates the more eficient deception. However, the effect of the additional information is questionable in the scenarios with high overlap of the mobility ranges of the sensors. Questionable is also the effect of the deception algorithms after multiple iterations of the sensor placement algorithms. That is why we evaluate the quality of the deception created by the proposed algorithms experimentally in Section 7.

7 Experimental evaluation

In this section, we experimentally evaluate the derived solutions of the formal single sensor deception game, the quality of the deception heuristics, as well as the performance of the proposed local search methods for solving the multi-agent deception aware problem defined in Section 3.

In most of the experiments, we followed the following scheme:

1. A random scenario is generated.
2. A method for creating deception modifies the scenario within the bounds.
3. The sensors placement algorithm observes only the apparent importances.
4. The placement is evaluated based on the original real importance values.

7.1 Formal Deception Game

In our first set of experiments we evaluate the strategies proposed in Section 4 for a single sensor covering one of several deceptive targets. The proposed methods are compared to three baseline algorithms.

- The naïve *Max* algorithm selects the target that seems to be the most important and it does not take into account any possibility of deception.
- The *Random* algorithm ignores the available information about targets' importance completely and selects one of the targets uniformly (randomly).
- The *Dom* algorithm is a simple heuristic that ignores all the targets that are certainly not the most important (i.e., dominated) and chooses randomly one of the remaining targets (i.e., choosing uniformly a target from $\{t \in T : \nexists t_2 \in T \quad bottom(t_2) > top(t)\}$).

For simplicity of presentation, we determine the capability of the adversary by the constant Δ, which is the same for all targets (see Section 3.3).

In this experiment we evaluated three methods proposed in Section 4. The simplified version of the deception aware method relative to the random strategy from Corollary 3 is denoted *LPrand*. We refer to the methods relative to the optimal selection from Theorems 2 and 3 as *LPratioOpt* and *LPdiffOpt* respectively.

In the first experiment, a single sensor had to choose to cover one of five targets. The real importance of each target was selected uniformly from the interval [0,29]. The graphs in Figure 7 show the mean (real) importance of the covered target from 10000 runs of the experiment. The horizontal axes of the graphs represents the capability of the adversary (Δ).

In Figure 7(a), the deception is created by the algorithm $DecieveMax$ described in Section 6. This method optimally deceives the *Max* algorithm and uses the remaining freedom in modifying the low importance targets to make the targets appear more similar to each other. The results show that all the proposed methods are better than *Max* for any selection of Δ. Moreover, thanks to the design of the *LPrand* algorithm, it is never worse than the random target selection. The algorithms relative to the optimal selection (*LPratioOpt* and *LPdiffOpt*) seem to be less careful. They perform better than *LPrand* for smaller capabilities of the adversary. Nevertheless, they perform worse than the random target selection for higher adversary capabilities.

In Figure 7(b), the setting was the same as in Figure 7(a), but the algorithm used for deception was $DecieveMax^{+}$ described in Section 6.1 that changes the importance

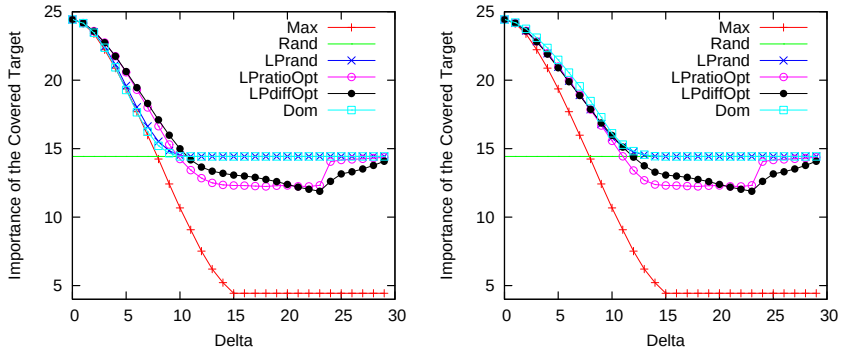

Fig. 7 The importance of the target covered by the proposed and two baseline algorithms in formal deception game scenario. The adversary uses its capabilities optimally against the naïve approach based on algorithm *DeceiveMax* (a) and its modification (b) from Section 6.1.

of all but one target to the lowest possible option. The algorithm still deceives *Max* optimally, but allows all the other algorithms to perform better. In this case, all the proposed algorithms perform the same for lower Δ and *LPrand* keeps its performance best even for higher Δ.

