Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Computer Science

DIPLOMA THESIS AGREEMENT

Student: Jan Starý

Study programme: Open Informatics
Specialisation: Artificial Intelligence

Title of Diploma Thesis: Scalable Probabilistic Planning for Decoupled Tasks using Multi-Agent Decomposition Guidelines:

Guidelines:

Multi-agent probabilistic planning (Dec-POMDP) is computaitionally hard and not scalable, as such problems induce a joint-action space with all possible combinations of agents' actions, thus resulting in a blowup exponential in the number of agents. In many real-world problems, the agents interactions are limited to the shared environment and the utility of the resulting solution. The thesis builds on the latest techniques of probabilistic and multi-agent planning in order to improve scalability of planning for such decoupled problems.

1) Study literature on multi-agent, classical and probabilistic planning
2) Formulate a theoretical model with desirable properties
3) Propose a solution for 2)
4) Implement the solution for a selected large-scale real-world problem
5) Experimentally evaluate the solution

Bibliography/Sources:

[1] Ronen I. Brafman, Carmel Domshlak: On the complexity of planning for agent teams and its implications for single agent planning. Artif. Intell. 198: 52-71 (2013).
[2] Daniel Claes, Philipp Robbel, Frans A. Oliehoek, Karl Tuyls, Daniel Hennes, and Wiebe van der Hoek: Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks. In Proceedings of AAMAS '15, pp. 881-890 (2015).
[3] Sara Bernardini, Maria Fox, Derek Long, Chiara Piancentini: Leveraging Probabilistic Reasoning in Deterministic Planning for Large-Scale Autonomous Search-and-Tracking. In Proceedings of ICAPS?16, (2016)

Diploma Thesis Supervisor: Bc. Michal Štolba, Mres.

Valid until the end of the summer semester of academic year 2017/2018

[illegible]

prof. Dr. Michal Pěchouček, MSc.

Head of Department

L.S.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta elektrotechnická -7-

[illegible]

prof. Ing. Pavel Ripka,CSc.

Dean

Prague, January 5, 2017

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Master's Thesis

Scalable Probabilistic Planning for Decoupled Tasks using Multi-Agent Decomposition

Bc. Jan Starý

Supervisor: Bc. Michal Štolba, MRes.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

January 9, 2017

Aknowledgements

I would like to thank everyone who has stood by me during the time that I have been writing this thesis. My thanks and deepest gratitude goes to my mentor Mr. Štolba, who always had time for my questions and he pushed me to accomplish something more. I would also like to thank Mr. Komenda for consulting the thesis and sharing his experience. Special thanks goes to my family, friends and my girlfriend for all their support and optimism. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042), is greatly appreciated.

Declaration

Abstract

The multi-agent non-deterministic planning in general has issues with its scalability. This thesis presents a solution for multi-agent planning of problems with decoupled tasks to overcome these issues. We have chosen a subclass of problems with limited interactions to exploit its characteristics. Those interactions are restricted to the shared environment and the utility. Our aim is to increase the scalability by using decomposition of multi-agent problem into number of subproblems centered around each individual agent together with the self-absorbed approximation and simulation of other agents to evaluate the effect of their action on global reward. The problem solution is defined and formalized for general problems with decoupled tasks and later applied on a concrete problem. The problem of multi-agent multi-target tracking is then implemented and compared with the (baseline) centralized solution and to the self-absorbed solution.

Abstrakt

Multi-agentní nedeterministické plánování má obecně problémy se škálovatelností. Tato práce se zabývá řešením řešením škálovatelnosti multi-agentních problémů s oddělenými úlohami. Vybrali jsme skupinu problémů vyznačujících se omezenými interakcemi s cílem využít jejich specifické vlastnosti. Tyto interakce jsou omezeny na sdílené prostředí a utilitu. Naším cílem je zvýšit škálovatelnost rozdělením multi-agentního problému na několik podproblémů spjatých s jednotlivými agenty. Za použití self-absorbed aproximace a simulace ostatních agentů hodnotíme vliv jejich akcí na společnou odměnu. Řešení problému je definováno a formalizováno pro obecný problém oddělených úloh a později použito k řešení konkrétního problému. Problém multi-agentního sledování více cílů je poté implementován a srovnáván s centralizovaným a self-absorbed řešením.

Contents

List of Figures

List of Tables

Chapter 1

1 Introduction

This thesis presents a solution for multi-agent planning with decoupled tasks. Planning is one of the techniques used in modern artificial intelligence. It is a technique that solves a problem by defining a sequence of actions. These actions gradually accomplish all the subgoals and lead us to a problem solution. We focus on the non-deterministic planning where the actions in the environment can have multiple outcomes. In the real world due to an error or bad luck we often see outcomes that we did not anticipate. So to match this non-determinism we also model it in the planning problems. This thesis focuses on tackling the scalability issues of multi-agent non-deterministic planning. We have chosen a subclass of problems that is specific by limited interaction between agents. Our aim is to increase the scalability by using decomposition of multi-agent problem into number of subproblems centered around each individual agent. We have also used the self-absorbed approximation together with simulation of other agents to evaluate the effect of their action on global reward. The presented solution based on these techniques should provide more scalability compared to the (baseline) centralized solution and better reward compared to the self-approximation solution.

The decoupled tasks can be found in real world problems and are specific by allowing only a limited interactions between agents. These interactions are limited to the shared environment and the utility of the solution. We want to exploit these characteristics to improve the scalability. In general, multi-agent planning based on Dec-POMDP [1] model is hardly scalable because of the combinatorial explosion in the number of joint action space. The aim of this work is to combine latest probabilistic techniques and multi-agent planning to solve these problems and propose a way how to take advantage of the specific properties of this class of problems.

The complexity of multi-agent planning arises from number of agents involved in the problem. A possible way to reduce the complexity is the decomposition of multi-agent problems. The decomposed sub-problems alone do not face such computation complexity but they lack the ability to capture the entire problem. The challenge is to use the latest techniques to optimize computation of individual decomposed sub-problems and to compensate for the lack of the global view. We focus on combining the knowledge about agent position and use available techniques to simulate and estimate their behavior. The goal is to design an improved algorithm that would benefit from decomposition while at the same time it will compensate for the loss of knowledge of other agents behavior.

The main idea used for the implementation part of this work is based on a concrete problem of multi-agent multiple target tracking problem. This problem was inspired by work of Bernardini et al. presented in [2], which describes a novel solution for tracking of the target with a single UAV. It is based on combination of deterministic planning and Bayesian inference. Our intention is to use the model of problem described in that article and extend it for multi-agent multiple target tracking. Then we identify its specifics and formally define and describe it as planning for decoupled tasks. We design a planner and a simulation environment based on latest probabilistic techniques found in literature by modifying and applying it to our problem.

The designed solution is then implemented and at the end of this thesis verified in a series of experiments. Both the implementation and the experiments show use of proposed solution on a concrete real world problem of multi-agent tracking of multiple targets.

1.1 Outline

This work is divided into 8 chapters. The first chapter is an introduction to the class of problems addressed by this thesis, its focus and goals. Next we offer a background information about the related work and more in-depth information about latest techniques in the field of planning. The third chapter provides general description of the decoupled tasks and their specifics in the context of planning. Also, we formally define a model for this planning problem and propose some helpful methods to decrease its complexity. In the next chapter we specify a concrete real-world problem, its environment and all the involved entities.Then, we formulate a concrete model based on the general problem formulation and we discuss its specifics and ways we can exploit them. In chapter five we describe the implementation of the planner and simulation, its structure, used algorithms, other improvements and the visualization of its data. Chapter six provides experimental evaluation and comparison with the (baseline) centralized solution and a solution from the literature, self-absorbed approximation. Last two chapters provide conclusion of this work and possible ideas and areas of future research in this field.

Chapter 2

Background and related work

In this chapter we talk about the background of planning and its use for multi-agent problem solving. We mention the most common models used for problem description and formalization and highlight its differences. Later we mention techniques and algorithms used in probabilistic planning and the most important algorithms used for computing a policy. Mutually we discuss the literature related to this work. We assess the differences and similarities of existing research and talk about the inspiration there.

2.1 Planning

Planning is a method used in AI for solving a large class of general problems. The problems have a common characteristic that they deal with action selection or control. To find a plan then means to select a sequence of actions that transform the problem from its initial state to a desired goal state. It is often used in multi-agent systems as a means of effective allocation of goals among the agents. Each agents then follows a plan for completing these task in a suitable order. Multi-agent planning can be divided into several subgroups depending on specifics of problems each subgroups solves. These groups are defined by these aspects [24]

- on-line vs. off-line
- centralized vs. distributed
- cooperative vs. self-interested
- dynamics (deterministic, non-deterministic, probabilistic)
- observation (none, partial, full)
- horizon (finite, infinite)

The off-line planning is typical by computing the plan before its execution starts. It is often used in situations where the environment is known in advance and we have enough time to carefully think about the whole course of the operation. In other cases where the environment is dynamic we can either use off-line planning with the added possibility to replan or we can compute the plan on-line as we execute it.

