W911NF-10-1-0112

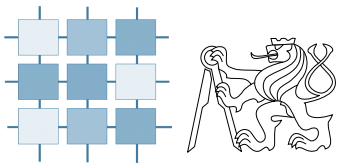
Tactical AgentScout 2

Deliberative and reactive planning in adversarial environments

Final report

April 2011

Agent Technology Center, Gerstner Laboratory
Department of Cybernetics
Czech Technical University in Prague



Branislav Bošanský, Michal Čáp, Antonín Komenda, Viliam Lisý

Project coordinator: Peter Novák
Principal Investigator: Michal Pěchouček

Contents

Executive summary

This document provides a final technical report on the work perfo0rmed in the context of the research project W911NF-10-1-0112, Deliberative and reactive planning in adversarial environments (Tactical AgentScout 2). The project aims at investigation of problems and challenges in implementation of *adversarial behavior in dynamic environments from game theoretic perspective* and investigation of issues in *long-term (deliberative) vs. short-term (reactive) multi-agent planning* in adversarial scenarios. The main research foci of the project include i) adversarial planning for patrolling of mobile targets (convoys), ii) adversarial planning for modeling smart targets, iii) multi-agent re-planning and plan repair as a vehicle for integration of reactive and deliberative action selection, and finally iv) modeling of coordination and teamwork in teams of heterogeneous cooperative agents. The empirical evaluation of the project aimed at validation in multi-agent simulations of urban warfare scenarios including heterogeneous teams composed of UAVs, VTOLs and UGVs.

The main achievements and innovations of the project include development and evaluation of:

1. extensions to state-of-the-art game-theoretic algorithms for controlling a mixed team of UAVs and UGVs attempting to detect, track and capture smart targets, i.e., targets which actively and purposefully avoid being detected, tracked, resp. captured;

2. novel game-theoretic algorithms for controlling a team of UAVs/VTOLs patrolling a number of mobile targets (e.g., convoys);

3. novel multi-agent plan repair algorithms based on distributed constraint optimization allowing modeling effective reaction and adaptation of a team of agents to unexpected events and changes in dynamic environment; and finally

4. extensions of state-of-the-art multi-agent coordination techniques for modeling of agent teams acting cooperatively in order to reach joint goals, while still maintaining team-, as well as individual-level reactivity to changes in dynamic environments.

In this report we provide a detailed account of the work performed in the context of the project, provide an extensive discussion of the technologies we proposed, designed and developed, or adapted for the purposes of the project. Where appropriate, we also provide a thorough evaluation of the proposed techniques in an example of a simulated ISTAR mission.

How to read this document

For a quick orientation in this report, additionally to the complete information extracted by reading through the whole of it, the document structure supports the following three modes of information extraction, corresponding to different levels of required information depth.

skim (~5min) the *Executive Summary* provides a basic overview of the project and its main objectives.

opinion (~15min) the *Overview* chapter provides sufficient information about the project context, its particular objectives, the work accomplished and finally main achievements and highlights. It provides enough information to support an interested reader on a pursuit for an informed overview about the project.

insights (~1hr) after reading the *Overview*, the subsequent four chapters provide a deep account of the individual workpackages of the project. Each of these chapters starts with a section *Summary of the workpackage* providing an overview of the particular techniques and approaches discussed in the chapter. Reading over these summaries is designed to provide a relatively informed account of the technologies, theoretical techniques, approaches, as well as related work built upon in the project.

1. Overview

This chapter provides an overview of the the final report for the work performed in the context of the project W911NF-10-1-0112, Tactical AgentScout 2. After an introduction into the broader context of the project and related research activities of ATG, we summarize its main objectives and subsequently introduce and discuss the main achievements of the project team together with the main results of the project. We conclude this overview with a brief recapitulation of the project execution from a project-managerial view and finally outline the structure of the remainder of this report.

1.1. Motivation and context

One of the major research tracks of the *Agent Technology Center* (ATG) in the last years is development of novel techniques and approaches for distributed multi-agent control of Unmanned Aerial Vehicles of various types. In the past, the achievements of the center in this context included development of the AgentFly technology suite, a large-scale simulation framework for development and evaluation of automatic mechanisms for next-generation air traffic control. The experiences from the past AgentFly projects provide a solid basis for a number of other research projects within ATG, including the here reported Tactical AgentScout 2 technology.

On the substrate of the AgentFly suite of technologies, our group was in the past involved in several projects aiming at investigation of information collection techniques for team of coordinated UAVs. Besides here reported W911NF-10-1-0112 project, these include:

- Tactical AgentFly phases I, II and III (W911NF-08-1-0521_1312AM0, W911NF-08-1-0521 and R&D 1408-CE-01), aiming at investigation of techniques for coordinated area information collection from basic ones implementing surveillance and tracking of a single target by a single UAV, to advanced ones enabling tracking of multiple targets with relatively small number UAVs. The third phase of the project series, an on-going project at the time of writing this report, aims at porting of the previously developed techniques to real hardware UAVs and VTOLs and is geared towards a real-world demonstrator in a mixed simulation.

- Tactical AgentScout phase I (BAA 8020902.A/W15P7T-05-R-P209), the direct precursor of the Tactical AgentScout 2, project reported here upon. It aimed at integration of aerial information collection in ISTAR-type missions with different

types of ground assets. The main focus of the project was on multi-agent planning and task-allocation problems.

We proposed the here reported project as a continuation of our long-term line of research towards deep understanding of information-collection tasks performed by a team of coordinated unmanned vehicles with a special focus on adversarial planning and investigation of issues involved in coordination of teams of heterogeneous assets.

ATG currently performs several on-going research activities aiming at transfer of the research results resulting from the above mentioned projects, including the one reported upon here, and their related technologies from mid- and large-scale simulations in the AgentFly framework to real hardware. In particular, there is an on-going project in parts supported by the Czech Ministry of Defence aiming at integration real UAV (*Unicorn Procerus*) into the AgentFly-based simulation and thus produce a mixed-simulation involving both simulated as well as real aircrafts. The objective is to develop a test-bed for multi-UAV air trajectory deconfliction by negotiation. Another, planned project will aim at transfer of the spatio-temporal planner for VTOLs to real quad-rotor UAV provided by *UAS Technologies*, Sweden, supported in part by the *SAAB* funded *LinkLab* - Center for Future Aviation Systems.

Figure 1.1.: The depiction of the *Procerus* UAV kit and the *LinkQuad* quad-rotor VTOL employed in the on-going research projects of the *Agent Technology Center*.

From the start on, the project was planned to loosely extend and build upon the work and results accomplished in the context of the two related project *W911NF-08-1-0521 Intelligent Software Agent Control of Combined UAV Operations for Tactical Missions* (Tactical-AgentFly 2) and *BAA 8020902.A/W15P7T-05-R-P209 AgentScout: Army Technology Objective Unmanned Systems C2 for Operations in Urban Terrain Planning Support* (Tactical-AgentScout 1).

1.2. Project description

The Tactical-AgentScount 2 project aims at studying various aspects of adversarial and cooperative behaviors in dynamic environments. In particular, the research objectives of the project can be grouped in two main research areas:

1. planning of agents' activities assuming that the team is facing a smart adversary, and

2. action selection for teams of cooperative agents embodied in dynamic environments.

The project itself was structured in four separate workpackages. The first two (WP1 and WP2) tackled issues from the first research area by employing game-theoretic techniques for implementation of strategies for convoy protection and strategies for detection, tracking and capture of smart targets in urban environments. The latter research area was approached in the other two workpackages (WP3 and WP4) which aimed at multi-agent plan repair techniques as an approach to deal with plan failures and complementary to that, at manual prototyping techniques for implementation of robust agent behaviors based on reactive-planning approach. In the following, we provide a brief summary of the particular project objectives of the individual workpackages.

WP1: Adversarial planning: patrolling of mobile targets

A complementary task to tracking a number of mobile adversary targets by a relatively small team of autonomous aircrafts (as investigated in the Tactical-AgentFly 2 project) is protection of mobile ground units against attacks from enemy units. The motivating scenario is an urban environment with a number of convoys passing through the area which should be protected by a small team of aircrafts, e.g., helicopters. In such a scenario, it is vital for the patrol not to execute a predictable patrolling strategy. The opposite would allow the opponent to optimize w.r.t. the patrol's strategy and attack the convoy in the worst timepoint, e.g., when the patrol just left the convoy it protects. The solution is use of randomized strategies which, however, still maintain certain average frequency of visits of each protected convoy.

The main objective of the workpackage was to develop algorithms for computation of *optimal strategies for protecting of mobile targets in adversarial environment*s. The basic underlying assumption driving the research was that the opponent is able to observe the patrol, as well as the convoys and is capable to launch an attack in any moment when the target convoy is left unprotected. Given a map of an urban environment, positions and plans of the convoys and a mobility model of opponent units, the particular goal was then to find an optimal randomized strategy for the patrol which minimizes the probability of attacks on the protected convoys. I.e., we are seeking patrolling trajectories which optimally cover the convoys and minimize the time when they are left unprotected, while making the movements of the patrol unpredictable.

WP2: Adversarial planning: smart targets

One of the objectives of the Tactical-AgentFly 2 project was to develop techniques for tracking a number of mobile targets by a team of UAVs (including both CTOLs, as well as VTOLs). However, no attention was paid to the capabilities, or, so to say, intelligence of the tracked targets. As an extension, in this project we consider *smart targets*, i.e., such which actively monitor their environment and act accordingly. Smart targets, when being tracked, are aware of that fact and actively try to avoid the tracking unit. Complementary, we consider trackers (be it UAVs, VTOLs, UGVs, or personnel) to be aware of the fact that the tracked targets are aware of their activities and try to act in best response to the whole setup.

The objective of the workpackage was to propose a formal game-theoretical model of pursuit-evasion scenario with heterogeneous teams of agents. The concrete goal was to develop algorithms for controlling a mixed team of UAVs and UGVs (pursuers) attempting to detect, track and finally capture a number of smart targets (evaders) so that they acted in the optimal way w.r.t. the actions performed by the evaders. Complementary, the actively evading smart targets were constantly optimizing against the actions of the trackers. In result the idea was to evaluate scenarios in which the individual players act according to their best response to the strategy of the opponents. We considered variants of the game with full, as well as incomplete information about the situation available to the players.

WP3: Multi-agent re-planning and plan repair

Actions of agents embodied in dynamic environments can fail. The more dynamic the environment is, the more often the actions fail. Unfortunately, most realistic environments are dynamic. This is even more so in military settings. One of the outstanding problems of artificial intelligence is the question how to implement behavior of intelligent agents in the face of environment dynamics, i.e., failing actions.

Classical-style planning is nowadays one of the most used techniques for automation of activities of intelligent agents. However, such plans are not robust w.r.t. unexpected events occurring in dynamic environments. The standard solution, in such cases, is to simply re-plan the agent's behavior from scratch and continue its actions according to the new plan.

In the Tactical-AgentScout 2, we consider heterogeneous teams of cooperative agents featuring different capabilities. In particular, we consider a heterogeneous units composed of human troops, UGVs and VTOLs cooperatively acting in order to achieve a joint team goal. While decentralized extensions of classical planning can be used to compute activities of the individual team members, full-scale re-planning of such multi-agent team plans in reaction to an unexpected event, or plan failure can become too costly. This is due to the costs of communication in multi-agent teams executing a decentralized planning algorithm. In scenarios where communication is costly, error-prone, or only low bandwidth is available, rather than optimizing for the computational efficiency of the planning process, we are

interested to minimize the communication complexity involved in it. In result, the main objective of this workpackage was to formalize multi-agent plan repair problem and devise and evaluate algorithms for solving it. The evaluation context was an urban military search and rescue mission involving information collection by the heterogeneous team of agents.

WP4: Coordination and teamwork

Reactive planning is an alternative approach to dealing with dynamics of the environment and resulting plan failures and unexpected events interrupting normal plan execution. It allows programmers to manually specify behaviors of agents in flexible manner so that agent's (resp robot's) action selection becomes efficient. The state-of-the-art techniques of reactive planning, however, do not support implementation of team-level behaviors. In particular, the main motivation for the final workpackage of the Tactical-AgentScout 2 project is the need to manually specify coordination mechanisms for a team of cooperative agents which pursue a joint goal (a mission), however none of them is able to carry out the mission on its own and in results, they are forced to coordinate.

The objective of this workpackage was to extend existing agent programming framework so that it can accommodate techniques for team-level coordination specification in terms of reactive plans executed jointly by the team members. In particular, we aimed at implementation of the *distributed commitment machines* approach in agent programming language *Jazzyk* implementing the framework of *Behavioural State Machines*. Finally, we aimed at evaluation of the implemented techniques in the context of military rescue/evacuation mission in urban environments.

Evaluation scenario

As a blueprint inspiring the project results evaluation scenarios, we consider a simulated military operation of a heterogeneous team in an urban area. We simulate a military evacuation mission implemented as and supported by several inter-dependent intelligent activities, such as traversal of an urban area, movement in a formation, information collection (surveillance, tracking of possibly smart targets, reconnaissance), heterogeneous team formation and coordination.

The scenario is put into a realistic context in that we additionally assume relatively unstructured and highly dynamic environment. Apart from a simulation of a physical environment (i.e., buildings, roads, landscape features, etc.), we assume presence of structural disturbances, such as blockades by collapsed buildings, placed improvised explosive device (IED), etc. at *a priori* unknown locations. The dynamics of the environment will be implemented by its partially hostile behavior, uncertainty and availability of only incomplete information facilitated by a number of actors, such as civilians and adversary units, etc. To cope with such a scenario, the task force has to comprise a heterogeneous team capable of cooperative and coordinated activities. In particular, we assume a number of unmanned aerial systems (UAS), possibly with a vertical take-off and landing capabilities

(VTOL), unmanned ground vehicles (UGV), a number of unattended ground sensors, such as cameras, acoustic and seismic sensors (UGS) and human troops in the area.

An example scenario showcasing the individual research objectives of the project could unfold as follows. The task force's mission is to find and evacuate a VIP from a location within a hostile urban area. The task force consists of a convoy consisting of a manned command vehicle supported by a number of UGVs and UASs. Possibly, there is a number of UGSs deployed, which form a field-deployed sensor network. In order to fulfil the mission, the aerial component of the task force must be able to perform a complex reconnaissance of the theater, be capable of continuous surveillance of areas within the theatre and must be able to detect, track and neutralize a number of moving ground targets, be it human units, or UGVs. In order to extract the VIP, the convoy navigates through the city and adapts its plans according to its information about blockades, IEDs or other obstacles to its path. The task of UGVs, supported by the UASs, VTOLs and UGSs in the vicinity is to protect the convoy from possibly hostile or non-cooperative actions of the adversary and civilians, as well as to support the surveillance and target tracking tasks, e.g., in areas inaccessible to UASs, etc. Upon VIP's extraction, the task force has to cope with a number of escaping adversaries implementing smart target strategy, i.e., intentionally avoiding and inhibiting the task force's information collection tasks.

1.3. Achievements and innovative claims

In the following, we summarize the main achievements and highlight the main contributions of the research and technological work performed in the context of the individual project workpackages. We conclude the section by a brief discussion of the implemented technological platform used for evaluation of the individual workpackages and highlight the list of the scientific papers and published either as a direct result, or inspired by the issues investigated in the context of the project.

WP1: Adversarial planning: patrolling mobile targets

Theoretical contributions:

1. we studied and subsequently extended game-theoretical etchniques for formalization of the problem of patrolling games focusing on mobile targets.

2. we proposed novel techniques for solving the problem on general street graphs without constrains on target (protected convoy's) movements, while considering the worst-case smart opponent.

3. we proposed and implemented an abstraction of the problem suitable for generation of a corresponding mathematical program.

Technological prototypes:

1. we implemented algorithms integrating and exploiting a state-of-the-art non-linear mathematical program solver for computation of patrol's strategies.

2. in order to support evaluation of the proposed techniques, we developed and implemented algorithms for smart attackers facilitating their realistic behaviors corresponding to maximizing the likelihood of a successful attack of unprotected convoys.

Evaluation and results:

1. we evaluated the proposed techniques and algorithms in a series of scenarios involving increasing numbers of attackers aiming at two independent convoys protected by a single UAV.

2. it turned out that proposed techniques for solving the patrolling of mobile targets problem are easily integrable with more realistic scenarios involving realistic sensor and actuators of physical agents, such as e.g., VTOLs implementing realistic physical movement dynamics.

3. *the abstracted techniques turned out to perform very well in comparison to fixed non-randomized policies. The use of optimal patrolling policies leads to 10% fewer attacks on convoys in the simulated urban environment than in the case with the non-randomized policies. The evaluation also shows, that the technique scales with the increasing number of considered attackers.*

WP2: Adversarial planning: smart targets

Theoretical contributions:

1. we investigated and formalized several models of pursuit-evasion games according to the state of the art in the domain. In particular, we devised universal algorithms for both player teams in the following settings:

   a) pursuit-evasion games in fully observable domains, i.e., both player teams have full information about the state of the world,

   b) visibility-based tracking game with imperfect information, i.e., the pursuer/tracker team tries to maximize the periods when the adversary is in its field of view, and finally

   c) visibility-based pursuit-evasion with imperfect information, i.e., the pursuer team aims at capturing the target(s), however only estimating its movements while it is not being visible.

2. the proposed techniques are based on our previous work on goal-based game-tree search.

Technological prototypes:

1. we implemented the algorithms for solving the problems in the three settings described above by employing anytime Monte Carlo Tree Search algorithm.

Evaluation and results:

1. we successfully performed a qualitative evaluation of the proposed techniques.
2. *we identified that the considered problem is highly sensitive to synchronization of behaviors of the individual agents.* In discretized setting, the agents implicitly arrive to their respective waypoints at the same time, however this is an unrealistic assumption in real-world scenarios due to e.g., imprecise path planning abstracting away possibly complex terrain features of the environment.

WP3: Multi-agent re-planning and plan repair

Theoretical contributions:

1. we formalized the multi-agent plain repair problem as an extension of the state-of-the-art multi-agent STRIPS-style planning framework.
2. we proposed three algorithms for the plan repair problem:

   NRA: *naive repairing algorithm*, based on iterative prolonging parts of the old plan and fragmentary repairs by re-planning from scratch,

   BRA: *blind repairing algorithm*, exploiting the possibility to push the failures in the semi-repaired plans forward into the future fix them opportunistically later, and

   LRA: *locality repairing algorithm*, based on the idea that localized repairs of plans do not influence other parts of the original plan, thus minimizing the need for heavy computational and communication.
3. we analytically studied computational and communication complexity of the proposed algorithms.

Technological prototypes:

1. we implemented the proposed algorithms in the form of a scientific software toolkit.
2. we integrated the proposed plan repairing algorithms with a state-of-the-art multi-agent planner.

Evaluation and results:

1. we experimentally evaluated computation and communication complexity of the NRA algorithms. *We verified the hypothesis that even the naive plan repairing algorithm performs significantly better that re-planning from scratch in the face of plan failures caused by a dynamic environment. The price of the reduction of communication complexity of the considered decentralized algorithms are plans which are slightly longer than optimal solutions of a classical planner.*

2. we evaluated performance of the NRA plan repairing algorithms in a tactical scenario in military urban mission involving a number of cooperating heterogeneous agents including UASs, VTOLs, UGVs and simulated human troops.

WP4: Coordination and teamwork

Theoretical contributions:

1. we extended and adapted the framework of *Distributed Commitment Machines*, a state-of-the-art theoretical approach to modeling multi-agent teamwork with explicit coordinationm, so that it can be used not only as an interaction monitoring tool (as originally devised), but also as a mechanism determining the behaviors of individual agents.

2. the shared team behavior specification yields a flexible mission specification language.

Technological prototypes:

1. we implemented the proposed coordination mechanism in a form of a multi-agent program written using a state-of-the-art agent programming language *Jazzyk*.

Evaluation and results:

1. we successfully performed a qualitative evaluation of the approach and implemented an example of a coordinated team behavior facilitating squad's movement through a simulated urban area in a formation, while still enabling adaptations to the local conditions (shape of the terrain), as well as swift reactions to unexpected events from the dynamic environment (encounter with an opponent).

2. *we verified the hypothesis that multi-agent coordination using the proposed techniques implemented in terms of a modular reactive plan is elaboration tolerant and provides a flexible representation of teamwork robust to changes in the code.* The technique allows rapid prototyping of team behaviors, as well as flexible adaptations thereof.

Technological platform

For the evaluation purposes of the project, we extended the technological platform incorporating the A-Lite toolkit also used in the past projects Tactical AgentFly 2 and in the precursor project Tactical AgentScout 1. The evaluation environment is a high-fidelity 3D environment dynamics of which is controlled by an event-driven simulation engine. In order to realistically model movements of the robotic vehicles in the world, the simulator integrates a physics simulator. In result, the unmanned aircrafts, as well as unmanned ground vehicles are capable of realistic movements modeling their physical properties. The Figure 1.2 depicts a screenshot from the simulated environment.

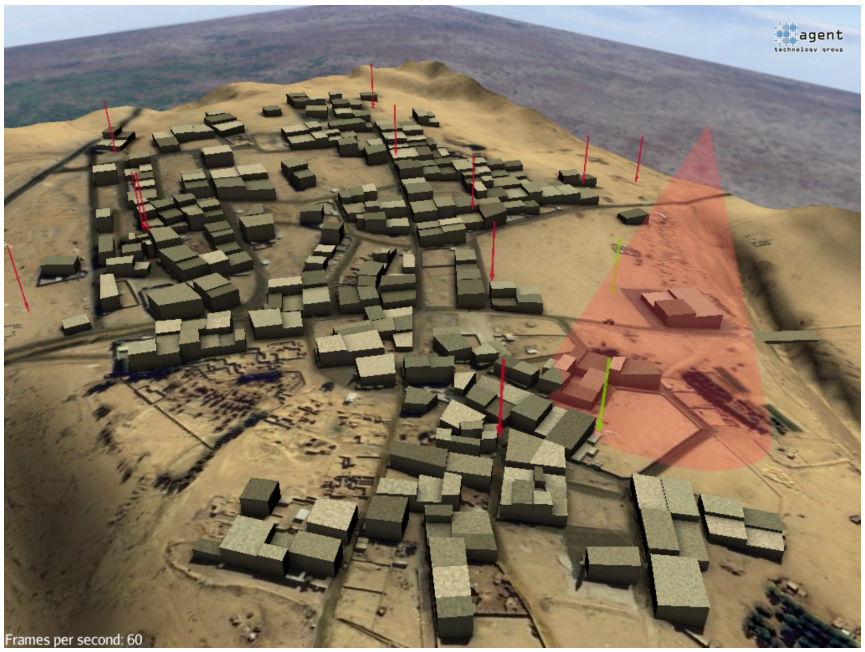


Figure 1.2.: Screenshot depicting a visualization of the high-fidelity urban environment model used in evaluation of the project workpackages.

Scientific output

As already highlighted above, the work on this project resulted in a number of scientific contributions to state of the art in the field of autonomous agents and multi-agent systems (subfield of Artificial Intelligence). As of writing this report, we already published two scientific papers at premier venues in Artificial Intelligence discussing the scientific contributions developed in the project's workpackages. Another two papers are currently under review at top-tier conferences and their affiliated workshops. The following lists summarize the scientific output of the project to date:

Published results:

- Branislav Boĺanský, Viliam Lisý, Michal Jakob and Michal Pěchouček: *Computing Time-Dependent Policies for Patrolling Games with Mobile Targets*, Proceedings of The Tenth International Conference on Autonomous Agents and Multiagent Systems, Taipei, Taiwan; AAMAS 2011, May 2011.

- Peter Novák and Wojtek Jamroga: *Agents, Actions and Goals in Dynamic Environments*; Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Spain; IJCAI 2011, July 2011.

Submissions under review:

- Antonín Komenda, Robert Lass, Michal Pěchouček, William Regli and Peter Novák: *Scalable and Robust Multi-agent Planning with Approximated DCOP*, submitted to the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI-11.
- Antonín Komenda and Peter Novák: *Multi-agent Plan Repairing*; submitted to Decision Making in Partially Observable, Uncertain Worlds, IJCAI 2011 co-located workshop.

After the project end, we still plan to further report on the results of this project, what should result in several additional scientific publications.

1.4. Managerial overview

The project unfolded as planned, without any significant problems and friction. Besides the principal investigator (Michal Pěchouček), the project involved a project coordinator (Peter Novák), staff of 4 full-time researchers (Branislav Bošanský, Michal Čáp, Antonín Komenda and Viliam Lisý) partially supported by one technical programmer and two undergraduate students.

Even though the original project proposal projected a single workpackage focusing primarily on adversarial planning for smart targets, in the course of the project execution it turned out that the topic of patrolling of mobile targets grew into a significant and self-encapsulated workpackage. Therefore, even though it wasn't highlighed as a separate package in the intermediary reports, this final report treats it a s full-fledged workpackage.

Generally, as also highlighted in above in this chapter, the project resulted in relatively deep investigation of the considered issues and inspired, or directly gave rise to a number of interesting and significant results – as also highlighted by the above discussed publication output.

1.5. Document structure

The remainder of this final project report is divided into four subsequent chapters 2, 3, 4 and 5 respectively providing a detailed account on the results of our work on the workpackages WP1, WP2, WP3 and WP4. The workpackage chapters feature a uniform structure. Firstly, a chapter provides an overview of chapter content in a *Summary of the workpackage* section. Subsequently, the section *Technology overview* introduces all the theoretical and implementation details developed and used in the project. Finally, sections titled *Evaluation and experiments* provide an overview of the project results evaluation and finally discuss and interpret the experiments conducted in its context.

The report concludes with Chapter 6 summarizing the main results of the report and providing an outlook on potential vectors of future work along the lines of research reported

upon here. Finally, the report is extended the Appendix A providing a brief overview of the implemented demonstrators delivered as a part of the project results package together with this final report..

2. Patrolling of mobile targets

2.1. Summary of the workpackage

In this workpackage we address the problem of patrolling. In patrolling scenarios a *defender* (also termed as a *patroller*) needs to protect multiple targets against an *attacker*. There are numerous examples of real-world scenarios that correspond to the patrolling problem – such as protecting a perimeter, covering some area, or protecting high-value targets. The common assumption in all of these examples is that the attacker can choose to attack any target at any time, and that the attack requires specific completion time (i.e. number of steps) during which the attacker can be discovered by the defender.

Game theory is a suitable framework for solving such problems as the solutions it provides are the optimal strategies for the defender given the opponents' information, capabilities and intentions. Game theoretic models have been successfully used to solve specific variants of the patrolling games in previous works [2, 3, 8, 9, 10] and there are several successful applications of similar game-theoretic models in real-world scenarios [71, 64]. We follow the previous work and extend existing patrolling game-theoretic models towards allowing the high-value targets to *change their positions* in time. Besides the theoretic work we integrated this method within the project Tactical AgentScout 2 and experimentally evaluated on a scenario, where an aerial patroller is protecting two convoys moving through an adversarial area. The experimental results showed that the game-theoretic model outperforms the baseline non-adversarial approach. Moreover, these results show that the game-theoretic model is better in spite of the fact that not all assumptions of the game-theoretic model were satisfied in the experimental settings.

This chapter is organized as follows. After reviewing the work related to our approach we provide a game-theoretic definition of the general problem of patrolling, following by the description of our approach for solving the patrolling game with mobile targets. Next, we describe the implementation phase – how this general mathematical model was applied in the project Tactical AgentScout 2 – together with the description of the experiments setting and the results. The theoretical essentials of our approach were published in [18].

2.2. Technology description

2.2.1. Related work

Two main classes of game-theoretic models are dealing with protecting targets or infrastructure from attacks of an adversary: *security games* and *patrolling games*. The main

common features of the games are (1) the presence of two players – the defender and the attacker; (2) a very limited amount of resources available for the task – the defender usually cannot guarantee preventing all the attacks, but it optimizes a utility based on the probability of a successful attack; (3) both classes seek the solution mostly in the form of a Stackelberg equilibrium – they seek a strategy that is efficient even if it is known to the attacker.

Security Games

In the security games [39, 75] model the defender protects the targets by simply allocating her limited resources to the targets. The attacker cannot observe current allocation of the resources and only selects the target to attack based on the known randomized strategy of the defender. There are several variants of security games, each reflecting an important characteristic in some real-world security scenario. The first variants of security games (in [39, 42, 37]) dealt with *heterogeneous resources* – the defender has several types of resources, each with different limitations on which target they can cover. Moreover, the authors presented concept of *schedules* (subsets of targets). Instead of assigning the resources to the individual targets, authors propose assigning them to a schedule meaning that the resource can simultaneously cover multiple targets. The schedules therefore represent a compact form, in which the game can be formulated.

Other variants of security games focus on the problem of multiple types of attackers – the model of *Bayesian security games* was studied in works [21, 63, 62]. In [75] authors explicitly defined a security game, in which the attacker also posses multiple resources and can attack multiple targets. This variant is a very important enhancement of the original security-game model and it is currently being analyzed by Korzhyk et. al. [43].

There are several existing applications based on security games (e.g. the system computing an optimal placement of checkpoints at LA International Airport [64], and assigning Federal Air Marshalls to protect flights in United States [71]).

Patrolling Games

In the patrolling games the defender protects the targets by moving through an area according to a strategy and periodically visiting the targets. The attacker can observe the current position as well as the strategy of the defender and in the right moment starts attacking some target. The attack takes some time and the goal of the defender is to interrupt this attack.

In [2], the problem of patrolling a perimeter is analyzed. The patrolled environment is modeled as a circle graph, where each node is a potential target. The authors seek the defender's strategy both as a simple Markovian policy and as a policy with an additional internal state. The implications of limiting the attacker's knowledge on the same game model are analyzed in [3].

The methods for perimeter patrol cannot be directly applied for patrolling environments

with more general topology hence the problem of patrolling on general graphs was studied in a sequence of works by Basilico et al. In [8] the authors define the patrolling problem on an arbitrary graph and provide a general model (termed BGA model) for computing the optimal strategy for the defender. The strategy is defined as a higher-order Markovian policy, though for computational reasons, only experiments with a first-order Markovian policy were performed. Further work in this line of research includes the analysis of the impact of the attacker's knowledge about the defender's policy on a general graph [9] and an extension of the model for multiple patrollers [10].

In our approach in this project we adopt the BGA model and further improve it in order to find optimal strategies for protecting mobile targets.

2.2.2. Patrolling game

A *patrolling game* is a *two-player* game between the defender and the attacker. The game model consists of a set of targets. The attacker may choose to attack any target at any time and the attack requires some time to complete. The defender tries to prevent the attacks by periodically visiting each of the targets. If the visit of a target occurs during the attack of this target, the attacker is caught. The initial condition for the patrolling-game model is, that the defender does not have sufficient time to deterministically visit all the targets to prevent all possible attacks and a *randomized strategy* has to be employed.

We discretize the environment, hence we assume that the game is played on a graph and in steps. In each step the defender and the targets can move to another vertex of the graph. The defender can move only to an adjacent vertex, but the movement of the targets in the graph is generally unlimited and they can move to an arbitrary vertex in the graph. Finally, in each step the attacker, while observing the situation, can choose whether to attack some target or wait. The attacker can choose to attack only one target during the course of the game.

Formal Definition

Environment The patrolling game is a game in *the extensive form* played on a directed graph $G = (V, E)$, where the targets Q and the defender can be positioned in any of $n = |V|$ vertices. We assume the set E is represented as an adjacency matrix $(e_{i,j})$, where $e_{i,j} = 1$ if there exists an edge from vertex i to j, $\{i, j\} \in E$, and $e_{i,j} = 0$ otherwise. The game is played in turns and we denote the set of turns T, indexed $t = 1 \dots |T|$[1]. The movement of the targets in the graph is modeled as a function $f : Q \times T \mapsto V$. We assume that a successful attack on a target takes D turns. The full information about the graph structure (G), information about current position of the defender and positions of the targets, as well as information about the movement schedules (f) is known to both players.

[1] When there is no confusion we use T to also denote the number of the turns $|T|$.

Strategies The goal of the defender is to move on the graph and to intercept an attack of the attacker, i.e. to come to a node where the attack is taking place. The randomized strategy of the defender has generally a form of a probability distribution over paths of length $|T|$ in the graph.

This, however, holds only if we assume that the game has a *perfect recall* – i.e. both players can remember history of all moves they have played before (i.e. the game can never return to exactly the same state, because at least the memory of the players would differ). The games with the perfect recall suffer from large space requirements – there are $O(N \cdot BF^T)$ states that needed to be explored (BF represents maximum degree of a vertex in the graph), which even for simple graph with 16 nodes (e.g. grid 4 by 4) and 14 turns can result in $\approx 10^{12}$ states.