7.1.1 Price of Paranoia

The results presented above show, that the proposed deception aware target selection methods are beneficial, if the adversary optimally uses its capabilities in order to deceive a sensor that uses the naïve *Max* algorithm to cover the most important target. In this situation, the best algorithm would be a combination of *LPdiffOpt* for smaller values of Δ and *Rand* for higher values. However, we intend to use this algorithm in a multi-agent setting, where agents change their positions and have limited mobility ranges. In this setting, it is impossible to create a fixed deception that would be the worst case in all the local deception games for the reasons discussed at the beginning of Section 6.2.

That is why we further evaluate the robustness of the proposed methods towards overestimation of the quality of the deception. We examine the decrease in the quality of the solution in case that no adversary is trying to deceive the sensor, but the sensor expects otherwise. We call the function, mapping the expected adversary capability to the decrease of the quality of produced solution (compared to the optimal solution without deception), *the price of paranoia*. For our scenario, the price of paranoia can be seen on Figure 8.

The optimal solution in case of no deception is the *Max* algorithm producing the upper straight line in the figure. With increasing paranoia (i.e. the false belief about the adversary activity), the performance of all the proposed algorithms gradually decreases to the mean real importance. This decrease is higher in case of the more careful *LPrand*, which performs as bad as a random selection when the capability of the adversary is equal or larger to half of the maximal importance of a target. However, the price of paranoya is much lower for the methods derived relative to the optimal selection. The

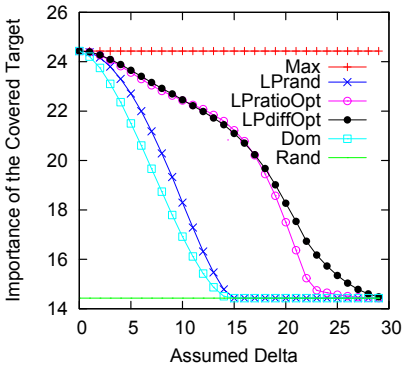

Fig. 8 The price of paranoia for the proposed algorithm on formal deception game scenario.

LPdiffOpt algorithm performs better than the random selection in all cases where the adversary is not assumed to present completly arbitrary numbers.

Because of the comparable results of all the deception aware methods in the case of exact knowledge about the used deception capabilities and clearly superior performance of *LPdiffOpt* in the case of overestimation of the adversary capabilities, we use *LPdiffOpt* as the local deception aware method in our multi-agent experiments.

7.2 Multi-agent Experiments Settings

This section describes the main settings we use in the multi-agent experiments and explains the rationale behind the parameter selection.

The main challenge when solving a problem with multiple sensors is the coordination among the agents. Since agents make local decisions on their movements lack of coordination can cause some targets to remain uncovered while other are covered by more sensors than required. An important extreme case are scenarios with relatively small number of agents in a large space. If the agents do not have any neighbors most of the time, the game reduces to a high number of simultaneously played single sensor games and there is no need for coordination. Thus, the benefit of using the deception aware method results directly from the higher expected coverage in each step of the single sensor games. Without the need for coordination, the overall quality of all the proposed deception aware local-search methods is the same. The main difference between the methods in such a sparse scenario is the speed of convergence. In the MGM_y and *maxStays* algorithms all (or almost all) agents are able to change their position in each step. In DSA_y only a portion of the sensors (proportionally to the parameter p) move, hence, the overall convergence is slower.

More interesting results can be expected for denser sensor networks with high overlap of the targets in individual the sensors' mobility ranges. We used two main experiment settings which both included 128 targets of random importance from (0,99] placed randomly on a 100x100 positions large grid.

Setting 1: In our first experimental scenario, each agent was able to fully cover the requirement of any target (i.e., $\forall a \in A \quad Cred_i = 100$). There were 64 agents with mobility range of 20 positions. This means that each sensor had approximately

10% of the area in its mobility range; hence it was able to cover on average 10% of the targets. With 64 uniformly randomly placed sensors, we can estimate from the binomial distribution that any position is covered by at least 2 agents in over 99% cases. Covering a target with more than one agent is useless in this setting so coordination among the agents is clearly needed.