We distinguish classical planning, that is defined [22] as problem solving method for environments that are fully observable, deterministic, finite, static and discrete. Static environment changes only when an agents uses an action. Classical planning problems are often represented using STRIPS [12] or PDDL [17] language, which defines the properties of domain and a concrete problem definition. Every action in the domain is represented using preconditions and effects. These definitions are then introduced into specialized solvers which finds a sound or in some cases optimal solution. The solvers are mostly based on a forward or backward state space search together with the use of a heuristic.

We focus on probabilistic planning, which is an extension of non-deterministic planning, where each outcome of an action can be defined by multiple successor states. The difference is that the transition between states is defined by a probability distribution. Many models were created to help describe various planning problems. These models are constructed on a basis of Markov decision processes. MDP is a mathematical framework for modeling decision making. An Effect of the decision does not always have deterministic outcome and we want to model its stochastic behavior.

Models

The subject of Markovian models is thoroughly described in article [14]. There is a number of well-known models that can be used to describe and formulate problems. They all belong to the class POSG which describes partially observable stochastic games. These models are then specifically extended for a given subclass of these games. For example MDP [14] model is a centralized model designed for fully observable sequential decision processes. On a contrary POMDP is a model for problems with partial observability.

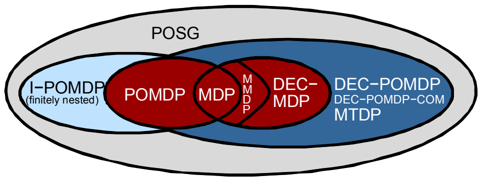

Figure 2.1: Relantionship among the models [24].

Most basic of these models is MDP. It is centralized and fully observable and it describes states, actions, transition probabilities and rewards in a given problem. The other models were derived from this model to describe larger classes of problems. Due to the nature of real-world problems there is a need for specific model for problems with partial observability. This model is called POMDP [15]. Other models were presented to support multi-agent problems with cooperation. MMDP [4] is often used for them. And also if we need to model decentralized problems we have the possibility to use Dec-MDP or Dec-POMDP [1]. All these models represent problems that are complex and hard to solve. Majority of them are NEXP-C or NP-C. Further information on complexities of these classes can be found in [24].

Other model closely related to our problem is so called factored Dec-POMDPs [18]. This model is based on a factorization of state into multiple fragments. The model then enables exploitation of independence between agents. It also allows variability over subset of agents. These models present an individual state and transition for each agent but for example they can be coupled by the reward function. The transition and observation independent, event-driven or temporally decoupled Dec-MDP and other models are described [18].

Algorithms

Planning algorithms can be divided based on the quality of their solution into optimal and approximate. Another difference is if they compute the solution with finite or infinite horizon. Probabilistic planning is often solved by using iterative methods. Two commonly known algorithms are value iteration [20] and policy iteration [5] . Value iteration is an algorithm that approximates the value of optimal plan with discounted rewards. In each iteration we sum up rewards from executing action *a* in current state *s* plus the discounted value of future rewards weighted by the transition probability *P(s'|s,a)*. We can iterate until the values converge, respectively as long as the future obtained reward after discount is greater than ε. Parameter ε is a threshold under which we consider the values negligible.

$$V'(s) \leftarrow R(s) + \gamma max_a \sum_{s'} P(s'|s,a)V(s') \tag{2.1}$$

Second well known algorithm is called policy iteration and is based on a different approach. Policy is a mapping from states to actions and it describes behavior. $\pi^*(s)$ is an optimal policy from state s. Policy iteration starts by selecting an arbitrary policy. This policy is then evaluated in iterations

$$V^{nt}(s) = R(s, \pi_t(s)) + \gamma \sum_{s' \in S} T(s, \pi_t(s), s')V^{\pi t}(s'). \tag{2.2}$$

Evaluation can be done by multiple steps of simplified value iteration where the action is defined by the policy. Next a policy improvement is computed for each pair of state and action. A new policy is finally created by choosing the best improvement to the policy. The policy is improved repeatedly until it converges.

These algorithms provide optimal solutions but are in practice intractable. To solve bigger problem instances an approximate algorithm can be used. As example of these algorithms [24] mentions JESP (Joint Equilibrium Search for Policies) and MBDP (Memory Bounded Dynamic Programming). Another example of algorithm capable of solving an on-line planning problem is MCTS. A thorough survey of Monte Carlo tree search and its enhancements can be found in [6]. MCTS is a method based upon search and approximation of states in a tree. These approximations are made by issuing a large number of simulation samples which estimate the reward value of these states. This simulation is controlled by a metric which ensure focus on promising regions of the state space search. It offers many advantages [23]. One of them is that the algorithm converges to optimal solution but can also be stopped at an arbitrary time. It samples state transitions instead of the whole state space. Also it is highly parallelizable and easy to implement. The algorithm can also be enhanced by the use of heuristics and by controlling the search.

MCTS is based on 4 steps repeated inside a loop. First a promising leaves node is chosen by the selection. Then the tree is expanded by possible actions. After selecting one of these expanded leafs we then perform the simulation. Simulation can be a random or a statistically biased sequence of actions that helps evaluate the expected outcome of selected node. When simulation is complete and rewards are collected we back-propagate the results on the way to the root of the tree. This way the back-propagation influences the next selection.

2.2 Related articles

In this chapter we present and discuss a number of articles that are closely related to the topic of the thesis. Mentioned articles propose some methods that can be used to improve scalability of planning for decoupled tasks. Other articles show improvements and possible solutions for problems related to the target tracking problem we have decided to implement for the purpose of testing the solution and comparison.

Effective approximations

The article [11] is about effective approximations for multi-robot coordination in spatially distributed tasks and is on of the substantial sources for this thesis. It proposes approximation methods that are able to lower the combinatorial growth in the size of the state space. It also presents the idea of subjective approximations that help decrease the number of joint actions. These approximations aggregate or neglect the effect of other agents. They inspired the subjective approximation that was used in this thesis and also pointed us to using the positional information of other agents and compose it in the obtained reward. The article further describes methods for empathy by predicting agents location and empathy by fixed weight discounting. Both offer some interesting ideas. For example a use of presence mass to describe possibility of other agents being present at a given location. It is a simplification due to the fact that identity of agent which is present is not important.

The second approximation is a method that provides an improvement in reducing the state space which effectively improves problem solving times. It resides in handling of the current "phase" of the problem rather than seeing each potential change that may arise. As presented by the authors this technique focuses only on currently active tasks and ignore the possibility of change in their set. In other words the tasks will be only services when they are relevant in the current time frame. This approximations reduces the otherwise rapidly growing state space to a more manageable size. The high dependence on current scenario is also mentioned in the article. It causes that in some cases this approximation does not provide any benefits at all. It also has the disadvantage that it adds the need for more complex temporal structure in cases where the sum of move and execute times is different for each task. The authors also mention that both of these approximations can be combined and further improved by a limit for k closest tasks. This combination provides even better reduction of complexity.

The article presents these methods on a class of Spatial Task Allocation problems. This SPATAPs [11] class is a subclass of MMDP [4]. As we are focusing on a different class of problems represented by Dec-POMDP we must be aware of their differences. In our case the agents model of interaction is different. SPATAP uses a model where interactions are locally restricted. Also it has negative interactions where each task can be serviced only once.In our case the decoupled tasks do not allow any direct agent interactions. These agents only influence themselves based on reward changes. Also unlike the SPATAPs decoupled tasks allow repeated completion of tasks. Only difference is that each completion can give a different reward based on other agents behavior.

Multi-UAV tracking

Another article [7] related to our thesis is the Decentralized cooperation of multiple UAS for multi-target surveillance under uncertainties. It focuses on solving a similar problem as the one presented in our planner for UAV tracking. In this article we can find some tricks to alleviate the complexity of given problem. One of them is a use of mixed observability. With mixed observability we consider some variables being observable while the others rest unobserved. It is suitable for representing UAV position as known. The second specific is a use of factored models, where each state is divided in factors. The complexity is reduced when some of these factors are reused and they do not have to be recomputed. Next they use roles, which are represented by optimal policy for single UAV models. Those roles are then combined and assigned to UAV during execution. The policy for multi-agent is not computed. The presented approach is then based on a decentralization and off-line pre-computation of these single agent policies. The agents then make an auction and the different roles are allocated to agents. This ensures that all roles are covered and that agents perform best suited actions in joint action space. The reward function is designed so that it penalizes if a target is already monitored by another UAV. And the reward is also affected by a time since the target was last seen. These techniques promote cooperation and distribution of targets between agents.

Our approach to the problem is more general and extends beyond the specific tracking problem. Specifically to the class of decoupled tasks planning. This article presents information about mixed observability. We can relate this with the need to know localization of other UAVs to improve upon the subjective approximation in our problem. The main difference compared to our problem is that we want our solution to be more scalable. This algorithm was tested only on two UAVs tracking two targets. Also we would prefer an on-line approach without the need to pre-compute single agent policies off-line. Moreover this article includes a model with limited communication where beliefs are shared when two UAV are in a near distance. More details on the topic of delayed or limited communication are in the articles [21, 8, 19]. We want to forbid any communication except the acquisition of UAV positions. In fact position can be obtained by sensors of some centralized authority. So the communication in our case is much more limited and we do not have to carry about connectivity issues or delays. On the other hand article offers great inspiration in computing the reward based on information about other UAV positions and also in the addition of last seen timer that promotes the search for lost targets.