Therefore in our approach we adopt the imperfect recall and we seek the strategy for a defender in the form of *first-order Markovian policy* – the defender makes her decisions based solely on the vertex, in which she currently is. The policy defines for each $i, j \in V$ and $t \in T$ a value $\alpha_{i,j}^t$ representing the probability that the defender present in vertex i in turn t moves to vertex j. We denote the set of all Markovian policies for the defender Θ_d.

The set of possible actions of the attacker $A_a = \{$*noop, attack(s,t,q)*$\}$ represents either the action *noop* (i.e. no attack), or starting the attack on a target q when the defender is in vertex s and it is the t-th turn of the game. If the attacker chooses one of the attack actions, it cannot perform any other actions and for next $d \in \mathbb{N}$ turns it can be captured in the vertices $\{f(q, t+1), \ldots, f(q, t+d)\}$. We assume that the attacker has a full knowledge of the stochastic strategy executed by the defender. This simulates the worst case attacker observing the defender for a long time before the attack or obtaining a reliable intelligence. The attacker's strategy is a response function ($AR : \Theta_d \mapsto A_a$), which selects an action for any of the strategies of the defender. We denote the set of all attacker's strategies Θ_a.

Utilities Finally, let us define the utility values for both players for each combination of their strategies. In general, there is a limited number of outcomes of the game. The attacker can either be captured, or it can successfully perform an attack on a target $q \in Q$. Following the BGA Model we define $X_0 \in \mathbb{R};\ X_0 \geq 0$ to be the reward for the defender when it captures the attacker and $X_q \in \mathbb{R};\ X_q \leq 0$ to be the loss of the defender when the attacker successfully performs an attack on a target $q \in Q$. Similarly, we define the loss and reward $Y_0 \in \mathbb{R};\ Y_0 \leq 0$, and $Y_q \in \mathbb{R};\ Y_q \geq 0$ for the attacker being captured and successfully attacking $q \in Q$, respectively.

As we stated above, the strategy of the defender is mixed (i.e. randomized), therefore the utility value assigned to a combination of two strategies is the expected utility. The values X_0 and X_q (or Y_0, Y_q respectively) are weighted by the probability of capturing the attacker (π_σ^q) in case that the defender plays ($\sigma \in \Theta_d$) and the attacker plays $AR(\sigma) \in \Theta_a$, which decides to attack the target q:

$$
\begin{aligned}
U_d, U_a &: \Theta_d \times \Theta_a \mapsto \mathbb{R} \\
U_d(\sigma, AR(\sigma)) &= X_0 \pi_\sigma^q + X_q \left(1 - \pi_\sigma^q\right) \\
U_a(\sigma, AR(\sigma)) &= Y_0 \pi_\sigma^q + Y_q \left(1 - \pi_\sigma^q\right) \\
U_a(\sigma, noop) &= U_d(\sigma, noop) = 0
\end{aligned}
\tag{2.1}
$$

Solution The defined problem corresponds to Stackelberg (or leader-follower) games and we search for a solution of the game in the form of a Strong Stackelberg Equilibrium (e.g. in [75]). The formal definition of this notion follows.

Definition 1. A pair of strategies $\langle \sigma, AR \rangle$ forms a Strong Stackelberg Equilibrium (SSE) if they satisfy the following:

1. The leader (defender) plays a best-response:
   $U_d(\sigma, AR(\sigma)) \geq U_d(\sigma', AR(\sigma')), \forall \sigma' \in \Theta_d$
2. The follower (attacker) plays a best-response:
   $U_a(\sigma, AR(\sigma)) \geq U_a(\sigma, AR'(\sigma)), \forall \sigma \in \Theta_d, AR' \in \Theta_a$
3. The follower breaks ties optimally for the leader:
   $U_d(\sigma, AR(\sigma)) \geq U_d(\sigma, AR'(\sigma))$

   $\forall \sigma$ and $\forall AR' \in \Theta_a$ satisfying 2.

SSE is a very suitable equilibrium for the security applications in the real world. First of all, the strategy in SSE is robust against the worst case opponents that have full knowledge of the strategy the defender is executing. Moreover, in many security games, the defender's solution for SSE is also a strategy in the NE of the game [75]. Hence, it is efficient also in the case that the attacker did not observe the defenders strategy and chose its action rationally only based on the definition of the game.

In our approach we assume that solution of this game cannot be deterministic – i.e. the defender cannot always protect the targets, nor the attacker can always perform a successful attack. Hence we seek non-deterministic solutions, where the defender is forced to randomize the movement to maximize the utility based on a chance of capturing the attacker.

Computing SSE in Patrolling Games with Mobile Targets

We use mathematical programming (MP) for computing a SSE in patrolling games with mobile targets – we formulate the problem as a mathematical program and use a generic solver (e.g. MATLAB®, or IBM CPLEX[1]) to find a solution. We adopt the BGA mathematical program [8] and further improve it in order to calculate the SSE for games with mobile targets (the details of the differences between our and BGA model can be found in [18]).

V, Q, T	the set of graph nodes / targets / time steps
X_q, Y_q	attacker's / defender's reward for successful attack on q
$\alpha_{i,j}^t$	prob. of using edge (i, j) if the defender is in i at turn t
$\delta_{i,q}^{h,t}$	prob. of moving from i to target q in *exactly* h steps starting at turn t
$\omega_{i,q}^t$	prob. of moving from i to target q in *at most* h steps starting at turn t

Table 2.1.: Quick reference of the notation and variable semantics. Indexes are omitted if not applicable.

Require: $G = (V, E)$ – graph; Q – targets
Ensure: σ – defender's strategy, v – strategy value
1: **for** $(s, q, t) \in V \times Q \times T$ **do**
2: $(v, \sigma) = MP(s, q, t)$
3: **if** $v > v_{max}$ **then**
4: $v_{max} := v$; $\sigma_{max} := \sigma$
5: **end if**
6: **end for**
7: **return** (σ_{max}, v_{max})

Figure 2.1.: The algorithm for computing the defender's policy for the game.

Developed MP searches for an optimal time-dependent policy $\sigma = (\alpha_{i,j}^t; i, j \in V; t \in T)$ for the defender. The reason for using time-dependent policy is that the defender can have substantially different strategy in the same vertex in different time steps because of the changed positions of the targets. Hence the $\alpha_{i,j}^t$ are the variables in our program. Besides that, we define an alternative set of variables $\delta_{s,q}^{h,t}$, which represent the probability that the defender positioned in the vertex $s \in V$ reaches the target $q \in Q$ in exactly $h \in \mathbb{N}$ steps while starting in the t-th ($t \in T$) turn of the game. In order to make the formulas more readable, we further define variables $\omega_{s,q}^t$ representing the probability that the defender positioned in vertex s visits the target q in *at most* D steps while starting in the t-th turn of the game. Table 2.1 summarizes used notation.

The mathematical program is constructed and ran for each attacker's action *attack(s,t,q)* as the best response. This reflects the motivation of the SSE – the attacker observes the defender and waits until the defender is located in the most convenient place for the attacker (s), it is the right turn of the game (t), and then starts the attack appropriate target (q). The main results of the program are the value of the game for the defender (i.e., maximized expected utility function for the defender) and defender's strategy $\sigma = (\alpha_{i,j}^t;\ i, j \in V)$.

Finally, as the overall solution of the patrolling problem we select those values of $\alpha_{i,j}^t$ that were found as the solution of the MP with the highest value of the objective function. The algorithm expressing the use of the MPs as sub-methods for finding a SSE is depicted in Figure 2.1. Each call of the MP in the algorithm optimizes the defender's policy $\sigma = (\alpha_{i,j}^t; i, j \in V; t \in T)$ under the assumption that $attack(s, q, t)$ is the optimal action of the attacker. The formulation of the MP for single configuration (s, q, t) is following:

$$\max_{\sigma} X_q \left(1 - \omega_{s,q}^t\right) + X_0 \omega_{s,q}^t \tag{2.2a}$$

$$\alpha_{i,j}^l \geq 0 \quad \forall i, j \in V;\ l \in T \tag{2.2b}$$

$$\sum_{j \in V} \alpha_{i,j}^l = 1 \quad \forall i \in V;\ l \in T \tag{2.2c}$$

$$\alpha_{i,j}^l \leq e_{i,j} \quad \forall i, j \in V;\ l \in T \tag{2.2d}$$

$$\delta_{i,g}^{1,l} = \alpha_{i,f(g,l+1)}^l \quad \forall i \in V;\ g \in G;\ l \in T \tag{2.2e}$$

$$\delta_{i,g}^{h,l} = \sum_{x \in V \setminus f(g,l+1)} \left(\alpha_{i,x}^l \delta_{x,g}^{h-1,((l+1) \bmod |T|)}\right) \quad \forall i \in V;\ g \in Q;\ h \in \{2, \ldots, d\};\ l \in T \tag{2.2f}$$

$$\omega_{i,g}^l = \sum_{h=1}^{d} \delta_{i,g}^{h,l} \quad \forall i \in V;\ g \in Q;\ l \in T \tag{2.2g}$$

$$Y_q \left(1 - \omega_{s,q}^t\right) + Y_0 \omega_{s,q}^t \geq Y_{q'} \left(1 - \omega_{s',q'}^{t'}\right) + Y_0 \omega_{s',q'}^{t'} \quad \forall s' \in V;\ q' \in Q;\ t' \in T \tag{2.2h}$$

The first two constraints (2.2b), (2.2c) ensure that the probabilities $\alpha_{i,j}^t$ represent a correct defender's policy σ, and constraints (2.2d) ensure that the defender moves only between two adjacent vertices. The constraints (2.2e)-(2.2f) define the probability $\delta_{i,g}^{h,l}$ of reaching a target in exactly h steps using the policy σ. To do so, we express the vertex of the target using the function f. If $h = 1$, δ is equal to probability connecting the current position of the defender i and the position of the target in the next turn $f(g, l + 1)$. For higher h, it is the probability of moving from the current position to some node x, which is different from the vertex of the target $f(g, l+1)$, multiplied with the probability of reaching the target g from the node x in exactly $h - 1$ steps. The constraints (2.2g) define variable ω and constraints (2.2h) ensure that no alternative attacker strategy can provide higher attacker's utility U_a. The optimized function (2.2a) expresses the expected utility U_d of the patroller's policy σ for a fixed combination of $s \in V,\ q \in Q$ and $t \in T$.

Solving the Mathematical Program If some solver can optimally solve the program defined above, we would have the optimal strategies for the patrolling problem. However,

solving this program is hard. The number of program constraints and variables in the improved stationary formulation is $O(|V|^2 \cdot |T| \cdot D)$ in the time-dependent case. Most of the constraints are bilinear; the remaining constraints as well as the optimized function are linear.

The formulation of the programs in previous sections was chosen with the readability as the main criterion. However, the exact form of the formulation can influence the computational complexity of solving the problem optimally as well as the potential for approximation. A different formulation of the problem can be constructed if some of the program variables are not represented explicitly in the program.

Bilinear MP The presented form of the programs expresses the optimization of a linear function over a region defined by (at worst) bilinear constraints. The size of the program is polynomial in the relevant problem parameters. However, solving a bilinear program is in general NP-hard [12]. Non-convexity of the feasible region that is defined by the bilinear equalities indicates that this particular problem is most likely not an exception. On the other hand, these programs are widely studied and many approximation algorithms are available.

Polynomial MP Some of the variables in the presented programs do not have to be represented explicitly in an actual program formulation. For example, the variables ω in program (2.2) can be clearly removed and all its occurrences can be substituted by the corresponding sum of variables δ. This modification still leads to a bilinear program. However, if we also remove the variables δ in the same way, we are in a different class of MPs. All the bilinear constraints are removed and only the linear constraints remain. However, the complexity of the optimized function increases dramatically. Instead of linear, it becomes polynomial with maximal degree D. As mentioned in [45], even unconstrained optimization of 4-degree polynomials is NP-hard, hence this formulation is also not likely to produce optimal solution for larger problems in reasonable time.

Experimental Evaluation – Synthetic Graphs

We present an experimental evaluation of the proposed approach on the synthetic graphs outside the Tactical AgentScout 2 domain. The focus of the experiments is on the quality of the solutions produced by the proposed non-linear program.

We used the following settings for the experiments: (1) we used two types of graphs – *grid* and *grid with holes*; (2) two targets were present in each setting and we used three different movement schedules for the targets; (3) we simplified the values of the targets and assume that all targets have the same value; (4) we compared the quality of produced time-dependent policies to an approximation calculated as a stationary policy.

We conducted the evaluation on two types of graphs inspired by a typical application domains: (1) *grid with holes* (see Figure 2.2(a) for an example) which may e.g. represent a road network, (2) *full grid* (see Figure 2.2(b) for an example) that corresponds to discretization of open space, such as ocean surface. In both figures, black nodes represent initial positions of targets and dashed arrows show motion patterns for the targets. In all

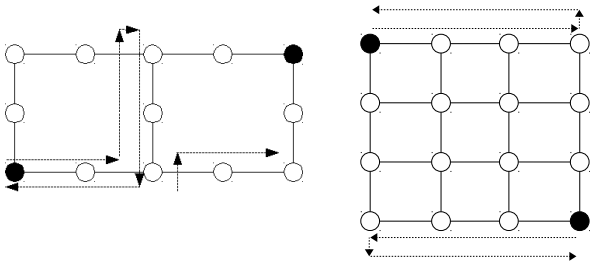


Figure 2.2.: The schema of the experimental scenarios. Black nodes denote target's initial poositions and arrows depict target's movement.

experiments, targets move once per two defender's moves; this reflects that the defender is faster than targets.

As adding more targets did not show any interesting changes in the results, we limit the presentation to experiments with two targets only. Three types of target movement with different implications on the distance between the targets were employed: (1) *alternating* where the distance between the targets is decreasing and increasing in time (see Figures 2.2(a),2.2(b)); (2) *equidistant* is defined only for grid graphs and involves simultaneous movement of targets along the top-most and bottom-most edges of the graph from left to right and back. In Figure 2.2(b), the target at the bottom starts from the left side and moves in the same way as the one on top; (3) *stationary* where targets remain in their initial positions. Finally, in all experiments we adopt the repeated version of the game – i.e. the targets are moving in cycles and the game repeats each $|T|$ turns.

Program for Time-Dependant Policies For explanatory reasons we simplified the values of the targets, that neither the attacker nor the defender has any preference among the targets – the attacker tries to maximize the probability that it will successfully attack some target and the defender aims to minimize this probability. This corresponds to an instance of the defined patrolling problem where $X_q = -Y_q = -1$; $\forall q \in Q$ and $Y_0 = X_0 = 0$. As we want to evaluate the probability of catching the attacker we assume that the attacker has to attack some target (i.e, we disallow *noop* action for the attacker).

Using above simplification the game became a zero-sum variant of the original problem, however, it does not substantially change the characteristics of the program, nor it is significantly computationally easier to solve compared to the original formulation due to the non-linearity in constraints of the MPs (δ variables). The only change is the simplification of the objective function and utility-based constraints, which enable us to formulate the MP as a single *min-max* optimization instead of a sequence of optimizations of MPs for each initial point, turn, and target. We are searching for σ that optimizes:

$$\max_{\sigma} \min_{s,q,t} \left(\omega^t_{s,q}\right) \tag{2.3}$$

s.t. (2.2b)-(2.2g)

Constraints (2.2h) are substituted by the maximization of the objective function and can be removed. We further refer to the value of the objective function (2.3) as the *reached value of the game*.

Program for Stationary Policy In order to evaluate the quality of the solutions based on a time-dependent policy we need to obtain a stationary policy, which still can be efficient even with moving targets in some cases (e.g. if the movement is limited). We compare the performance of these two formulations in terms of the reached game value and computation time in the game with moving targets.

In order to obtain a stationary policy, we have slightly modified the MP (2.3) – we have removed the time index from all α variables in all constraints. All δ and ω variables keep the time index in order to take target's movement into account. This modification is especially useful if the variables δ and ω are not explicitly represented in the implementation of the program. In that case, the number of real variables in the program decreases significantly. Besides the comparison reasons we used the stationary policy as an initial point for solver for calculating the time-dependent policy.

Implementation We implemented the proposed mathematical programs in MATLAB® using the *fminimax* function for the optimization. For both programs – the MP for the time-dependent and for the stationary policy – we use only α as variables; variables δ and ω are not explicitly represented as variables of the MP. The set of α variables is limited to those $\alpha^t_{i,j}$ for which there exists an edge between vertices.

Internal MATLAB parallel methods were used during the optimization, hence the duration of the experiments is expressed in the total CPU time (in seconds) consumed on all cores.

Results The results on the synthetic graphs proved that in the game with mobile targets it is reasonable to use the time-dependent policy. In most of the experimental settings usage of time-dependent policy led to significantly higher utility value than the approximation using a stationary policy but currently at the expense of significantly higher computational costs.

The most representative results, in terms of the reached game value and the average computation time, from two graphs (shown in Figures 2.2(a) and 2.2(b)) were selected and depicted in Table 2.2.2. Note that the value reached in the zero-sum variant represents the worst-case probability that the defender catches the attacker during the attack on some target. As expected, the dynamic policy is significantly better than the stationary approximation, as the defender can better adapt to the movement of the targets. For the third target movement type, i.e. stationary targets, both methods converged to the same values and policies.

Graph	Mov. Type	Policy	d	*value*	*time* $[s]$
grid 4x4	alternating	stationary	8	0.19	6.60
		dynamic		0.50	3516.20
		stationary	9	0.33	30.81
		dynamic		0.89	14063.46
	equidistant	stationary	9	0.32	37.32
		dynamic		0.50	333.22
		stationary	10	0.37	39.81
		dynamic		0.69	1338.19
grid-hole n13	alternating	stationary	8	0.17	4.83
		dynamic		0.50	3194.19
		stationary	9	0.26	13.58
		dynamic		1.00	9859.08

Table 2.2.: Comparision of the reached value of the game (equals the probability that the defender catches the attacker) and the average copmutation time; d denotes attack duration.

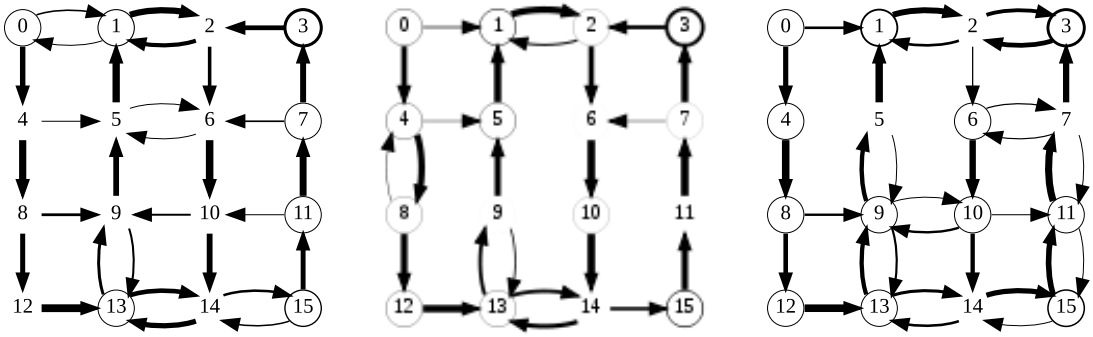


Figure 2.3.: Defender's policies. Two targets move right from vertices (0,12) to (3,15) and back. The probability of using an edge corresponds to thickness of the respective edge or circle (in the case of loops). A stationary policy (Figure (a)) and two snapshots of a time-dependent policy are shown – turn 6 with targets at (2, 14) (Figure (b)) and turn 7 with targets at (3, 15) (Figure (c)).

The frequent appearance of 0.5 as the reached game value in Table 2.2.2 stems from having two targets. In many settings, the defender cannot protect both targets and thus it non-deterministically "chooses" just one of them; the attacker then succeeds if it attacks the other target. Note that the MP also found a deterministic policy that always leads to catching the attacker (reached value is 1.00).

The differences between stationary policy and time-dependent policies for the defender can be seen in Figure 2.2.2: the stationary policy 2.3(a) covers all positions of the targets in time, while the time-dependent policy can utilize the knowledge of the current positions of targets (vertices 2 and 14 in 2.3(b) showing turn 6) and also future positions of targets (2.3(c) shows turn 7 of the game with targets in vertices 3 and 15). Note, that thanks to the time-dependent policy, the defender can in turn 7 reach the target in vertex 3 from the vertex 2 in one move, however, there is no such possibility in the stationary policy.

2.2.3. Integration with the simulator

Experiments on synthetic graphs showed that the time-dependent policy can significantly improve the utility of the defender compared to the stationary policy. Therefore, it is reasonable to consider the application of the time-dependent policies into real-world scenarios. However, high computational requirements on the computation of the randomized policy prohibit us from using the method directly in the Tactical AgentScout 2. In this section we therefore first describe the scenario, in which we intend to utilize the patrolling, following by approximation method based on so called *distance graph* and experimental results.

Tactical AgentScout 2 – Patrolling Scenario

The screenshot of the patrolling scenario in the project Tactical AgentScout 2 is depicted in Figure 2.2.3. There are two convoys (i.e. two targets) crossing an adversarial area, and there is a single patroller providing an aerial support. The goal of the patroller is similar to the standard definition in a patrolling game – i.e. to periodically visit both convoys and prevent any possible attack from the adversaries.

Distance Graph

This scenario can be modeled as the mathematical program from Section 2.2.2. The size of the program will be, however, too large, hence unsolvable for any mathematical solver. Therefore, we use an abstraction termed the *distance graph*. Note, that to find the optimal policy it is unnecessary to model the entire environment – i.e. to exactly compute the solution of the patrolling game on the graph of the complete road network. Firstly, the patroller is an aircraft vehicle, hence its movement is not limited to the road network. Secondly, modeling the precise structure of the graph is not necessary for finding the optimal policy for the patroller – the essential is the distance between patroller and the targets. The patroller can in each step choose to come closer to a target, or move away from a target. Therefore, in the scenario with two mobile targets, the optimal strategy of

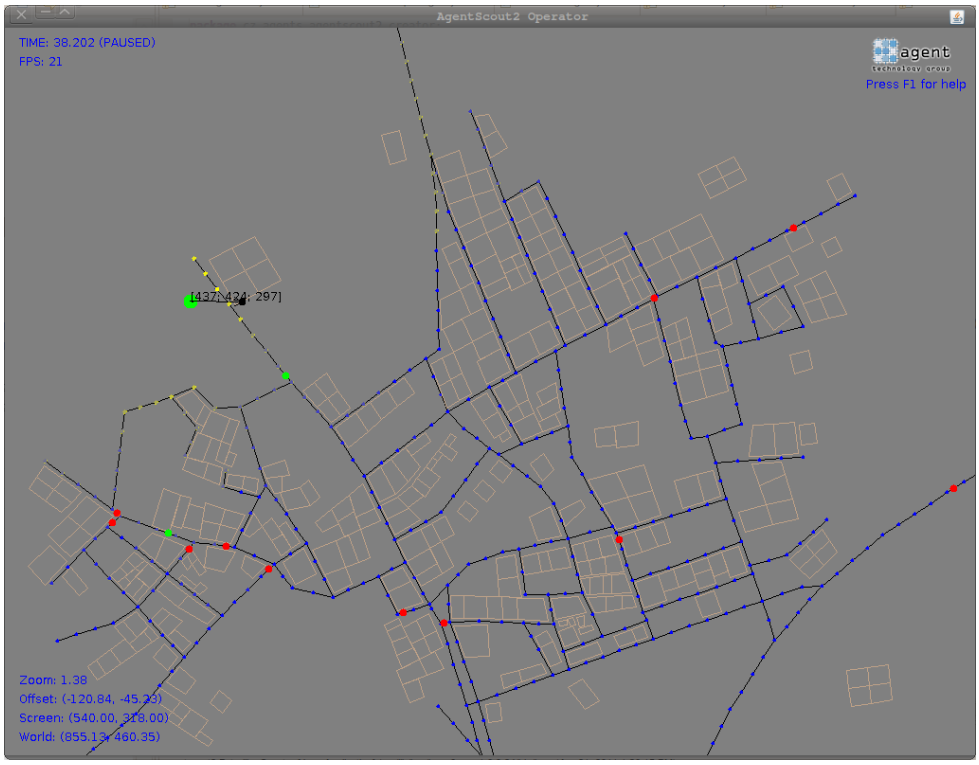


Figure 2.4.: Patrolling scenario implemented in the Tactical AgentScout 2 environment. The small green nodes represent convoys that need to cross an area, in which multiple adversaries operate (red nodes). They are protected by a single patroller (large green circle). Blue nodes represent street-graph nodes, and yellow nodes represent the sensor of the patroller.

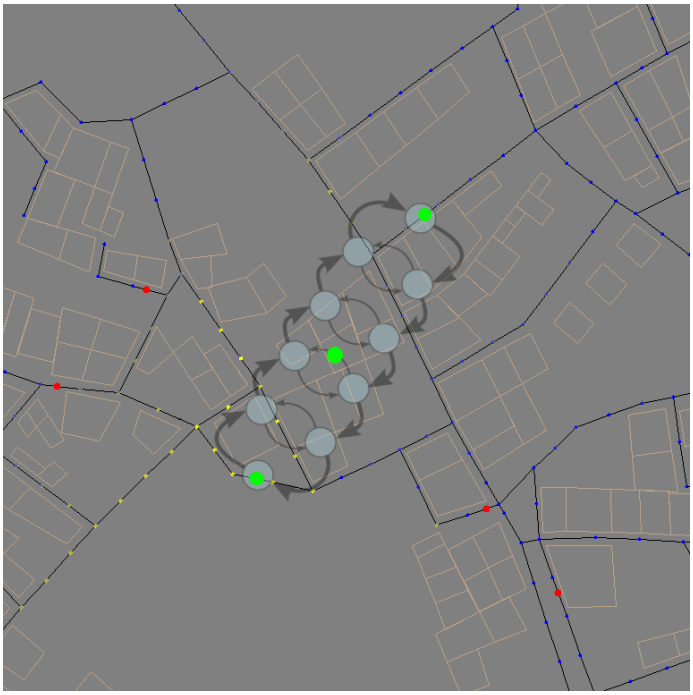


Figure 2.5.: Distance graph in the patrolling scenario. The patroller follow policy calculated on the distance graph. The targets can change their position in the distance graph in time reflecting changes in the distance between them in the environment.

the patroller is a randomized walk on the shortest path between the targets – on a distance graph between the targets. The distance graph for the patrolling scenario in the Tactical AgentScout 2 is depicted in Figure 2.2.3. The targets are placed in some vertices of the distance graph according to their distance in the environment (note that the distance can change in time and therefore the targets can also change their position in the distance graph) and patroller follows a policy calculated on this graph – in each decision point it can decide whether to follow its current heading towards a target or it can turn around. For scenarios with multiple targets, the distance graph will be constructed similarly as a complete graph between all the targets, however, some additional vertices will have to be added to keep the solution computed on the distance graph to be optimal in the original game as well.

2.3. Evaluation and experimental results

We implemented the model of the distance graph and the patroller following the optimal time-dependent policy computed for the distance graph into the Tactical AgentScout 2 project. We used two ground convoys with scripted paths through the area, and a single patroller modeled using model of a helicopter with simulated dynamics. Compared to the game-theoretic model we use different representation of the adversaries. The game-theoretic model assumes that there is only a single attacker observing the situation and waiting for the right moment to start to attack some target. Such assumption is rarely fully corresponding with the real-world scenarios. Therefore, instead of using an omniscient adversary that can attack at any time any target we use multiple simple adversarial units. They follow the convoys, but they are deterred by the patroller. We project the sensor of the patroller to the nodes of the road network (yellow nodes in Figure 2.2.3) – these nodes represent the "active range" of the patroller and they are selected based on the current velocity vector of the patroller. If the patroller turns the other way, the nodes are slowly fading representing the aging of the information. This sensor together with the simple model of adversaries sufficiently reflect desired situation. The adversaries tend to go to a convoy, however, they do not want to be "seen" by the sensor of the patroller.

To experimentally evaluate the performance of our game-theoretic approach (further referred to as the *randomized strategy*) in this scenario, we compared it with the classical algorithm for protecting targets – deterministic policy, where the patroller visits each target in a deterministic manner (further referred to as the *fixed strategy*). We implemented both methods using the same helicopter agent and the only difference was in the algorithm, which selects a convoy to visit next. We compare these two methods by means of number of occurrences of so called *dangerous situations* – a dangerous situation occurs if an adversary is close to a convoy and it is not "seen" by the sensor of the patroller. Dangerous situations reflect possible attacks on the convoy and to goal is to prevent them.

We executed experiments on the patrolling scenario in the Tactical AgentScout 2 environment with fixed paths of the convoys, and with the increasing number the adversaries

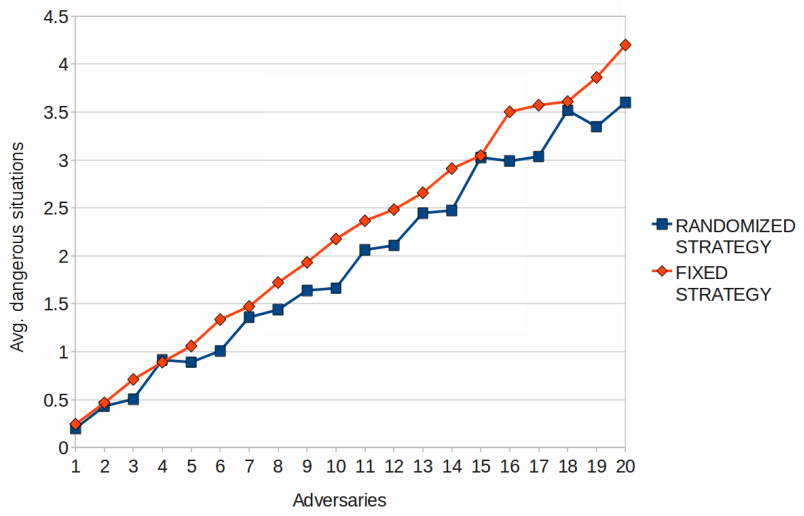


Figure 2.6.: Comparison of performance between the randomized policy based on the game-theoretic approach and the deterministic approach. Lower number of average dangerous situation is better.

present in the environment. The adversaries were placed randomly in the road network and for each configuration (i.e. for each method and for each number of adversaries) over 400 different samples were executed. The results, depicted in Figure 2.3, show that the randomized strategy based on our approach of solving patrolling games with mobile targets is not worse compared to the fixed strategy and in many configurations is significantly better. On average, the usage of the randomized strategy lowers the number of dangerous situations by 10%.

3. Modelling smart targets

3.1. Summary of the workpackage

In this work package, we developed algorithms for controlling a smart target that actively avoids detection, tracking and capture by a team of pursuers. Furthermore, we developed algorithms for the team of pursuers that perform these tasks.

We first provide a brief review of the related research in the areas mathematical pursuit-evasion games and tracking, more practical probabilistic pursuit evasion games, and general algorithms for playing games of imperfect information. Next, we formally define the problem solved in this chapter. We assume a suitable discretization of the space and provide a formalization that allows describing pursuit-evasion games with teams of agents that have different mobility and field of view restrictions. The objective in these games can be either to capture the evader or to move the pursuers in a way that maximizes time in which the evader can be observed by the pursuers. We further define several incrementally complex sub-games of the complete games that are investigated separately.

The algorithms we provide for creating behaviors in the games are based on general algorithms for playing (imperfect information) games. For games with full observability of the agents, we base the algorithms on the Monte-Carlo Tree Search (MCTS) algorithm with action selection based on the Upper Confidence Bound that has been recently very successful in various domains. In the partially observable games, we base our approaches on the paranoid version of information set search. We use it both with the more standard minimax algorithm and with MCTS. We discuss several adjustments of these algorithms needed in order to make them applicable in our domain. The main problems are caused by

- the presence of simultaneous action of the players,
- the scalability with increasing number of units available to the players and the size of the graph.

We tackled the *scalability* issues by adding domain specific knowledge to the game playing algorithms. We designed domain-specific "heavy" simulation methods for MCTS and we used procedural knowledge heuristic to prune the search space. For improving scalability with the number of agents, we analyzed an option of weakening the coupling the actions of agents within one team. However, this approach does not seem to be promising in this domain. When dealing with the *simultaneous moves*, we used the standard approach used

in successful players in general game playing competition. Even though we have identified serious weaknesses of the method, it is still usable in our domain.

We have implemented the agents in a simulation and performed basic feasibility experiments. The presented algorithms are able to produce reasonable behavior in the simulation in real time. However, in the current form, the algorithm does not scale favorably with the number of units in the scenario. The behavior produced by our anytime algorithms exhibit serious flows when used with more than four units and "real-time" CPU restrictions.