Setting 2: The second experimental setting required more sensors to cover a single target. In this case if the number of sensors is small, the situation does not change dramatically from the first setting. With higher amount of uncertainty about the target importance, it is mostly beneficial to cover each target by only a single sensor. More interesting dynamics appear if the number of sensors is higher than the number of targets; hence they are forced to cover the targets by more than one sensor and the question is which targets to cover with more sensors than others and which of the targets should not be covered at all. Thus, the parameters of the second setting were the same as in the first one with the exception of the number of agents, which was 150 and the credibility of the agents that was set to 50.

7.3 Deception Algorithms Evaluation

In order to evaluate the benefit of using the deception robust algorithms in the multi-agent setting, we first need to evaluate the success of the proposed deception creating methods in deceiving naïve algorithms. In the following experiments, we randomly generate scenarios, apply an algorithm to create deception and then run the naïve MGM algorithm (See Section 5.1) for 100 iterations.

We compare two algorithms for creating deception, which were described in Section 6.2, with two baseline algorithms. The evaluated methods are:

- The algorithm *GlobalSteps* depicted in Figure 4.
- The algorithm *LocalSteps* depicted in Figure 6.
- *Random*, which selects the apparent importance uniformly in the interval that is available for the adversary.
- *Mean*,which computes the mean importance of all targets in the scenario and sets the apparent importance of the targets with real importance above the mean as low as possible and the remaining targets as high as possible.

The quality of the deception generated by the algorithms in *Setting 1* can be seen in Figure 9(a-b). We plot the quality of the coverage achieved after 10 iterations (Figure 9(a)) and 100 iterations (Figure 9(b)) of the algorithm. The results after 10 iteration give us an idea about the quality of the achieved coverage in time critical situations and the results after 100 iteration generally represent a stabilized solution of maximal quality achievable by the investigated methods.

The weakest deception in this setting (for any Δ) is produced by the *Random* algorithm. With increasing delta, the coverage quality gradually decreases to the half of the value achieved without any deception consistently after 10 as well as after 100 steps. For $\Delta = 90$, there is practically no correlation between the observed value and the real importance of the target. Thus, the targets to be covered by the MGM algorithm are picked randomly. As a result, the achieved coverage values constitute the lower bound on any reasonable deception aware algorithm in the same way as the random algorithm does for the single-sensor case. This result is always achievable for