Combining probability reasoning with deterministic planning

One of the interesting works in this field is hybrid approach to Search and tracking problem [2]. It focuses on large scale SaT missions and presents a solution that employs both deterministic planning model in combination with Bayesian inference. The authors are presenting a novel solution to a problem of autonomous surveillance by a UAV. The goal of this UAV is to search for and subsequently follow the target to its destination. The destination is unknown, but there is a set of possible candidate cities. The main focus of this work lies in improving of the search part of SaT. As the article points out purely probabilistic solutions are often usable only for small scale problems and other previously used methods may struggle when the target's behavior is unpredictable. Because of that the authors have decided to take a different approach and combine automated planning with Bayesian reasoning. What makes it unique is that it uses Bayesian inference and past observations to make a prediction about target's position.

There basic model of the problem was adopted from this work and generalized for use of multiple UAVs tracking multiple targets. The model specifies that the target is a road vehicle and it is located somewhere on a road network. The motion of both the target and the UAV is uniform and it goes to a specific location by using an efficient path. In the process of localizing the target the UAV maneuvers over locations with high probabilities of target incidence. The goal is to find a set of maneuvers that would maximize probability of detecting the target. The search is described as being a sequence of standard patterns (spirals and lawnmowers).

The solution is based on candidate pattern selection by Monte Carlo search (MCS). The authors use it to find points with the highest probability of finding the target in certain time periods. MCS is run over a shortest path graph connecting last known position and all the destination nodes. Two most probable locations are selected for each temporal section. Then they use these information to find the best sequence of these patterns. The selected sequence maximizes the probability of locating a target and preserves UAV resources like fuel. The planner also has to predict where the target is heading and it has to update this prediction in time. For this it uses the information about its previous predictions together with the fact that since it hasn't found the target the last search sequence has failed. The authors exploit the fact that this provides an opportunity to get negative information about target's location and improve estimation of its real destination. The target is either found or the information serve as an improvement increasing the probability of finding it next time. This work was tested on real-world problems with long temporal horizons and large mission areas.

This article had inspired the actual problem solved in the implementation of this thesis. We took the basic problem from this article and improved it to solve a similar one containing multiple UAVs and targets. This way we will verify our decoupled task planners ability to solve complex large scale problems. The structure of the problem is similar including the model of targets movement on the roads and their destinations being a random location from set of cities on the map. The solution is then very different. It is based purely on probabilistic reasoning together with complexity enhancements and MCTS more of a core algorithm than an instrument for sampling. Since the problem for more UAVs is more complex we have relaxed the constrains. For example we consider the observations strictly true so that the outcome of search action is always accurate.

Benefits of multiple UAVs

Important information about requirements on number of UAVs with bounded speed can be found in article [13]. Main focus lies on determining the least number of UAV required for continuous tracking of n targets. Additionally the authors study how speed and tracking range influences number of needed UAVs. This article is unique by the approach to reformulate this problem as network flow problem. The article shows that setting a lower bound on number of UAV to have a perfect tracking of n targets is NP-complete. Even in situations where target movement is known in advance the complexity still persists. The use of knowledge of exact trajectory of target movement instead of uncertain movement make that an optimization problem that can be solved by network flows. Although this approach seems very distant from our vision of the solution it still offers important information about how speed, tracker radius and number of UAVs influence each other. This information can be later used for example to set a reasonable speed to UAVs in our problem.

Multi-target Detection and recognition

The article[9] is closer to our the problem we are trying to solve. It is a multi-target detection and recognition problem. It also build on an uncertain probabilistic environment with the exception that it only operates with a single UAV. Also the goal of this problem lies in recognition of targets and not in their surveillance. The mission is based on moving between zones, changing height, recognition done by image processing and a use of database of models. The proposed technique in this article is an on-line algorithm for planning under strong time constraints. The optimize-while-execute framework is an interesting optimization which while executing the current action uses the time to anticipate and plan for next possible belief state. Duration of executed action a its probabilistic effects is used to construct possible belief states and then proportional time to their probability is spent computing policy for each of them.

Chapter 3

Problem description

In this thesis we focus on a problem class of multi-agent planning for decoupled tasks. In general we are set in a situation, where we have multiple agents and the goal is to find and complete a number of tasks in the environment. Each agent can service any of these tasks without restrictions. In planning of decoupled tasks interactions are limited to the shared environment and interactions by means of utility. They do not in any case prevent an agent from executing chosen action.

To successfully solve this problem, we have to make sure these tasks are serviced by one of the agents. Of course any overlapping in task allocation may cause the solution to be sub-optimal. Since there are multiple agents involved, we face the problem of exponential growth in the joint action space and the state space. Consequently this makes it more difficult to find an optimal solution. In order to make this problem more scalable, we have to use special methods like partition organization or self-absorbed approximation[11]. In contrast with SPATAP class of problems in this article [11], we do not face spatial constraints but rather constraints regarding agent interaction. As we want to use this model for dealing with real-life problems, using some of them does not provide desired benefits. For example use of the environment partitioning would only be a waste our resources on locations, where the distribution of tasks is not even.

Therefore we propose using a model based on Dec-POMDP [1] with some special features unique to this class of problems. Dec-POMDP is a framework that uses decentralized approach on multi-agent decision making under uncertainty. Unlike POMDP each agent uses only its local information based on their own observations. Given these partial observations the agents try to maximize a single reward function. Solving Dec-POMDP is NEXP-complete [3]. So without the use of approximations and exploits of additional problem characteristics it is not tractable to solve larger problems.

Mainly we want to exploit the property of decoupled tasks that interactions between agents are limited. An action performed by one of the agents does not directly effect other agents. Furthermore there are no actions that need agents' cooperation. So in a short summary actions of a single agent cannot prevent other agents from following a certain goal, block his path, or cause other similar influences. The only influence permitted is established by the shared environment and by means of obtained reward which is a crucial tool for effectively splitting the task handling among the agents.

By a property of the decoupled tasks, each of our agents is autonomous and unaware or the actions and beliefs of the others. This leads us to another specific of this problem and that is limited communication. Communication between the agents is minimized to the extent that they only know position of each other. This is one of the reasons why we want to build the model on a decentralized and partially observable basis.

3.1 Formal description

As mentioned before the formal model is based on Dec-POMDP.
By definition [18], Dec-POMDP is a tuple <D,S,$\mathcal{A}$,T,R,**O**,O,h,I>.

- $D=\{1,\ldots,n\}$ is a set of agents.
- S is a finite set of states of the environment.
- $\mathcal{A}$ is a finite set of joint actions.
- $T : S \times \mathcal{A}\times$S'$\rightarrow \mathbb{R}$ is a transition probability function.
- $R : S \times \mathcal{A}\times$S'$\rightarrow \mathbb{R}$ is a function that specifies obtained rewards.
- **O** is a finite set of joint observations.
- O is the observation probability function
- h is the horizon of the problem.
- I is the probability distribution of $s_n \in S$ being the initial state.

A joint action $\mathcal{A}$ is a combination of actions available to individual agents. Each agent has a different set of available actions. The outcome of a joint action is determined by the transition probability function T. This function T(s,a)=s' specifies all the states that are reachable from state s and probabilities Pr(s'|s,a) that using action a in state s leaves the environment in state s'.

Every time a joint action is used the environment provides a joint observation $\mathbf{O}$. From this joint observation each agent only perceives its own private observation o_k. Because we minimize communication between agents we can omit any communication about agent behavior, cost of this communication and its effects and we do not have to model it. This is in agreement with Dec-POMDP model where each agent acts only based on his own observations. The only thing that remains is that each agent has the ability to observe positions of other agents. This can be modeled as a part of the observation where each agent receives information about one anothers position as a part of their individual observation. Since this information is the same for every agent joint observation$\mathbf{O}^i$ can be modeled as a vector $< \alpha, o^1, o^2, \ldots, o^n >$.