3.2. Technology description

The problem of collaborative tracking or surveillance of targets that are being aware of being monitored (further termed as 'smart' targets or evaders) is formally described by various variants of *pursuit-evasion games* (PEGs). There are several different characteristics of variants of PEGs. The differences can be found in environment (it can be a graph, polygonal environment, or free space), different relation of the pursuers' and evaders' maximal speed (the speed can be bounded or not, or the speed of the evaders can be superior or not), different range of sight (whether there is a limited range of visibility or the angle of the sight (e.g. camera)).

Even though there is limited number of results on PEGs in continuous space, they are generally limited to open space with no obstacles, or very restricted topology. As we aim to apply the developed methods in high fidelity simulation and later in the real world, we focus on approaches with discretized time and space. In general, we are interested in games on graph, which progress in discrete time steps.

There are two classes of methods that have been used for solving PEGs and games in general: mathematical programming (MP) and state space search (SSS). For pursuit-evasion games on graphs with hundreds nodes that are needed to represent our domain, methods based on state space search (SSS) (e.g. alpha-beta) are more suitable. The advantage of MP is that it provides provable optimal solutions or approximations of the optimal solutions with bounded error. However, these methods generally require huge amount of memory and computational resources. As a result, they cannot provide real-time performance with larger spaces or teams with higher amount of agents. The SSS methods often sacrifice optimality and often even completeness in order to quickly provide a reasonable solution. Therefore the SSS approach is more suitable for time-critical tasks of searching and tracking where the position of the targets changes unpredictably.

3.2.1. Related work

Capturing a smart evader in an environment is the basic task in pursuit-evasion games (note, that searching for an evader can be seen as a special case of capturing, where the evader is considered captured when the visual contact is established). One of the best explored variant is introduced in [70], where a group of pursuers is searching for a single evader with unbounded speed in a polygonal environment. This variant was further tackled

in [30] where a problem of determining the minimum number of pursuers needed for which exists a strategy to capture the evader was shown to be NP-hard. Several works were focused on removing various restrictions such as focusing on curved environment [47], or limiting the visibility (by terms of field of the view [28], or the range of sensors [35]). The formal definition of the problem of finding a strategy for pursuers to locate an evader with unbounded speed on a graph was formalized in [41] as a GRAPH-CLEAR problem showing to be NP-complete.

All of above approaches was concerning the problem of capturing the evader in case the pursuers do not know the position of the evader. On the other hand, the game of *cops and robbers* introduced in [34] describes a game, where a group of cops are trying to catch equally fast robber with known position. In general, the game is solved by a variant of SSS (as e.g. *TrailMax* algorithm in [51]). Multiple variants of the game have been introduced over time (e.g. limited visibility of the cops was analyzed in [36]), however none of the models match all our requirements (see Section 3.2).

Tracking of a smart target is another task related to the problem. In [46] authors define tracking (also being called as a problem of maintaining visibility) of an unpredictable evader in a polygonal environment with obstacles in discrete time. A game-theoretical framework introduced in this work was further improved in [25] where a position uncertainty of a robot pursuer was considered. Different variant was considered in work [7] where stealth target tracking was introduced – pursuer is tracking an unpredictable moving target in an unknown environment with obstacles and, at the same time, remains hidden from the target. On the other hand, the game-theoretical framework of a continuous case of tracking for holonomic agents in known environment with obstacles was described in several works by Bhattacharya and Hutchinson ([13], [14]), finally defining strategies being in Nash Equilibrium in [15].

All of mentioned works considered only the problem of tracking with one pursuer and one evader. In the domain of collaborative target tracking the target is usually moving in random direction (as e.g. as in the work of Vidal et al. [73] where a group of UASs and UGVs is tracking and pursuing a ground evader) and the issues of communication in an agent network are being tackled (e.g. in [22]). The game theoretic approach is applied in works of Harmati and Skrzypczyk [33] where a group of pursuers is tracking an evader. The coordination, collision avoidance and keeping the shape of a formation is formalized using game-theoretical framework and appropriate utility functions, while for solving they introduce a semi-cooperative Stackelberg equilibrium.

The algorithms designed for specific classes of games can be very efficient, but they usually strongly depend on specific assumptions and are not usable at all if they do not hold. In order to achieve high flexibility and produce a reasonable (even if suboptimal) course of action in any situation, we aim to base our methods on general game playing algorithms that proved to be successful in variety of domains.

The most prominent algorithms in this area are the *information-set search* [60] and Monte-Carlo tree search algorithms [40]. Although there are no guarantees for these algorithms to find even approximate equilibrium in general extensive-form games, they often

perform well in a real play. These algorithms are described in more details in Section 3.2.3.

Alternatively, approaches based on learning could be utilized. Fictitious play [20] is a powerful mechanism that allows learning equilibrium strategies in many games. It has been successfully used e.g. in the network security domain [52]. Similar concept has been used in the algorithm called Counter-factual regret minimization [77], one of the top performing algorithms for playing poker. It iteratively computes (approximate) NE based on regret matching [29] and it is guaranteed to converge to a NE in zero-sum games in finite number of iterations. It has a low memory requirement (linear, compared to possibly exponential requirements of mathematical programming) and the speed of convergence can further be improved my Monte-Carlo sampling [44].

3.2.2. Formal game definition

The related research section of this chapter shows that there is a large body of research of several variants of pursuit evasion games (PEGs). It includes multiple options for representation of the space and motion dynamics of the players. However, none of the existing formalization fully captures all aspects of the game required for the scenarios supported by the project. The main differences in our scenario compared to the standard pursuit evasion games are following.

- **Multiple Evaders** Most of the games assume single evader trying to escape from one or a small team of pursuers. In our scenario, multiple smart targets are trying to escape from the team of pursuers.
- **Heterogenity** The pursuers and evaders in standard PEGs are assumed to move in the same space and each pursuer has the same capability to capture the evader. In our scenario, we have several sources of heterogeneity. First, different agents have different *movement restrictions*. The evader can cross obstacles like scarce bushes, narrow corridors, or stairs, while they can be impassable for UGVs. The movement of UASs is not influenced by the obstacles on the ground at all, but it often has a more complex movement dynamics restriction that does not allow it to move arbitrarily. Moreover, our scenario assumes that only the UGVs can **capture** the evader, which is another source of heterogeneity.
- **Limited Visibility** The smart targets are likely to use terrain features, like trees, awnings, or obstacles by buildings in narrow streets to prevent detection. Therefore, we need to assume different visibility models for the pursuers.

Even though we aim to apply the developed algorithms in high fidelity simulator with continuous time and space, we use a discrete model for formal definition of the game due to lower computational requirements compared to continuous case.

Complete Game Definition

The PEG with Heterogeneous Agents (PEHA) is a tuple $(I, N, pos_0, t, v, w, h)$ where

- $I = O \cup P \cup E$ is a set of agents, which is composed of disjoint subsets of observers (O), pursuers (P), and evaders (E)
- N is a common set of nodes (positions) for the agents
- $pos_0 : I \to N$ are the initial position of the agents. Generally, we refer to the position of the agents at (the end of) time step l as pos_l.
- $t : I \times N \to \mathcal{P}(N)$ is the accessibility mapping that assigns to each agent and each place a set of places the agent can move to from the given position
- $v : I \times N \to \mathcal{P}(N)$ is the visibility mapping that assigns to each agent and each place a set of places in which the agent can see to other agents from the given position
- $w : E \to \mathbf{N}$ is the value of capturing an evader
- h is the number of time steps in which the game is played (i.e., the fixed horizon)

Definition 2. We say that an evader $e \in E$ is *captured* by a pursuer $p \in P$ in time step s iff

$$pos_s(e) = pos_s(p)$$

Definition 3. We define a legal path (strategy) of agent a as a mapping $\rho_a : \{1, \ldots, h\} \to N$, such that

$$\forall s \in \{1, \ldots, h\} \;\; \rho_a(s+1) \in t(a, \rho_a(s))$$

We further denote the joined path (strategy profile) of a subset of agents $G \subseteq I$ by the same symbol with set subscript ρ_G.

The objective of the pursuers is to capture as many valuable evaders as possible within the limited time horizon. They are supported by the observers, which provide additional information about target positions. More formally, the game objective is

$$\textbf{objective} : \arg \max_{\rho_{(O \cup P)}} \min_{\rho_E} \sum_{\{e \in E : \exists s \in \{1,\ldots,h\}, p \in P \; \rho_e(s) = \rho_p(s)\}} w(e) \tag{3.1}$$

The solution of the game are the legal paths for the observers and pursuers that maximize the importance of the captured evaders in the worst case (i.e. optimal play by the smart targets).

Simplified Game Variants

In solving the game defined in the previous reports, we use the standard approach. We add several simplifying assumption, solve the problem and based on the experience obtained and techniques developed, we gradually remove the simplifications. In order to do that, we define two subsets of the complete game.

Fully Observable Pursuit-Evasion Game This simplest game model we use adds the following assumptions to the game definition.

- **Full observability:** the visibility mapping v assigns each agent and each possible position the whole set of nodes on the map.
- **Homogeneous movement:** the movement restrictions are the same for each of the agents. In particular, the "speed" of the agents is unified.
- **Single evader:** the set of evaders has only one element ($|E| = 1$).
- **No observers:** we assume that the set of observers is empty ($O = \emptyset$). I.e. all the agents of the blue team are able to catch the evader.

With full observability and unified movement restrictions, the game we solve is similar to the well-studied cops and robbers game. The first report of this project contains more details on the similar game models. Our game differs from the classical cops and robbers game in one key element – simultaneous moves. The agents choose their next position on the graph simultaneously and hence they are introducing a simple form of imperfect information to the game. The agents are uncertain about the positions of their opponents in the following move. Hence, they are also uncertain about the real outcomes of their actions.

Visibility Tracking Several state related research works consider a model, where the task of the team of pursuers is not to capture the evader (i.e. be collocated with him), but to keep him in sight. These games can also be set into our general model, but we have to modify the objective and add a few restrictions. We assume the following:

- **All observers:** we assume there are only observer and evader agents ($P = \emptyset$). I.e. the game always continues for the evader until the time horizon is reached.
- **Modified objective:** the objective in the visibility tracking game is to see the important evaders as many times as possible. Formally:

$$\textbf{objective} : \arg\max_{\rho_{(O)}} \min_{\rho_E} \sum_{s\in\{1,\dots,h\}} \sum_{\{e\in E:\exists p\in P \ \rho_e(s)\in v(\rho_p(s))\}} w(e) \tag{3.2}$$

3.2.3. State-of-the-art approaches

In this section, we briefly sketch the intended approach to solving the problem. So far, we have focused more on the patrolling problem in the domain; hence we do not deliver a more detailed analysis of the solution of this problem in this report.

In our scenario, we need real-time decision making in fairly large game, hence we have decided to adopt the state space search (SSS) approach as discussed above. We intend to implement some of the state-of-the art methods surveyed in the related work section of

this chapter. Due to expected high computational complexity we intend to focus mainly on incomplete methods that cannot offer theoretical guarantees on the produced solution, but which has been proven to be successful in other domains. These methods include mainly Monte-Carlo tree search method (such as UCT[40]) and the methods for exploiting various forms of sparseness in game domains, which have been developed by ATG in its previous projects [48, 49].

Monte Carlo Tree Search

All the algorithms presented in this section are based on Monte Carlo tree search (MCTS) techniques. These techniques are suitable in our domain for a number of reasons. First of all, they proved to be successful in a number of different domains. An agent based on MCTS technique called UCT [27] was the winner of the general game playing (GGP) competition organized by AAAI in 2007 and MCTS agents stayed on the top position every year, including the last competition in 2010.

The key advantage of the algorithm for our application domain is its anytime property. The algorithm can use any amount of computational resources available to produce near optimal strategy, but even after a short time, it is already able to output a reasonable strategy and quickly decide about the next move. In our application, it allows dynamic balancing of the quality of the pursuit process and the resources needed for more crucial tasks performed by the agent, such as collision avoidance and motion control.

The third advantage is that the MCTS algorithms in general require minimal amount of domain specific knowledge to be encoded. They require only the implementations of the actions available to the agents, their effect on the game state and the description of the desired outcome of the game. No additional knowledge about specific opponent strategies or suggested plans for the agents is needed. On the other hand, if this knowledge is available, it is usually possible to incorporate such knowledge in order to increase efficiency of the algorithm.

The algorithm maintains a tree corresponding to a part of the search space close to the initial state. At the beginning, this tree consist of only the root node and then it gradually grows more in the areas that are perspective to at least one of the players. The main skeleton of any MCTS algorithm consists of four procedures that are repeatedly called in the following order:

- **Selection:**Starts in the root note and descends down the already constructed part of the tree based on the statistical information stored in the tree nodes.
- **Expansion:**After it reaches the leaf of the already constructed tree, it adds a small sub-tree rooted in the leaf to the maintained tree.
- **Simulation:**It chooses one of the leafs of the newly added tree and runs a of the play behind that point using random actions for each of the players and evaluate its result.

- **Backpropagation:**It returns back to the root of the tree and on the way, it updates statistics in all the nodes using the result of the simulation.

The main idea of the algorithm is to use earlier iterations to create statistics that allow guiding the later iterations to the portions of the search space that are more relevant for the game.

The output of the algorithm are the actions in the root node that lead to statistically most favorable results for the agents that runs the algorithm.

Goal-based game tree search

GB-GTS [48] is a search-based algorithm that incorporates procedural background knowledge into classical adversarial search in order to simplify searching in multi-agent games. A similar approach based on HTN has been used to create a competitive bridge playing program [69].

The procedural knowledge is represented in the form of so called *goals*, where each goal consist of three elements: (1) a set of initial conditions specifying the states in which the goal is applicable, (2) a set of conditions that define the world states in which the goal is satisfied, and (3) a procedure that for each state of the world returns the next action on the path towards the goal satisfaction, or marks the goal unsatisfiable. Instead of searching on the level of atomic actions, the GB-GTS plans on the level of high-level goals. This way only meaningful branches of the game tree would be evaluated, which substantially reduces the search space. On the other hand, the GB-GTS does not reduce the problem to the pure usage of high-level goals instead of atomic actions because the simulation of the future development in the world is still performed on the atomic level. Hence the GB-GTS is similar to a cut-off heuristic in a classical adversarial game tree.

The algorithm runs as follows. The search starts with all agents marked as idle. In any run of the main loop, if there is an idle agent, the algorithm creates a new branch of the search for each goal that is applicable for the agent in the current situation, assigns it the goal (the agent is not idle anymore) and continues to the next loop. If all agents have some goals assigned, GB-GTS simulates the future development of the game move after move, until at least one agent becomes idle again by satisfying a goal or detecting its goal is unsatisfiable in the given state. In such case, the main loop of the algorithm starts again. The algorithm iterates a pre-determined fixed number of moves to the predicted future (look-ahead), evaluates the reached position and the interim states that led to the position, and propagates the value back using a value back-up function.

Information Set Search

Information set search (ISS) [60, 61] is a heuristic method for playing imperfect information extensive form games that has been successful for example in Kriegspiel (imperfect information variant of Chess). However, it is a heuristic algorithm that does not provide any bounds on the quality of the produced solution, unless we have a perfect model of the

opponent's behavior. The main idea of the algorithm is substantial reduction of computational complexity by searching only through the information sets that belong to the player using the algorithm. The actions of the opponent are substituted by operations on the information sets that could possibly be the effect of the opponent's actions.

The algorithm is based on the same principles as the standard minimax algorithm used to play perfect information games like chess. It traverses the game tree by a depth-limited depth first search. The tree nodes are the information sets in the game. In the root (first layer) and every odd layer, the edges that lead to the following layer represent the actions that can be performed by the player. Note that in all states in one information set, the applicable actions of the player must be the same. The information set in the next layer represents all the states that can follow after applying the action in the parent information set. The edges going from the even layers of the tree do not represent actions, but possible observations. Each observation leads to a different information set.

When the algorithm reaches the pre-defined depth, it applies a static evaluation function and computes the utility value. The values of all the child nodes are then used to compute the value of a parent. In the action selection nodes, the value of the child after the action that maximizes the searching player's utility is used. In the observation selection moves, the situation is much more complex. The right value to select is based on the opponent model. Two basic opponent models are explored in literature.

Paranoid The paranoid opponent model assumes that the observation received is the least convenient one. This is consistent with the minimax assumption used in perfect information games. However, in imperfect information games, this expects the opponent to make the worst possible move even if he does not have enough information to determine it. The information of the searching player about what would be the move that would harm him the most is usually more accurate than the information available to the opponent.

Overconfident The overconfident assumption is the other extreme. As the opponent often does not have the information to determine, which of his moves lead to the worst possible observations, this assumption expects that the acquired observation is uniformly randomly selected from all possible observations. This approach performs better than the paranoid in games with strong information asymmetry.

Based on these opponent models, the value in the "observations" node would be the minimum (paranoid) of mean (overconfidence) of the values in the child nodes. The advantages, disadvantages and bounds of these two opponent models are further examined in [61].

Visibility-based pursuit evasion The information wet search has been successfully applied for generating the pursuers' behavior in visibility-based pursuit evasion domain [66]. The actions of the pursuers are all possible moves of the agents. The possible observations revealing the position of the evader on each position that became visible by the previous move of the pursuer or no observation. This "observation" leads to a state that represents

all possible locations of the evader that it could reach from its previous position and are not currently observable.

The paper presents several evaluation function in the leafs. Their experiments indicate that the most successful function for tracking is the one they called RLA. The function evaluates a leaf information set proportional to the size of the set of nodes that can be reached by the evader in less than d steps, but cannot be observed by any of the pursuers in this time.

3.2.4. Designed algorithms

In this section, we explain the details of the algorithms we have designed for creating behavior in the PEGs with heterogeneous teams. All the algorithms are based on (a combination of) the existing approaches introduced above.

MCTS for Fully Observable Pursuit-Evasion Game

The algorithm we use for this problem is a variant of MCTS with UCT [40].

Simultaneous Moves UCT was developed for alternating moves games. The advantage of these games is that the players always know what the next state of the world is after they apply an action. It does not hold in games with simultaneous moves. The state after applying an action depends on the action the other players chose to apply. If we want to apply MCTS to simultaneous moves games, we have two basic options.

Moves serialization: The moves can be reasoned about as if they occurred in sequence. As a result, the player that is considered to move second knows the move of the first player and can adapt its strategy accordingly. The order, in which the moves are considered, causes the algorithm with serialized moves to be either too optimistic or too pessimistic. This can lead to bad play in some cases, but in general, the approach has been shown efficient in several domains (e.g. [48]).

Simultaneous updating: The adversarial planning methods based on MCTS allow also direct consideration of the simultaneous moves. The definition of the game tree is modified so that a subset of players moves in one node of the tree, not a single player. In the selection phase, an action for each of the players is acquired in each tree node based on the statistics of the action's expected utility and the number of times the action was sampled. In the backpropagation phase, statistics for each of the actions are adapted separately. This approach avoids the issue with optimism/pessimism, but the concurrent adaptation of the strategies of multiple players based on the same samples may lead to suboptimal balanced strategies that can be exploited [68]. However, this method proved to be successful in CADIA player [27] and we adopt it in the current work.

Monte-Carlo Tree Search The implementation of the game state in the fully observable case consists only of a vector of physical positions of the agents. The UCT algorithm

creates a new root node for the observed state of the world and repeats the following four phases.

Selection: The selection phase starts in the root node and descends down the previously constructed portion of the game tree. It selects an action for each of the players in the current tree node based on the statistics in the nodes and the UCT formula:

$$action.expValue + C\sqrt{\frac{ln(node.nbSamples}{action.nbSamples}}$$

$expValue$ is the expected value of taking the action and $nbSamples$ is the number of previous samples going through the current node and selecting the action. The action that maximizes the value of the formula is selected and performed in order to identify the next node that will be set as the current node. At the end of this process, either the end of the game is reached or the selection selects a node that has not been previously visited.

Expansion: When a new leaf node is reached, the node is expanded. All the game states that can follow the current leaf node in one move are generated and appended to the tree. One of them is selected and the simulation phase is initiated. When the game is ended in this node, constant node with value 2 for the pursuers is created.

Simulation: Simulation tries to estimate the quality of the state reached at the end of the repeated selection process. We have designed two simulators for the perfect information version of the game.

The simpler *random* simulator selects completely random moves for all the players until the evader is caught, or a pre-specified horizon is reached. If the random simulation does not lead to capture of the evader, it is assumed to escape and the value of the simulation is 0 for both players. Otherwise $(1+1/t)$, where t is the round in which the evader was captures is returned. This motivated the pursuers to capture the evader as early as possible.

The canonical random simulation in MCTS can be substituted for so-called *heavy* simulations [23] that use domain-specific knowledge to produce simulation that are likely to occur in the game. The heavy simulations in our implementation prohibit for all agent staying in a node and going to the position where the agent was in the previous time step. Moreover, always when an agent arrives to a crossroad, it computes which of the notes it can move to is (a) the furthest node from the closest pursuer for the evader agents and (b) the closest node to the evader from a pursuer agent. Then with probability 0.5 it chooses the selected node and with probability 0.5 it chooses any random node. The simulation value in the case of capture is the same as for the random simulation. The simulation value i the case of reaching the simulation horizon of 30 steps without a capture is $(1/sumDistance)$, where the denominator is the sum of distances between the evader and the pursuers. In order to compute the heavy simulations efficiently, we use a pre-computed cache of distances between each pair of nodes in the graph.

Backpropagation: In the last phase, the result of the simulation is propagated back towards the root in the tree. The statistics of the actions and nodes on the path from the simulated node to the root are updated. The number of samples in each of the nodes

and selected actions is incremented and the expected values of the actions on the path are updated by the result of the simulation.

Search Tree Progression In order to be able to model the topology of the space precisely, we use a graph with high number of nodes. Hence, the distances between two neighboring nodes are quite small. The standard approach used with game tree search algorithms is to execute a new search after each move is applied. This would give us only very limited amount of time for reasoning about the next move. Moreover, the situation after a single move is generally very similar to the situation in the previous time step, which indicated that most of the reasoning could be reused.

In order to reuse the previous computations and use the available time as much as possible, we do not start a new search in new step. After the agents submit their moves to the simulation and start to move, they observe the actions being performed by the opponents (i.e. the directions, in which they move). They find child of the current root node of their MCTS trees that corresponds to the combination of the actions that are being performed. The replace the current root node by its child and run the simulation until the end of the move. At the end of the move, they select the actions to play based on the statistics an observe the nest opponents move. From the design of the UCT selection mechanism, large portion of the simulation in the current tree used the child that leads to the actual situation in the next move. These simulations create the tree for the next move that is further refined.

Team actions We have restricted the game to contain only one evader, but we assume to have multiple pursuers. While being two entities in the simulation, they have exactly the same goal and they can communicate to coordinate their actions. From the game theoretical perspective, they are single player with actions corresponding to the Cartesian product of the actions of individual actors – pursuers.

On the other hand, working explicitly with the whole set of possible combinations of actions of the individual players may be inefficient for several reasons.

- If the numbers of actions available to the pursuers in a node are $(b_1, \ldots, b_{|P|})$ and we keep the action statistics for each action from the Cartesian product, we have to keep $b_1 * b_2 * \cdots * b_{|P|}$ records. If we consider each agent an individual player, we just need to keep $b_1 + b_2 + \cdots + b_{|P|}$ records. This means not only a substantial reduction of the memory requirements of the algorithm, but also saving of computational time, as much smaller number of actions need to be considered in the selection process in each node. Considering the agents to be separate players improves the scalability of the approach.

- In many domains, such as ours, some moves by one agent may be superior to other moves no matter what the other pursuers do. For example, no pursuer should enter an empty death end. When all the pursuers are considered single player, the player

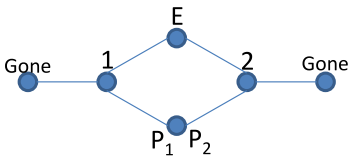


Figure 3.1.: Simultaneous UCT update pathology example.

has to learn that going to the dead end with the agent is bad for each and every combination of moves of the other pursuers. This leads to spending a lot of computational resources on parts of the tree that are unlikely to bring interesting good outcomes.

- Using the Cartesian product of actions complicates distribution of the algorithm to multiple agents. Agents have to reason about all the other agents and their actions and assume high level of coordination.

The alternative approach is considering all the teammates to be separate players with the exact same utility and simultaneous decision (as in case of the opponents).However, this approach also has drawbacks on the fundamental level as well as more practical issues with the specifics of the UCT algorithm. The problem with UCT is that the concurrent deterministic update of the actions' quality statistics of the individual agents may converge to suboptimal strategies. One example of such pathology is following.

Example 4. Assume that two players of the same team have both two actions in a node. We denote them $\{a_1^1, a_2^1\} and \{a_1^2, a_2^2\}$. The UCT selection rule gives the highest priority to the actions that has not been tried in the node yet. As a results, the first two samples will cause consecutive selection of (a_1^1, a_1^2) and (a_2^1, a_2^2). The agents from the same team, hence the simulations results to the same value for both the agents and hence their estimate of the expected value of the actions is the same. Any time the UCT visits this node, the number of samples and the returned values for the action on the same position for the two agents will be the same, hence the UCT formula will choose the actions on the same position for both the agents. The pairs of actions on different positions, such as (a_2^1, a_1^2), will never be examined. However, this action can be sub-optimal as shown in Figure 3.1. Each pursuer should go to a different node in order to capture the evader before it reaches the nodes marked *Gone*.

Goal-based extension The performance of the SSS techniques in many domains can be substantially improved using procedural knowledge heuristic about reasonable behaviors in the game. In our setting, we used a simplified version of GB-GTS for this purpose. We augmented the state of the world with the current goal for each agent. The goal in our case was a node in the graph where the agent wants to arrive using the shortest path. In the MC tree search algorithm, if an agent has a goal assigned, the only action considered for

the agent is moving to the next node on the shortest path to the goal node. If it reaches the node, it becomes idle. In that case, the next action for the agent is assigning any of the applicable goals (important crossroads near-by) in our case.

Just using the close crossroads helped significantly reduce the size of the search space. However, the GB-GTS method can be used to incorporate more sophisticated background knowledge. The goals can be pre-computed nodes with favorable topological properties for either the pursuer of the evader or hand-made smart heuristics. Because of limited amount of time and satisfiable results with simpler goals, we did not pursue this line of research further.

Monte-Carlo ISS for Visibility-based PEG

We have described the state of the art method for visibility-based tracking [66] in Section 3.2.3. It is based on the information set search and static evaluation function. There are two main problems we have to solve if we want to adapt it to PEGs with heterogeneous teams in realistic setting. The first problem is that we need a different *evaluation function*. The RLA heuristics works well for the tracking problem, but it is not usable for capturing the evader. The other problem is that the vanilla minimax-based ISS needs *variable time* to compute the next action. If an agent is in an area with higher number of crossroads, the amount of possible future courses of action within a horizon is larger and the computation time needed to traverse it longer. This effect is further strengthened by various pruning mechanisms such as alpha-beta. Unless the algorithm finishes, it cannot provide any solution.

This issue is critical in our application, because in practice, we have a strictly limited period of time for computation. The time is given by the speed of the agents and the time, when the opponent arrives to the next node of the graph. If we do not have the next move by the time, we cannot apply the next action, we might have missed the opportunity to catch the evader and we have to start over again. The whole tree (until the selected look-ahead depth) has to be traversed again in the next move.

This problem does not have a satisfactory solution within the classical game tree search approaches. The solution offered are either wasteful to computational resources of requires move ordering knowledge that might be impossible to acquire. The first solution could be to estimate the upper bound on the computation time needed and select the search depth accordingly. However, the amount of computation needed is exponential in the search depth. As a result, it is not likely that it will be possible to find a search depth that uses the whole period of time available for computation. The player would be most likely idle long before it has to decide about the next move. It a reliable upper bound cannot be computed, iterative deepening is an option, but it has the same problem. The last iteration will not be finished in time and it will most likely use most of the available time because of exponential increase of the required time with the search depth.

Both the problems with evaluation function and effective use of resources can be solved by MCTS approach. That is why in the following, we combine the ISS with MCTS. Thanks

to the anytime properties of MCTS, the confidence in the right solution increases in time and we can always get a good solution exactly at the end of the move. Moreover, thanks to reusing of the search tree between the nodes, we can transfer a lot of the search effort from previous time steps to the following. Even the problem with designing a good leaf node evaluation function is relaxed in MCTS. One of the main reasons why MCTS has been successful in the general game playing competition [27] is that (mostly) random simulations are often more efficient in estimating quality of a node than a static evaluation function.

Monte-Carlo ISS The proposed method is application of the MCTS to the same kind of tree that is built in the ISS. However, we do not limit it to a fixed depth and rather construct the tree gradually more intensively in the more promising parts. The state of the game is represented as a vector of positions of the agents of the searching player and a vector of the sets of the possible positions of the opponents.

Selection: The selection mechanism we use is the same as we applied in the game with full observability. We use UCT selection in the nodes where action is selected. In the nodes where the observation is selected, we use both the paranoid and overconfident model. The paranoid model is emulated by UCT selector with negative values of the nodes. The overconfident model corresponds to uniformly choosing a random observation.

Expansion: The expansion of the node in which an action is selected is straightforward. All the applicable actions are identified and applied to get the child nodes. In the case of the nodes with observations, the expanded children correspond to each of the positions where the opponent could appear and possible one information set where it stays unseen. There is one exception. We define a threshold on the maximal possible size of the set of possible positions of an unseen agent. When the critical size is reached, we assume that the agent stays in the region that does not grow any further until it is spotted again of the parts of the region are explored and the agent is not detected. The reason for that can be seen in the pathological behavior of the evader. If the evader does not see a pursuer for a long time, it always considers possible that it is handing in each unseen location. If we did not apply the threshold, the observation nodes would always produce the pursuer appearing in exactly at the next seen position in all possible escape directions. Both overconfident and paranoid algorithm would reach the conclusion that any further course of action is equally bad regardless on other seen pursuers that might be approaching the evader. Ignoring the long unseen pursuer consistently produces better behavior.

Simulation: In the simulation stage, we need to estimate the quality of an information set, not a single state as it was in the case of the fully observable game. For that, we use perfect information sampling in our algorithm. We create 5 random samples of full states that are part of the information state. We run the same biased simulation as in case of the full observability and return the mean of the results of these 5 simulations.

Choosing the samples randomly in simulation also corresponds to an overconfident assumption. A model of opponent behavior could be used to bias the selection of the stated for simulation. It the opponent has sufficient amount of information, the simulation results

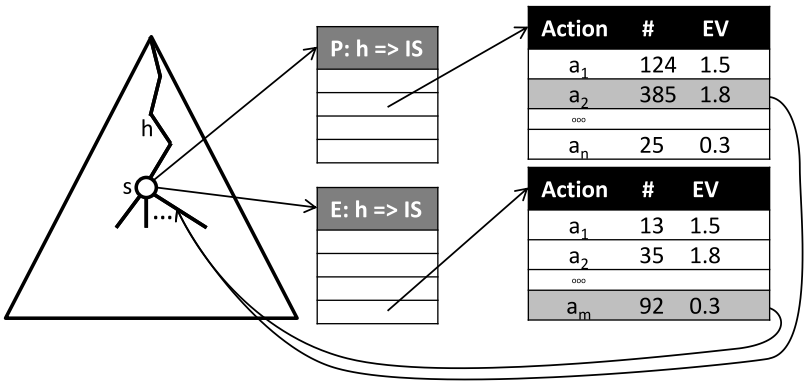


Figure 3.2.: UCT selection for partial information games. "#" stands for the number of samples and EV is the expected value.

for states that are unfavorable are more likely to be representative for estimating the value of the node.

Backpropagation: The backpropagation phase is the same as in the previous cases. After the simulation, the numbers of times the actions have been used as well as the expected value they led to are updated in all the information sets that have been traversed in the current sample.

Novel Approach Suggestion

The need for opponent model or at least the paranoid/overconfident assumption in ISS results from the fact that the search is performed only on the information sets of single player. The information about the other player has to be provided externally. In order to overcome this problem, we have proposed an alternative solution. This section introduces the main concepts of the algorithm, which has not been implemented within the limited timespan of this project.

The algorithm can be perceived form two perspectives. The first perspective is generalization of the UCT algorithm for the simultaneous moves as shown in the precious section. The other perspective is simultaneous learning of the best response strategy to the actual strategy if the opponents. First we explain the algorithm for the case that we know exactly what the initial state of the game is and how this limitation can be dropped.