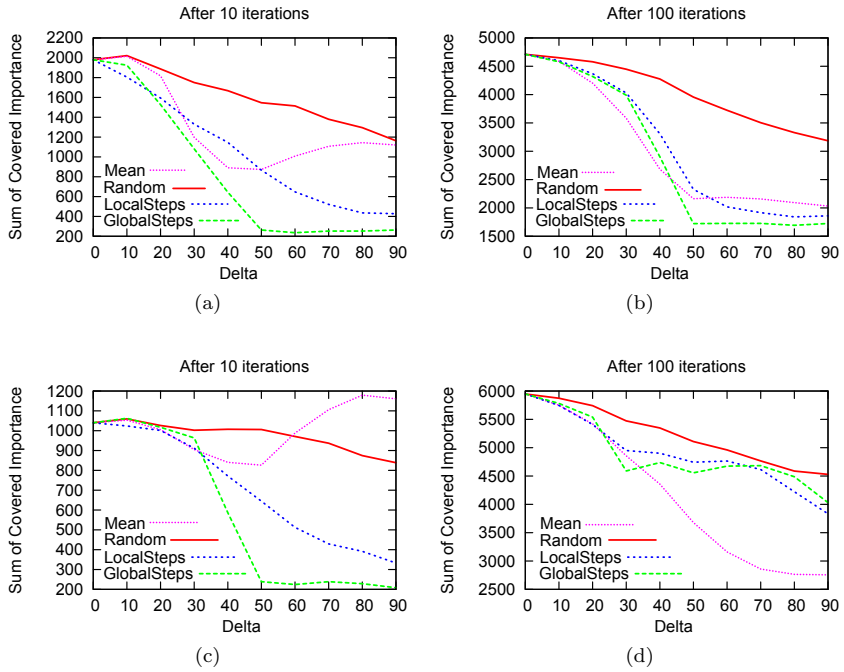

Fig. 9 Evaluation of the proposed methods for creating deception in multi-agent setting after 10 and 100 iterations of the naïve MGM algorithm. Figures (a-b) represent the reached coverage for the case with 64 agents with credibility 100% covering 128 targets (i.e, *Setting 1*). (c-d) are for the case of 150 agents with credibility 50% (i.e, *Setting 2*). The x axis is the adversary capability Δ in all figures.

the sensors team by assigning random importance values to the targets and covering them using the naïve algorithm.

All the other deception algorithms are clearly superior to the *Random* deception in this setting. The coverage achieved by the naïve algorithm facing any of them was lower for all values of Δ. Surprisingly, the results show that using the additional information about positions of the targets and the sensors in the simple way as it is done by the *LocalSteps* algorithm is counterproductive. The *GlobalSteps* method creates strongest deception in the short run (10 iterations) and a more similar but still stronger deception in the long run (100 iterations). The reason is that fixing the deception to be efficient from one position restricts the options of creating deception from another. This is true not only for two neighboring sensors in one iteration, but also for two subsequent positions of the same sensor. Even if we run the experiment on the same setting with just a single sensor, the *GlobalSteps* methods causes the naïve algorithm to cover a less important target than *LocalStep* after second iteration. The deception caused by *LocalSteps* is stronger only after the first iteration of the naïve algorithm.

The results for measuring the quality of the naïve MGM algorithm facing different forms of deception in *Setting 2* are shown in Figures 9(c) and 9(d). The situation after the first 10 iterations is similar to the case of *Setting 1* with approximately half

the coverage. The reason is that even if the agents have $Cred_i = 50$, they generally cover exactly the same targets in the first iterations of the algorithm. Once a target is covered by one sensor, its importance drops by 50, which makes it unlikely to be the apparently most important target in the mobility range from the sorounding positions. As a result, the first 20 sensors are placed in a very similar fashion in both settings.

The number of concurrently moving agents for different settings of Δ is roughly the same in *Setting 2* and *Setting 1*. However, the improvement in the performance of the naïve algorithm for *Mean* deception with increasing values of Δ, seems to be stronger in Figure 9(c). The explanation is that for lower values of Δ, agents tend to cover first the targets of high real importance. Sensors with credibility 50 cannot fully satisfy their requirement and hence the overal coverage is only slightly higher than half of the coverage achieved by sensors with credibility 100. On the other hand, in case of strong deception, the sensors first cover the less important targets. Their requirement can be fully covered even by the sensor with credibility 50; hence the achived coverage is almost the same as in the case of sensors with credibility 100.

The results after 100 iterations reveal that the the proposed deception generating algorithms are less effective when sensors do not have enough credibility to fully cover targets. The more sophisticated deception algorithms perform well only for smaller setting of Δ, but for higher values, they perform only slightly better than the random deception.

The simplest idea of taking the importance of the important targets down and the remaining targets up works best for larger deception capabilities from all the proposed deception algorithms. For high values of Δ, this deception causes the targets of low importance to be covered by two sensors before the targets of high importance are covered at all.

Based on the results in this section, we further use the strongest deception in our experiments. Namely we use algorithm *GlobalSteps* for all sizes of Δ in *Setting 1* and $\Delta \leq 30$ in *Setting 2*. We use the algorithm *Mean* for creating deception for remaining values of Δ in *Setting 2*.

7.4 Multi-agent Local Search Algorithms

Next, we present results that indicate that the proposed deception aware local methods (presented in Section 4) can be successfully used as the method executed by each agent in the decentralized local search algorithm for solving the sensor placement problem defined in Section 3.

The performance of all the local search algorithms (suggested in Section 5) in *Setting 1* with deception created by algorithm *GlobalSteps* is presented in Figure 10. As before, we assume that the adversary capabilities are defined by a fixed constant Δ for all the targets. The presented results are the mean values form 50 random instances of the problem, which were the same for all algorithms.