We can also exploit a set of special information about the environment. To be more specific they are potential task locations κ, their possible movement $\mathcal{M}$ and the agents position λ. This information is gathered from the environment map that can be represented

freely by any means of grid or a graph. This additional structure affects how we transition between states and how reward is computed. To represent it in the model we make some changes to the representation of the state $s =< \lambda, \kappa >$. In this tuple λ now represents a vector of agent positions and κ the potential position of tasks in the environment. $\mathcal{M}$ is an additional function that specifies the probability of task changing its location. Transition from state s to s' is then described by transition function $T(s', s)$ giving us the probability (3.1).
p_x^T is the probability of transition of $target_x$ and p_n^M is probability of movement of UAV_n .

$$Pr(\lambda', \kappa' | \lambda, \kappa, a, \mathcal{M}) = \prod_{x \in \mathcal{M}} p_x^T(\kappa'_x | \kappa_x) \prod_{n \in \mathcal{D}} p_n^M(\lambda'_n | \lambda_n, a_n, \kappa) \tag{3.1}$$

If we take a look at the reward function (3.2) we need to specify what behavior should affect the decision making. It can be broken down into reward received for accomplishing the task R_x and a reward for agents cooperation R_D. The second one can be modeled either as a positive reward for completing tasks far away from other agents or negative one for being too close together. The reason behind this second reward is to force agents to focus on different tasks without the knowledge of their action selection thus reducing unnecessary overlapping. The reward function is computed from these two factors. R_U is the reward for localization of $target_x$ by the UAV_y and R^D is the reward influence caused by other UAVs.

$$R(s, a_y) = \sum_{x \in \kappa} R_x^U(\lambda_y, \kappa_x, a_y, \mathcal{M}_x) + \sum_{i \in \mathcal{D} \setminus \{y\}} R_i^D(\lambda_i, \kappa, a_i, \mathcal{M}) \tag{3.2}$$

3.1.1 Independence of actions

Compared to an ordinary Dec-POMDP we have the specific that the interactions between agents are limited. These limits include a characteristic that no actions of two arbitrary agents are dependent. So to say the set of dependent action $\mathcal{A}_d = \emptyset$. For example there are no actions that would require another agents cooperation in its positioning or use of of his actions. This property is important in later development of our solution. If an agent n_1 in state s uses the action a, then the transition probability function of other agents in the newly-emerged state s' is the same as in state s.

3.1.2 Proposed methods

In this section we propose some methods that will enable us to solve the larger scale problems defined in the last chapter. Use of these methods was inspired by an article on solving multi-agent coordination in spatially distributed tasks [11]. Our goal is to use specific features of described problem to reduce the complexity of its joined state and action space. A substantial part of the complexity lies in a fact that both the transition function and reward function is dependent on multiple agents behavior. The number of agents then affects the complexity of given problem exponentially. We use the property of decoupled tasks that each of our agents is autonomous and unaware of others behavior. To split the problem in a number of similar sub-problems using self-absorbed approximation. Next we further decrease the complexity by reducing the state space by using phase approximation.

3.1.2.1 Multi-agent decomposition

A suitable method that can be used is multi-agent decomposition. It is based on the principle of splitting the problem into smaller subproblem that can be then solved separately. When we decompose the problem we either lose the information about other agents or we are forced to simulate their behavior. Complete simulation of other agents would add even more to the complexity of the problem. That is why we want to use the self-absorbed approximation with an addition of limited horizon simulation and you its results to alter the obtained reward of the UAV.

3.1.2.2 Self-absorbed approximation

The self-absorbed approximation [11] is one of the subjective approximations used to reduce complexity created by number of agents in joined action space. It is an approach where we decide to reduce the problem to a single agent scenario. From there we act as this agent is the only one in the environment and all the state transitions and reward function are computed as if no other agents were present. This method reduces the problem action space from $|\mathcal{A}|^{|D|} to |\mathcal{A}|$.

As there are no other interactions between the agents other than by means of obtained reward we can use combination of the two previously presented methods and split the problem in $|D|$ sub-problem each a self-absorbed version of one of these agents. We then improve this technique by simulating other agents as a part of the environment. Using information about their position presents an opportunity to affect obtained rewards of self-absorbed actions in such a way that actions with higher probabilities of being serviced by another agent yield less reward. Thus promoting selection of actions that have higher value in means of agent cooperation and preventing agents from servicing only the most valuable task.

Chapter 4

Tracking using decoupled tasks

We demonstrate the use of the techniques presented in chapter 3 on a particular example based on a real-world problem. It is a multi-agent multiple-target tracking problem and we show how to solve it by means of planning for decoupled tasks. The basis of this tracking problem comes from an article [2], which presents a solution for single-agent tracking problem. It focuses on large scale search and tracking (SaT) missions and presents a solution that employs both deterministic planning model in combination with Bayesian inference. In this thesis we model a similar problem and push it even further by not limiting the number of both UAVs and the targets that are being tracked. We use the same model of the targets with the exception that our search results are considered errorless. In contrast to [2], we propose a purely probabilistic solution to this problem.

4.1 Problem definition

Our goal is to design an on-line planner based on general model described in chapter 3 and use it to solve this tracking problem. For the purpose of this example we have to model an environment with n unmanned aerial vehicles (UAV) and m targets. In this environment each UAV has its own starting position. The goal of these UAV is to continuously track targets' position. We assumed that number of the targets can be greater than number of the UAV at our disposal. This prevents us from allocating each specific UAV to track a single target. Using this specific *1:1* allocation was also proven sub-optimal [7], since there may arise a situation where swapping targets may increase the tracking efficiency. Since it is impossible in some cases to continually track m targets with n UAVs we focus on acquisition of the latest and most accurate data about their positions.

Another important element of this problem is that the targets have the ability to move. Available moves are restricted by a defined road network. There are also several important locations in each environment. We can see them as large cities and they represent a possible final destination of the targets. Every time each UAV locks its selected action for execution a target chooses a move action towards its destination and executes its movement. UAVs are unaware of the destination of the targets. As the number of these potential destinations rises, the probability of successfully predicting targets next move decreases. On the other hand the UAVs have a total freedom of movement since they are not dependent on the road network. It is natural that road transport has speed limits and congestions. That is why we allow UAVs to move at a higher speeds.

In order to solve this problem we use the techniques mentioned in section 3.1.2 to reduce the complexity caused by number of agents involved in this problem. Also we present additional probabilistic techniques to handle the uncertainty of the targets' positions.

4.1.1 Environment

We now define the environment of the problem as

$$< P, S_0, P_E > . \tag{4.1}$$

$P =< p_1, p_2, ..., p_x >$ is a set of locations inside the environment map.

$S_0 \subseteq S$ defines a starting state of our problem.

The initial state $S_0 =< \lambda_0, \kappa_0, t_0 >$ defines the setting of the environment before we start the planning phase. It defines position of all the UAVs $\lambda_0 =< p_{uav_1}, p_{uav_2}, \ldots, p_{uav_n} >$, the distribution of probabilities of target locations $\kappa_0 =< p_{t_1}, p_{t_2}, \ldots, p_{t_n} >$ and $t_0 =< t_0^1, t_0^2 \ldots, t_0^n >$ which is a vector that describes age of the information about targets' positions.

$P_E \subseteq P$ is a subset defining possible destinations for targets. The exact assigned values for each target are unknown.

4.1.2 Agent

The unmanned aerial vehicles are modeled as the agents. The UAV is autonomous and can complete any task. They all possess the same actions and are homogeneous. The movement of each UAV is limited by a representation of the environmental map and is defined by its constant speed. The only ability given to these UAVs is a combined move and search action. This action $\mathcal{A}_{ms}$ represent a movement to specified location automatically followed by a scan for targets in the destination area. Also the UAVs are forced to move every turn because we don't expect them to levitate. This limitation is present to enable the use of plane-like UAVs.

4.1.3 Targets

The targets are an unspecified object which can represent any type of moving road vehicle. For every target a specific destination location is selected at the start of the problem execution. This destination is final and cannot be altered during the execution phase. Each time step the target takes a step towards its destination. This step is computed by a means of shortest path algorithm from its current position to the destination.

4.1.4 Interaction with the environment

As specified above the interaction between the environment and the UAVs is represented by actions. Each executed action has specific effects on the environment and it also provides a partial observation for a given agent that executed it. Together with this observation also comes a feedback in a form of reward that evaluates effectiveness of this decision.

Another interaction is by means of the obtained reward. The reward evaluates the quality of actions performed by the UAVs and affects their decisions. It is computed based on reward for localizing the target, time passed since their previous encounter and reward derived from the positions of other UAVs.

4.2 Model

This chapter describes the model constructed for multi-agent multi-target tracking problem. We formally define the states of the environment together with definition the actions that cause transition between them.