The main idea of the algorithm repeatedly traverse the game tree as in UCT, but to use the current information of individual agents instead of the exact state of the world in the selection and backpropagation phases. In order to do that, we need a data structure that will be able to quickly return an information set corresponding to any sequence of moves by the agents. With each of the information sets, we store the actions available in the information set and we with all actions, their expected value and the number of times the action has been tried-out.

Selection The selection process starts from the initial state of the game, which is assumed to be known to both players. During the whole selection phase, we are still working with a complete state of the game, regardless what the agents know about it. The selection process is depicted in Figure 3.2. In any node, the full state of the world (s) including the full history of the actions of all the agents (h) is transformed to the information sets of each of the players. The information set corresponds to the information the players would have if they played h in the game and update their believes about the state of the world based on the observations they obtain. The pursuer (P) knows the exact position of its agents, but it might be uncertain about the position of the evader, if it cannot see it in state s.

With each information set that has been previously encountered in the search, the agent stores the following information. (1) All the actions that are available in the information set. For each action, the statistics about (2) how many times was each of the actions used in the information set and (3) the mean value that was gained by the player for playing the action. Based on this information, each agent can use the UCT formula to select the next action it should explore in order to optimally balance the exploration and exploitation of the information obtained in previous samples.

The following node in the selection process is uniquely identified by the combination of actions each of the agents selects. The selection process continues in the same way in the following nodes, until the combination of actions in a node leads to constant leaf node (i.e. the game ends in the node) or to a situation s_f, for which at least one of the players does not have a corresponding information set in the table of already encountered information sets.

Expansion The expansion means adding the information sets corresponding to the state s_f to the table of encountered information sets for all the players, which do not already have it there. All the actions available to the player in the information set must be generated and the corresponding data structure initiated.

Simulation The simulation phase is exactly the same as in the full observability version of the game. It starts from the full game state that is available all the time in the algorithm. In that state, all the agents take random moves until the evader is captured, or a predefined horizon of moves is reached. The horizon in our experiments is set to 15 moves. If it is reached without capturing the evader, the value returned by the simulation is zero for all the players. If the evader is captured in move t of the simulation, the value returned is $(1 + 1/t)$ for the pursuers and the same value with the negative sign for the evader.

Backpropagation The backpropagation phase is also similar to the full observability case. After the simulation, the numbers of times the actions have been used as well as the expected value they led to are updated in all the information sets that have been traversed in the current sample.

Uncertain root node The algorithm described above always starts from a known game state in the root node. While this assumption can be used in academic partial information games, such as Kriegspiel, it is not realistic in the pursuit evasion game. We might know that the evader is located somewhere on the map, but if it is not visible in the root note of the game, the root represents an information set that consists of a large number of possible game states. Even if one player knows the positions of its agents, it is not known to the other player. In general, only the information that is **common knowledge** in the game can be used for unbiased pruning of the search.

We assume that the only common knowledge at the beginning of the pursuit evasion game with partial information is that all the agents are on the map and in case that the initial state is not unique, both players know that they cannot see each other. The game states consistent with the common knowledge for the case of two pursuers and one evader are following.

$$R = \{(pos(p_1), pos(p_2), pos(e)) \in N{\times}N{\times}N : pos(e) \notin v(p_1, pos(p_1)) \;\&\; pos(e) \notin v(p_2, pos(p_2))\}$$

With limited visibility range, the size of the set is almost as large as the number of all possible states of the game. If this set is really too large, just a sample of a limited size is selected from the set R.

The algorithm then runs the same way as before, but the initial position in the rood is selected uniformly from (the sample of) the set R. After certain time, the best action from the information set representing the real knowledge of the agent in the current state of the world is performed.

Information update After the game progresses by another move, new common knowledge is generated. The agents either see each other or not. In the first case, the pursuer knows the exact state of the game, but the evader typically learns only the position of one of the pursuers. In the second case, both players learn that they do not see each other.

Even if the agents do not see each other, some of the positions of the opponent, which are apparently bad, become less likely than others. The uniform distribution over the states we used in the root node to represent complete lack of knowledge about the opponent's position is already not precise. The opponent's position is the outcome of its strategy and not some random effect of the environment. Therefore, creating a probability distribution over the opponent's possible positions is impossible without additional assumptions. As usual in game theory, we are trying to create a method that limits the assumptions on the opponent's behavior only to its rationality. We do not know what the best moves are for the opponent, so we use the information from the search.

One option to use the information can be creating a new root node and sample a new set R based on the most likely moves in each of the roots in the current set. This approach is difficult, because it is not clear how to sample the set to keep the size of R constant and preserve both the uniform distribution of possible starting positions and the high chance of taking good moves by the agents.

That is why as the first step to a good resampling strategy, we suggest delaying the resampling to the special situations, in which the uncertainty about the opponent's positions is quite low. In general setting we preserve the root of the search three in past and each Monte Carlo simulation starts in the root, the selection process goes through one of the information sets one of the players may assume to be possible in the current state and continue in to the future. When the players obtain new information, all the information sets in the current time step and in the past are checked and removed is they are inconsistent with the observations. Note, that the actions in the predecessors of the information sets that lead to the removed sets are removed as well. Moreover, the contributions of the actions to the number of samples and expected value of the whole information set are removed as well.

The following MCTS samples that start in the root of the tree (in the past) arrive to a possible current state of the world with the probability proportional to the current estimate of the optimal strategies, all the observations, and the (uniform) distribution of the possible start states in the root node of the search.

Discussion The main advantage of the proposed approach is that it is not biased to paranoia or overconfidence, as most of the state of the art techniques. It reasons for both players equivalently, thanks to the requirement of the symmetric knowledge.

The main disadvantage is naturally computational complexity of the approach. It may have to run for a long time in order to produced reasonably good strategy. The analysis of the balance between the combinatorial explosions caused by all the combinations of possible moves and the tree reductions possible thanks to the observation by the agents is part of the future research.

3.3. Evaluation and experiments

With the exception of the algorithm described in Section 3.2.4, we have implemented all the algorithm described above within the multi-agent simulation platform. We start the evaluation of the algorithms by the simpler case with full observability and then examine the generalization. We focus on analysis of

- suitability of the basic principles of the proposed algorithms with respect to the real world requirements,
- scalability of the proposed algorithms to problems of realistic size,
- and basic parameter choises suitable for our setting.

Figure 3.3.: An example situation in the game with full observability. The green dots are the pursuers and the red dot is the evader.

3.3.1. Fully observable setting

Minimax vs. MCTS

In the first set of experiments, we evaluated our assumption that the minimax-based algorithms will have serious problems assuring stable performance in limited time. We run the minimax algorithm with a simple evaluation function that returns the sum of distances between the evader and the pursuers or -1 in the case of capturing the evader. The example problem includes single evader and two pursuers. The screenshot of the schematic view of the scenario is in Figure 3.3.

Without the simple goal-based extension that removed the decisions on the interim unimportant nodes in the graph, neither of the approaches performed well. The fine discretization needed to provide exact model of the scenario topology caused huge state space. With the goal-based approach, the minimax algorithm was able to search 10 moves ahead. This look-ahead was generally sufficient for capturing the evader in our domain with the simple evaluation function. However, the deterministic nature of the algorithm caused that it often also got stuck in a cyclic situation that repeated forever without a successful capture.

As we expected, the main problem with this approach was the variable time it needed for computation. The histogram of the computation times needed for a sample initial setting is shown in Figure 3.4.

Most of the computations were performed quickly. However, the pruning was sometimes

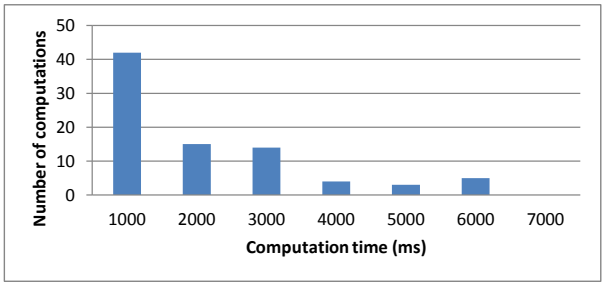


Figure 3.4.: The histogram of computational times (ms) needed to compute solution with the minimax algorithm and alpha-beta pruning.

not efficient and more complex situations caused the algorithm to run even 7 seconds. This uncertainty could be lowered by fine-tuned move ordering and situation complexity assessment, but it cannot be completely removed. Sometimes, the algorithm would not produce any solution in time and the agents would have to move heuristically.

The experiments with MCTS algorithm are more promising. The first advantage is that it is an anytime algorithm. Hence, we can limit the computation time to any time period we need. In our experiments, we used 1 second. Even with this strict limitation, the algorithm performs reasonably well. If both agents use MCTS to produce their behavior, the pursuers are able to eventually capture the evader in any starting configuration of the simulation. The current implementation with the same procedural knowledge heuristic as we used for the minimax algorithm and with the biased simulation described in Section 3.2.4 was able to perform approximately 10 000 simulations per second. However, with reusing of the simulations from previous moves and approaching end of the game, the decisions were eventually made with over million simulations performed through the root of the tree.

Based on these findings, we have determined that MCTS is a more suitable algorithm in our setting.

Scalability with additional pursuers

The standard way to reason about games among relatively small teams of agents' controlled by a small number of players is to consider all combinations of actions of the agents to be an action of the player. However, this approach does not scale well with increasing number of agents. The number of actions of each of the players is exponential in the size of his

team.

In MCTS algorithms stores statistics about each player's action. With exponential number of actions, the memory complexity as well as computational complexity of processing single tree node becomes exponential in the size of the team. An alternative approach we explored is to consider the team of agents to be separate (simultaneously deciding) players within the MCTS algorithm. These players share the same utility function and implicitly coordinate based on the results of the simulation. After sufficient number of trials, each team member select the best action most often and the other team members can adapt their strategy to achieve good results. The simultaneous adaptation of agents' strategies can be guaranteed to reach the Nash equilibrium [20]. A good update strategy could lead to reasonable coordinated play even without the need to reason explicitly about all possible combinations of the actions of the agents.

In order to examine this hypothesis, we have performed the following experiments.

Experiment setting All the experiments reported in this section were performed on a regular 10×10 grid graph. Agents can move in at most 4 directions, or they can stay at the same node. One evader and two pursuers start on a random location. Two instances of the UCT algorithm are run. The first is run by the evader agent and the second by one of the pursuers. This pursuer decides about the actions of all its teammates, sends them a message and they just perform it without any reasoning. We have used a fixed evader setting for this experiment. The evader used

- the presented Monte Carlo search with UCT parameter set to 3
- considered each evaders actions separately (not their product)
- used 1 second of CPU time to come up with a next move.

Results The first experiment validates that the described UCT based Monte Carlo method is capable of producing rational behavior of the pursuers within reasonably short computational time.

We varied the parameters of the pursuer player in two basic dimensions. First, we tried to give it various time intervals for reasoning about the next move. Second, we compared the quality of its play in case of using the Cartesian product of the actions and two separate players with their own actions. In the product case, the UCT parameter was set to 3. In the other case, it was set to 3 for one player and 4 for the other player in order to avoid the issues with deterministic synchronized action quality exploration.

The results of the experiment are summarized in Figure 3.5(a). First of all, we can see that the agents were performing reasonable strategies. Catching an evader on a grid generally requires pushing it to a corner and intercepting its attempt to flee from there. If it manages to flee from the corner (which cannot be prevented for 100% by two pursuers), it can run to the opposite corner of the grid, which buys it approximately 20 moves, in which it cannot be captured. With only 1 second of reasoning, the suggested methods were

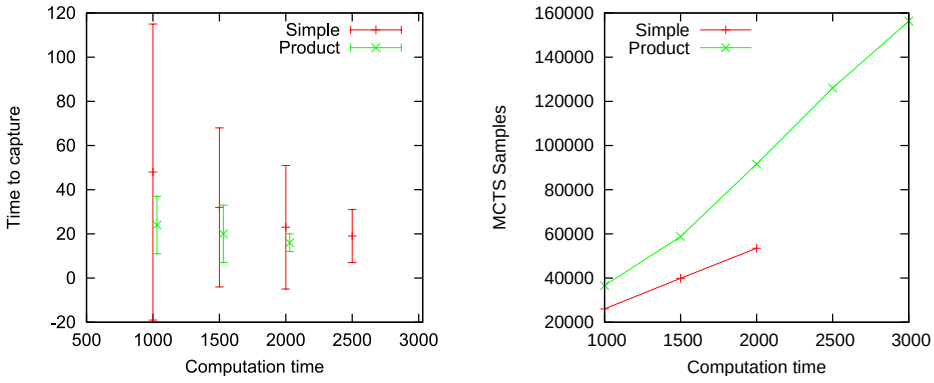


Figure 3.5.: (a)The mean number of moves needed in order to capture the evader with two pursuers on 10x10 grid graph. (b) The mean number of samples made by each algorithm in the given time.

able to catch the evader on average in less than 50 moves. Naturally, the actions of the agents also seemed reasonable in the simulation.

The second result of this experiment is the fact, that using cross product of the actions in case of two pursuers is more efficient the separate consideration of the agents. The number of moves the agents needed to catch the evader was in average smaller in the earlier case. Moreover, the standard deviation of the number of moves needed was substantially smaller for the cross product setting. Even though the selection process has higher computational complexity, it allows more precise guidance of the search and pays-off at the end.

This claim is supported in Figure 3.5(b), which shows the number of Monte Carlo samples individual methods managed to make in the given time. Even though the computation in each tree node is more complex, the better guidance of the search assured that the samples were ended sooner and hence, more samples could be made.

The last result that can be seen on these graphs is confirmation of the anytime property. The more time the agents used for the reasoning, the shorter was the time needed to catch the evader and the smaller was the deviation.

Based on these experiments, we can see that using simple actions with the basic UCT formula is not beneficial. However, we assume that it is caused by the described pathology and the situation can be different a different selection rule.

3.3.2. Partially observable setting

Based on the experience with the scenario with full observation, we could make several design decisions in advance. We focus on MCTS that is more suitable in this setting. As we aim to use the very successful UCT selection method, we further assume the actions of the player to be reasoned about as all the combination of the actions of the agents in her team.

Based on the MCTS IIS algorithm described in Section 3.2.4, we were able to create a proof of concept players for both the pursuer and the evader agents in a game with the team of pursuers composed of three units. Two ground units that are able to capture the evader and one observer UAV that provides information. Only one evader is present in the scenario.

The performance of the players is far from perfect. We believe that the main reason of the limited performance is that the GB-GTS method be used add procedural knowledge heuristic to the search algorithms in the full information case cannot be directly applied for ISS. Hence, the algorithm currently considers all possible sequences of movements of the actions.

However, in limited situation, they can find interesting solutions within a short period of time. The proof of concept implementation indicates that the chosen methods can be successfully used for the problem solved in this chapter in its full width. Achieving this even without any procedural knowledge heuristic indicates that we can expect very good play once the prune out all the clearly inferior courses of action. A good solution might be a combination of the agent oriented GB-GTS and the HTN-based approach of Smith et al. [69], which can deal with the imperfect information present in bridge.

4. Multi-agent re-planning and plan repair

4.1. Summary of the workpackage

The following chapter fleshes out the results of our research in the context of interaction of multi-agent and multi-robot systems with dynamic environments, such as the urban warfare missions are. The workpackage tackled two main issues. Firstly, we studied interactions of agent and multi-agent systems with dynamic environments in which actions of agents embodied in dynamic environments can fail. Secondly, we tried to tackle the problem of managing such interactions by automated planning approach. In particular, in the face of plan failure, we propose to attempt to repair the plan by the team of agents, instead of re-planning from scratch.

Our study of interactions agents with their environments, we followed a rigorous formal approach and proposed a formalism allowing us to finally classify various combinations of agent-vs.-environment systems according to estimation of plan execution success rate, resp. confidence level. In the core of the framework lies modelling of environments in terms of stochastic Markov Decision Process in which agent's actions do not automatically succeed, but rather fail with some probability. On top of this formal model, we propose a novel formal temporal modal logic called pDCTL* which allows us to exactly capture the success of execution of agent programs attempting to satisfy a given goal, an individual mission. IN result, the formalism allows us to formulate the notion of a *planning horizon*, i.e., timeframe for which planning still works reasonably well. That is, the plan will succeed with some sufficient probability. In the future, we plan to extend this line of research further to multi-agent case, i.e., investigation of interactions of cooperating agents working in teams towards satisfaction of some joint goal, or a mission.

The second, and in fact the main pragmatic topic of this chapter deals with the problem of plan repair. In particular, we recognize that classical-style planning is nowadays one of the most used techniques for automation of activities of intelligent agents, however we also observe that such plans are not robust w.r.t. unexpected events occurring in dynamic environments. The standard solution, in such cases, is to simply re-plan the agent's behavior from scratch and continue its actions according to the new plan. In multi-agent case, when decentralised planning method is used, this leads to significant communication among the multi-agent team members. In situations where communication is costly, or undesired (e.g., it could reveal the position of the agent to an adversary), there is a ample interest to reduce the communication to a bare minimum, even though the price for it might be prolonged non-optimal plans and/or imprecise plans.

We propose four algorithms solving the problem of multi-agent plan repair. The algo-

rithms differ in the level of complexity of the plan repair process, analytical properties, such as teh communicartion complexity, as well as in the quality and efficiency of the resulting plans. We also evaluated and compared the implemented algorithms with other state of the art multi-agent planners and the optimal output plans in synthetic domain of box movers. The domain is rich enough to facilitate modelling and study of properties of the proposed algorithms in a domain, where teamwork is essential for the multi-agent system's success.

Finally, we employed one of the plan repair algorithms in an integrated scenario of team mission execution in urban warfare setting. We model a multi-robot team consisting of several micro-UASs (modelled as AESIR Vidar single-rotor VTOLs, about 1m in diameter, 10kg payload capacity) and medium-range unmanned helicopters (modelled as Saab Skeldar V-200 VTOL, 40 kg payload capacity) supporting a human squad on their VIP evacuation mission in an urban terrain. While the secondary task of the robot team is to perform their own information collection mission in the town, their primary task is to provide situational awareness and reconnaissance for the human squad. In result, while the robotic team has its own plan to carry out their joint planned mission, they are not aware of all the movements of the human squad and have to repair their original plans according to the changes invoked by the activities of the squad. The results of this qualitative evaluation show a lot of promise w.r.t. the future improvements and in-field applications of the plan repair algorithms.

4.2. Technology description

4.2.1. Interaction of agents with dynamic environments

In dynamic environments agent's actions can fail. This might be either directly due to failing effectors of the agent, or indirectly due to incompleteness of information. The latter situation may arise either because precise and complete information is inherently impossible (think of a robot dealing with complex physical phenomena, such as the weather), or technically infeasible, resp. undesirable in the given application domain (e.g., high space complexity of the environment representation, or high rate of environment change in relation to the speed of the agent's deliberation).

Construction of agents in the face of unexpected failures is difficult. As a reaction to difficulties with classical planning in dynamic environments, the paradigm of agent-oriented programming (AOP) based on reactive planning (e.g., [16]) became one of the state-of-the-art techniques for construction of intelligent agents. Even though the motivation behind AOP is rooted in the idea that more reactive style of deliberation is more appropriate for agent's interaction with a dynamic environment, the precise characteristics of the relationship have not yet been deeply studied.

In this subsection we precisely investigate *the relationships between capabilities and control mechanisms of an agent, its design objectives, the goal, and characteristics of the environment it is embodied in.* As a conceptual framework supporting the discourse, in Section 4.2.1 we introduce a series of agent behaviour performance measures. We start

from relating the set of agent's generic capabilities to an environment in which actions can fail. Subsequently, we introduce agent programs constructed from the basic actions and based on their performance in the environment, we provide a taxonomy of their mutual matchings. Finally, in Section 4.2.1 we consider the relationship between a program, its design objective, a goal it is aimed to fulfill, and characteristics of the environment. We study the relationships of the triad using *Probabilistic Dynamic CTL** logic (pDCTL*), a novel temporal logic framework facilitating reasoning about agent system specifications and actual programs aimed at realising it. We observe, that imperfect performance of agent programs situated in dynamic environments w.r.t. their goals can in fact be caused by two distinct phenomena. While the environment by its dynamics can make the program fail, it can be also the program itself, which is not implemented perfectly w.r.t. the goal. We conclude the discourse of the subsection by discussion in Section 4.2.1 of how program construction influences its performance w.r.t. the design specification, and finally, in Section 4.2.1 we formally relate the two causes of program imperfection. In particular, we introduce an *impact metric*, a measure indicating how much the imperfections in program implementation influence the chances for reaching the goal.

The following section is based on our research paper [58].

Agents, actions, environments

We focus on actions of an agent acting in an environment. The agent's actions can change the state of the environment, possibly in a probabilistic way. Other agents, if present in the system, are not relevant at this stage and are assumed to be appropriately modelled as a part of the environment. Due to this characteristics, we will model environments as Markov decision processes (MDP's) [11]. By this, we implicitly assume that agents can always exactly recognise the current state of the environment. We leave analysis of the more general case (partial observability) for future work.

Basic notions

Definition 5 (**Environment**). *Environment* E is modelled as a Markov decision process $(\mathcal{S}, \mathcal{E}, P)$ where $\mathcal{S}$ is a set of states the environment can be in, $\mathcal{E}$ is a set of events which can happen in E and $P : \mathcal{S} \times \mathcal{E} \times \mathcal{S} \to [0, 1]$ is a probabilistic transition function with transitions labelled by events. That is, $P(s, e, s')$ defines the probability that, upon occurrence of the event e in the state s, the next state of the environment will be s'. We will adopt the convention that if e is not enabled in s then $P(s, e, s') = 0$ for all $s' \in \mathcal{S}$.

Furthermore, we assume that some propositional language $\mathcal{L}$ is available to characterise properties of states of E. $\mathcal{L}$ comes with a standard satisfaction relation $\models$, with $E, s \models \phi$ meaning that the formula $\phi \in \mathcal{L}$ holds in the state s of the environment E.

An agent consists of a template that specifies how it *can* act (i.e., provides a set of basic operations available to the agent) and a program that prescribes how it *will* act (e.g., by defining its deliberation mechanism). An agent must match its environment in the sense

that its actions must be events in the environment. Moreover, each action is annotated with a specification of its envisaged effects. In an ideal environment, the annotation should hold after the action has been executed.

Definition 6 (**Agent template**)**.** Let $E = (\mathcal{S}, \mathcal{E}, P)$ be an environment. An *agent template* $(Act, \mathfrak{Ann})$ situated in E specifies the set of basic actions (capabilities) $Act \subseteq \mathcal{E}$ that the agent can execute in E, together with the function $\mathfrak{Ann} : Act \to \mathcal{L}$ that annotates the agent's actions by formal descriptions of their expected effects.

Definition 7 (**Macro actions and traces**)**.** A *macro action* is a possibly infinite sequence of actions $\rho = a_1, \ldots, a_n, \ldots$ with $a_i \in Act$.

An execution trace λ of macro action ρ rooted in a state $s_0 \in \mathcal{S}$ of environment E is a (finite or infinite) sequence of labelled transitions $s_0 \stackrel{a_1}{\to} s_1 \stackrel{a_2}{\to} \cdots \stackrel{a_n}{\to} s_n \stackrel{a_{n+1}}{\to} \cdots$ such that $s_i \in \mathcal{S}$ and $P(s_{i-1}, a_i, s_i) > 0$ for all i. By $\lambda[i] = s_i$, we denote the ith state on λ, and $\lambda[i..j] = s_i \stackrel{a_{i+1}}{\to} \ldots \stackrel{a_j}{\to} s_j$ denotes the "cutout" from λ from position i to j. The ith prefix and suffix of λ are defined by $\lambda[0..i]$ and $\lambda[i..\infty]$, respectively. $|\lambda|$ denotes the number of transitions in λ. In the case λ is infinite, we write $|\lambda| = \infty$. For a given trace $\lambda = s_0 \stackrel{a_1}{\to} s_1 \stackrel{a_2}{\to} \cdots \stackrel{a_n}{\to} s_n \stackrel{a_{n+1}}{\to} \cdots$, we denote $\rho(\lambda) = a_1, a_2, \ldots, a_n, \ldots$ the macro action, execution of which resulted in λ.

Definition 8 (**Situated agent**)**.** An *agent* is represented by a tuple $A = (Act, \mathfrak{Ann}, \tau)$ where $(Act, \mathfrak{Ann})$ is an agent template situated in an environment E, and τ is an *agent program* over Act. The set $\mathcal{P}$ of well-formed programs is defined as follows:

- a is a program for every $a \in Act$;
- If τ, τ' are programs, then also $\tau \cup \tau'$, $\tau; \tau'$ and τ^* are programs denoting nondeterministic choice, sequential composition and unbounded iteration respectively;
- Additionally, fixed iteration τ^n is defined recursively as $\tau^1 = \tau$ and $\tau^{n+1} = \tau^n; \tau$.

The semantics of an agent program is defined in terms of the set of traces $\mathcal{T}_E(\tau, s_0)$ it induces in the environment E, rooted in some initial state s_0. Formally, $\mathcal{T}_E(\tau, s_0)$ is defined inductively

- $\mathcal{T}_E(\langle a \rangle, s_0) = \{s_0 \stackrel{a}{\to} s_1 \mid P(s_0, a, s_1) > 0\}$,
- $\mathcal{T}_E(\tau; \tau', s_0) = \{\lambda \mid$ there exists $i : \lambda[0..i] \in \mathcal{T}_E(\tau, s_0)$ and $\lambda[i..\infty] \in \mathcal{T}_E(\tau', \lambda[i])\}$,
- $\mathcal{T}_E(\tau \cup \tau', s_0) = \mathcal{T}_E(\tau, s_0) \cup \mathcal{T}_E(\tau', s_0)$,
- $\mathcal{T}_E(\tau^*, s_0) = \{\lambda \mid \lambda[0] = s_0, k_0 = 0$ and there exist $k_1, k_2, \ldots$ s.t., $\lambda[k_i, k_{i+1}] \in \mathcal{T}_E(\tau, \lambda[k_i])\}$.

Additionally, $\mathcal{T}_E(\tau)$ denotes the set of all traces induced by τ in states of E, i.e., $\mathcal{T}_E(\tau) = \bigcup_{s \in \mathcal{S}} \mathcal{T}_E(\tau, s)$. Moreover, given a set of sequences X, we will use $Fin(X)$ to denote the set of finite prefixes of the sequences from X. For example, $Fin(\mathcal{T}_E(\tau))$ is the set of finite histories that can occur during execution of τ in E.

Note, that the generic form of agent programs we define in Definition 8, serves only for exposition of ideas in this paper. In fact, any succinct way of encoding of agent's behaviour in terms of enabled execution traces (intended system evolutions) would serve equally well. This broader understanding of agent behaviours naturally includes various planning mechanisms, as well as most state-of-the-art AOP languages.

Hereafter, unless specifically stated otherwise, we will assume an agent $A = (Act, \mathfrak{Ann}, \tau)$ situated in an environment $E = (\mathcal{S}, \mathcal{E}, P)$, s.t. the annotation function $\mathfrak{Ann}$ is expressed in some propositional language $\mathcal{L}$ with a satisfaction relation $\models$ that interprets formulae of $\mathcal{L}$ in states of E.

Probabilistic execution of actions and programs Annotations play dual role in the specification of agents' capabilities. On one hand, they put forward the envisaged outcome of an action in an ideal environment. On the other, they allow to define the notion of successful execution of the action (and, dually, the notion of execution failure).

Definition 9 (**Success and failure of actions**)**.** A transition $s \xrightarrow{a} s'$ is a *successful execution* of a if $s' \models \mathfrak{Ann}(a)$, otherwise it is a *failure*.

Given a state $s \in \mathcal{S}$, the probability of successful execution of action $a \in Act$ in s is defined as

$$P_{ok}(a, s) = \sum_{s' \models \mathfrak{Ann}(a)} P(s, a, s')$$

Straightforwardly, the probability of failure of a in s is $P_{fail}(a, s) = 1 - P_{ok}(a, s)$.

A macro action is successful if and only if all its components succeed along the trace.

Definition 10 (**Success of macro actions**)**.** Let $\rho = a_1, \ldots, a_n$ be a macro action. The probability of successful execution is extended to macro actions as follows:

$$P_{ok}(\langle a_1, \ldots, a_n \rangle, s_0) = \sum_{\substack{\lambda = s_o \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n \\ s_i \models \mathfrak{Ann}(a_i)}} \prod_{i=1}^{n} P(s_{i-1}, a_i, s_i).$$

Note that P_{ok} is a probability distribution determining execution success of sequences of actions in states of E.

Now we are ready to introduce probability-based measures of successful execution of actions and programs in an environment.

Definition 11 (**Execution success measures: actions**)**.** Let ρ be a (macro) action. The minimal certainty of successful execution of ρ in the environment E is defined as:

$$P^{-}_{ok}(\rho, E) = \min_{s \in \mathcal{S}} P_{ok}(\rho, s).$$

Similarly, we define the maximal certainty of successful execution of ρ in E:

$$P^{+}_{ok}(\rho, E) = \max_{s \in \mathcal{S}} P_{ok}(\rho, s).$$

Definition 12 (**Execution success measures: programs**)**.** Let $\tau \in \mathcal{P}$ be an agent program over the actions of some agent A. The minimal certainty of successful execution of τ w.r.t. a state s in the environment E is the minimal probability of execution success among the individual traces induced by the program rooted in s:

$$P^{-}_{ok}(\tau, s) = \min_{\lambda \in \mathcal{T}_E(\tau,s)} P^{-}_{ok}(\rho(\lambda), s)$$

Moreover, the minimal certainty of execution success for program τ in the environment E is defined as

$$P^{-}_{ok}(\tau, E) = \min_{s \in \mathcal{S}} P^{-}_{ok}(\tau, s).$$

The notions of maximal certainty of successful execution $P^{+}_{ok}(\tau, s)$ and $P^{+}_{ok}(\tau, E)$ for an agent program are defined analogically.

Classification of environments w.r.t agent templates In the first approach, we gauge how well the agent's basic capabilities (e.g., effectors of a robot) match the environment in which the agent is to be situated.

Definition 13 (**Taxonomy of environments w.r.t. success of actions**)**.** Given an agent template $(Act, \mathfrak{Ann})$ in an environment E, we can distinguish several relations between the two, depending on how well the template specification meets the dynamics of the environment. Formally, we say that the matching between $(Act, \mathfrak{Ann})$ and E is:

ideal iff $P^{-}_{ok}(a, E) = 1$ for all $a \in Act$,

consistent iff $P^{-}_{ok}(a, E) > 0$ for all $a \in Act$,

strictly consistent iff it is consistent and $P^{+}_{ok}(a, E) < 1$ for all $a \in Act$,

inconsistent otherwise.

We argue, that the most interesting cases of relationship between an agent and an environment is when from the agent's perspective the environment is *strictly consistent*, or at least contains a significant *strictly consistent fragment*. For a mobile robot, a controlled indoor environment is usually ideal. Examples of (strictly) consistent agent-environment systems include e.g., outdoor robots operating in rain, snow, on icy, or sandy surfaces, or on gravel roads. From a robot's perspective, an environment is also strictly consistent when it features only imprecise sensors and/or unreliable effectors. For the agent's controller, such situation is indistinguishable from the case when the failures are truly exogenous. A controlled indoor environment is ideal for the robot. A standard outdoor environment is manageable only, and a crowded outdoor environment could even turn hopeless. This classification can give a good formal hint on limitations of usability of the robot.

The view promoted in the Definition 13 is rather pessimistic. Especially for agent templates based on large libraries of actions, even if one action does not match the environment,

the whole agent template is seen as a mismatch. In a more complex agent program, designer might want to take into account also some failures of actions e.g., by encoding various contingencies, or refining the conditions under which actions and plans can be executed. The following refinement of the environment vs. agent classification takes the agent program as a basis for the matching.

Definition 14 (**Environments vs. agents**)**.** Given an agent $A = (Act, \mathfrak{Ann}, \tau)$ in an environment E, we say that the matching between A and E is:

ideal iff $P^{-}_{ok}(\tau, E) = 1$,

consistent iff $P^{-}_{ok}(\tau, E) > 0$,

strictly consistent iff it is consistent and $P^{+}_{ok}(\tau, E) < 1$,

inconsistent otherwise.

Still, this notion of matching between agents and environments is not perfect. In particular, if τ includes unbounded iteration (an infinite deliberation cycle in an event-driven agent is a good example) then it is easy to see that all the environments are either ideal or inconsistent w.r.t. such an agent. This is because the notion of success relates executions to the annotations of all the actions that are going to be performed, regardless of their relevance to a larger context. In many scenarios, such context is provided by an objective, a goal, that the agent is supposed to pursue. We will formalise the concept and discuss the consequences in the next section.