The graphs show the quality of target coverage as the sum of real importances of the covered targets. The first three graphs depict the development of the coverage quality with increasing number of iterations for three values of adversary capability $\Delta = \{30, 50, 80\}$. The last graph shows the quality of the coverage after 10 iterations of running the algorithm for the whole range of Δ values.

The novel local search method introduced in Section 5.3 (*maxStays*) consistently performs best in the presented experiments. It reaches its maximal value after less

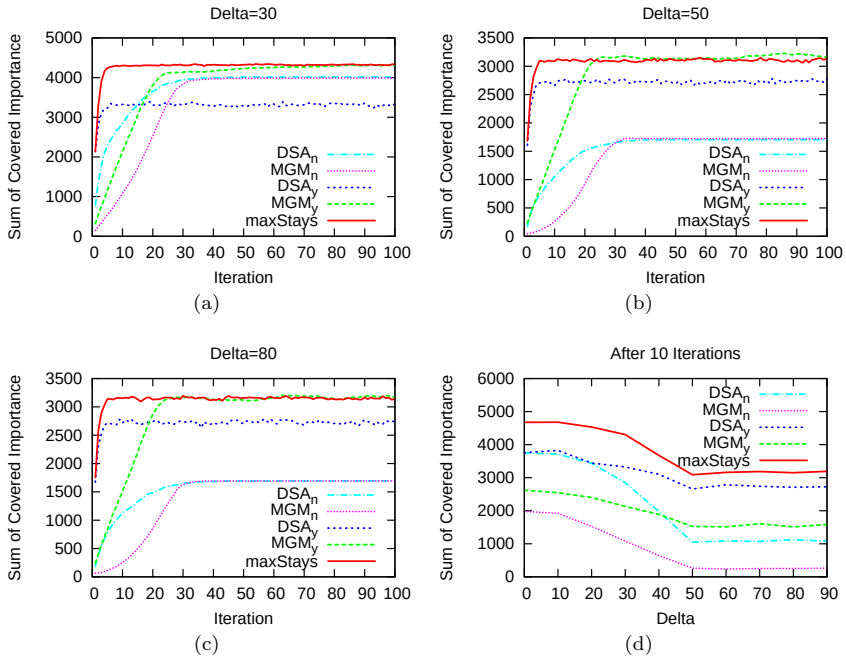

Fig. 10 Comparision of the sum of real importances of the covered targets for various methods on a scenario with general MR and minimal SR. The deception aware methods were based on the *LPdiffOpt* algorithm and they are indicated by "y" in the legend. (a-c) Show the convergence of the methods on a scenario with adversary capabilities $\Delta = \{30, 50, 80\}$ respectively. (d) Shows the coverage reached in the 10th iteration of the algorithms.

than 10 iterations and keeps the same quality of coverage with minimal oscillations. The fast convergence is caused by allowing all agents to change their assignment in each iteration. This is also the main reason why the *maxStays* method outperforms all the other methods for all settings of Δ after 10 iterations (Figure 10(d)).

Besides it's fast convergence, *maxStays* is also robust against deception. Even if we run the experiment long enough to allow all the other methods to converge to a stable quality, *maxStays* still clearly outperforms all the naïve methods. Moreover, the quality of the coverage resulting from the *maxStays* method is the same as the quality reached by the deception aware MGM_y after a higher number of itteration (see Figures 10(a-c)). MGM_y avoids any coordination problems for the cost of slow convergence. This indicates that coordination provided by *maxStays* is really efficient and does not decrease the quality of the solution.