This UAV mission planning problem is defined using modification of the general model as

$$< D, S, \mathcal{A}, T, \mathcal{M}, R, \mathbf{O}, O, h, S_0, \circ > . \tag{4.2}$$

- $D = \{1, \ldots, n\}$ is a set of UAVs.
- S is a finite set of states of the environment, where state $s =< \lambda, \kappa, t >$.
  - λ is a vector of UAV positions, $\lambda =< p_1, p_2, \ldots, p_n >, p_k \in P$ is position of UAV_k, where $k \in< 1, n >$.
  - κ specifies potential target position, $\kappa =< p_1, p_2, \ldots, p_n >, p_l \in P$ is position of $Target_l$, where $l \in< 1, |\kappa| >$.
  - t is a vector of integers, $t =< t^1, t^2, \ldots t^n >$ where t^1 is time since $target_1$ has been seen.
- $\mathcal{A}$ is a finite set of joint actions.
- $\circ$: is operator that describes application of an action in state $\circ : S \times \mathcal{A} \rightarrow S'$.
  - $\mathcal{A}_{ms}$ in the only type of action which secures both UAV movement and search for targets.
  - $S \circ \mathcal{A}_{ms} \rightarrow S'$
  - Each joinet action $\bar{a} =< a^1, a^2, \ldots, a^n >$ consists of individual actions of the UAV.
- $T : S \times \mathcal{A} \times S' \rightarrow \mathbb{R}$ is a transition probability function.
- $\mathcal{M} : S \times S' \rightarrow \mathbb{R}$ is movement probability of targets

The probability of transition between states is computed from the probability of transition of $target_x$ p_x^T and the probability p_n^M of movement of UAV_n.

$$Pr(\lambda', \kappa' | \lambda, \kappa, a, \mathcal{M}, T) = \prod_{x \in \mathcal{M}} p_x^T(\kappa_x' | \kappa_x) \prod_{n \in \mathcal{D}} p_n^M(\lambda_n' | \lambda_n, a_n, \kappa) \tag{4.3}$$

- $R: S \times \mathcal{A} \rightarrow \mathbb{R}$ is a function that specifies obtained rewards.

The reward obtained after the execution of an action is composed of R_U, a reward for localization of $target_x$ by the UAV_y and R^D, which is the reward influence caused by other UAVs.

$$R(s, a_y) = \sum_{x \in \kappa} R_x^U(\lambda_y, \kappa_x, a_y, \mathcal{M}_x) + \sum_{i \in \mathcal{D} \setminus \{y\}} R_i^D(\lambda_i, \kappa, a_i, \mathcal{M}) \tag{4.4}$$

- $\mathbf{O} =< \alpha, o^1, o^2, \ldots, o^n >$ is a finite set of joint observations.
  - It consists of agents individual observations o^i.
  - and an observation of UAV positions α
- $O: S \times \mathcal{A} \rightarrow \Delta(O)$ is the observation probability function
  - $O(o^i | s, a)$ is a probability of agent i seeing observation o^i after using action a from the state s.
- h is the horizon of the problem.
  - In our case the horizon is infinite and we use a discount factor $0 \leq \gamma < 1$.
- S_0 is the initial state .

4.3 Solving the tracking problem

Using the specifics of our problem with the application of self-absorbed approximation we manage to break this problem down into n problems of a smaller size. Each represented as if only one of the UAV was present. This decomposition is possible due to the specifics of decoupled tasks and it does not change the size of UAV's action pool or its abilities to reach any of the goals.

Next idea is to use the information about UAV positions. They can be easily modeled as part of the current environment state and in fact can be completely considered a property of the environment as opposed to a information from other agent. These localization information enable us to reason about other agents actions and how they affect the global reward with the advantage that we do not have to specifically model all action combinations in the joint action space. We model only its effects as approximated rewards.

These values are computed from shared information about the positions of other UAVs. The UAVs are unaware of each others actions, but they still tend to choose their behavior based on reward function that promotes actions with bigger effect on global reward. Generally speaking we want to exploit the fact that close proximity of other agent to a task increases the chance that this task is serviced by it. By this assumption we should encourage other agents to pursue completion of other tasks.

All these small details lead to a decision on how to solve this problem. Since we want to decompose the problem and we prefer to use an on-line planning approach which is more agile and can be used in many real life situations we want to apply this concept together with the MCTS algorithm and see how will it affect its performance and scalability.

We decompose the problem to a number of UAV centered subproblems and define the problem as

$$< M, T, N, L(_{n_1,n_2}), D > . \tag{4.5}$$

The definition of the environment can be used globally for all the subproblems.
$M =< m_1, m_{2,...}, m_x >$ is a set of Sectors in our mission map. $m_s =< l_1, l_2, ..., l_x >$ is a Sector that is comprised of a number of locations. Specifically all locations that can be accessed in 8-connected neighborhood graph in one step given the speed of the UAV . T is finite number of enemy targets we are searching for. N is a finite number of UAVs available for this operation. We also define a function $L(_{x,y}) : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$ that returns distance between sectors x and y, s.t. $x, y \in \{0, 1, \dots M\}$.

The number of subproblems is equal to the number of UAVs present in the problem instance. The set of UAV decompositions is $D =< d_1, d_{2,...}, d_n >$. Each decomposition is defined as

$$d_i =< S_i, A_i, U_i(x, y), Pr_a(s, s'), R_a(s, s'), \gamma > . \tag{4.6}$$

Since we use the self-absorbed approximation for UAVs each of the decomposed subproblems now has its own set of states. S_i is a finite set of states

$$s =< P_i, E_i, I_i > . \tag{4.7}$$

The state describes position of UAV, enemies and times since the targets were encountered. P is a n-tuple of UAV positions $P =< P_1, P_2, \dots, P_n >$. E is a T-tuple of vectors with possible Enemy positions $E =< E_1, E_2, \dots, E_t >$.

$U(x, y)$ is a function that returns probability for every pair of position x and entity y. Probability of transition between states is based on probabilities that the UAVs are in given sector $U(m_k, j)$ and the enemies are at given locations $U(l_g, e)$. In case of $U(m_k, j)$, $m_k \in M$ is a position of UAV_j, M set of sectors. $s \leq n$ is an identifier of UAV and n is number of UAVs. Again the same applies for targets where e is the identifier of a target and l_g the location of its incidence. Location $l_g \in m_f$ is a part of sector m_f and $g \leq |m_f|$.

Last part of the state is vector of integers I_i. It is a T-tuple of timers holding last enemy localization values $I =< i_1, i_2, \dots, i_t >$. For each transition between states all the values of I_i are incremented by one. The only exception is when an UAV finds a target. Then $i_e = 0$ because $target_e$ has been recently found.

After the definition of state the next important topic is actions. A_i is a finite set of actions usable by $agent_i$. The operator $\circ$ defines the application of actions $\circ : S_i \times A_i \to S_i^{'}$. Action $\bar{a}$ is applied $s_t \circ a_i \to s_t^{'}$ and state $s_t^{'}$ models the effects of this action on the environment.

We define the type of moveSearchAction as $A_{ms} \subset A_i$. This action combines UAV movement with the subsequent search. It enables the UAVs to move from one location to the other and at the destination location it searches for present targets. The moveSearchAction a_{ms} is defined by destination of movement m_k, identifier of UAV j and the distance of movement $L(p_j, m_k)$ has following effects

- $s_t \circ a_{ms} \to s_t^{'}$.
- $< P, E, I > \circ a_{ms} \to < P', E', I' >$
- $P' =< m_k, p_2, \ldots, p_N >$ positions of UAV after action $\mathcal{A}_{ms}$.
- $E' =< e_1^{'}, e_2^{'}, \ldots, e_T^{'} >$ positions of target after action $\mathcal{A}_{ms}$.
- $I' =< i_1^{'}, i_2^{'}, \ldots, i_t^{'} >$, where each $i_v^{'} \in I = i_v + 1$ or 0 if $target_v$ was found by this action.

Outcome of these action depend on the probability that describes the transition between current and the next state. The action $a_t \in A_{ms}$. Probability $Pr(s, s') = pr(s'|s, a_t)$. The movement part of the action is deterministic and is completely dependent on previous state. The change of state caused by the search is then dependent on probabilities of targets locations. The probability that a target is found is

$$\prod_{e_k \in E'} \prod_{t \in e_k \cap P'} U(l_t, k). \tag{4.8}$$

In other words the probability is dependent on probabilities of incidence where both the UAV and the target is in the same location.

The reward function specifies the utility of a transition between two states. It depends on number of targets found and also on time that they were last seen. The localization of targets in a single time step can be obtained from vector I_0, which contains all the indexes k of vector I s.t. $i_k = 0$.

$$R_a(S, S') = (\sum_{x \in I_0} Rc \cdot x) + R_o. \tag{4.9}$$

The reward is computed using information about found targets, time x that represents how long was a target lost and the approximated reward for other agents as we try to simulate their presence. Rc - is a reward constant that specifies a gain for locating a single target.

The reward function employs a discount factor to prefer immediate reward compared to a future one. The discount factor is set to $0 \leq \gamma \leq 1$. Problem of finding the best actions throughout the mission can be classified as a maximization of this sum

$$\sum_{t=0}^{\infty} \gamma^t R a_t(S_t, S_{t+1}). \tag{4.10}$$

$a_t = \pi(S_t)$ is the action selected at time step t according to policy π.

Chapter 5

Implementation

In this chapter we present how to build a planner and simulator specifically tailored for this multi-agent multi-target tracking problem. Making this implementation also enables us to perform extensive testing of our approach and a comparison with a centralized implementation. To do so we have to create a reality simulation. In the forthcoming text we present how individual parts are constructed and how they work together. A special attention is given to the representation of the environment, methods used to decompose the problem and solve the sub-problems, and usage of probabilities to our advantage when searching for a solution.