Reasoning about temporal goals

In the previous section, we showed how "perfectness" of an environment can be classified with respect to the agent's repository of actions and/or its main algorithm. However, the classification is quite rough in the sense that it depends on *all* the actions behaving as expected (according to the provided annotations). An alternative is to consider a particular objective, and to measure how the environment reacts in the context of the objective.

Objectives that refer to execution patterns (like *achievement* of a property sometime in the future, or *maintenance* of a safety condition in all future states) can be conveniently specified in *linear time logic*. LTL [65] enables reasoning about properties of execution traces by means of temporal operators $\bigcirc$ (in the next moment) and $\mathcal{U}$ (strong until). To facilitate reasoning about finite sequences of actions and compositions thereof, we will use a version of LTL that includes the "chop" operator $\mathcal{C}$ [67].

Temporal goals: LTL Formally, the version of LTL used in this paper is defined as follows.

Definition 15 (**LTL**)**.** The syntax of LTL is given by the following grammar:

$$\phi ::= \mathsf{p} \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \mid \phi \mathcal{C} \phi.$$

Other Boolean operators (disjunction $\vee$, material implication $\rightarrow$, etc.) are defined in the usual way. The semantics is defined through the clauses below (where E is an environment and λ is an execution trace of some macro action of an agent A situated in E):

$E, \lambda \models \mathsf{p}$ iff $E, \lambda[0] \models \mathsf{p}$,

$E, \lambda \models \neg\phi$ iff $E, \lambda \not\models \phi$,

$E, \lambda \models \phi \wedge \phi'$ iff $E, \lambda \models \phi$ and $E, \lambda \models \phi'$,

$E, \lambda \models \bigcirc \phi$ iff $E, \lambda[1..\infty] \models \phi$,

$E, \lambda \models \phi \mathcal{U} \phi'$ iff there exists $i \geq 0$, such that $E, \lambda[i..\infty] \models \phi'$, and $E, \lambda[j..\infty] \models \phi$ for every $0 \leq j < i$,

$E, \lambda \models \phi \mathcal{C} \phi'$ iff there exists $i \geq 0$, such that $E, \lambda[0..i] \models \phi$ and $E, \lambda[i..\infty] \models \phi'$.

Additional operators $\Diamond$ (sometime in the future) and $\Box$ (always in the future) are defined as $\Diamond\phi \equiv \top \mathcal{U} \phi$ and $\Box\phi \equiv \neg\Diamond\neg\phi$. Note that *achievement goals* can be naturally specified with formulae of type $\Diamond\phi$, whereas a goal to maintain ϕ corresponds to the formula $\Box\phi$. Finally, we say that LTL formula ϕ is *valid in E w.r.t. a program τ* (written $E, \tau \models \phi$) iff for all $s \in \mathcal{S}$, ϕ holds on every trace $\lambda \in \mathcal{T}(\tau, s)$.

For an agent situated in an environment, we can easily define how likely the agent is to bring about a given goal.

Definition 16 (**Probabilistic fulfilment of goals**)**.** Given an agent program τ situated in an environment $E = (\mathcal{S}, \mathcal{E}, P)$ and an LTL formula ϕ, we first define the probability space $(\mathcal{T}_E(\tau, s), \mathit{Fin}(\mathcal{T}_E(\tau, s)), pr)$ induced by the next-state transition probabilities P. In this space, elementary outcomes are runs from $\mathcal{T}_E(\tau, s)$, events are sets of runs that share the same finite prefix (i.e., ones from $\mathit{Fin}(\mathcal{T}_E(\tau, s))$), and the probability measure $pr : \mathit{Fin}(\mathcal{T}_E(\tau, s)) \rightarrow [0, 1]$ is defined as

$$pr(s_0 \xrightarrow{a_1} \cdots \xrightarrow{a_n} s_n) = P(s_0, a_1, s_1) \cdot \ldots \cdot P(s_{n-1}, a_n, s_n).$$

Then, the probability of *fulfilling goal ϕ from state s on* is defined through the following Lebesgue integral:

$$P_{ok}(\tau, s, \phi) \;=\; \lim_{k \to \infty} \sum_{\lambda \in T^k_\phi(\tau, s)} pr(\lambda), \text{ where}$$

$$T^k_\phi(\tau, s) \;=\; \{\lambda \in \mathit{Fin}(\mathcal{T}_E(\tau, s)) \mid E, \lambda \models \phi \text{ and } |\lambda| = k\}.$$

The interested reader is referred to [38] for details of the construction and a proof of correctness.

Analogously to Section 4.2.1, the basic measure of fulfilment for a whole environment is based on the worst case analysis.

Definition 17 (**Fulfilment of goals:** P^-_{ok}, P^+_{ok})**.** Let $\tau \in \mathcal{P}$ be an agent program over the actions of some agent A. The minimal certainty of fulfilment of goal ϕ by τ in an environment E is the probability of fulfilment from the "worst" state in E:

$$P^-_{ok}(\tau, \phi) = \min_{s \in \mathcal{S}} P_{ok}(\tau, s, \phi).$$

The maximal certainty of fulfilment is defined analogously.

Reasoning about goals in pDCTL* In order to reason about the expected fulfilment of goals, we propose a refinement of the branching-time logic CTL* [24] with explicit quantification over program executions and probability thresholds. In the extension, $[\tau]_\zeta \phi$ reads as "agent program τ fulfils goal ϕ with probability at least ζ". The logic is called "*probabilistic Dynamic CTL**" (pDCTL*). It is a straightforward extension of "dynamic CTL*" from [57] (which in turn can be seen as a variant of Harel's process logic [32]) along the lines of probabilistic temporal logics [5, 31].

Definition 18 (**pDCTL***)**.** The syntax of pDCTL* is defined as an extension of LTL by the following grammar:

$$\begin{array}{lcl} \theta & ::= & \mathsf{p} \mid \neg\theta \mid \theta \wedge \theta \mid [\tau]_\zeta \phi \\ \phi & ::= & \theta \mid \neg\phi \mid \phi \wedge \phi \mid \bigcirc \phi \mid \phi \mathcal{U} \phi \mid \phi \mathcal{C} \phi \end{array}$$

where p is a propositional formula from $\mathcal{L}$, and τ is an agent program.

The semantics of pDCTL* extends that of LTL by the clauses below:

$E, \lambda \models \theta$ iff $E, \lambda[0] \models \theta$,

$E, s \models \mathsf{p}$ iff $s \models \mathsf{p}$,

$E, s \models \neg\theta$ iff $E, s \not\models \theta$,

$E, s \models \theta_1 \wedge \theta_2$ iff $E, s \models \theta_1$ and $E, s \models \theta_2$,

$E, s \models [\tau]_\zeta \phi$ iff the probability of fulfilling ϕ by τ from s on is at least ζ, i.e.,
$P_{ok}(\tau, s, \phi) \geq \zeta$.

pDCTL* formula θ is *valid in* E (written $E \models \theta$) iff $E, s \models \theta$ for every state s of E. Finally, ψ is a semantic consequence of ϕ (written: $\phi \Rightarrow \psi$) iff for every environment E, $E \models \phi$ implies $E \models \psi$.

The following proposition is straightforward and shows a strong relationship between formulae of pDCTL* and the measures of goal fulfilment introduced in Section 4.2.1.

Proposition 19. *$E \models [\tau]_\zeta \phi$ iff $P^-_{ok}(\tau, \phi) \geq \zeta$.*

Additionally, we define $[\tau]\phi$ as $[\tau]_1\phi$. It is easy to see that the semantics of $[\tau]\phi$ in pDCTL* and DCTL* coincide. Moreover, pDCTL* validity corresponds to LTL validity w.r.t. a program (the proofs are straightforward).

Proposition 20. *For every environment E, program τ and DCTL* formula ϕ, we have $E, s \models_{DCTL*} [\tau]\phi$ iff $E, s \models_{pDCTL*} [\tau]\phi$.*

Proposition 21. *For every environment E, program τ and LTL formula ϕ, we have: $E, \lambda \models_{LTL} \phi$ for every $\lambda \in \mathcal{T}_E(\tau)$ iff $E \models_{pDCTL*} [\tau]\phi$.*

We note that the operator dual to $[\tau]_\zeta$ has almost the same meaning, except of being underpinned by strict, instead of weak inequality.

Proposition 22. *Let $\langle\tau\rangle_\zeta\phi \equiv \neg[\tau]_{1-\zeta}\neg\phi$.*
Then $E, s \models \langle\tau\rangle_\zeta\phi$ iff $P_{ok}(\tau, s, \phi) > \zeta$.

Proof. $E, s \models \langle\tau\rangle_\zeta\phi$ iff $E, s \not\models [\tau]_{1-\zeta}\neg\phi$ iff $P_{ok}(\tau, s, \neg\phi) < 1-\zeta$ iff $1 - P_{ok}(\tau, s, \phi) < 1-\zeta$ iff $P_{ok}(\tau, s, \phi) > \zeta$. □

Thus, pDCTL* allows also to refer to the exact probability of fulfilment by $[\tau]_{=\zeta}\phi \equiv [\tau]_\zeta\phi \wedge \neg\langle\tau\rangle_\zeta\phi$.

Classifying environments w.r.t. goals Assuming a particular temporal goal allows for a finer-grained taxonomy of environments than we proposed in Subsection 4.2.1.

Definition 23 (Taxonomy of environments w.r.t. goal fulfilment). Given an agent program τ and a goal ϕ, we can distinguish between several types of environments according to the probability of fulfilment of ϕ by τ. We say that E is:

ideal iff $P^-_{ok}(\tau, \phi) = 1$ (alternative formulation: $E \models [\tau]\phi$),

manageable iff $P^-_{ok}(\tau, \phi) \geq \zeta$ for some $\zeta > 0$ (equivalently: $E \models [\tau]_\zeta\phi$ for some $\zeta > 0$). Note that, for finite environments, it is equivalent to $P^-_{ok}(\tau, \phi) > 0$ (and $E \models \langle\tau\rangle_0\phi$);

strictly manageable iff the environment is manageable and $P^+_{ok}(\tau, \phi) < 1$,

hopeless $P^-_{ok}(\tau, \phi) = 0$ (or equivalently: $E \models [\tau]_{=0}\phi$).

Consider the mobile robot from Subsection 4.2.1. Aiming to move along a pre-defined path, an outdoor environment is manageable for the robot, but a crowded place could even turn hopeless due to continuously moving people interfering with the robot. In effect, such a classification can give a good formal hint on limitations of usability of the robot.

For agent programs inducing only finite traces, we can distinguish environments w.r.t. the degree of iteration needed for the program to achieve a goal.

Definition 24 (**Taxonomy of environments cont.**). Given a program τ such that $|\lambda| < \infty$ for every $\lambda \in \mathcal{T}_E(\tau)$, we can distinguish between several additional types of environments according to the certainty of fulfilment of ϕ by τ in E:

iteratively manageable w.r.t. some $\zeta > 0$ iff there exists $k \in \mathbb{N}$ s.t. $E \models [\tau^k]_\zeta \phi$,

completely hopeless iff $E \models [\tau^*]_{=0}\phi$.

Additionally, we define the following limit cases:

- $E \models [\tau^*]\phi$, but there is no $k \in \mathbb{N} : E \models [\tau^k]\phi$,
- there exists $\zeta > 0$, s.t. $E \models [\tau^*]_\zeta \phi$, but there is no $k \in \mathbb{N} : E \models [\tau^k]_\zeta \phi$, and
- for all $k \in \mathbb{N}$, there exists $\zeta > 0$, s.t. $[\tau^k]_\zeta \phi$, however $[\tau^*]_{=0}\phi$.

This line of thought can be elaborated upon further by considering special types of programs. E.g., $\tau = a_1 \cup \cdots \cup a_n$, where $Act = \{a_1, \ldots, a_n\}$. Now $\mathcal{T}_E(\tau)$ include all the possible plans which can be constructed out of the actions in Act.

Program composition vs. goals

In this section, we turn our interest to the the following question: *how program composition affects the certainty level of goal fulfilment by agents in dynamic environments?* In particular, we are interested in how likelihood of fulfilling general maintenance and achievement goals (involving the $\Box$ and $\Diamond$ modalities respectively) relates to the way how the corresponding programs are constructed. Throughout this section, we implicitly assume that environments are strictly manageable w.r.t. programs and goals in consideration.

The following theorem articulates the intuition, that concatenation of programs leads to a strict decrease of the certainty of goal fulfilment by the joined program.

Theorem 25. *If τ_1, τ_2 are programs, s.t., $E \models [\tau_1]_{=\zeta}\phi_1 \ \wedge \ E \models [\tau_2]_\zeta \phi_2$ and at the same time $E \models \neg[\tau_1]_\zeta \phi_2$, then $E \models \neg[\tau_1; \tau_2]_\zeta \phi_1 \mathcal{C} \phi_2$.*

Proof sketch. The idea behind the proof is that due to the independence of τ_1 and τ_2 w.r.t. ϕ_2 ($E \models \neg[\tau_1]_\zeta \phi_2$), to establish $P_{ok}((\tau_1; \tau_2), s_o, \phi_1 \mathcal{C} \phi_2)$, we must consider traces induced by τ_1 *prolonged* by traces induced by τ_2. By approximating the sum of probabilities for sets of such prolongations rooted in terminal states of traces of τ_1 by $\xi = P^+_{ok}(\tau_2, E, \phi_2)$, we arrive to the inequality $P_{ok}((\tau_1; \tau_2), s_o, \phi_1 \mathcal{C} \phi_2) \leq \xi \cdot \zeta$. Since the environment is only *strictly manageable* w.r.t. τ_2 and ϕ_2, by necessity $\xi < 1$, hence $\xi \cdot \zeta < \zeta$, i.e., the minimal certainty of execution success of $\tau_1; \tau_2$ w.r.t. the goal $\phi_1 \mathcal{C} \phi_2$ and the environment E is strictly less than the original ζ.

□

In Theorem 25 we used the $\mathcal{C}$ operator for joining the goal formulae. Concatenation and further iteration of the same program leads to the following property of maintenance goals involving $\Box$ modality on ever longer traces.

Corollary 26. *Given τ is a program, s.t., $E \models [\tau]_{=\zeta}\phi$, then $E \models \neg[\tau^k]_{\zeta}\Box\phi$.*

Let's assume that the behaviour of an agent conforms to some performance quality measure when continuously operating at a particular level of probability of fulfilment of its goal by the agent program in a particular environment. We can formulate the following informal consequence of Theorem 25 and Corollary 26: *in dynamic environments, behaviour specifications involving temporal maintenance goals of the form $\Box\phi$ are undesirable.* The main reason behind this conjecture is, that to ensure satisfaction of strict maintenance goals, such as $\Box\mathsf{p}$, necessarily programs have to be joined by sequential composition, what leads to decrease of the level of certainty of execution success w.r.t. the envisaged goal. In fact, by subsequently prolonging the induced traces by sequential program composition, the level of certainty eventually eventually drops below the minimal required performance threshold.

Let's turn our attention to achievement goals, i.e., those which involve the $\Diamond$ modality.

Theorem 27. *Given τ is a program, s.t., $E \models [\tau]_{\zeta}\phi$, then $E \models [\tau^k]_{\zeta}\Diamond\phi$ for every $k > 0$.*

Proof sketch. The idea behind the proof is that by inductively iterating τ^k, the traces which satisfy $\Diamond\phi$ are i) those on which the goal was already satisfied for lower k's and prolonged by any trace regardless whether it satisfies $\Diamond\phi$, or not; plus ii) all the traces which did not satisfy $\Diamond\phi$ for lower k's prolonged by the traces induced by τ which satisfy $\Diamond\phi$. In result, for every k, the number of traces which satisfy the formula $\Diamond\phi$ is not decreasing, nor is the ratio to those which do not satisfy the goal. □

In fact, we hypothesise that if $E \models [\tau]_{=\zeta}\phi$, then we should have $E \models [\tau^k]_{=\xi}\Diamond\phi$, where $\xi > \zeta$ and $k > 0$.

Program perfection vs. fulfilment

Agent programs in dynamic environments can perform in an imperfect manner w.r.t. their goals due to two reasons. Firstly, it can be the environment by its dynamics which can cause the program not to fulfil its goal. Secondly, it can be the implementation of the program itself, which results in execution traces along which the goal is not fulfilled. In the following, we look at the probability of goal fulfilment *had the implementation of the agent's capabilities been ideal.* This allows us to measure the impact of the imperfection in the implementation on the fulfilment likelihood.

Definition 28 (Idealised environment). The idealised variant of an environment $E = (\mathcal{S}, \mathcal{E}, P)$ with respect to an agent template $(Act, \mathfrak{Ann})$ is defined as the environment $E^{\mathfrak{Ann}} = (\mathcal{S}, \mathcal{E}, P^{\mathfrak{Ann}})$ where the new probabilistic transition relation $P^{\mathfrak{Ann}}$ is as follows:

$$P^{\mathfrak{Ann}}(s,a,s') = \begin{cases} 0 & \text{iff } s' \not\models \mathfrak{Ann}(a) \\ \frac{P(s,a,s')}{\sum_{s'' \models \mathfrak{Ann}(a)} P(s,a,s'')} & \text{otherwise} \end{cases}$$

That is, we take the transition relation P in E and remove all the transitions that do not conform with $\mathfrak{Ann}$ (normalising P afterwards). Note that this scheme can be seen as a probabilistic version of *model update* similar to the one in *Public Announcement Logic* [6].

The idealised probability of fulfilling a goal ϕ is the probability of fulfilment of ϕ under the assumption that the agent's actions will behave as specified.

Definition 29 (Idealised fulfilment of ϕ). $P_{ok}^{\mathfrak{Ann}}(\tau, s, \phi)$ (resp. $P_{ok}^{\mathfrak{Ann}}(\tau, \phi)$) in an environment E is simply defined as $P_{ok}(\tau, s, \phi)$ (resp. $P_{ok}^{-}(\tau, \phi)$) in the idealised environment $E^{\mathfrak{Ann}}$.

Now we can measure the impact of imperfect implementation as the difference in certainty of fulfilment between the ideal and the real case:

Definition 30 (Impact metric *Imp*). Given environment $E = (\mathcal{S}, \mathcal{E}, P)$, agent $A = (Act, \mathfrak{Ann}, \tau)$, and goal ϕ, we define

$$Imp(\phi) = P_{ok}^{\mathfrak{Ann}}(\tau, \phi) - P_{ok}^{-}(\tau, \phi).$$

$Imp(\phi)$ indicates how much the imperfections in implementation of the agent's capabilities influence the chances for reaching the goal ϕ. In this context, it can be interpreted in two ways:

- *Imp* indicates *how inaccurate the agent implementation is with respect to the given goal*. In contrast, $P_{ok}^{-}(\tau, \phi)$ shows only the general inaccuracy of the agent implementation.
- Assuming that the designer has some control over the deployment of the agent in the environment, *Imp* can be understood as a measure of *how much they should improve the implementation of the agent's capabilities in order to get the objective met.*

Concluding remarks

The discussion in the last two sections only scratched the surface of what the introduced conceptual framework allows to rigorously investigate. One of interesting examples is the notion of a *planning horizon*. I.e., given a set of agent's capabilities, a goal and a specific level of certainty with which it should be satisfied, we can ask: *what is the maximal plan length beyond which the probability of fulfilling the goal decreases below the required quality threshold?* An estimate of the planning horizon could have an impact on parametrisation of planning algorithms used in dynamic environments, or can lead to constraints on lengths of plans in a library of a reactive planner. Similarly, deeper insights into how program composition influences the impact metric of the resulting programs have a potential to influence formulation of useful code patterns for agent programming, such as those introduced in [57]. We leave these interesting issues for further work along this line of research.

4.2.2. Plan repairing and re-planning

Plan repairing is a process of partial adaptation of a plan during its execution according to new conditions in the environment. The plan has not to be altered as a whole, but only its local part which is inconsistent with the new conditions, has to be changed. On the contrary, *re-planning* is a process of restarted planning during execution of a plan. The new planning process starts from the current state under the current context. Re-planning do not reuse any parts of the old (inconsistent) plan.

The research challenge for the frame of plan repairing and re-planning include finding answers for questions as: *"For what problem types we need plan repairing and for what re-planning?"* The hypothesis supporting plan repairing approach requires local and relatively rare unpredictable effects. The re-planning approach do not care the amount or impacts of the unpredictable effects, as the planning process has to be always run from the current state to the goal state. On the other hand, re-planning requires larger amounts of information to be shared among the planners (agents) in order to reconstruct the global state and context of other agents.

Additionally, we hypothesize two other important types of unpredictable world effects from the perspective of the deliberative/reactive planning systems and approaches. The first one is a *stolid effect* and the other one is a *opportunistic effect*. A stolid effect causes only a specious plan inconsistency (e.g. solvable by simple ignoring of the effect) where such effects can be effectively tackled by the plan repairing approaches. An opportunistic effect can even improve the whole plan provided that it is properly exploited. The opportunistic effects should be probably more re-planning friendly, as the plan repairing is a local process and can not simply consider the global state and the neighboring agents' context.

For an experimental validation and evaluation of the outlined hypotheses we have designed a testing scenario in two levels of abstraction. The next section summarizes the key principles and elements of the scenario.

Multi-agent Planning Formalization

Various aspects of multi-agent planning were studied in the course of the last decade (cf. e.g., [72]), surprisingly a formal treatment and a complexity analysis of the general problem of multi-agent planning appeared only recently in the work of Brafman and Domshlak [19]. In the following we recapitulate and extend their formalism of multi-agent planning (Subsection 4.2.2), introduce an abstract plan-execute-monitor architecture capable to detect plan failures (Subsection 4.2.2) and finally introduce the multi-agent plan repair problem (Subsection 4.2.2).

Multi-agent planning Let from now on $\mathcal{L}$ be a propositional language. The language $\mathcal{L}$ comes with the entailment relation $\models: 2^{\mathcal{L}} \times \mathcal{L} \to \{\top, \bot\}$ mapping a set of propositions, a theory in $\mathcal{L}$, and a formula to Boolean truth values according to the set inclusion relation. I.e., having a theory $S \in 2^{\mathcal{L}}$ and a proposition $\varphi \in \mathcal{L}$, $S \models \varphi$ if and only if $\varphi \in S$. W.l.o.g.,

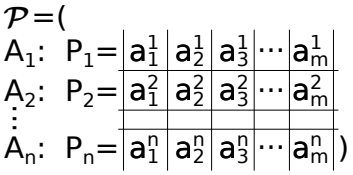


Figure 4.1.: Depiction of a multi-agent plan structure.

we assume $\top, \bot \in \mathcal{L}$.

A *multi-agent planning problem* is defined as a tuple $\Pi = \langle \varphi, S, s_{\text{init}}, S_{\text{goal}} \rangle$ where φ is a set of *agents* $A_1, \ldots, A_n$, S denotes the set of all possible states and $s_{\text{init}} \in S$ and $S_{\text{goal}} \subseteq S$ respectively denote the initial state of the multi-agent system and the set of desirable goal states. Each agent is characterised by sets of their respective individual capabilities. The capabilities are described using STRIPS as a set of quadruples $\phi_{pre} a \phi^+_{post} \phi^-_{post}$. $\phi_{pre} \in 2^{\mathcal{L}}$ denotes the set of preconditions of the action a, where a is the action's label. $\phi^+_{post}, \phi^-_{post} \in 2^{\mathcal{L}}$ respectively denote the sets of add and delete effects of execution of the action a. To say that an agent A is capable to execute the action a, from now on we simply write $a \in A$. Furthermore, we assume that each agent is capable to execute the empty action $\{\top\}\epsilon\{\top\}\{\top\}$, i.e., $\epsilon \in A_i$ for every $0 \leq i \leq n$.

Note that our version of STRIPS action specification language does not involve variables. Assuming only finite domains of variables, w.l.o.g., we assume already grounded action specifications, i.e., such where a first-order action specification with variables is translated into a set of primitive ground actions without variables by considering all instantiations of the action over the domains of the involved variables.

Having a multi-agent planning problem (MA-plan) $\Pi = \langle \varphi, S, s_{\text{init}}, S_{\text{goal}} \rangle$, we are looking for a set of individual plans $P_1, \ldots, P_n$ for the agents $A_1, \ldots, A_n$ in the form of sequences of actions, such that (i) each agent is capable to carry out its individual plan on its own, and (ii) after the synchronised execution of the individual plans, the system is in one of the goal states S_{goal}. Formally, considering agents $A_1, \ldots, A_n$, we are seeking a n-tuple $\mathcal{P} = (P_1, \ldots, P_n)$, such that each $P_i = a^i_1, \ldots, a^i_m$, where $a^i_j \in A_i$ and $|P_i| = |P_j| = m$, for every $1 \leq i, j \leq n$. We also denote $|\mathcal{P}| = m$. In the following we write $P[k]$ to select a_k, the k-th action of the individual plan $P = a_1, \ldots, a_k, \ldots, a_m$. Consequently, we write $\phi[k]_{pre}$, $\phi[k]^+_{post}$ and $\phi[k]^-_{post}$ do denote the pre- and post-conditions of the action $P[k]$. The structure of a MA-plan is depicted in Figure 4.1.

Since every agent is capable to perform the empty action ϵ, the constraint for the same length of plans can be trivially fulfilled for any multi-agent plan by adding ϵ-padding to the end of shorter individual plans.

Synchronised execution of a multi-agent plan $(P_1, \ldots, P_n)$ in a state s_0 is characterised by a sequel of states $s_0, s_1, \ldots, s_m$ where each state s_j is obtained from s_{j-1} by executing the joint action of the agents $\langle P_1[j], \ldots, P_n[j] \rangle$ for all $0 < j \leq m = |P_1|$. We write

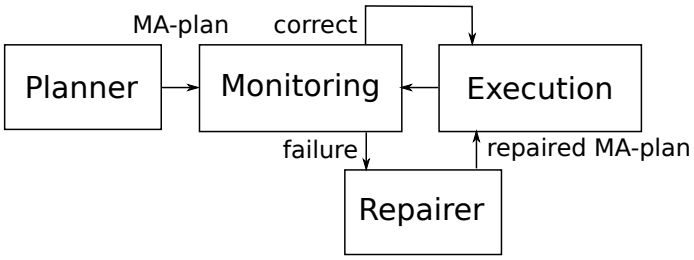


Figure 4.2.: The plan execution and monitoring architecture.

$s_j = s_{j-1} \oplus \langle P_1[j], \ldots, P_n[j] \rangle$, formally

$$s_j = s_{j-1} \setminus \{\bigcup_{i=1}^{n} \phi_i[j]^-_{post}\} \cup \{\bigcup_{i=1}^{n} \phi_i[j]^+_{post}\}$$

The joint action $\langle P_1[j], \ldots, P_n[j] \rangle$ is *executable* in a state $s_{j-1} \in S$ if and only if preconditions of each individual action are satisfied in s_{j-1}, i.e., $s_{j-1} \models \phi_{pre}$ for every $\phi_{pre} \in \bigcup_{i=1}^{n} \phi[j]_{pre}$.

We say that the multi-agent plan $(P_1, \ldots, P_n)$ is *sound* w.r.t the MA-plan problem $\Pi = \langle \varphi, S, s_{\text{init}}, S_{\text{goal}} \rangle$ iff the synchronised execution of $(P_1, \ldots, P_n)$ is characterised by the sequel of states $s_0, s_1, \ldots, s_m$, such that $s_0 = s_{\text{init}}$, $s_m \in S_{\text{goal}}$ and each joint action $\langle P_1[j], \ldots, P_n[j] \rangle$ is executable in the state s_{j-1}.

Plan execution and monitoring In dynamic and uncertain environments, agent's actions, plans, must not always lead to the desired consequences, or can turn out to be not executable. To account for such cases, a cautious agent must be able to not only to execute its actions, but also monitor its own progress and detect failures. Generally speaking, besides a planning component, implementation of an agent should also include a monitoring and plan repair components. Figure 4.2 depicts a generic multi-agent plan-execute-monitor architecture. More concretely, considering a multi-agent plan produced by some suitable MA-plan problem planner, the abstract execution monitoring algorithm checks in every state the soundness of the next step before advancing. If necessary, it invokes the plan repair procedure. The Algorithm 4.1 lists a sketch of such an algorithm.

The multi-agent planning function *plan*(...) takes as an input an instance of the MA-plan problem and returns a multi-agent plan $\mathcal{P}$. *simulate*(...) denotes the ideal plan execution function if everything works out right according to the joint action specification (w.r.t. the effects of the involved actions). Formally, $simulate(\mathcal{P}, s, k) = s \setminus \{\bigcup_{i=1}^{n} \phi_i[j]^-_{post}\} \cup \{\bigcup_{i=1}^{n} \phi_i[j]^+_{post}\}$ simulates the ideal execution of the joint action $\langle P_1[k], \ldots, P_n[k] \rangle$ on the state s. Function *exec*(...) executes a joint action defined by sets ϕ^+_{post} and ϕ^-_{post} and returns changed state.

Plan failure is detected by the condition $cannot\_proceed(\mathcal{P}, s_{\text{curr}}, k)$ which is true iff there exists an individual plan P_i of the agent A_i, such that the precondition $\phi[k+1]_{pre}$ of the

Algorithm 4.1 Generic plan execution and monitoring algorithm.

1: $\mathcal{P} = plan(\Pi)$
2: $k := 0$
3: **while** $k \neq |\mathcal{P}|$ **do**
4: $\quad s_{\text{curr}} := exec(s_{\text{curr}}, \bigcup_{i=1}^{n} \phi_i[k]^{-}_{post}, \bigcup_{i=1}^{n} \phi_i[k]^{+}_{post})$
5: $\quad$ **if** $cannot\_proceed(\mathcal{P}, s_{\text{curr}}, k)$ **then**
6: $\quad\quad \mathcal{P} = repair(\Pi, \mathcal{P}, s_{\text{curr}}, k)$
7: $\quad\quad$ **if** $\mathcal{P} = Fail$ **then**
8: $\quad\quad\quad$ **return** $Fail$
9: $\quad\quad$ **end if**
10: $\quad$ **end if**
11: $\quad k := k + 1$
12: **end while**
13: **return** $True$

action $P_i[k]$ is not satisfied in s, i.e., $s \not\models \phi[k+1]_{pre}$. Hence, the next joint action of the plan $\mathcal{P}$ cannot be executed as planned because one of the preconditions necessary for the step is not satisfied.

Upon plan execution failure, the plan repairing function $repair(\Pi, \mathcal{P}, s_{curr}, k)$ takes as an input the considered instance of the multi-agent plan problem, the executed multi-agent plan $\mathcal{P}$, the current state s_{curr} in which the execution of $\mathcal{P}$ failed (could not proceed any more) and k the step in which the execution of the plan $\mathcal{P}$ failed in.

Multi-agent plan repair As already sketched in the previous subsection, upon detection of plan execution failure, a plan repair should be attempted. In general, a plan repair problem $\mathcal{R}$ can be defined as a tuple

$$\mathcal{R} = (\Pi, \mathcal{P}, s, k)$$

comprising the original MA-plan problem Π, the currently executed multi-agent plan $\mathcal{P}$, the state s in which the execution of $\mathcal{P}$ was interrupted (couldn't proceed anymore) and $0 < k < |\mathcal{P}|$ the number of the step in which the execution of $\mathcal{P}$ was interrupted. Furthermore, we assume that there was a reason for the joint plan so that it couldn't proceed further n state s. Provided the execution of $\mathcal{P}$ is characterised by the sequence of states $s_0, \ldots, s_{|\mathcal{P}|}$, there must be an individual plan P_i in $\mathcal{P}$, such that the precondition $\phi[k+1]_{pre}$ of the next action to be executed $P_i[k]$ is not satisfied in $s = s_k$, i.e., $s \not\models \phi[k+1]_{pre}$. A solution of the plan repair problem (MA-repair) is of the same type as that of the plain multi-agent planning problem Π, i.e, a joint plan $\mathcal{P}'$, which is sound w.r.t. the MA-plan problem $\Pi = \langle \varphi, S, s, S_{\text{goal}} \rangle$. Optionally, we can require the resulting multi-agent plan $\mathcal{P}'$ to be constructed from $\mathcal{P}$ with use of minimal number of changes. Since the optimal plan repair problem is not the focus of this paper, we leave out its precise formal definition.