Both the naïve local search methods converge to a solution of the same quality in this setting. They reach the maximal performance after almost the same number of iterations, but an interesting phenomenon is that the graphs for DSA_n are concave and the grahs for MGM_n are convex in Figures 10(a-c). The behavior of DSA_n is more intuitive, first all the sensors cover some targets, moving from their initial random positions where they most likely do not cover anything. However, due to the random

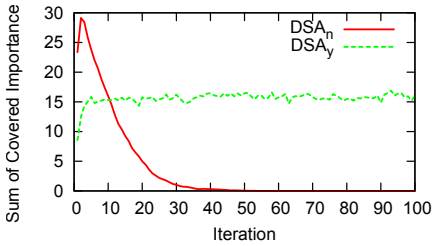

Fig. 11 The number of redundant agents in the deception aware (DSA_y) and the naïve (DSA_n) version of the DSA algorithm corresponding to Figure 10(a).

coordination, it is possible that multiple sensors will cover the same target, which is inefficient in this setting. With increasing number of iterations, sensors would switch to another target if their current target is covered sufficiently and in more cases do not have such high local reductions. On the other hand, with MGM_n almost the same (lower) number of agents move in each of the first 30 iterations. When using the *GlobalSteps* algorithm for creating deception (see Figure 5), the targets that seem to be most important (cause maximum local reduction message) are actually the strongest decoys. That is why the improvement of coverage quality caused by covering these targets is quite low. However, after these decoys are covered, more and more important targets are selected and the quality of the coverage rise faster.

The efficiency of the deception robust local method in this setting can be seen also in Figure 10(d) showing the situation after 10 iterations. The quality of coverage for $\Delta = 0$ is generally caused mostly by the amount of concurrent movements allowed by individual methods. However, even after this little number of iterations, we can clearly see that the quality of coverage by the same base method (DSA/MGM) always decrease much faster in the naïve version compared to the versions with deception robust local method.

7.4.1 Redundant Agents

One interesting phenomenon in the results presented in Figure 10(a), is that for a small capability of the adversary, even the naïve version of *DSA* outperforms the deception aware DSA algorithm after running for enough (20 or more) iterations. The *MGM* algorithm does not exhibit this pathology. Since MGM does not allow neighboring agents to move in the same iteration, it makes sense to assume that the reasons are the collisions of the assignments of neighboring agents that move in the same iteration. In order to validate this claim, we experimentally evaluated the number of collisions in this scenario. In *Setting 1*, the credibility of each agent is equal to the maximum possible importance of a target; hence if a target is covered by more than a single agent, the remaining agents covering the target are redundant. The mean number of redundant sensors in a setting with $\Delta = 30$ are plotted in Figure 11.

The number of redundant agents with the deception aware DSA stays relatively high with increasing number of iterations. Fifteen targets are redundant on average. This is not true for the case of naïve agents. At first, the naïve agents are not covering any targets and only few targets in each neighborhood are considered most important, hence all the agents head to cover these targets and high redundancy is generated.

However, after 40 iterations of the algorithm, the solution is stabilized and none of the agents are redundant. This stabilization corresponds also to reaching the maximal coverage in Figure 10.

Redundant agents appear when two agents consider the same uncovered target, among the targets within their mobility ranges, to be the most important, and decide to cover it (i.e., change their position). In the case of naïve agents, this happens less with an increasing number of iterations. For example, assume two sensors covering three targets with apparent importances (8,7,6). After a couple of iterations, the naive DSA_n algorithm covers the targets with importance 7 and 8. This situation is stable. None of the agents has an incentive to change its assignment and no redundant sensors can appear in future iterations. On the other hand, in the same situation, the deception aware agent knows it could have been deceived and randomizes over all the targets. That is why, even in the situation of covering the targets which appear to be most important (7 and 8), the agent assumes that there is a chance that the target with apparent importance 6 is actually the most important one; hence it covers the target with a nonzero probability. If both the agents decide to cover it, a redundant assignment can occure. In the complete experimental setting, these effects cause that that redundant assignments are always present for the DSA_y algorithm. This effect is reduced for smaller values of the probability parameter for changing position p, but even for very low values of p it cannot be completely avoided. Moreover, with sufficiently small p that reduces this effect, the speed of convergence is no longer faster than the speed of convergence of MGM based algorithms that completly avoid redundant assignments. As a result, we conclude that DSA_y is not a suitable algorithm for this setting.

The *maxStays* algorithm explicitly avoids redundant agents assignments and assures maximal concurrency, which are the main reasons of its clearly superior performance in this scenario.

7.5 Sensors with Limited Credibility

This section reports the results of the local search algorithms in *Setting 2* (i.e, 150 sensors with credibility 50). Some targets should be covered by two sensors.