5.1 On-line planning

When obtaining the plan for our agents we want them to be autonomous and able to react to changes in the environment. On-line planning is ideal for our purpose of solving a problem in a decentralized way. It fits in our strategy of interleaving planning phase with action execution. Also it provides an advantage in situations where an off-line planner would have to re-plan part of its plan due to unexpected changes in the environment. These planning methods have demonstrated good performance in solving large scale POMDPs. On the other side with size of the state space it becomes intractable. We are trying to combat the intractability by improvements mentioned in chapters above.

5.2 High-level view

If we look at the architecture (viz. Algorithm 1) of this simulator, it can be broken down into tree main parts. The first of those three parts is the actual planning component. It works with the information from the environment. The information can be pre-processed but also with the dynamic information acquired from the agents. The second part is called the Environment and it specifies features of the environment, how is it decomposed into smaller regions and what possibilities from the perspective of movement it allows. The last component is the reality and it substitutes the perception of the real world. By other words it provides observations and effects for actions executed by the UAVs.

Algorithm 1: Planner - main loop

```
1:  reality ← createReality();
2:  mctsSolvers ← initMctsTrees();
3:
4:  while(reality.getTime() < limitSteps)
5:     reality.timeTick();
6:     reality.moveTargets();
7:     for all tree in mctsSolvers
8:         nextAction ← tree.getNextAction();
9:         outcome ← reality.execute(nextAction);
10:        reward += outcome.getReward();
11:        consumption += outcome.getConsumption();
12:        tree.restart(nextAction,outcome);
13:    endfor
14:    for all tree in mctsSolvers
15:        tree.shareUavPositions();
16:    end for
17: end while
```

5.3 Algorithm description

Let us break down each part of the simulation and show how exactly it works and on which principles and algorithms is it built.

5.3.1 Environment

First, we focus on the Environment to gain closer understanding of the problem and what information is then available in the problem solving algorithms. The definition of the environment is read from an external file. In folder *./resources/* are located some examples of map specification files. These files are in a format called JAVA property files. It is a type of XML file used for storing and loading configurations for JAVA applications. As XML is a simple language we have decided to use this format external map information storage.

A map files specifies information about size of the environment, its concrete structure including roads, starting positions for UAVs and targets and many more information.

```
<entry key="dist0">2211</entry>
<entry key="dist1">1222</entry>
<entry key="dist2">1211</entry>
<entry key="dist3">1222</entry>
```

Figure 5.1: Example of map representation in properties file. (1 - empty location, 2 - road)

This file is loaded by *EnvLoader*() and the information is then processed into a usable structure. Once all of the information from the map file is loaded and processed the algorithm then uses it to create an inner instance of the environment which the suits as a storage of these information. The pre-processing of the information decides the inner interpretation of the environment. The environment map input and inner structures are separated by this. It can be later extended to allow other input formats and its translation into the inner graph representation. Based on the input pre-processing affects possibilities of movement and number distinct moves from a specific location.

The environment contains a special data structure. We are creating a two layered grid graph for movement. First layer is a simple graph made of a grid by connecting neighboring tiles. This graph has the information about neighboring locations and also about neighboring roads if there happen to be any. The second graph is build over the first one. It is used for UAV movement and actions and it decomposes the locations into overlapping sectors.

Sectors

A sector is a set of neighboring locations that can be accessed by the UAV in one move. Its size depends on the speed parameter of the UAV. Sectors are created from set of locations called hub locations. They are in a grid distance of one UAV step apart. Each hub location represents the center of the sector. Locations that are part of the sector are acquired using an 8-neighborhood. Each location that can be accessed in a number of steps equal to the speed of the UAV is considered part of the sector. UAV movement is then realized on a graph of neighboring sector. These graphs are constructed using a recursive search in all directions with the restrictions on distance and map border check. Before the planning begins each UAV is also associated with one of these Sectors so that it could start using its connection network.

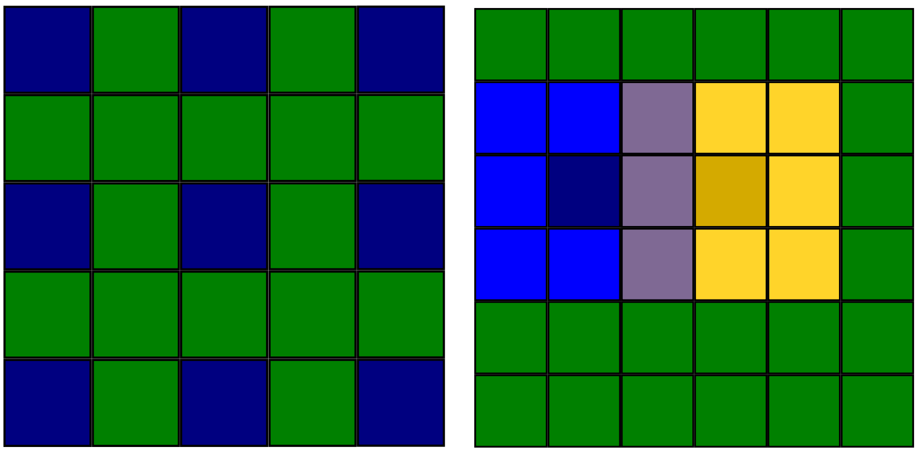

Figure 5.2: Example of hub locations (speed=2) and overlapping of neighboring sectors.

When both sector graph and road graph is created the algorithm sets destination locations that have been read from the map file. With the application of Dijkstra's shortest path algorithm search for all the paths leading to the destination nodes. We have decided not to use A* since it only provides optimal path between two locations and we want to collect shortest paths for all combinations of locations and destinations. That is also the reason why we are running this algorithm in opposite direction. Starting at the destination it obtains all of the distances to other locations. Also it locates the next steps which is always on the shortest path leading the UAV to a desired destination. These paths are then kept as part of the environment representation. Each path graph is distinguished by id of the destination. Every location is also marked by ids of paths that lead through it. This enables us to later to identify a correct next step for a certain destination in cases where multiple paths cross in that location.

Pre-processing helps us reduce operations inside the planning phase where they would have been needlessly repeated. Now that we have shown creation of the environment and preparations we can move forward and explain the decomposition of problem used by the planner.

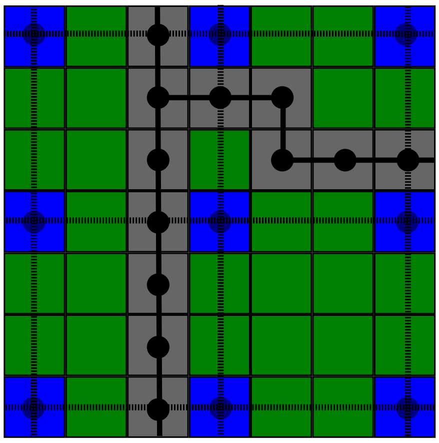
Figure 5.3: Example of movement graphs. Filled line represents graph for movement of targets, dashed line graph for UAVs.

5.3.2 Handling decomposed solvers

As we have mentioned it the theoretical part of this thesis we want to increase effectiveness of problem solving by decomposition. In this chapter we talk about how we decompose the problem and solve a sub-problem for each of the UAVs present in the simulation. Simulations can be run with an infinite number of planned steps or if we want to limit the problem we can set a finite number of steps that are calculated by the planner. The main routine of our planner is a while loop that can be terminated after specified number of steps. This loop secures execution of several subsequent actions. First we register change of time in our timer. Following that we look for target movement actions and execute their position change in the reality representing module. Next we iterate over all the decomposed solvers. There are n solvers, each planning action for one UAV in a self-absorbed environment with simulation that substitutes other UAVs. After the retrieval of the next action we apply it and receive the effects from the reality. Now that we know if we have successfully located some enemies we can use these observations to identify the state of the planner. From these observations we are also able to count the reward participation of this UAV in the ongoing step. Last thing that needs to be done is to set the planner to a state corresponding to the reality a observation made for execution of next iteration. When this loop finishes we are able to share changes in UAV positions and we are ready for planning of another step.

5.3.3 Solvers based on MCTS

The most interesting part of this application is the Monte Carlo Tree Search algorithm. It is a part of the planner that creates a tree upon receiving the environment object with the initial state of the world. A Monte Carlo tree search provides a solution for each UAV's decomposed sub-problem. It also requires some parameters. These settings change how accurate the solution is. But on the other hand they can drastically decrease its performance when chosen poorly. We focus more on setting these parameters in chapter 6. Nevertheless we are able to set the number of iterations of MCTS and number of samples generated before it returns next action.

MCTS

The core of our system uses a Monte Carlo Tree Search (MCTS) algorithm implemented specially for this piece of software. This technique enables us to find a good policy for problems with high branching factors where it would be impossible to calculate the exact best sequence of actions. It is based on an idea that with increasing number of samples we are getting a better approximated image of the reality. One of its key features is that it can be stopped in arbitrarily chosen time and it always presents the best solution found within this time limit or number of steps. Bear in mind that it is always a compromise between the computation time and the quality of the solution. The MCTS algorithm is comprised of four main parts. They are selection, expansion,simulation and finally backpropagation. These four main steps repeat in a loop and in every iteration converges closer to an optimal solution.