Algorithm 4.2 Naive plan repair algorithm

1: $s_{\text{curr}} := simulate(\mathcal{P}, s_{\text{init}}, k)$
2: **while** $S_{\text{goal}} \not\subseteq s_{\text{curr}}$ **do**
3: $\quad \mathcal{P}' := plan(\varphi, S, s_{\text{F}}, s_{\text{curr}})$
4: $\quad$ **if** $\mathcal{P}' \neq \emptyset$ **then**
5: $\quad\quad$ **return** $\mathcal{P}' \cdot tailplan(\mathcal{P}, k)$
6: $\quad$ **else**
7: $\quad\quad k := k + 1$
8: $\quad\quad s_{\text{curr}} := simulate(\mathcal{P}, s_{\text{init}}, k)$
9: $\quad$ **end if**
10: **end while**
11: **return** $Fail$

In the above paragraph, we focus only on a single type of plan failure, the critical one, i.e., when the cooperative plan simply cannot proceed anymore. The reasons for such a situation might be twofold. Either (i) some previous individual action of the multi-agent plan failed, i.e., some post-condition of the action did not become true after the action's execution, or (ii) the environment interfered and invalidated a previously established condition so that the forthcoming action couldn't be safely executed. In both cases, the failure is detected at the latest possible point, i.e., when the condition becomes vital for the subsequent step in the joint plan, that is its precondition.

In general, we can identify several types of plan failures. Alternatively, post-conditions of actions could be checked after their execution in order to detect action failures. This approach is however vulnerable to checking irrelevant conditions, i.e., such which are never used as a precondition of some future action in $\mathcal{P}$.

Plan repairing algorithms

Naive Repairing Algorithm Algorithm 4.2 lists the simplest algorithm designed. It uses iteratively prolonged repairing plans from the point of the failure. The complexity of the Naive Repairing Algorithm (NRA) respects the inner planner complexity. The worst case complexity is *PSPACE-complete* (since a general planning fits the *PSPACE-complete* complexity class).

Blind Repairing Algorithm The Blind Repairing Algorithm (BRA) tries to solve a failure by adopting an alternative action(s) solving the failed effects. Firstly it finds agent(s) which caused the problem (their effects does not meets the simulated state). A after the agents are found, for each agent an alternative action is tried to be found. If the personal alternative cannot be found a wide alternative is tried (another agent can adopt an action which solves the failure). Finally, even if the wide alternative cannot be found, the problem is ignored to be solved in the next steps prospectively, which can cause opportunistic solution of the

Algorithm 4.3 Blind Repairing Algorithm

1: $\mathcal{P}' := (P'_1 = \epsilon, ..., P'_n = \epsilon)$
2: $s_{\text{curr}} := simulate(\mathcal{P}, s_{\text{init}}, k)$
3: $\varphi^* := (A | A \in \varphi \wedge \exists a \in A : \phi^{a+}_{post} \subseteq s_{\text{curr}} \setminus s_{\text{F}} \vee \phi^{a-}_{post} \subseteq s_{\text{F}} \setminus s_{\text{curr}})$
4: **for all** $A_i \in \varphi^*$ **do**
5: **if** $\exists a_r : a_r \in A_i, \phi^{a_r}_{pre} \subseteq s_{\text{F}} \wedge \phi^{a_r+}_{post} \subseteq s_{\text{curr}} \wedge s_{\text{curr}} \not\subseteq \phi^{a_r-}_{post}$ **then**
6: $P'_i[1] = a_r$
7: **else**
8: **if** $\exists A_d \exists a_d : A_d \in \varphi, a_d \in A_d : \phi^{a_d}_{pre} \subseteq s_{\text{F}} \wedge \phi^{a_d+}_{post} \subseteq s_{\text{curr}} \wedge s_{\text{curr}} \not\subseteq \phi^{a_d-}_{post}$ **then**
9: $P'_d[1] = a_d$
10: **else**
11: noop()
12: **end if**
13: **end if**
14: **end for**
15: **if** $\exists i : P'_i \neq \epsilon$ **then**
16: **return** $\mathcal{P}' \cdot tailplan(\mathcal{P}, k)$
17: **else**
18: **return** $tailplan(\mathcal{P}, k)$
19: **end if**

failure in future. If a failure is ignored the repairing algorithm is called again in the next step by the main monitor-execution loop. Algorithm 4.3 lists the pseudo-code of the blind plan repairing algorithm.

The algorithm is not sound in the pure form as it can miss a solution of the repairing problem. Its modification towards the soundness can be managed simply by adding a fail-safe call of NRA if the algorithm iteratively fails at the goal state. The iterative form of the algorithm is listed in Algorithm 4.4.

The non-iterative form has linear complexity (iteration over the number of the agents). The iterative extension has minimal quadratic complexity (number of the agents times the length of the plan). Maximal complexity reflects the complexity of the NRA as it can be used as a fail-safe process.

Locality Repairing Algorithm The last repairing algorithm (see Algorithm 4.5) is based on an assumption, the communication is not costless, i.e. we count amount of information needed to be transmitted for each repairing of an action (for the Naive Repairing Algorithm, it is the complete global state plus the final state to one repairer agent, and then back to the result to the rest of the agents $c = 2 * 2 * n$, n is number of the agents, i.e. number of the state to transfer and c is communication volume for each (re-)planning process; for the Blind Repairing Algorithm the mean value is $c = n/2$, i.e. a median of the probability of number of agents involved in reparation of one action).

Algorithm 4.4 Iterative Blind Repair Algorithm

1: $k^{\text{orig}} := k$
2: $s_{\text{F}}^{\text{orig}} := s_{\text{F}}$
3: $\mathcal{P}' := \mathcal{P}$
4: **while** $S_{\text{goal}} \not\subseteq s_{\text{curr}}$ **do**
5: $\quad \mathcal{P}' := blindRepair(\Pi, \mathcal{P}', s_{\text{F}}, k)$
6: $\quad$ **if** $\forall i \in (0, ..., m) : \phi_i'[k]^+_{post} \subseteq s_{\text{F}} \wedge s_{\text{F}} \not\subseteq \phi_i'[k]^-_{post}$ **then**
7: $\quad\quad$ **return** $\mathcal{P}'$
8: $\quad$ **else**
9: $\quad\quad k := k + 1$
10: $\quad\quad s_{\text{F}} := simulate(\mathcal{P}', s_{\text{init}}, k)$
11: $\quad$ **end if**
12: **end while**
13: **return** $naiveRepair(\Pi, \mathcal{P}, s_{\text{F}}^{\text{orig}}, k^{\text{orig}})$

Before presenting the final plan repair algorithm, the locality repairing, let us introduce the notion of a *causality graph* exploited by the algorithm.

Each MA-plan $\mathcal{P}$ can be represented as a graph G, where vertices V represent the actions and edges E represent causal links among the actions:

$$\begin{aligned} G &= (V, E) \\ V &= \{v_i | \forall i : a_i \in \mathcal{P}\} \\ E &= \{e_{ij} | e_{ij} = (v_i, v_j) \textbf{ iff } \phi^{a_i}_{post} \cap \phi^{a_j}_{pre} \neq \emptyset\}. \end{aligned}$$

Using a causality graph G, a causality relation $\leftarrow$ can be defined as $a_i \leftarrow a_j$ **iff** $e_{ij} \in E$. A transitive closure of $\leftarrow$ (oriented path in the causal graph) will be denoted as $\overset{*}{\leftarrow}$.

The proposed plan repairing algorithm is based on the planning technique presented in [53]. From the communication perspective, it tries to solve the problem as locally as possible, iteratively including more and more agents which tries to solve the problem also locally. Therefore, the agents communicate as few as possible information (failed actions). Algorithm 4.5 lists the pseudo-code of the Locality Repair Algorithm (LRA).

The vector $\overline{v}$ represents three counters: v_1 number of agents involved in the repairing process, v_2 represents length of the repairing plans, and v_3 represents number of actions removed from the old plan. The counters are increased if a valid plan is not found using a constant vector $\overline{V}$. Various parametrisation of the algorithm can be done using the $\overline{V}$ vector (e.g., $\overline{V} = (0.1, 1, 0)$ represents increase of the number of the involved agents for each 10 actions of the repairing plan length).

The $combination(\varphi, A, n')$ method generates a set of possible combinations of n' agents prioritising combinations with the agent A. The function $A : alterns(q, r)$ generates alternative repairing plans from the perspective of the agent A of length q with r removed actions from the current plan. The process of action removing follows a path in the causal-

Algorithm 4.5 Locality Repairing Algorithm

```
1: v̄ := (1, 1, 0)
2: loop
3:    φ* := combinations(φ, A, ⌈v1⌉)
4:    π = {ρ|∀A ∈ φ* : ρ = A : alterns(⌈v2⌉, ⌈v3⌉)}
5:    P = compatible(π)
6:    if P ≠ ∅ then
7:       return P
8:    end if
9:    v̄ := v̄ + V̄
10: end loop
```

ity graph G, formally the process removes actions on path $a_k \overset{*}{\leftarrow} a_{k+r}$. The generated plan alternatives have to satisfy the initial conditions defined by the state s_{F} and the final conditions defined by state s_{k+r}. The function $compatible(\pi)$ returns a plan based on the alternatives in the set π, where only mutually compatible alternatives are preserved. Two alternatives are compatible if delete effects of one do not preclude usage of the other. The first compatible alternative tuple found is used.

Repairing Dimensions The Locality Repairing Algorithm is based on the principle of the Blind Repairing Algorithm. The main difference is the algorithm is sound inherently in the pure form.

The main problem of the BRA is the repairs can be done only by one changed action for an agent A in the step k. Additionally, the changes cannot be backtracked as the process works only in the successive manner. On one side, this facts simplifies the process and implies the process has only linear complexity. On the other side, the process can miss quite simple solutions using only two successive actions provided by the failure causing agent. The Locality Repairing Algorithm takes into an account all these possibilities and thus searches through the complete space of possible repairs.

More precisely, there are three dimensions in the space of the possible repairs. The first dimension v_1 is common with the BRA and it describes which agents undertake the repairing actions. The second dimension v_2 describes how many and which actions one agent uses for its local repair. The last one v_3 describes how deep the repair goes through the current plan. The last dimension is similar to the successive direction of the BRA. The differences are (i) the repairing process can backtrack and (ii) it primarily follows the causality graph in an opposite direction against the causality links.

In the exhaustive case, all the combinations of the future actions have to be treated as possibilities for repairing. The only aspect decreasing the full range of $2^{|A|\lceil v_2 \rceil}$ combinations for one agent A is that only compact successive sequences of the actions based on the causal links need to be considered. The number of the combinations thus changes to $(\lceil v_2 \rceil + 1)^{|A|}$.

Plan Repairing for DCOP-based Multi-agent Planning The plan repairing technique (NRA particularly) was used in extension of a state-of-the-art Multi-agent Planner presented in [54]. The planner consists of two phases of planning. In the first phase, a multi-agent plan is formed (with help of the DCOP solver). This plan contains only public actions. A public action is an action which can affect other agents' actions. In the second phase, the plans are locally completed using the private actions, which ensure transition from one public action to the next one for each agent.

Using distributed constraint reasoning (DCR) to solve the planning problem shifts the complexity from the planner to the DCR solving algorithm, and these algorithms cannot scale to large problem sizes. The scalability of these algorithms is especially relevant in planning, where the domains grow exponentially with δ, the length of allowable action sequences.

We proposed using an approximation algorithm for solving the constraint reasoning representation of the planning problem, and then using plan repair to remove infeasible parts of the plan. In our empirical results, we used the Max-Sum algorithm [26], which gracefully degrades solution quality with message loss, and produces complete solutions for acyclic constraint graphs.

The outline of the approach can be summarized as:

- formulation and definition of the planning problem
- generation of candidate public action sequences (separately by each agent)
- selection of appropriate sequences (distributively by the agents)
- reparation of invalid sequences (distributively by the agents)
- private planning of local plans based on the public sequences (separately by each agent)

Details of the planning processes and particular techniques can be found in .

To ensure completeness of the overall multi-agent plan, the plan repair phase solves potential infeasible plans returned by the approximate DCOP algorithm. The generative graph of the planner is preserved from the planning phase for the plan repairing phase, which implies, the process reuses a data structure from the planning phase and exploits it for more efficient plan repairing similarly to [4]. Since the planning problem in the plan repairing phase is simpler, we can use optimal DCOP to ensure soundness of the repaired plan.

4.3. Evaluation and experiments

The presented algorithms were evaluated in a synthetic domain and verified in an environment simulating real-world urban area. The following sections summarizes details of the prototypes together with and the results, measurements and conclusions for the particular cases of usage.

4.3.1. Planning algorithms

The planning and plan repairing algorithms were implemented and tested in synthetic environment based on well known planning domain. The details follows in the next subsections.

Simple Box-Mover Prototype

For experimental purposes, a prototype of the plan repairing algorithm was implemented. The prototype contains two main parts: a multi-agent planner and plan repairing algorithm. As a planner, a state-of-the-art technique [53] was used. The planner is based on compatible formalisms as the proposed ones in Section 4.2. It is a distributed planner based on two-phase planning. In the first phase, each agent prepares a set of possible sequences of synchronization actions. These sequences are combined into a large set of all possible tuples. The combined complete plans are checked for (a) soundness, and (b) for goal satisfaction constraints. For instance, the soundness check tests if there are no two simultaneous actions using one resource. Additionally, the test also checks linkup of actions of different agents – a box cannot be loaded if it is carried by another agent. In the second phase of the planning process, private plans complete the public actions towards executable private plans of particular agents. The private plans cannot affect the other agents and they are prepared separately by each agents. The private planning phase ends with prepared personal plans usable for the execution.

The particular planning problem implemented in the prototype is a simplified instance of classical blocks-world problem with resources. Initial state of problem is following:

$$\begin{aligned} S_I \;=\; & \{mover(Mover1), mover(Mover2) box(Box1), box(Box2), \\ & at(Mover1, Point[0,0]), holds(Mover1, -), \\ & at(Mover2, Point[0,5]), holds(Mover2, -), \\ & at(Box1, Point[9,0]), at(Box2, Point[8,0])\}. \end{aligned}$$

The goal state conditions are:

$$S_G \;=\; \{at(Box1, Point[0,9]), at(Box2, Point[9,9])\}.$$

In Figure 4.3, the initial state and one of possible final states are depicted.

There are three possible actions (i) move, (ii) load, and (iii) unload. The STRIPS definition of the actions contain three sets for each action:

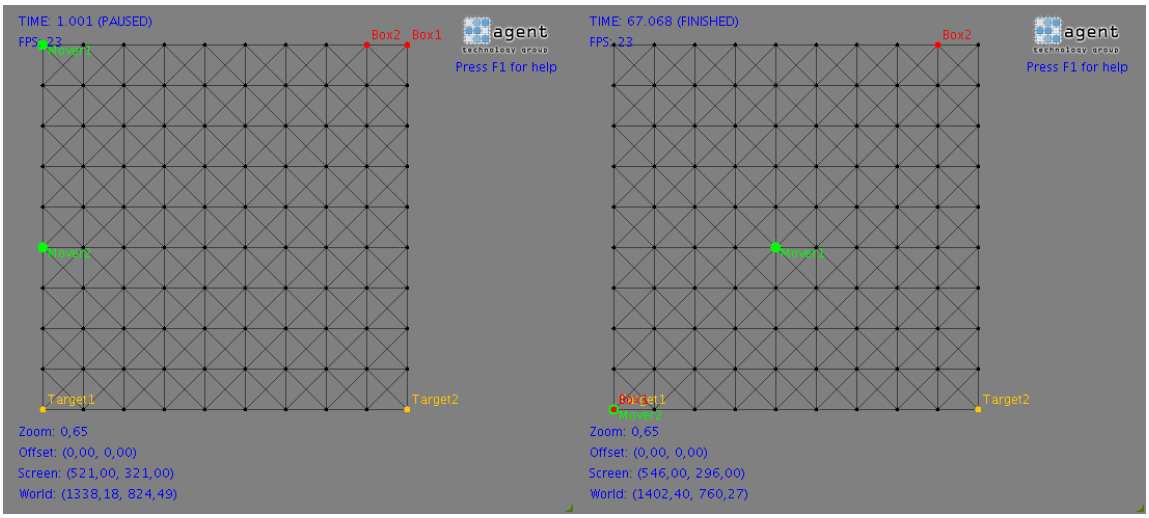


Figure 4.3.: Initial state of the planning problem (on the left) and one of the possible final states (on the right). The goal state contains only $Box1$.

$$
\begin{aligned}
pre(move(M,P)) &= \{mover(M), at(M, OldPoint), \\
&\quad (M = Mover1) \implies P_y < 5, (M = Mover2) \implies P_y > 5\}, \\
del(move(M,P)) &= \{at(M, OldPoint)\}, \\
add(move(M,P)) &= \{at(M,P)\},
\end{aligned}
$$

$$
\begin{aligned}
pre(load(M,B)) &= \{mover(M), at(M,P), holds(M,-), box(B), at(B,P)\}, \\
del(load(M,B)) &= \{at(B,P), holds(M,-)\}, \\
add(load(M,B)) &= \{holds(M,B)\},
\end{aligned}
$$

$$
\begin{aligned}
pre(unload(M,B)) &= \{mover(M), at(M,P), holds(M,B), box(B)\}, \\
del(unload(M,B)) &= \{holds(M,B)\}, \\
add(unload(M,B)) &= \{at(B,P), holds(M,-)\}.
\end{aligned}
$$

The planning algorithm implemented in the prototype covers the core principle of the proposed technique in [53], however it differs in particular implementation. Figure 4.6 provides the listing of a simplified planning algorithm for the agent $Mover1$.

The algorithm for $Mover2$ only receives the sent action sequence and after private planning process, the agent executes it. The main difference against the algorithm in [53] is that the planning algorithm is not using CSP solver for finding an appropriate

Algorithm 4.6 Pseudocode of the planning algorithm for the $Mover1$ agent.

Require: s_{init} ... initial state
Require: S_{goal} ... goal state
Ensure: P_{Mover1} ... executable local plan of *Mover1*

```
1: for δ = 1; δ = δ + 1 do
2:    ρ = generateActionSequences(δ, s_init, S_goal)
3:    ρ' = RECEIVE(ρ from Mover2)
4:    for all P ∈ ρ do
5:       for all P' ∈ ρ' do
6:          if S = simlulate(s_init, s_init, ρ, ρ') then
7:             for all s_i ∈ S do
8:                P_Mover1 = P_Mover1 + doPrivatePlanning(s_{i-1}, s_i)
9:                if P_Mover1 = ∅ then
10:                  continue
11:               end if
12:               SEND(ρ' to Mover2)
13:               return P_Mover1
14:            end for
15:         end if
16:      end for
17:   end for
18: end for
```

pair of the public action sequences, but a exhaustive search is used and implemented as two nested loops on lines 4 and 5. The statements $SEND$ and $RECEIVE$ represent inter-agent communication (which is in the prototype replaced by direct variable access). The function $generateActionSequences(\delta, s_{\text{init}}, S_{\text{goal}})$ generates a set of all possible public action sequences of length δ. The sequences are pre-filtered and only locally sound sequences are included in the set. An example of an invalid sequence is $(load(M, Box1), load(M, Box2)$, since $Box2$ cannot be loaded if $Box1$ is already carried. The function $simlulate(s_{\text{init}}, s_{\text{init}}, \rho, \rho')$ returns a sequence of global states anticipated if the action sequences ρ and ρ' would be executed from state s_{init} without any failure in the execution. The goal state set S_{goal} is provided to allow the function to filter sequences which do not end in the goal state. Additionally, the function also checks feasibility of the planned sequences and interactions between the agents. Finally, the function $doPrivatePlanning(s_{i-1}, s_i)$ creates a private action sequence including one pubic action at the end of the sequence. The chaining of the private sequences $P = P + \ldots$ with public action at the end builds the final executable local plan.

The second part of the prototype includes two methods for failure handling: re-planning from scratch and the Naive Plan Repairing Algorithm. Re-planning runs the complete planning process again if a failure is detected during the execution. The Naive Plan Repairing Algorithm updates the current plan only locally to solve the failure. The local change is done from the state of the failure to the first public action state.

The model of the failures is based on a probability p with a uniform distribution. The probability describes if a move action carrying a box drops the box. The detection of the failure is reliable which means the failure is always detected. $p = 1$ represents a failure of each executed move action (since only a move action can fail and the failure occurs after the action execution, even for $p = 1$ the goals can be fulfilled – the box is dropped after each move). If $p = 0$, the execution is deterministic and without failures.

In Figure 4.4, it is demonstrated the length, of the plans generated by the plan repairing technique, is only slightly larger (the maximum is 10% for $p = 1$) than the plans fixed by re-planning from the scratch. However, the number of generated action sequences (the main complexity indicator of the algorithm) is by 73% higher for the planning from the scratch than for the plan repairing algorithm. The Figure 4.5 depicts measurements of the action sequence generators.

General Box-Mover Prototype

The Simple Box-Mover Prototype was extended towards variable numbers of the entities in the environment, i.e. there are k mover agents (representing a robot and crane as one entity), which can each move in a $w \times h$ grid and b distinguishable boxes each of which has one target position. The boxes cannot be laid upon each other. Each grid point can be accessed by only one mover with the exception of s points, which can be accessed by two neighbouring agents (hand-over points). Each mover can carry only one box at a time. The initial state defines points where the movers start and where the boxes lie. The movers has

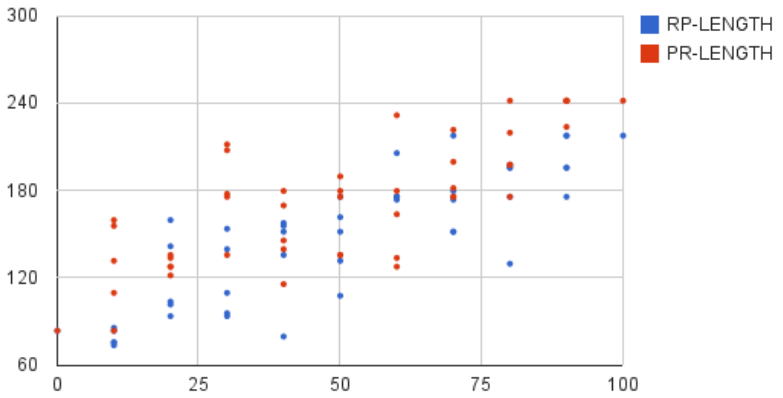


Figure 4.4.: Relation of a total length of a plan (y-axis is counted in number of actions) and probability p of the failures in percents (x-axis). RP represents re-planning algorithm and PR represents Naive Plan Repairing Algorithm.The goal state contains only $Box1$. The domain contains two movers.

similarily three possible actions (i) move, (ii) load, and (iii) unload. A mover can *move* in its accessible grid by one cell horizontally, vertically, and diagonally (including the hand-over points) whether or not it is carrying a box. A mover can *load* a box, if it is at the same point as the box. A mover can *unload* a box if it is carrying it, and there is not another box in the unload position. Each box has a target point. A solution of the problem is a plan of actions for each mover which relocates the boxes from their initial points to their target points. The model of the failures is same as for the Simple Box-Mover Prototype.

An example instance of the domain is depicted in Figure 4.6, where $k = 4, w = 6, h = 6, b = 2$, and $s = 4$.

The experiments are comparing the Naive Repairing Algorithm and re-planning from scratch for varying probability p. There are three comparison metrics used: (i) computational complexity, (ii) communication complexity, and (iii) length of the resulting plans. The computational complexity measure is based on a number of action sequences generated by the planning, re-planning, and plan repairing processes. All such sequences have to be checked for soundness and mutual compatibility, which makes them the most complex part of the algorithm. The communication complexity counts a number of messages, which are necessary to coordinate the planning processes. The messages contain only comparable sizes of data (the largest structure transferred by one message is a single-agent action sequence). More complex data packages are split into more messages. The length of the final

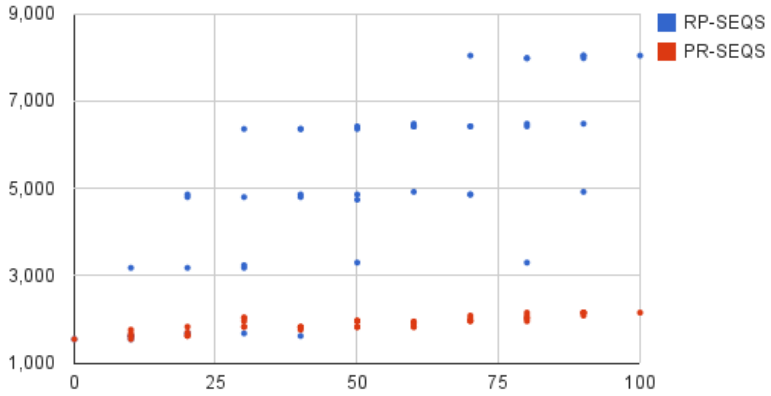


Figure 4.5.: Relation of a total number of generated action sequences (y-axis) and probability p of the failures in percents (x-axis). RP represents re-planning algorithm and PR represents Naive Plan Repairing Algorithm.The goal state contains only $Box1$.The domain contains two movers.

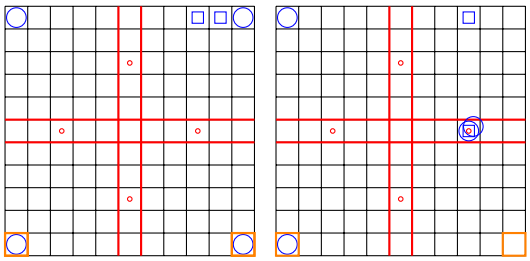
plan includes also all repairing plans.

The results are depicted in three figures according to the mentioned metrics. As we can see in Figure 4.7, the complexity of the plan repairing algorithm (NRA) is considerably lower (12%) than the complexity of the re-planning. The plan repairing technique uses the planner during the execution only for short and simple planning problems. On the other hand, the re-planning approach in the earlier phases of the execution has to plan complete plans towards the goal state.

The communication complexity (see Figure 4.8) of the plan repairing algorithm is also lower than the re-planning from scratch (57%). As the planning problems creating repairing plans are simpler than the complete re-planning problems, the planning process need less messages to find the compatible personal action sequences of the agents. The more advanced plan repairing techniques as Locality Repairing Algorithm are theorised to even more decrease the number of the messages required for the repairing process according to the communication dimension of the vector $\overline{V}$.

The plan lengths, depicted in Figure 4.9, shows the increase of the plan length for the plan repairing technique (59%). It is caused by repeated extension of the initial plan by the repairing plans. Study and reduction of this effect can be one of the possible future works.

The results of the Box-Mover Experiments support the premise that the plan repairing



(a) (b)

Figure 4.6.: The example environment, where circles represent movers and squares represent the boxes. The orange cells are the respective target points and the red points are the hand-off points. (a) The initial configuration for 4 movers and 2 boxes. (b) Box1 (10, 0) is being handed from Mover3 (top-right) to Mover4 (bottom-right).

algorithms are from the perspective of algorithm complexity much more suitable for highly dynamic environments sacrificing only a small increase of the plan length.

Evaluation of Plan Repairing for DCOP-based Multi-agent Planning

The plan repairing algorithm (NRA particularly) was used to fix soundness of plans generated by the DCOP techniques. This enabled tackling of larger planning problems.

Once the length of the plan δ is big enough to reach any of the goals, a plan is produced which does so. That this is one of the key differences between our approach and a constraint satisfaction algorithm. For example, the results in the Box-Mover domain with two agents for $\delta = 1 \ldots 3$ was a set of ϵ for each agent. For $\delta = 4 \ldots 5$ it produces a plan that moves one of the boxes to the goal. For $\delta \geq 6$ it produces a plan that meet both goals.

When the number of agents was increased to four, which resulted in cyclic constraint graph, DCOP solver failed to construct an optimal plan. At this point, plan repair was used to construct the optimal plan. The search space (i.e. the number of public plan combinations need to be evaluated) is shown in Figure 4.10. The complexity of plan repair was the same for any number of agents, because δ was fixed at 6.

The result shows the plan repairing technique can be also used in cooperation with other problem solving approaches for additional adaptation or fixing of more efficiently generated plans. The complexity of the problem it thus transformed into the complexity of the plan repairing problem, which can, using known planning approaches, provide better insights into the problem and well-known heuristics.

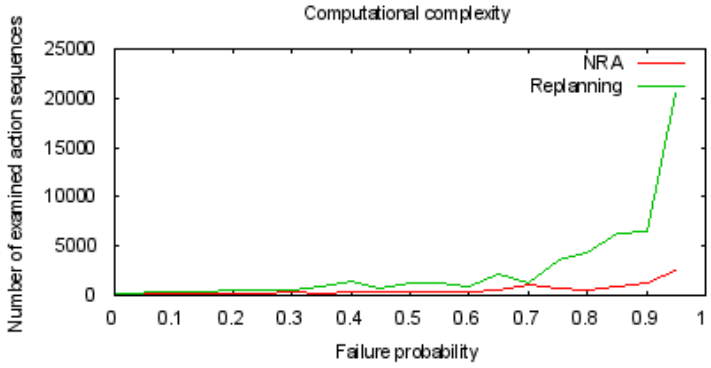


Figure 4.7.: Comparison of NRA and re-planning from scratch – computational complexity metrics.

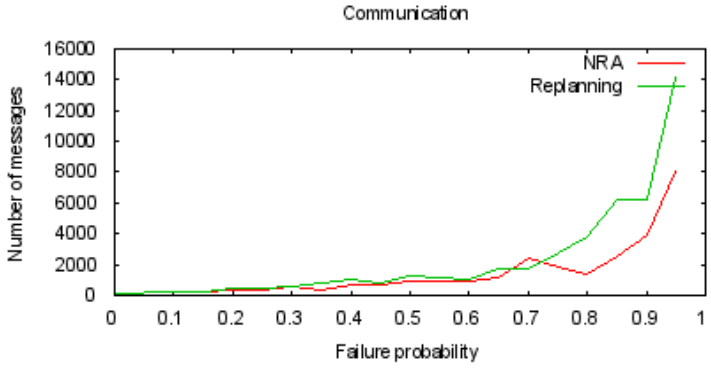


Figure 4.8.: Comparison of NRA and re-planning from scratch – communication complexity metrics.

4.3.2. Plan repairing in dynamic urban environment

Mission

The high-level mission goal is to extract a VIP from a safe-house in a hostile urban area (village) by a blue force units. The topology of the village is known a priory from a satellite recon. The troops are supported by a heterogeneous team of autonomous robotic assets (2 Vidar VTOL MUASs, 2 Skeldar VTOL UASs, 1 MDARS UGV), see Figure 4.11. The environment unpredictability and the blue forces involved cause perturbations in the plan execution, and the autonomous units has to adapt (repair plans) accordingly to be consistent with the behavior of the blue forces. The adaptation should take into an account the high-level plan, which should be always the goal of the blue forces. The assets are controlled by multi-agent planning and plan repairing system and they must behave quite

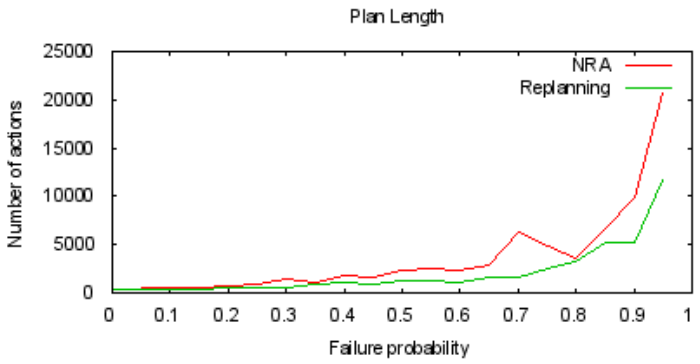


Figure 4.9.: Comparison of NRA and re-planning from scratch – plan length metrics.

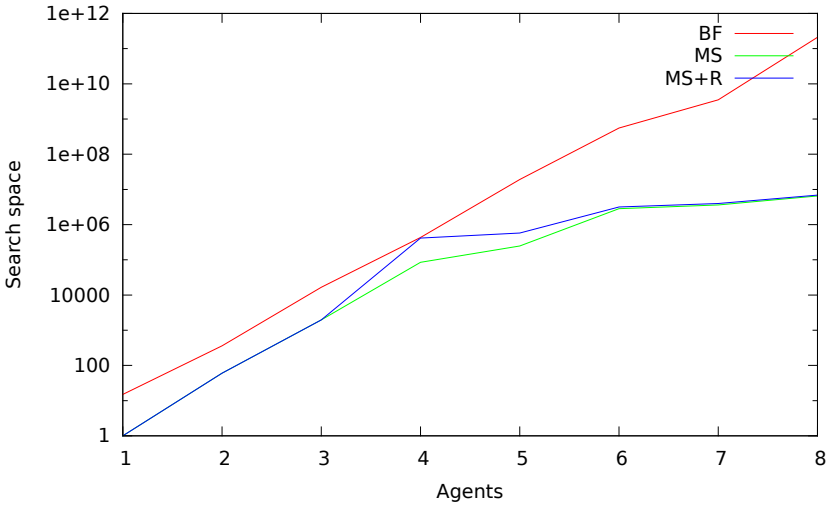


Figure 4.10.: Growth of the search space for $\delta = 6$ as the number of agents increases for brute-force (BF), DCOP (Max-Sum) (MS), and DCOP (Max-Sum) with plan repairing (MS+R).