The results are presented in Figure 12. We do not include the results for the DSA algorithm in this experiment. The previous experiments show that the naïve DSA_n performs comparable to MGM_n and the deception aware DSA_y exhibits problems with redundant assignments suitable only with low values of the parameter p, which makes it perform similar to MGM_y.

The dominance of MGM_y over MGM_n shows that it is beneficial to use the deception aware local method even if the targets cannot be fully covered by single sensor. The deception aware algorithm both converges faster and reaches higher performance after high number of iterations. The convergence of the MGM_n slows down for a while after certain number of iterations and then speed up again. This is because the algorithm first covers the apparently most important targets by one sensor. This causes the (remaining) apparent importance of the targets to drop and the other agents do not pick these targets for covering. However, after a large portion of the apparently important targets are covered, the sensors start covering the targets that are already covered by one agent (because of limited credibility). This is where the slow down starts. If the apparently important targets are decoys created by the adversary, the first sensor

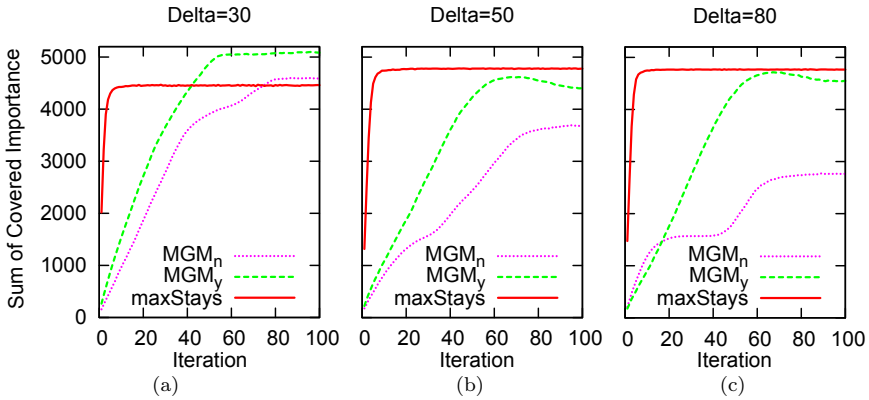

Fig. 12 Comparision of the sum of real importances of the covered targets for various methods on a scenario where more sensors are required to fully cover the targets. The deception aware methods were based on the *LPdiffOpt* algorithm and they are indicated by "y" in the legend. The graphs show the convergence of the methods on a random scenario for adversary capabilities $\Delta = \{30, 50, 80\}$ respectively.

already fully covers the low real importance of the target. Adding the second sensor to this target does not have any effect on the optimized value.

The *maxStays* algorithm that dominated over all the other algorithms in the first setting, where a single sensor was enough to cover a target completely, did not always perform best in this setting. For small values of Δ, it was outperformed even by the naïve MGM. The reason is the relatively high number of agents that give up on covering any of the targets due to coordination. Agents in *maxStays* gives up on covering a target if the sum of credibilities of other agents that intend to cover this target with higher probability than their own, is higher than the maximal possible importance of the target. If an agent gives up on covering all the targets in its mobility range, it performs a random move. This random move might result in a location from which targets which were previously in range are not considered any more. Thus if other agents did not move to cover such a target, it will remain uncovered. This also happened in *Setting 1*, but the effect was not as strong because of the smaller number of competing agents.

8 Conclusion

Realistic military applications of mobile sensing agents networks, are expected to include adversaries which try to reduce the network efficiency. Previous attempts to address the problem of effectively monitoring (covering) an area using a network of mobile sensing agents did not consider such an adversary in their model.

In this paper we focused on the ability of an adversary to use means of deception (i.e. decoys, camouflage) to draw attention to less important targets and keep the more important targets uncovered. The adversary is assumed to be able to either increment or decrease the perceived importance of targets by a bounded amount.

Our paper achieved three main contributions. The first, was a formalization of the problem of deception in mobile sensor networks as an extension of the existing DCOP_MST model.