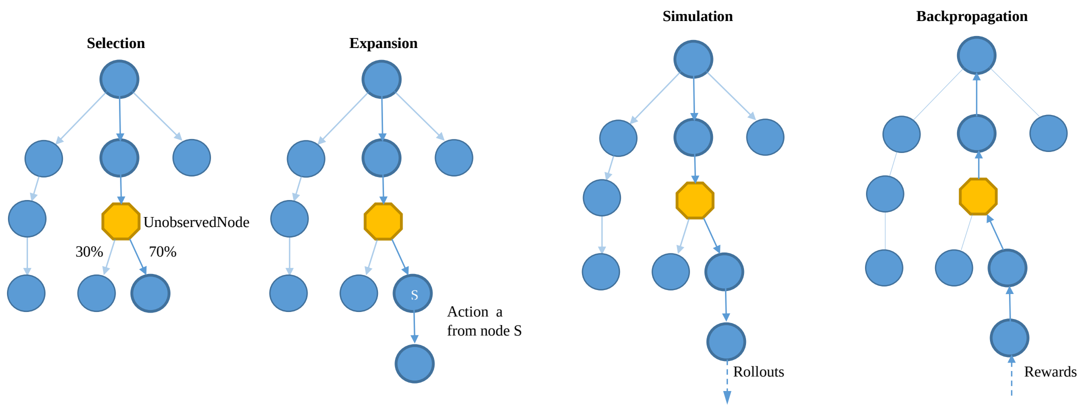

Figure 5.4: Four steps of the MCTS algorithm.

Selection using UCT

The selection step of the algorithm employs a function for selection of the most profitable action from current state. Without any knowledge from MCTS the UAVs would have to make a random decision here. But because of a great number of samples that can be executed in any node in the tree we are able to make an assumption about average reward received after the use of this action. Our MCTS implementation uses the upper confidence bound applied to trees (UCT) [16] .

$$\pi UCT(s) = \underset{a}{argmax}\{Q_{UCT}(s,a) + 2C_p\sqrt{\frac{\log n(s)}{n(s,a)}}\} \tag{5.1}$$

$Q_{UCT}(s,a)$ is estimated value of state-action pair (s,a) taken to be a weighted average of its children's values. $C_p > 0$ is a constant that influences exploration of state space.
$n(s)$ is a total number of rollouts starting from state s and $n(s,a)$ is the number of rollouts that execute action a at state s.

UCT is a metric that helps us in selection process to decide which action is better than the others. Selection traverses the tree and finds a leaf node at the end of the path with biggest UCT values. Selection may also end at a node that has not been previously evaluated and has no UCT value calculated. Selecting this node is then more favorable than any off the others. Our implementation is then slightly altered to work with nodes that represent uncertainty. Each of these uncertain nodes holds multiple child nodes representing the possible outcomes of given action. Each time the selection procedure encounters this type of node, it has to decide the outcome from given probabilities and select the corresponding node.

For example our node represents a situation where we execute the search action and the outcome of this action is stochastic. The probability of finding the target is 50%. The search node has two child nodes. One represents the possibility of finding the target, while the other one represents unsuccessful search. So statistically in one half of the selections at the node we need the algorithm to select a child node where the target is found and in the second half the second one.

Expansion

The second step of MCTS handles the expansion of the tree. In our case we have chosen to expand selected node and create all the child nodes in one step . It is easier to find all actions available from this node and expand them all then holding a list of unexpanded actions. Since we are using combined move and search actions each of the actions available to us creates one child node in the tree that represents movement. This child node then again has several child node each representing a different outcome of the search part of this action. These nodes are linked to their parent and the other way around. Also each node has to be identified in such way that we can easily reach for it later (fig.5.5). From these expanded nodes one leaf nodes is selected.

The creation of child nodes representing the search outcomes is done in a sophisticated way. The algorithm only creates those nodes representing possible search outcomes. In cases where we have N enemies the total number of combinations of enemies is 2^N. This would in most cases make the tree grow at a pace that it would considerably slow the computation times and heavily increase the memory consumption.

Having this in mind we have proposed an expansion method that first looks on the probabilities of enemy presence in given sector. Then based on these probabilities we construct an enemy presence vector which is a sequence of ones and zeros. Each number one represents that the target with id matching the position of this number in this sequence has a non-zero probability of presence. In this sequence we can count the number of ones. Let's say we only counted two. If we generate all binary numbers from zero to $2^2 - 1$. If we take each of these binary numbers in terms and replace the ones with the ones in the presence vector we end up with all the possible combinations of found enemies. This combination then becomes an identifier for a particular child node. Because the probability of all the enemies being in the same locations is not large, this method should be in most cases very effective in reducing the number of state nodes.

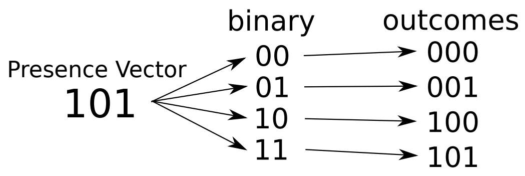

Figure 5.5: Generating expansion nodes identifiers.

Simulation

Next step in our tree search is called simulation. This part is what makes MCTS so effective. From the node selected in previous steps we run a sample. It is a random sequence of actions played from this point until we hit the action limit. Each subsequent action is always random and we only allow those actions that are available in the state we're currently in. When the simulation is carried through we simply gather all the rewards received in its progress. Rewards help us evaluate the importance of completion of a certain sub-tasks. This system then promotes use of actions with better rewards.

In our case the simulation phase has been implemented to enable parallelization. As the literature indicates there are a few types of MCTS parallelization[10]. Specifically leaf parallelization, root parallelization, and tree parallelization. We are using leaf parallelization which runs multiple threads for the simulation and when all of them are finished gather the rewards that have been collected.

The simulation is in our case represented in the Simulator class. It requires a link to the environment model, and the leaf state that has been selected in previous MCTS phase. However there are also 4 parameters that specify how the simulation proceeds. It is number of repetitions of this simulation, a limit of simulated steps, and a discount factor.

Loop of the simulation phase starts by making a copy of the initial state that has been selected. The information in this state are probabilistic and thus must be sampled from given probabilistic distributions. We have implemented a Decider class that handles sampling of enemy positions and their destination locations. Now that we have converted the state to be deterministic, we can start to simulate individual time steps. First we generate the actions executed by our moving targets. They simply follow the shortest route to their destination. Next we generate a random action for each of the simulated UAVs. This is done prior to selection of main UAV's action. These simulated UAVs have the ability to find the targets and thus lower the reward received by the main UAV if it discovers it too. Last to select and evaluate its action is the main UAV. It collects its rewards and we increment the timer by one step.

When we finish all the iterations of this simulation we then evaluate the terminal state. This is done because of the nature of this problem. The duration of the simulation can be infinite as opposed to other problems where we can simulate until we reach a state with all goals complete. By limiting the number of simulation steps we speed up the algorithm while at the same time we cut of part of its ability to describe the quality of given state. A solution that we used is to employ a discount factor for rewards together with heuristic that evaluates the end state of the simulation. The discount factor $0 < \gamma < 1$ is a tool that scales down reward values. The rewards is calculated in a way, that it promotes immediate gain of reward as seen in the following formula

$$\sum_{t=T}^{end} \gamma^{t-T} R(t). \tag{5.2}$$

With the use of a discount factor after threshold of n steps, any reward that is smaller then ε and can be neglected. But in our case we may need to finish the simulation s steps where $s < n$. This is why our implementation uses an evaluation of the end state. It is a heuristic that evaluates distance of UAVs to the nearest target and also takes into the account how many UAVs are the closest trackers. Ultimately the states with all targets close to an UAV and all UAVs being a closest tracker to at least one of the targets should be preferred.

5.4 Implementation details

In this section we introduce some of the details about the implementation. We will discuss the the process of generating actions for UAVs and we provide more details about how we work with the probabilities.

5.4.1 Generating actions

The dynamic objects in our environment are the UAVs and the targets. Both have the ability to execute an action which has certain effect on the environment. To model these actions we use a general class Action from which all the specific actions inherit its general characteristics. The system is designed to support addition of new types of actions. For example we can easily add an action to refuel UAVs. In current state we only need one combined action ("moveAction") for movement and search for UAV and one movement action for our targets ("moveTargetAction").

These actions are indispensable in the MCTS expansion phase to determine the expanded states of the tree and also in simulation to execute the random actions performed by the

UAVs. To generate these actions we have created a class called ActionGenerator which handles their creation. This class has methods for creating all possible actions or a randomly selected action. It requires a reference to a node of the MCTS tree, reference to the environment and mostly an id of the UAV. Only when creating moveTargetAction it instead requires a set of possible destinations because all target actions are created as one action. This is possible due to the probabilistic representation of target positions. The action generation uses the map graph for generation of move actions to neighboring sectors or locations. In case of the UAVs the fuel consumption restrictions are applied and all the locations in the sector are searched to build the presence vector of the enemies. This vector can be used later if the action is executed and we need to expand its outcome nodes.