Figure 4.11.: The ground forces (green) supported by the robotic team (2 Vidar VTOL MUASs – orange, 2 Skeldar VTOL UASs – red, 1 MDARS UGV – brown) in the simulated urban environment.

intelligent with regard to the mission (which description includes also the needs of the soldiers in form of action preconditions). Additionally, any asset can be destroyed, if it behaves inadvisedly. The positions and numbers of the hostiles is not known a priory. The behavior of the blue and red forces is based on the reactive behaviors described in Chapter 5.

The communication among the assets is considered to be reliable and limitless (the blue forces are spatially compact and thus they keep nearly permanent visual contact). The computation load is limited by the simulating computer which corresponds with computational power of the the on-board computers . The sensors are simulated as deterministic, precise, and use ground-to-ground occlusion simulation. The actuators differ for the various assets. The Vidars have deterministic and precise move actuators, the Skeldars use dynamics simulation and the MDARS is simulated using physics simulation engine.

Planning and plan repair

The initial plan of each asset is to move to the safe-house and than to the extraction point. Possible actions of the assets are:

noop() The preconditions are $\emptyset$ and the action can be used by all asset types. The action represents doing nothing, i.e. wait.

move-to(*position***)** The preconditions are $\{covering \wedge \neg colliding\}$ and the action can be used by all asset types. The asset is moved from the current position to new *position*.

personal-cover(*person***)** The preconditions are $\{covering \wedge \neg colliding\}$ and the action can be used by the Vidars. The action moves the asset above the position as the *person*

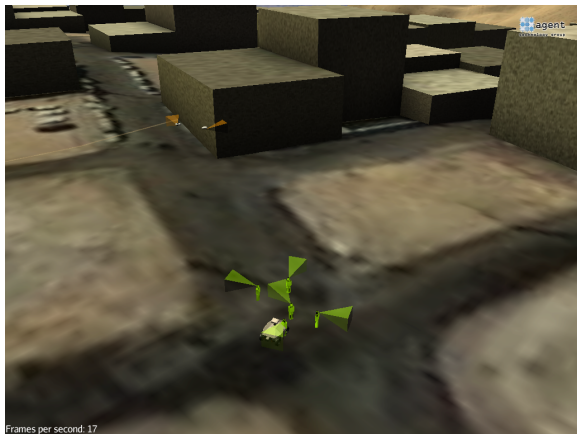


Figure 4.12.: Two Vidar MUASs providing street recon actions in front of the team.

is and orients it oppositely than the *person* is directed.

street-recon(*street***)** The preconditions are $\{allAlliedCovered \wedge \neg colliding\}$ and the action can be used by the Vidars. The action moves and orients the asset as it can look through the *street* (see Figure 4.12).

street-surveillance(*street***)** The preconditions are $\{allAlliedCovered \wedge \neg colliding\}$ and the action can be used by the MDARS and Skeldars. The action moves the asset along the *street*.

crossing-surveillance(*crossing***)** The preconditions are $\{allAlliedCovered \wedge \neg colliding\}$ and the action can be used by the Skeldars. The action positions the asset above the *crossing* and makes it to observe the area around the *crossing*.

There are three used preconditions:

colliding The condition holds if the action is spatio-temporally colliding with action(s) of other assets.

covering The condition holds if the action causes an allied troop is covered (the asset is in a proximity).

allAlliedCovered The condition holds if all allied troops are covered (in the proximity of all troops is at least one asset).

The conditions can form a propositional formulae.

The plan-repairing mechanism is solving the problems caused by the unpredictable movement of the troops. The first **move-to** action in the initial plan

$$(moveto(safehouse), moveto(extractionpoint))$$

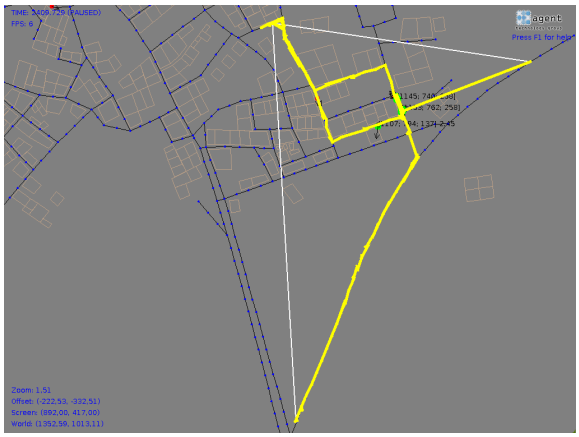


Figure 4.13.: An individual plan of one of the Vidar MUAVs. The white lines represent the initial mission plan (move to the safehouse – move to the extraction point). The yellow lines depict a spatial representation of the current repaired plan of the asset.

has the precondition *covering*, which implies the action cannot be used and causes a failure of the plan. To solve the failure, the NRA technique (see Section 4.2.2) adopts a action solving the problem which is $personal - cover(person)$ with a randomly chosen allied troop. Usage of the action solves the failure (and made *allAlliedCovered* holding) and thus the other assets can use other the possible actions (surveillance and recons). The basic deconflicition of the assets is caused by the *colliding* precondition, because of which the assets are always adopting actions at distinct positions. The initial mission plan and the repaired plan of one Vidar VTOL is depicted in Figure 4.13.

The demonstration shows the multi-agent plan repairing technique can be used for distributed continual planning of robotic team in highly dynamical and unpredictable environment.

5. Coordination and teamwork

5.1. Summary of the workpackage

In the coordination an teamwork research track, we studied methods that will allow us to specify the joint mission of a number of agents. In particular, we were focusing on how we can specify the desired sequence of actions an agent needs to perform in order to effectively pursue the joint goals of the team.

For this purpose we were examining commitment machines – a framework for monitoring of communication protocols. In this framework, each synchronization point in the protocol can be defined as a commitment one agent makes to another agent. A protocol cannot proceed to later stages until all the relevant commitments are discharged.

Such a mechanism turns out to be also suitable for the specification of desired interactions between two or more agents. Distributed commitment machines represent a more realistic flavor of commitment machines that allows the mechanism to be executed in a distributed way, i.e. by each involved agent.

In order to evaluate the method, we have implemented a simple example scenario in which a team of troops moves in formation through an urban area. The modelled team consists of one leader with four subordinate troops. The team moves in a coordinated fashion so that they jointly cover the surrounding area.

The mechanism was implemented using Jazzyk agent programming language, which provided us an advantage of being able to combine different knowledge representation techniques in our program. Using the technologies mentioned above, we were able to build agents exhibiting robust coordinated behavior, which was specified in form of a short declarative program.

5.2. Technology description

One of the objectives of this workpackage is to develop a technology allowing us to specify and subsequently execute a large range of mission scenarios composed of a number of basic building blocks, such as *exploration*, *surveillance*, *tracking* on the side of the mission assets, such as e.g., UAVs, as well as behaviors including e.g., *securing an area*, *movement to a pre-defined location*, *tracking* or *evasion* on the side of ground forces, be it allied, or foe troops.

We designed and implemented a basic framework allowing us to encode specifications of missions as a number of interdependent commitments representing the dependencies

betwen the partial goals the team has to achieve in order to successfully fulfill the mission. Our mission specification framework is based on the formalism of distributed commitment machines.

5.2.1. Distributed Commitment Machines

Commitment Machines

Commitment machines, originally developed by Yolum and Singh in [76], is an approach for flexible and succinct specification of inter-agent coordination protocols, are well suited for high-level specification inter-relationships among individual agent's goals.

Inter-agent commitments are a structure capturing a relationship between goals and beliefs of different agents working together. Commitments usually involve 2 parties, a *debtor* who commits to a *creditor* by a concrete commitment *content*. Below, we treat commitment and meaning as abstract types defined over some language. In the following we assume that a language $\mathcal{L}$ is a set of formulae coming with a relation of set inclusion $\in$ and a relation of provability $\vdash$ w.r.t. a set of formulae.

The set of commitments over a language $\mathcal{L}$ are formulae constructed as follows:

- $C(c, d, \varphi)$, where $\varphi \in \mathcal{L}$ is a commitment,
  - provided p and r are commitments, then also $C(c, d, p)$ and $C(c, d, p \rightsquigarrow r)$ are commitments as well.

c and d above denote the creditor and the debtor of the commitment. $C^{\mathcal{L}}$ denotes the set of all commitments over $\mathcal{L}$.

Whenever the context is clear, we omit the creditor and debtor designation. In the following, we simply assume that the debtor of a commitment is a particular team member and the creditor is the team in general.

The set of meanings $\mathcal{M}_{\mathcal{L}}$ over $\mathcal{L}$ is defined as follows:

- all formulae $\varphi \in \mathcal{L}$ are meanings in $\mathcal{M}_{\mathcal{L}}$,
  - all commitments $\phi \in C^{\mathcal{L}}$ are meanings in $\mathcal{M}_{\mathcal{L}}$ too, and finally
  - provided ϕ and ψ are meanings, then $\phi \wedge \psi$ and $\neg\phi$ are meanings as well.

A finite set of meanings $M \subseteq \mathcal{S}$ is said to be minimal iff $\top, \bot \in M$ and

$$\forall m_i, m_j \in M : m_i \equiv m_j \Longrightarrow m_i = m_j$$

Furthermore, we say that a set of meanings $F \subseteq M$ is consistent w.r.t. M iff any meaning that is stronger than a meaning from F necessarily also belongs to F, i.e.,

$$\forall m_i \in F, m_j \in M : m_j \vdash m_i \Longrightarrow m_j \in F$$

An action $\phi_{pre} : a : \phi_{post}$ is a tuple of a label a uniquely identifying the action and $\phi \in \mathcal{M}_{\mathcal{L}}$, the associated meaning it causes.

Whenever the context will be clear, since the pre- and post-conditions of actions are conjunctions of literals, we treat them as sets and write $\varphi \in \phi$ whenever ϕ is of the form $\phi = \psi_1 \wedge \cdots \wedge \varphi \wedge \cdots \psi_n$.

A Commitment Machine (CM) is a triple $\langle M, \Delta, F \rangle$, where M is a finite minimal set of meanings over some language $\mathcal{L}$, $F \subseteq M$ denotes a set of finite meanings, a set of meanings consistent w.r.t. M and finally Δ is a finite set of actions.

The standard commitment machines semantics in [76] is defined using the action update function $s \oplus a : \phi = s'$, such that $s \wedge \phi \vdash s'$.

Let $\mathcal{L}$ be a language and $cm = \langle M, \Delta, F \rangle$ a commitment machines over $\mathcal{L}$. Furthermore let S be a set of states and $\pi : S \to 2^{\mathcal{M}_{\mathcal{L}}}$ be a propositional labelling function mapping each state to some set of meanings over $\mathcal{L}$.

An action $\phi_{pre} : \mathbf{a} : \phi_{post}$ induces a transition between two states s and s', also denoted $s \overset{\phi_{pre}:\mathbf{a}:\phi_{post}}{\longrightarrow} s'$, whenever the following conditions are satisfied

- $s \vdash \phi_{pre}$,
  - $s \wedge \phi \vdash s'$,
  - if $s \vdash C(p)$ and $s' \vdash p$, then $s' \nvdash C(p)$,
  - if $s \vdash C(p \rightsquigarrow r)$ and $s' \vdash r$, then $s' \nvdash C(p \rightsquigarrow r)$, and finally
  - if $s \vdash C(p \rightsquigarrow r)$ and $s' \vdash p$ and $s' \nvdash r$, then $s' \vdash C(r)$.

The semantics of cm is defined as a set of all paths $s_1, \ldots, s_n$, such that $s_1 \in M$, for each $1 \leq i < n$ there exists $\phi_{pre} : \mathbf{a} : \phi_{post} \in \Delta$, s.t. $s_i \overset{\phi_{pre}:\mathbf{a}:\phi_{post}}{\longrightarrow} s_{i+1}$ and there exists a final meaning $\psi \in F$, s.t. $s_{i+1} \vdash \psi$.

Distributed Commitment Machines

In **[74]**, Winikoff introduces the framework of *Distributed Commitment Machines*, an extension of the original CM framework. The main idea behind DCMs is that each agent reasons about its commitments locally, however some of its actions result in changes in the global state of commitments in the team, thus communication is necessary in order to notify the parties involved in the updated commitments to adapt their local states accordingly. To that end, the framework of DCMs introduces two important notions: the set of roles agent's play in the mission, and communication queues for each agent.

[Distributed Commitment Machine] A Distributed Commitment Machine (DCM) is characterized by

- a set of roles R,
- a set of possible fluents F,

- a set of possible commitments C,
- a set of possible actions A, where for each action $a \in A$, the following is defined:
  - a role r_a which is supposed to perform that action
  - a set of effects divided into two sets, addition effects E_a^+ and deletion effects E_a^-. Both E_a^+, E_a^- are sets of formulae and are disjoint. Each commitment an action adds, or deletes must be a commitment made by the corresponding role r_a,
  - a set of roles $R_I = R \setminus r_a$ which should be informed upon performing the action a.

The state of a role r is defined by the tuple $S_r \stackrel{def}{=} \langle X, Q \rangle$, where X is a set of formulae (fluents and/or commitments), and Q is a message queue containing messages which are yet to be processed by r. The messages are generated upon performing actions from the DCM protocol specification according to the set of roles which are to be notified upon the action performance (R_I). The state of the overall DCM comprises the set of partial states of the individual roles. I.e., $\mathcal{S} \stackrel{def}{=} \{r_i \mapsto \langle X_{r_i}, Q_{r_i} \rangle \mid r_i \in R\}$.

The system's operational semantics is provided by two types of transitions. Either a role (an agent) performs an action, or a role processes one message from its message queue. Formally, the state change of the role (agent) which performs an action is defined as follows:

$$\begin{aligned} \mathcal{S}^a \stackrel{def}{=} & \{(r_i \mapsto \langle X_i, Q_i \rangle) \mid (r_i \mapsto \langle X_i, Q_i \rangle) \in \mathcal{S} \wedge r_i \notin R_I \wedge r_i \neq r_a\} \\ \cup & \{(r_i \mapsto \langle X_i, Q_i \oplus a \rangle) \mid (r_i \mapsto \langle X_i, Q_i \rangle) \in \mathcal{S} \wedge r_i \in R_I\} \\ \cup & \{(r_a \mapsto \langle \mathcal{N}((X_{r_a} \cup E_a^+) \setminus E_a^-), Q_{r_a} \rangle)\} \end{aligned}$$

where $\mathcal{N}(X)$ denotes normalization of the theory X as defined in [74]. Informally, normalization is a procedure "minimizing" the theory X, i.e., removing all facts which follow from a minimal, normalized, core of the theory. For further details, we refer to the original publication.

The message processing step by a role is formally defined as follows:

$$\begin{aligned} \mathcal{S}_r^a \stackrel{def}{=} & \{(r_i \mapsto \langle X_i, Q_i \rangle) \mid (r_i \mapsto \langle X_i, Q_i \rangle) \in \mathcal{S} \wedge r_i \notin R_I \wedge r_i \neq r_a\} \\ \cup & \{(r_a \mapsto \langle \mathcal{N}((X_{r_a} \cup E_a^+) \setminus E_a^-), Q_{r_a} \ominus a \rangle)\} \end{aligned}$$

where a is the most current message in the message queue Q_{r_a}. The operations $\oplus$ and $\ominus$ add a new/remove a message from the role's message queue.

Final states of a distributed commitment machine are such, in which no commitments remain outstanding and at the same time no messages remain in the message queue.

On DCM usage

DCM is a framework allowing to model both local, as well as team-level commitments between agents and inter-relationships thereof. I.e., each agent possesses its own CM program which i) steers its own actions in that it provides incentives for actions; and ii) it uses the CM specification to correctly delete, update, or derive new commitments whenever it's local CM enforces it, or whenever a team-level public action is executed by another team member and communicated to the team.

5.2.2. Jazzyk/BSM

Taking into consideration our past experience with agent-oriented programming, we choose the state-of-the-art techniques from the field of reactive planning and agent-oriented programming languages as the starting point for our implementation of the flexible mission specification and execution framework. In particular, our design of the fully configurable multi-agent mission simulation framework is based on the *Belief-Desire-Intention* (BDI) agent architecture. The BDI architecture dictates a decomposition of agent's knowledge bases into several components storing information the agent has about its environment, itself and its team peers (beliefs), the information about its goals (in this case, basically the mission specification together with the agent's internally adopted goals) and a representation of the agents plans, or programs, execution of which ensures establishment of the agent's goals, i.e., the agent's beliefs match its goals in that the agent believes that it had already achieved the goals eventually in the future.

Concretely, our approach to the design of the flexible mission execution framework is based on the framework of *Behavioural State Machines* by Novák **[55, 59]** and its associated agent-oriented programming language *Jazzyk*. Using this framework, we implemented a modular BDI architecture, that is similar to that described by Novák and Dix in [56]. In the following, we first briefly introduce the framework of *Behavioural State Machines* and subsequently describe out adaptation and extension of the original *Jazzyk* interpreter, which we developed for the purposes of this project.

Behavioural State Machines In [55], Novák introduced the framework of *Behavioural State Machines* (*BSM*). *BSM* framework is a reactive-planning approach to programming cognitive agents based on the *Belief-Desire-Intention* hybrid architecture. The *BSM* framework draws a clear distinction between the *knowledge representation* and *behavioural* layers within an agent. It thus provides a programming system that clearly separates the programming concerns of *how to represent an agent's knowledge* about, for example, its environment and *how to encode its behaviours*. In the core of the framework is a *generic reactive computational model* inspired by Gurevich's *Abstract State Machines* [17], enabling for efficient structuring of the program code. This section briefly introduces the *BSM* framework. Below, we introduce an extension of the *BSM* framework which we developed for the purposes of this project. For the complete formal description of the *BSM* framework, see [55].

Syntax *BSM* agents are collections of one or more so-called *knowledge representation modules* (KR modules), typically denoted by $\mathcal{M}$, each representing a part of the agent's knowledge base. KR modules may be used to represent and maintain various mental attitudes of an agent, such as knowledge about its environment, or its goals, intentions, obligations, etc. Transitions between states of a *BSM* result from applying so-called *mental state transformers* (*mst*), typically denoted by τ. Various types of mst's determine the behavior that an agent can generate. A *BSM agent* consists of a set of KR modules $\mathcal{M}_1, \ldots, \mathcal{M}_n$ and a mental state transformer $\mathcal{P}$, i.e. $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$; the mst $\mathcal{P}$ is also called an *agent program*.

The notion of a KR module is an abstraction of a partial knowledge base of an agent. In turn, its states are to be treated as theories (i.e. sets of sentences) expressed in the KR language of the module. Formally, a KR module $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$ is characterized by a knowledge representation language $\mathcal{L}_i$, a set of states $\mathcal{S}_i \subseteq 2^{\mathcal{L}_i}$, a set of query operators $\mathcal{Q}_i$ and a set of update operators $\mathcal{U}_i$. A query operator $\models \in \mathcal{Q}_i$ is a mapping $\models : \mathcal{S}_i \times \mathcal{L}_i \to \{\top, \bot\}$. Similarly an update operator $\oplus \in \mathcal{U}_i$ is a mapping $\oplus : \mathcal{S}_i \times \mathcal{L}_i \to \mathcal{S}_i$.

Queries, typically denoted by φ, can be seen as operators of type $\models : \mathcal{S}_i \to \{\top, \bot\}$. A primitive query $\varphi = (\models\phi)$ consists of a query operator $\models \in \mathcal{Q}_i$ and a formula $\phi \in \mathcal{L}_i$ of the same KR module $\mathcal{M}_i$. Complex queries can be composed by means of conjunction $\wedge$, disjunction $\vee$ and negation $\neg$.

Mental state transformers enable transitions from one state to another. A primitive mst $\oslash\psi$, typically denoted by ρ and constructed from an update operator $\oslash \in \mathcal{U}_i$ and a formula $\psi \in \mathcal{L}_i$, refers to an update on the state of the corresponding KR module. Conditional mst's are of the form $\varphi \longrightarrow \tau$, where φ is a query and τ is a mst. Such a conditional mst makes the application of τ depend on the evaluation of φ. Syntactic constructs for combining mst's are: non-deterministic choice $|$ and sequence $\circ$.

[mental state transformer] Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules of the form $\mathcal{M}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{Q}_i, \mathcal{U}_i)$. The set of *mental state transformers* is defined as below:

- **skip** is a *primitive* mst,
  - if $\oslash \in \mathcal{U}_i$ and $\psi \in \mathcal{L}_i$, then $\oslash\psi$ is a *primitive* mst,
  - if φ is a query, and τ is a mst, then $\varphi \longrightarrow \tau$ is a *conditional* mst,
  - if τ and τ' are mst's, then $\tau|\tau'$ and $\tau \circ \tau'$ are mst's (*choice*, and *sequence* respectively).

Even though it is a vital feature of the *BSM* theoretical framework, for simplicity we omit the treatment of variables in the definitions of query and update formulae above. For a full fledged description of the BSM framework consult [55].

Semantics The *yields* calculus, summarized below after [55], specifies an update associated with executing a mental state transformer in a single step of the language interpreter.

It formally defines the meaning of the state transformation induced by executing an mst in a state, i.e. a mental state transition.

Formally, a *mental state* σ of a *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \tau)$ is a tuple $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of KR module states $\sigma_1 \in \mathcal{S}_1, \ldots, \sigma_n \in \mathcal{S}_n$, corresponding to $\mathcal{M}_1, \ldots, \mathcal{M}_n$ respectively. $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$ denotes the space of all mental states over $\mathcal{A}$. A mental state can be modified by applying primitive mst's on it and query formulae can be evaluated against it. The semantic notion of truth of a query is defined through the satisfaction relation $\models$. A primitive query $\models\phi$ holds in a mental state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ (written $\sigma \models (\models\phi)$) iff $\models(\phi, \sigma_i)$, otherwise we have $\sigma \not\models (\models\phi)$. Given the usual meaning of Boolean operators, it is straightforward to extend the query evaluation to compound query formulae. Note that evaluation of a query does not change the mental state σ.

For an mst $\oslash\psi$, we use $(\oslash, \psi)$ to denote its semantic counterpart, i.e., the corresponding *update* (state transformation). Sequential application of updates is denoted by $\bullet$, i.e. $\rho_1 \bullet \rho_2$ is an update resulting from applying ρ_1 first and then applying ρ_2. The application of an update to a mental state is defined formally below.

[applying an update] The result of applying an update $\rho = (\oslash, \psi)$ to a state $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ of a *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$, denoted by $s \bigoplus \rho$, is a new state $\sigma' = \langle \sigma_1, \ldots, \sigma'_i, \ldots, \sigma_n \rangle$, where $\sigma'_i = \sigma_i \oslash \psi$ and σ_i, $\oslash$, and ψ correspond to one and the same $\mathcal{M}_i$ of $\mathcal{A}$. Applying the empty update **skip** on the state σ does not change the state, i.e. $\sigma \bigoplus \mathbf{skip} = \sigma$.

Inductively, the result of applying a sequence of updates $\rho_1 \bullet \rho_2$ is a new state $\sigma'' = \sigma' \bigoplus \rho_2$, where $\sigma' = \sigma \bigoplus \rho_1$. $\sigma \overset{\rho_1 \bullet \rho_2}{\rightarrow} \sigma'' = \sigma \overset{\rho_1}{\rightarrow} \sigma' \overset{\rho_2}{\rightarrow} \sigma''$ denotes the corresponding compound transition.

The meaning of a mental state transformer in state σ, formally defined by the *yields* predicate below, is the update set it yields in that mental state.

[yields calculus] A mental state transformer τ yields an *update* ρ in a state σ, iff $yields(\tau, \sigma, \rho)$ is derivable in the following calculus:

$$\frac{\top}{yields(\mathbf{skip}, \sigma, \mathbf{skip})} \qquad \frac{\top}{yields(\oslash\psi, \sigma, (\oslash, \psi))} \qquad \text{(primitive)}$$

$$\frac{yields(\tau, \sigma, \rho), \sigma \models \phi}{yields(\phi \longrightarrow \tau, \sigma, \rho)} \qquad \frac{yields(\tau, \sigma, \rho), \sigma \not\models \phi}{yields(\phi \longrightarrow \tau, \sigma, \mathbf{skip})} \qquad \text{(conditional)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1), yields(\tau_2, \sigma, \rho_2)}{yields(\tau_1 | \tau_2, \sigma, \rho_1), yields(\tau_1 | \tau_2, \sigma, \rho_2)} \qquad \text{(choice)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1), yields(\tau_2, \sigma \bigoplus \rho_1, \rho_2)}{yields(\tau_1 \circ \tau_2, \sigma, \rho_1 \bullet \rho_2)} \qquad \text{(sequence)}$$

$$\frac{yields(\tau_1, \sigma, \rho_1), \rho_2 \neq \mathbf{skip}}{yields(\tau_1 / \tau_2, \sigma, \rho_1)} \qquad \text{(chain preference)}$$

$$\frac{\forall \rho_1 \colon yields(\tau_1, \sigma, \rho_1) \wedge \rho_1 = \mathbf{skip}, yields(\tau_1, \sigma, \rho_2)}{yields(\tau_1 / \tau_2, \sigma, \rho_2)} \qquad \text{(chain preference)}$$

We say that τ yields an *update set* ν in a state σ iff $\nu = \{\rho | yields(\tau, \sigma, \rho)\}$.

The mst **skip** yields the update **skip**. Similarly, a primitive update mst $\oslash\psi$ yields the corresponding update $(\oslash, \psi)$. In the case the condition ϕ of a conditional mst $\phi \longrightarrow \tau$ is satisfied in the current mental state, the calculus yields one of the updates corresponding to the right hand side mst τ, otherwise the no-operation **skip** update is yielded. A non-deterministic choice mst yields an update corresponding to either of its members and a sequential mst yields a sequence of updates corresponding to the first mst of the sequence and an update yielded by the second member of the sequence in a state resulting from application of the first update to the current mental state. Finally, not appearing in the original *BSM* framework by Novák, we added the chain preference operator / which executes only the first, non-empty (**skip**) update yielded in the sequence of mst's.

The following definition articulates the denotational semantics of the notion of mental state transformer as an encoding of a function mapping mental states of a *BSM* to updates, i.e. transitions between them.

[mst functional semantics] Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be KR modules. A mental state transformer τ encodes a function $\mathfrak{f}_\tau : \sigma \mapsto \{\rho | yields(\tau, \sigma, \rho)\}$ over the space of mental states $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle \in S_1 \times \cdots S_n$.

Subsequently, the semantics of a *BSM* agent is defined as a set of traces in the induced transition system enabled by the *BSM* agent program.

[BSM semantics] A *BSM* $\mathcal{A} = (\mathcal{M}_1, \ldots, \mathcal{M}_n, \mathcal{P})$ can make a step from state σ to a state σ', iff $\sigma' = \sigma \bigoplus \rho$, s.t. $\rho \in \mathfrak{f}_\mathcal{P}(\sigma)$. We also say, that $\mathcal{A}$ induces a (possibly compound) transition $\sigma \xrightarrow{\rho} \sigma'$.

A possibly infinite sequence of states $\sigma_1, \ldots, \sigma_i, \ldots$ is a run of BSM $\mathcal{A}$, iff for each $i \geq 1$, $\mathcal{A}$ induces a transition $\sigma_i \rightarrow \sigma_{i+1}$.

The semantics of an agent system characterized by a BSM $\mathcal{A}$, is a set of all runs of $\mathcal{A}$.

Additionally, we require the non-deterministic choice of a *BSM* interpreter to fulfill the *weak fairness condition*, similar to that in [50], for all the induced runs.

Condition 1 (weak fairness condition)**.** *A computation run is weakly fair iff it is not the case that an update is always yielded from some point in time on but is never selected for execution.*

Jazzyk *Jazzyk* is a programming language implementing the computational model of the *BSM* framework. Originally introduced in [55], the language comes with a standalone interpreter which can transparently incorporate a number of heterogeneous knowledge representation modules ranging from logic-programming-based, object-oriented programming language interpreters to interfaces to robotic simulators. However, since the *A-Globe/AgentFly* technological infrastructure is based on the *Java Virtual Machine* run-time environment, the original language interpreter was not a suitable option for use in the context of this project. For the purposes of this project, we ported the *Jazzyk* programming language to the *Java* platform and developed a new incarnation of the *Jazzyk* interpreter capable of running in the *Java Virtual Machine*. The implemented full-fledged language interpreter allowed us to implement the behaviours of agents in our MAS simulation in a highly

modular and flexible manner. The heterogeneity of the *BSM/Jazzyk* suite allowed us to exploit the strengths of declarative technologies, such as *Prolog* together with object-oriented technologies, i.e., *Java* language as a natural components of a BDI-based architecture for behaviour-rich , heavily deliberating, yet responsive and reactive agents in our simulated missions.

Language The original syntax of the *Jazzyk* language is an instantiation of the abstract mathematical syntax of the *BSM* theoretical framework. when then construct encodes a conditional *mst* $\phi \longrightarrow \tau$. Symbols ; and , stand for choice | and sequence $\circ$ *BSM* operators respectively. To facilitate operator precedence, mental state transformers can be grouped into compound structures, blocks, using curly braces {...}. Since the original *BSM* framework did not include the *unless* operator /, we additionally implemented it in our extension.

Interpreter Our implementation of the Jazzyk programming language, called *JazzykJVM*, was developed in a modern functional-object-oriented programming language *Scala*[1] which compiles into *Java* byte code, in result featuring a tight and transparent integration with the *Java* infrastructure.

Together with the *Jazzyk* programming language interpreter, we implemented two general-purpose knowledge representation modules: *Prolog* module and *Java* module. As the basis for the *Prolog* module accessible from the *Java* platform, we used the *tuProlog* v2.1.1[2] developed by *Department of Electronics, Informatics and Systems* of *Alma Mater Studiorum-Università di Bologna*, Italy. The *Prolog* module allowed us to exploit the conciseness and declarative language strengths of logic-programming. The *Java* module facilitating object-oriented is based on a *Java* scripting language interpreter *BeanShell*[3]. We describe the concrete usage of the modules within our implementation later in this chapter.

To better support source code modularity and re-usability, *Jazzyk* interpreter integrates a *Java* implementation of the popular *GNU M4*[4], a state-of-the-art macro preprocessor. The concrete *M4 Java* port we used is the *MNI Macro Processor (MMP)* package developed by Burkhardt Renz at *University of Applied Sciences*, GieSSen-Friedberg, Germany.

Macros are a powerful tool for structuring and modularizing and encapsulating the source code and writing code templates. Before feeding the *Jazzyk* agent program to the language interpreter, first all the M4 macros are expanded and only afterwards the plain *Jazzyk* program is fed to the interpreter for execution.

[1] `http://www.scala-lang.org/`
[2] `http://tuprolog.alice.unibo.it/`
[3] `http://www.beanshell.org/`
[4] `http://www.gnu.org/software/m4/`

5.2.3. Implementation description

The framework facilitating flexible mission specification and execution is heavily based on exploitation of the strengths of the framework of *Behavioural State Machines* and the associated programming language *Jazzyk*. In the following, we briefly describe the agent architecture developed for the purposes of the project and the details of mission-specific agent behavior implementation. For illustration purposes, we only provide a description of the implementation of ground allied agents whose implementation features the richest range of behaviours.

Agent architecture Above, we already noted that the architectural decomposition of the simulated entities is heavily inspired by the BDI architectural scheme. The agent architecture was implemented for utilization in the context of a mission described in the previous project report.

The agents' belief base is decomposed into two sub-modules: a *Prolog*-based component storing the agent's information about its environment and itself, and a *Java* component. The former stores primarily persistent information such as the agent's belief about being (in-)secure, or static information about the meeting point location, safe house location once received from the UAV team, etc., together with a logic-based inference engine allowing to draw deductive conclusions from this information. The second sub-component facilitates mainly manipulation with information about the topology of the simulated environment, i.e., the map of the urban area of the operations theater and the related reasoning mechanisms, such as e.g., path planning on the map graph, etc.

The traditional role of a goal in BDI architecture is replaced by a commitment. An internal goal of an agent is modelled as a commitment to itself, i.e. the agent is both the creditor and debtor of the commitment. For the simplicity of implementation, we consider an agent's commitment as a special belief stored together with other beliefs in the Prolog component of the agent's belief base. This way we can easily implement interactions between the agent's beliefs and commitments.