The second contribution was a method for selecting one of several targets in a way that optimizes the expected real importance of the selected target. To this end, we defined a formal deception game between a sensor choosing to cover one of several targets and an adversary modifying their apparent importance in order to deceive it. Using game theoretic techniques, we derived formally sound strategies that optimize the selection of the target in case of the worst possible real importance of the targets, consistent with the available observation. This approach results in several methods, which are robust against deception. One of the strategies never performs worse than a complete random selection which ignores all the (possibly misleading) information observed. Another exhibits a very good performance in case of overestimation of the adversary capabilities while still being robust against the well assessed deception. All of the derived strategies are stochastic (mixed) and ensure the good performance over a large number of game instances.

The third contribution was modifying the known local search algorithms for DCOP_MST to accommodate stochastic local strategies and proposing a novel local search algorithm which was designed directly for problems with stochastic local strategies. As far as we know, stochastic local strategies have never been used in local search for DCOP before. We show that DSA-C is not suitable in this domain because of high number of conflicting variable assignments. A modified version of the MGM algorithm performs well once it converges, but the convergence is rather slow. The novel *maxStays* algorithm always converges very quickly. It reaches the same performance as MGM in only a fraction of steps if a single sensor is sufficient to fully cover a target, but its performance is weaker if the credibility of the sensors is limited.

In future work we intend to investigate the use of distributed local search algorithms with stochastic local strategies in the context of general DCOP.

Acknowledgements This research was partially funded by AFOSR MURI award FA9550-08-1-0356, AFOSR USAF grant number FA8655-09-1-3060 and by the Ministry of Education of the Czech Republic grants ME09053 and MSM6840770038.

References

Fristedt B (1997) The deceptive number changing game, in the absence of symmetry. International Journal of Game Theory 26(2):183–191

Hespanha J, Ateskan Y, Kizilocak H (2000) Deception in non-cooperative games with partial information. In: Proceedings of the 2nd DARPA-JFACC Symposium on Advances in Enterprise Control

Jain M, Taylor ME, Yokoo M, Tambe M (2009) Dcops meet the real world: Exploring unknown reward matrices with applications to mobile sensor networks. In: Proc. Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09), Pasadena, CA, USA

Lee K (1993) On a deception game with three boxes. International Journal of Game Theory 22(2):89–95

Maheswaran RT, Pearce JP, Tambe M (2004) Distributed algorithms for dcop: A graphical-game-based approach. In: Proc. Parallel and Distributed Computing Systems PDCS), pp 432–439

Marecki J, Schurr N, Tambe M, Scerri P (2005) Dangers in multiagent rescue using defacto. In: Workshop on Safety and Security in Multiagent Systems, AAMAS '05

Matsui T, Matsuo H, Silaghi M, Hirayama K, Yokoo M, Baba S (2010) A Quantified Distributed Constraint Optimization Problem. In: Proceedings of the Ninth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)

Michael J, Riehle R (2001) Intelligent software decoys. In: Proc. Monterey Workshop: Eng. Automation for Software Intensive Syst. Integration, Citeseer, pp 178–187

Modi PJ, Shen W, Tambe M, Yokoo M (2005) Adopt: asynchronous distributed constraints optimization with quality guarantees. Artificial Intelligence 161:1-2:149–180

Petcu A, Faltings B (2005) A scalable method for multiagent constraint optimization. In: IJCAI, pp 266–271

Root P, De Mot J, Feron E (2005) Randomized path planning with deceptive strategies. In: American Control Conference, 2005. Proceedings of the 2005, pp 1551–1556

Shoham Y, Leyton-Brown K (2008) Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. Cambridge University Press New York, NY, USA

Spencer J (1973) A deception game. American Mathematical Monthly pp 416–417

Waun S, Ozguner U (2004) A coordination strategy for cooperative sensor network deception by autonomous vehicle teams. In: 43rd IEEE Conference on Decision and Control, 2004. CDC, vol 4

Zhang W, Xing Z, Wang G, Wittenburg L (2005) Distributed stochastic search and distributed breakout: properties, comparishon and applications to constraints optimization problems in sensor networks. Artificial Intelligence 161:1-2:55–88

Zivan R, Glinton R, Sycara K (2009) Distributed constraint optimization for large teams of mobile sensing agents. In: International Joint Conference on Web Intelligence and Intelligent Agent Technology, pp 347–354