The Figure 5.1 provides a schematic overview of the developed agent architecture.

Commitment Machines in Jazzyk

As mentioned above,an agent's commitments are stored in its Prolog belief base. To distinguish commitments from other beliefs and control the dynamics of commitments, we have introduced three special-use predicates shown in Table 5.1. In Figure 5.2, we present an example Prolog belief base containing a commitment to be at position (n5,n15) and a rule specifying that the commitment will be discharged as soon as the agent starts believing that it is indeed at position (n5,n15).

Agents' overall behavior is implemented as a set of self-encapsulated composable behaviours encoded as reactive plans, policies, in the programming language *Jazzyk*. The commitments are used to motivate the behavior of the agents, to govern their interactions

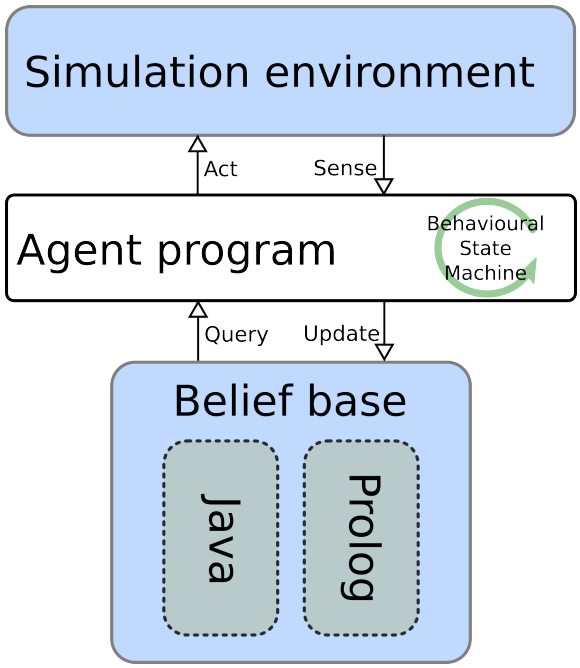


Figure 5.1.: Architecture of ground agents in the *AgentScout2* simulation.

$c(\varphi)$	denotes that the agent is committed to φ
$cc(\varphi, c(\psi))$	denotes a conditional commitment, i.e. whenever an agent beliefs φ, it adopts the commitment ψ
$ccd(\varphi, c(\psi))$	denotes a conditional commitment deletion, i.e. whenever an agent beliefs φ, it discharges the commitment ψ[5]

Table 5.1.: Commitment related predicates

and to specify their joint mission. An example of commitment machine specifying the mission from the perspective of the leader agent is listed in Figure 5.3.

The first three lines contain logical derivation rules used to derive facts about the world. The fourth line contains a commitment denoting that the leader agent is committed to be at the location of safehouse. On the following two lines, we can see a conditional commitment prescribing that once the whole team is at the location of safehouse, a new commitment c(extracted(vip)) is adopted. Similarly, once the VIP is extracted, the leader will commit to be at the collection location. Finally, the two following lines specify when can be commitments discharged. The leader can only discharge a commitment of being at a certain location if his entire team is there as well. The commitment to extract the VIP can be discharged as soon as the leader agents beliefs that the VIP has been extracted.

The Jazzyk agent program contains behaviours that actively pursue the current com-

```
c(at(n5,n12)).
ccd(at(N,T),c(at(N,T))).
```

Figure 5.2.: Example of commitment machine in Prolog

```
front_soldiers_at(N,T) :- at(adam,N,T), at(brian,N,T).
back_soldiers_at(N,T) :-  at(chris,N,T), at(david,N,T).
all_soldiers_at(N,T) :- front_soldiers_at(N,T), back_soldiers_at(N,T).
c(at(SAFEHOUSE_N, SAFEHOUSE_T)).
cc( (at(SAFEHOUSE_N,SAFEHOUSE_T),all_soldiers_at(N,T)), c(extracted(vip)) ).
cc(extracted(vip), c(at(COLLECTION_N, COLLECTION_T))).
ccd( (at(N,T), all_soldiers_at(N,T)), c(at(N,T)) ).
ccd( extracted(P), c(extracted(P)) ).
```

Figure 5.3.: The belief base of the leader agent

mitments of the agent. An example of a behavior that pursues the commitment c(at(N,T)) is listed in Figure 5.4.

Additionally, agents often feature implicit behaviours, i.e., such which were not provided in the mission specification, but the agent adopts their associated goals because of its own decision in run-time. The Figure 5.5 lists an example of such a behavior implementing an agent's reaction when a foe character is encountered.

Finally, the behaviours are composed into an agent program. Using the *BSM* composition operators, the agent program defines the interrelationships among the behaviours of the agent. In the case of our implementation, listed in Figure 5.6, the behaviours are prioritized using chain preference operator. In result, for example, the agent prefers to recover from collisions and react to foe characters over the commitment-pursuing behaviors.

5.3. Evaluation and experiments

5.3.1. Scenario

In order to evaluate the proposed techniques, we have implemented a multi-agent simulation of the following tactical scenario. We assume a scenario of a simulated military operation of a ground team in an urban area. We simulate a military evacuation mission. A team of allied troops traverses the urban area and approaches a safe house located at an apriori known map position, where the team extracts the VIP. After completing the extraction task, the troops meet at the collection location where the mission finishes. Throughout the whole mission, the allied forces maintain their formation so that they cover the surrounding area and protect themselves. Besides the team of allied forces, there is a number of foe characters operating in the urban area. The foe characters are generally trying to avoid allied troops and hide away. Nevertheless, every time an allied troop spots a foe character, it starts monitoring it until the threat disappears.

```
define('PURSUE_AT_COMMITMENT', '
{
  when query beliefs (Target, Current) [{ c(at(Target,_)), at(Current,_). }] then
  {
    when query beliefs (Target) [{ plan(Target, _). }] then
    {
      /* If the front line is on their positions, make a move */
      when query beliefs (Target,Next,T) [{ plan_head(Target,Next,T), front_soldiers_at(Next,T). }]
           and query body (Next,T) [{ self.isOneStepAway(Next,T) }]
           and query body [{ self.getMovementTarget() == null }] then
      {
        act (Next,T) [{ self.goToNodeAndTurn(Next,T); }]
      },

      /* Pop first element of the plan */
      when query beliefs (Target, Current, Rest)
           [{ plan_head(Target,Current,T), all_soldiers_at(Current,T), plan(Target, [Current|Rest]) }] then
      {
        update beliefs (Rest) [{ update_plan(Target, Rest). }]
      },

      /* Order the team to do the next step */
      when query beliefs (TargetNode,N,T) [{ plan_head(Target,N,T), not(next(N,T)).}] then
      {
        update beliefs (N,T) [{ action(next(N,T)). }],
        update body (N,T) [{ self.broadcast("action(next("+N+","+T+"))"); }]
      }

    } else
    {
      /* Plan a path to the target node */
      when query body (Current, Target, Plan) [{ Plan = self.planPath(Current, Target); true; }] then
        update beliefs (Target, Plan) [{ update_plan(Target, Plan).}]
    }
  }
} ')
```

Figure 5.4.: *Jazzyk* code implementing the behavior that pursues the agent's `c(at(N,T))` commitment

```
define('REACT_TO_FOES', '
{
  when sense [{ self.seesFoes() }] then
  {
    when not B [{ sees_foes. }] then
    {
      update beliefs [{ does_see_foes. }],
      when sense [{ self.isMoving() }] then act [{ self.halt(); }],
      act [{ self.turnToClosestFoe(); }]
    }
  },

  when B [{ sees_foes. }] and not sense [{ self.seesFoes() }] then
  {
    update beliefs [{ does_not_see_foes. }]
  }
}
')
```

Figure 5.5.: *Jazzyk* code implementing the agent's reaction to foe characters

```
SENSE_CURRENT_NODE,
PROCESS_MESSAGES,
PROCESS_COMMITMENTS,
RECOVER_FROM_COLLISION /
{
        REACT_TO_FOES /
        PURSUE_COMMITMENTS
}
```

Figure 5.6.: The top-level *Jazzyk* code of allied team leader

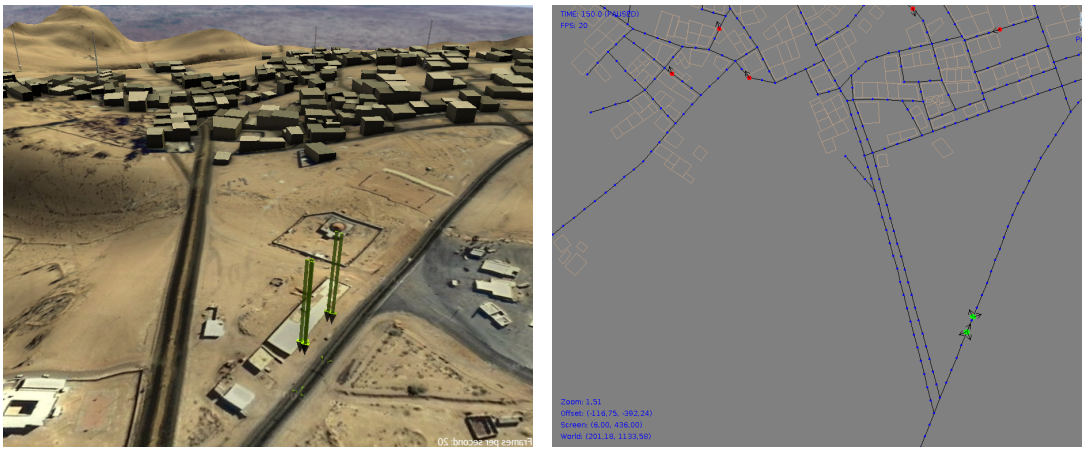


Figure 5.7.: 3D visualization and schematic visualization of the simulated environment

We have implemented the agents in the scenario using the aforementioned techniques and evaluated their performance in a multi-agent simulation. The Figure 5.7 shows the 3d visualization and the schematic visualization of the simulated environment. The green icons (spots) represent the allied troops, the red icons (spots) represent the foe characters. The agents operate in a village consisting of over 300 buildings. The street network of the urban area is represented by a graph which has 142 nodes and 176 edges.

The allied troops plan their route to the safe house using the A* algorithm and consequently execute it by means of a commitment machine which ensure that the squad moves in a proper formation. The formation typically maintained by the team is shown in Figure 5.8.

On their way to the safe house, and back to the collection point, the allied team may come across foe characters that are randomly moving in the environment. Such an encounter will interrupt the moving, since the team has to deal with a potential threat. In our scenario, the team members that spot a foe (the ones for which the foe is not occluded by any of the buildings) halt their movement and monitor the threat, i.e. turn at the foe. An example of such an encounter is shown in Figure 5.9. Once the threat disappears, the coordinated movement should continue from where it was interrupted. We included these situations in the scenario in order to test the modularity and robustness of our approach. We first developed a commitment machine that encoded the coordinated movement. Later, we added the behavior that handles the encounters with foe characters. The new behavior could be included in the agent program with almost no modification in the existing code.

Further we were testing the effort needed to change the formation pattern in which the allied squad traverses the area. The two possible patterns of movement are depicted in Figure 5.10. By default we use the pattern illustrated in the first row. The change of the formation pattern to that shown in the second row is only matter of one minor change in the behavior that pursues the commitment, i.e. the behavior whose code is listed in

Figure 5.8.: Typical formation of the allied team

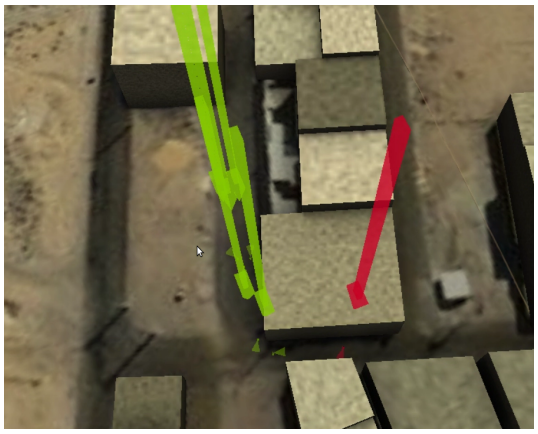


Figure 5.9.: One of the allied troops encounters a foe character.

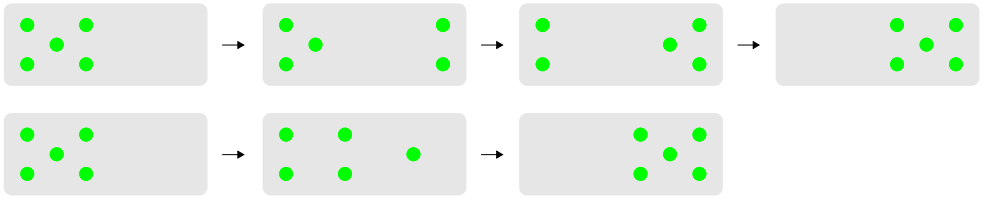


Figure 5.10.: Two patterns of allied team formation movement

Figure 5.4. In particular, the condition on line 8 needs to be replaced by the following.

```
when query beliefs (Target,Next,T) [{ plan_head(Target,Next,T), all_soldiers_at(Next,T). }]
    and query body (Next,T) [{ self.isOneStepAway(Next,T) }]
    and query body [{ self.getMovementTarget() == null }] then ...
```

Based on these experiments, we consider the chosen approach suitable for concise specification of agents and especially their interactions. It exhibits high degree of flexibility and good elaboration tolerance.

6. Discussion and outlook

We conclude the project final report with an outlook and discuss possibilities for future work along the lines of research we performed in the context of this project.

6.1. Adversarial planning: patrolling of mobile targets

In this workpackage we developed a novel game-theoretic model – patrolling games with mobile targets – together with algorithms that are able to solve these games. Moreover, we successfully integrated this game-theoretic model within the Tactical AgentScout 2 project and we improved the performance in the problem of protecting convoys crossing an adversarial area compared to the existing algorithms.

Successful results of this workpackage give several directions that can be further explored. From the theoretic view, the notion of the distance graph has to be analyzed in more detail to establish a clear definition of a usable abstraction for the multi-target scenarios. Proper definition of the distance graph and proofs of the equivalence (or approximation) of the solutions will further improve the applicability of the game-theoretic model of patrolling games to real-world scenarios. Moreover, the relation to the security games, and the extensive form of security games need to be established in order to improve the algorithms computing the solutions of patrolling games, and to utilize existing fast algorithms for computing solutions in security games.

On the other hand, several practical questions have appeared during the implementation phase of the workpackage. Currently, the patroller had only a single task – to follow the randomized policy in the patrolling scenario. However, while creating a more complex behavior of an agent a combination of such tasks can be desirable. Hence new methods of implementation of the randomized policy to the agents' behavior will have to be explored. Completing this task will improve possibilities of deployment of this approach on real unmanned aerial systems.

6.2. Adversarial planning: modelling smart targets

In the workpackage WP2, we developed algorithms for controlling a smart target that actively avoids detection, tracking and capture by a team of pursuers. Furthermore, we developed algorithms for the team of pursuers that perform these tasks. The techniques are relatively computationally demanding and in result do not scale very well with the number of players in the game. As one of the potential vectors of further development, in

our future work we will attempt to speed up the approach by incorporation of procedural background knowledge about the particular instances of the pursuit-evasion game into the state-space search. Another interesting, and highly actual vector of further research along these lines is decentralization of the proposed algorithms and consequently their scalability to larger numbers of players. Finally, a lesson learned from the work on the finished project is that in the future incarnations of the technique, much more attention has to be paid to performance tuning of various domain-dependent parameters. As one of interesting lines of further work is development of heavy simulations for the Monte Carlo search algorithms.

6.3. Multi-agent re-planning and plan repair

In the workpackage WP3, we successfully designed, implemented and evaluated a series of multi-agent plan repairing algorithms. We finally also integrated the algorithms with the simulation platform and developed an integrated scenario employing one of the plan repair techniques as a means of coordination of two, otherwise independent, multi-agent teams.

In our future work, we plan to further extend this line of research and work towards a number of open challenges. In particular, we will perform a more thorough evaluation of the more advanced multi-agent plan repair algorithms (BRA and LRA) in both synthetic, as well as real-world scenarios. We plan to use the research community benchmarks from the ICAPS planning competition, as well as implement further multi-agent planning/plan repair scenarios in the future projects of our research group. In order to facilitate easy integration with software components using general planning algorithms and test our approaches in head-to-head settings, we plan to implement modules facilitating standardized planning input languages, such as e.g., PDDL.

One of the observations from the finished project is that there is still a lot of space for further improvements in multi-agent plan repair algorithms. One of the interesting vectors is a thorough analysis of the problem's communication complexity and the related trade-offs w.r.t. resulting plan quality.

6.4. Coordination and teamwork

In the context of the project, we advanced our approach to mission specification frameworks, we started to work towards in the context of the project Tactical AgentFly 2. Even though the results are promising, they leave quite a lot to be desired. In particular, in our future work, we will work toward development of a complete formal model of teamwork. The approach used in this project was based upon the formalism of *Distributed Commitment Machines* and, even though it seemed to be promising at the beginning of the project, during the application of the approach to the multi-robot scenario, we identified a number of issues with it. One of most pressing problems is its tight integration with particular actions/capabilities of the team members. In the future, we will work towards development of a purely declarative framework, which will allow us to specify missions in terms

of plain achievement and maintenance goals, completely disregarding the particularities of the agents' capabilities and treating these as pluggable behaviors.

Bibliography

[1] IBM ILOG CPLEX optimizer, `http://www.ibm.com/software/integration/optimization/cplex-optimizer`.

[2] N. Agmon, S. Kraus, and G. A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *ICRA*, pages 2339–2345, 2008.

[3] N. Agmon, V. Sadov, G. A. Kaminka, and S. Kraus. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *AAMAS*, pages 55–62, 2008.

[4] T.-C. Au, H. Muñoz-Avila, and D. S. Nau. On the complexity of plan adaptation by derivational analogy in a universal classical planning framework. *Advances in Case-Based Reasoning*, pages 199–206, 2002.

[5] A. Aziz, V. Singhal, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Proceedings of CAV*, volume 939 of *LNCS*, pages 155–165, 1995.

[6] A. Baltag. A logic for suspicious players. *Bulletin of Economic Research*, 54(1):1–46, 2002.

[7] T. Bandyopadhyay, Y. Li, M. H. A. Jr., and D. Hsu. Stealth tracking of an unpredictable target among obstacles. In *Proceedings of the International Workshop on the Algorithmic Foundations of Robotics*, 2004.

[8] N. Basilico, N. Gatti, and F. Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, volume pages, pages 57–64. International Foundation for Autonomous Agents and Multiagent Systems, 2009.

[9] N. Basilico, N. Gatti, T. Rossi, S. Ceppi, and F. Amigoni. Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings. In *WI-IAT*, pages 557–564, 2009.

[10] N. Basilico, N. Gatti, and F. Villa. Asynchronous Multi-Robot Patrolling against Intrusion in Arbitrary Topologies. In *AAAI 2010*, 2010.

[11] R. Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6:679–684, 1957.

[12] K. Bennett and O. Mangasarian. Bilinear separation of two sets inn-space. *Computational Optimization and Applications*, 2(3):207–227, 1993.

[13] S. Bhattacharya, S. Candido, and S. Hutchinson. Motion strategies for surveillance. In *Robotics: Science and Systems*, 2007.

[14] S. Bhattacharya and S. Hutchinson. Approximation schemes for two-player pursuit evasion games with visibility constraints. In *Robotics : Science and Systems IV*, 2008.

[15] S. Bhattacharya and S. Hutchinson. On the existence of nash equilibrium for a two-player pursuit-evasion game with visibility constraints. *Int. J. Rob. Res.*, 29(7):831–839, 2010.

[16] R. H. Bordini, L. Braubach, M. Dastani, A. E. F. Seghrouchni, J. J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30:33–44, 2006.

[17] E. Börger and R. F. Stärk. *Abstract State Machines: A Method for High-Level System Design and Analysis*. Springer, 2003.

[18] B. Bosansky, V. Lisy, M. Jakob, and M. Pechoucek. Computing time-dependent policies for patrolling games with mobile targets. In *Tenth International Conference on Autonomous Agents and Multiagent Systems (to appear)*, 2011.

[19] R. I. Brafman and C. Domshlak. From one to many: Planning for loosely coupled multi-agent systems. In J. Rintanen, B. Nebel, J. C. Beck, and E. A. Hansen, editors, *ICAPS*, pages 28–35. AAAI, 2008.

[20] G. W. Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 1951.

[21] V. Conitzer and T. Sandholm. Computing the optimal strategy to commit to. In *Proc. of the 7th ACM conference on Electronic commerce*, pages 82–90. ACM, 2006.

[22] J. Derenick, J. Spletzer, and A. Hsieh. An optimal approach to collaborative target tracking with performance guarantees. *J. Intell. Robotics Syst.*, 56(1-2):47–67, 2009.

[23] P. Drake and S. Uurtamo. Move ordering vs heavy playouts: Where should heuristics be applied in Monte Carlo Go. In *Proceedings of the 3rd North American Game-On Conference*. Citeseer, 2007.

[24] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 995–1072. Elsevier Science Publishers, 1990.

[25] P. Fabiani, H. H. Gonzalez-Banos, J. C. Latombe, and D. Lin. Tracking an unpredictable target among occluding obstacles under localization uncertainties. *Robotics and Autonomous Systems*, 38(1):31 – 48, 2002.

[26] A. Farinelli, A. Rogers, and N. R. Jennings. Bounded Approximate Decentralised Coordination using the Max-Sum Algorithm. In *Distributed Constraint Reasoning 2009*.

[27] H. Finnsson and Y. Bj
"ornsson. Learning Simulation Control in General Game-Playing Agents. In *Proceeding of AAAI*, 2010.

[28] B. P. Gerkey, S. Thrun, and G. J. Gordon. Visibility-based pursuit-evasion with limited field of view. *I. J. Robotic Res.*, 25(4):299–315, 2006.

[29] G. J. Gordon. No-regret algorithms for online convex programs. In *Advances in Neural Information Processing Systems (NIPS)*, pages 489–496, 2006.

[30] L. J. Guibas, J.-C. Latombe, S. M. LaValle, D. Lin, and R. Motwani. A visibility-based pursuit-evasion problem. *Int. J. Comput. Geometry Appl.*, 9(4/5):471–, 1999.

[31] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[32] D. Harel and D. Kozen. Process logic: Expressiveness, decidability, completeness. *Journal of Computer and System Sciences*, 25(2):144–170, 1982.

[33] I. Harmati and K. Skrzypczyk. Robot team coordination for target tracking using fuzzy logic controller in game theoretic framework. *Robotics and Autonomous Systems*, 57(1):75 – 86, 2009.

[34] T. Ishida and R. E. Korf. Moving target search. In *IJCAI'91: Proceedings of the 12th international joint conference on Artificial intelligence*, pages 204–210, San Francisco, CA, USA, 1991. Morgan Kaufmann Publishers Inc.

[35] V. Isler, S. Kannan, and S. Khanna. Randomized pursuit-evasion with local visibility. *SIAM J. Discret. Math.*, 20(1):26–41, 2006.

[36] V. Isler and N. Karnad. The role of information in the cop-robber game. *Theoretical Computer Science*, 399(3):179–190, June 2008.

[37] M. Jain, E. Karde, C. Kiekintveld, F. Ordóñez, and M. Tambe. Optimal defender allocation for massive security games: A branch and price approach. In *Autonomous Agents and Multi-Agent Systems*, 2010.

[38] J. G. Kemeny, L. J. Snell, and A. W. Knapp. *Denumerable Markov Chains*. Van Nostrand, 1966.

[39] C. Kiekintveld, M. Jain, J. Tsai, J. Pita, F. Ordóñez, and M. Tambe. Computing optimal randomized resource allocations for massive security games. In *AAMAS*, pages 689–696, 2009.

[40] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. *Lecture Notes in Computer Science*, 4212:282, 2006.

[41] A. Kolling and S. Carpin. Multi-robot surveillance: an improved algorithm for the graph-clear problem. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2360–2365. IEEE, 2008.

[42] D. Korzhyk, V. Conitzer, and R. Parr. Complexity of Computing Optimal Stackelberg Strategies in Security Resource Allocation Games. In *AAAI Conf. on Artificial Intelligence*, 2010.

[43] D. Korzhyk, V. Conitzer, and R. Parr. Security games with multiple attacker resources. In *IJCAI (to appear)*, 2011.

[44] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling. Monte carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1078–1086, 2009.

[45] J. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[46] S. M. LaValle, H. H. GonzÃąlez-BaÃśos, C. Becker, and J. claude Latombe. Motion strategies for maintaining visibility of a moving target. In *In Proc. of the IEEE International Conference on Robotics & Automation (ICRA*, pages 731–736, 1997.

[47] S. M. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. In *ICRA*, pages 1677–1682, 1999.

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

[49] V. Lisý, B. Bošanský, R. Vaculín, and M. Pěchouček. Agent subset adversarial search for complex non-cooperative domains. In *IEEE Conference on Computational Intelligence and Games*, pages 211–218, 2010.

[50] Z. Manna and A. Pnueli. *The temporal logic of reactive and concurrent systems.* Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[51] C. Moldenhauer and N. R. Sturtevant. Evaluating strategies for running from the cops. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*, pages 584–589, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[52] K. C. Nguyen, T. Alpcan, and T. Basar. Security Games with Decision and Observation Errors. *Computing Research Repository*, pages 510–515, 2010.

[53] R. Nissim, R. I. Brafman, and C. Domshlak. A general, fully distributed multi-agent planning algorithm. In W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *AAMAS*, pages 1323–1330. IFAAMAS, 2010.

[54] R. Nissim, R. I. Brafman, and C. Domshlak. A general, fully distributed multi-agent planning algorithm. In *Autonomous Agents and Multiagent Systems*, pages 1323–1330, 2010.

[55] P. Novák. Jazzyk: A programming language for hybrid agents with heterogeneous knowledge representations. In *Proceedings of the Sixth International Workshop on Programming Multi-Agent Systems, ProMAS'08*, volume 5442 of *LNAI*, pages 72–87, May 2008.

[56] P. Novák and J. Dix. Modular BDI architecture. In H. Nakashima, M. P. Wellman, G. Weiss, and P. Stone, editors, *AAMAS*, pages 1009–1015. ACM, 2006.

[57] P. Novák and W. Jamroga. Code patterns for agent oriented programming. In *Proceedings of AAMAS'09*, pages 105–112, 2009.

[58] P. Novák and W. Jamroga. Agents, Actions and Goals in Dynamic Environments. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, Barcelona, Spain; IJCAI 2011*, July 2011.

[59] P. Novák and M. Köster. Designing goal-oriented reactive behaviours. In *Proceedings of the 6th International Cognitive Robotics Workshop, CogRob 2008, ECCAI co-located workshop, July 21-22 in Patras, Greece*, pages 24–31, July 2008.

[60] A. Parker, D. Nau, and V. Subrahmanian. Overconfidence or paranoia? search in imperfect-information games. In *AAAI Conf. on Artificial Intelligence*, volume 21, page 1045, 2006.

[61] A. Parker, D. S. Nau, and V. S. Subrahmanian. Paranoia versus overconfidence in imperfect-information games. In *Heuristics, Probabilities, and Causality: A Tribute to Judea Pearl*, pages 63–87, 2010.

[62] P. Paruchuri, J. Pearce, J. Marecki, M. Tambe, F. Ordonez, and S. Kraus. Playing games for security: an efficient exact algorithm for solving Bayesian Stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, number Aamas, pages 895–902. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[63] P. Paruchuri, J. P. Pearce, M. Tambe, F. Ordonez, and S. Kraus. An Efficient Heuristic for Security Against Multiple Adversaries in Stackelberg Games. In *AAMAS*, 2007.

[64] J. Pita, M. Jain, J. Marecki, F. Ordó nez, C. Portway, M. Tambe, C. Western, P. Paruchuri, and S. Kraus. Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles Int. Airport. In *AAMAS*, pages 125–132, 2008.

[65] A. Pnueli. The temporal logic of programs. In *Proceedings of FOCS*, pages 46–57, 1977.

[66] E. Raboin, D. Nau, U. Kuter, S. Gupta, and P. Svec. Strategy generation in multi-agent imperfect-information pursuit games. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 947–954. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[67] R. Rosner and A. Pnueli. A choppy logic. In *Proceedings of LICS*, pages 306–313, 1986.

[68] M. Shafiei, N. Sturtevant, and J. Schaeffer. Learning Simulation Control in General Game-Playing Agents. In *IJCAI Workshop on General Game Playing*, 2009.

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

[70] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM J. Comput.*, 21(5):863–888, 1992.

[71] J. Tsai, S. Rathi, C. Kiekintveld, F. Ordóñez, and M. Tambe. IRIS - A Tool for Strategic Security Allocation in Transportation Networks Categories and Subject Descriptors. In *AAMAS*, pages 37–44, 2009.

[72] R. van der Krogt and M. de Weerdt. Self-interested planning agents using plan repair. In *Proceedings of the ICAPS 2005 Workshop on Multiagent Planning and Scheduling*, pages 36–44, 2005.

[73] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: Theory, implementation and experimental evaluation. *IEEE Transactions on Robotics and Automation*, 18:662–669, 2002.

[74] M. Winikoff. Implementing flexible and robust agent interactions using distributed commitment machines. *Multiagent and Grid Systems*, 2(4):365–381, 2006.

[75] Z. Yin, D. Korzhyk, C. Kiekintveld, V. Conitzer, and M. Tambe. Stackelberg vs. Nash in security games: Interchangeability, equivalence, and uniqueness. In *AAMAS*, pages 1139–1146, 2010.

[76] P. Yolum and M. P. Singh. Commitment machines. In J.-J. C. Meyer and M. Tambe, editors, *ATAL*, volume 2333 of *Lecture Notes in Computer Science*, pages 235–247. Springer, 2001.

[77] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems (NIPS)*, 20:1729–1736, 2008.

A. Demonstrators

One of the main deliverables of the Tactical AgentScout 2 project is a series of executable demonstrators showcasing the achievements of the individual workpackages, together with a commented videos annotating their execution. In the following, we briefly describe the demonstrators and videos linking them to the individual workpackages.

Prerequisities

WP1: Patrolling of mobile targets

Demonstrators and videos

wp1-patrolling the demonstrator shows patrolling scenario in 3D urban environment with two convoys protected by a single VTOL. The task of the convoys is to drive through the urban area according to some pre-defined path. The task of the VTOL is to guard the convoys by keeping them in the range of its sensors, depicted by a red cone. The VTOL flies according to a pre-planned randomized trajectory so as to minimize the probability that the convoys will be attacked during the period when the VTOL is not watching the particular convoy.

WP2: Modelling of smart targets

Demonstrators and videos

wp2-full-info the demonstrator shows two pursuers (green) attempting to detect, track and finally capture an evader (red) in a 3D urban environment. The players are moving in a synchronous manner and try to react to each others moves in the mest possible fashion (best-response strategy).

wp2-heterogeneous shows the pursuit-evasion game with a heterogeneous team of pursuers. One of the pursuers is a fixed-wing UAV with a cone camera sensor. The demonstrator showcases the flexibility of the used approach allowing straightforward integration of heterogeneous assets with differing properties w.r.t. the movement model, speed and type of sensors used.

wp2-occlusions demonstrates the pursuit-evasion game with incomplete information and highlights the visibility model used by the players and its consequences on their behaviour computed by the developed adversarial planning algorithms.

WP3: Multi-agent re-planning and plan repair

Demonstrators and videos

wp3-plan-repair the demonstrator showcases the functionality of the plan repair algorithm. A squad of troops is moving on the ground (green) and jointly works towards accomplishing a mission modelled as VIP evacuation from a safe house in the town. The team is supported by two Skeldar VTOLS providing an overwatch for the team (red cone camera sensor) and two small-sized AESIR Vidar single-rotor VTOLS providing autonomous reconnaissance and cover services to the team. As the squad moves through the town, the multi-robot team is continuously adapting its plans so as to both: satisfy their own mission (obtain imagery from the area around the safe-house) and at the same time, provide protection to the human squad.

WP4: Coordination and teamwork

Demonstrators and videos

wp4-teamwork the final demonstrator shows the detailed behaviour of the human squad from the demonstrator of the workpackage WP3. The team is moving through the 3D urban area so as to continuously protect itself and flexibly react to unexpected interruptions from the environment – modelled as encounters with red troops. When a green team member spots a red unit in its visibility range, it points its sensors to it and waits until the entity moves away. The behaviour of the red troops is to model avoidance of the red troops which watch it (modelling an adversarial encounter between the two).