The method for generating random action is used to generate actions for UAVs in the simulation phase. It optimizes the speed of the simulation where it is undesirable to generate all possible actions when only a single one is used.

5.4.2 Working with probabilities

The positions of targets and UAVs are represented as part of the states. The representation is made using a HashMap, because of the fact that each position is stochastic. This way we have avoided using a large sparse arrays for position representation. These large arrays would have slowed the computation by over and again copying large number of values representing a 0% probability of occurrence. Even though the proposed solution consumes part of the saved time by declaring this data structure, it is still greatly beneficial. Probabilities are always stored using the location as a key and the probability being the value.

These probabilities present a method for storing the uncertainty of a state in the environment. We use these values to generate outcomes of stochastic actions, to sample a deterministic state from these probabilities and to change these probabilities. Probabilities are handled by *ProbDistribution*,*Decider* classes and by the expansion part of MCTS algorithm . The first one handles the changes of probabilistic distribution in locations issued by the outcome of search action. Second on is present to sample a deterministic image of the state from its stochastic template.

The change of probabilities is executed each time when we expand the Monte Carlo Tree and we want to predict change of targets positions. From the current probability distribution of target locations we take all the particles. A particle in this case means a probability with specified location and target identifier of this particle. For each of these particles we can identify all the possible destinations depending on current position and their target affiliation. Next we find their next steps towards these destinations and make a unique collection. These next steps are then used to create new particles which then carry a proportional part of the probability in the expanded nodes.

Probability distributor is needed to set the probabilities in nodes that represent outcome of a stochastic action. In our case we use it to define the probabilities that match the state after execution of search action. Each child node of the search node represents a different possible outcome of the search for the targets. We have already covered in chapter about MCTS expansion how it creates child nodes are how to distinguish them later. What we did not mention was how exactly the state of these child nodes is different and how the outcome of search is projected into probabilities describing target's location. Each target's position is defined by $< 1 - L >$ probabilities, where L is the upper bound set by number of locations in the environment. As a part of the expansion of a search node we have the enemy presence vector. This vector identifies which enemies have a non-zero probability of occurrence in given sector where search was activated. Based on that we can pre-compute probabilities of each target in both cases it has been found or otherwise. Then we combine these pre-computed chunks according to individual outcome nodes. We are again trying to avoid repetitively computing the same thing.

A single outcome of a search for an enemy changes the probabilities as follows. If a target has been found there, we simply throw away all particles representing location of this target outside of given sector. The location inside is then set to 1. At the other end if we were not able to localize the target, then all of the probabilities inside of the sector are redistributed among the particles outside. The sector where search was executed is searched for target occurrence and the found probability is then relatively distributed among the particles outside of the search sector.

The last paragraph is dedicated to a sampling class called Decider. This class fulfills the role of sampling deterministic objects from a stochastic representation in nodes. It is used to sample position of targets from a probabilistic distribution and to sample a destination for each of the moving targets. To sample these values from given probabilities we are using the roulette wheel selection known for example from genetic algorithms. In our case the probability values in particles representing a position of one target sum up to 1. In order to choose a sample according to these probabilities we have to generate a random real number from a range $(0, 1>$. Then we go through these particles and sum up their probabilities. When the summed value exceeds the random number we can stop and we select the last particle as our representant in the sample. This way we can sample all the representants in our sample and we have a completely deterministic representation of positions.

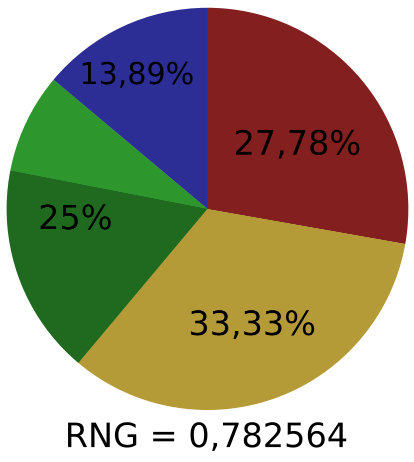

Figure 5.6: Example of roullete wheel sampling.

5.4.3 Visualization

For better observation of the progress of the simulation we have implemented a simple visualization interface. It is based on java Swing and it repaints the UAV a target positions into the environment map after each planning step. In consists of few data structures for storing the information about the environment and the objects in it. Then few methods dedicated to update these information and a paint method which is able to draw the information into the window.

The visualization shows several important things. First thing it show the layout of the map, all the roads in gray color and the possible destination locations in darker shade of gray. Then after each iteration of the main planner loop when all UAVs executed their actions it repaints the position of UAVs a targets. UAVs are shown as blue squares, while targets are the yellow ones.

Another interesting ability of this visualization is to show for each UAV the locations with non-zero probability of enemy incidence. These probabilities are shown as a small red or black squares. One square represents a probability for each target with the most left one being the first targets probability. The closer the probability gets to 100% the more they shine in red. To be able to change between the individual beliefs of the UAVs we have implemented a keyboard listener. By pressing a number on the keyboard the view is automatically change to view of UAV with the corresponding id. To know which view is currently active the UAV is marked in cyan color.

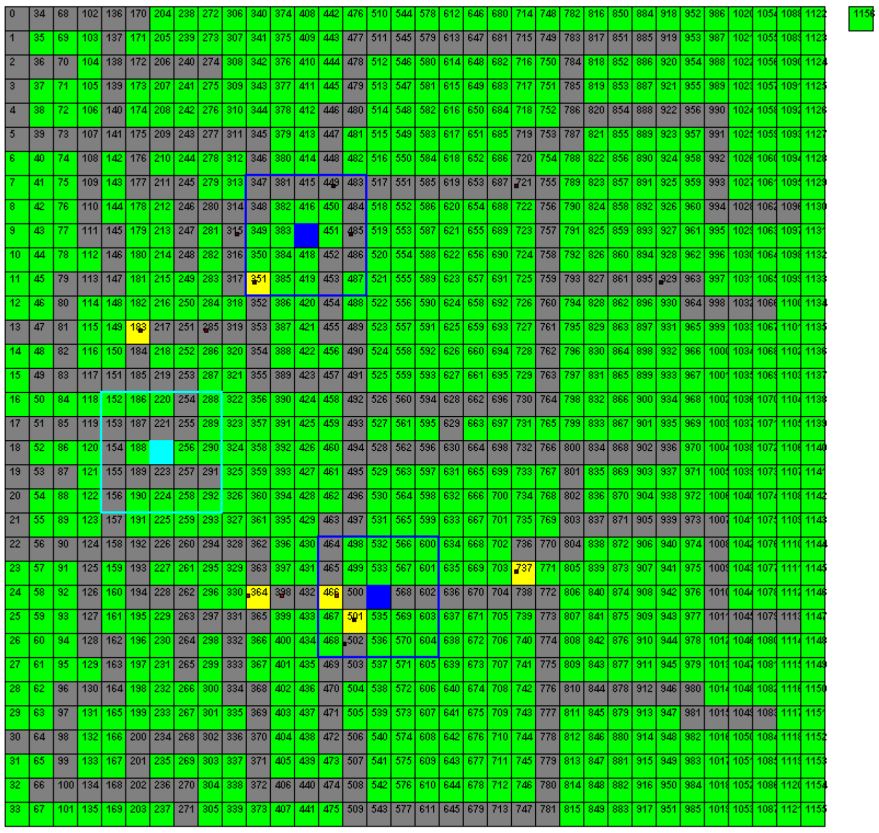
Figure 5.7: Visualization of 3UAVs tracking 6 targets.

Chapter 6

Experiments

This chapter evaluates proposed solution from the perspective of quality, scalability and in direct comparison with other previously known methods used to obtain solution. At first we describe the testing, its environment and design of these experiments. Then we conclude the experiment and discuss the results and competing methods. The output files generated after each experiment provide a complete information about experiment setting. Detailed reports about results of the experiments can be found on the included CD.

We planned out a series of experiments to test out our implementation. First we have observed and decided which factors we want to test and for what purpose will they be conducted. Also we wanted to compare different setting of parameters. The results for tested problem instances are dependent on the setting of main MCTS parameters. If we want to maximize the utility and offer results comparable to other proposed solutions, we need to compare several combinations of parameter settings and decide which among them promise biggest utility gains. Following that we have identified variables that we want to study. Our intention is to observe changes in obtained reward, time of execution and how they depend on various variables. We want to compare the results of the decentralized solution developed in this thesis with a centralized and a self-absorbed solution. The comparison is based on giving the centralized solution n-times the samples of decentralized solution, where n is the number of UAVs in given scenario.

The main properties we want to identify are

- Setting of MCTS parameters that maximize obtained utility.
- Compare reward obtained in presented settings.
- Compare scalability in the individual settings.

Environment

The whole planner and simulator is written in the JAVA language. The implementation should be multi-platform and not dependent on any other software or library version. It is build and compiled into a one jar file and uses external map files located inside the resources folder. To run an experiment we recommend to use the same version as was used in the process of development. The tested version is `JAVA 1.7.0_67-b01` (64-bit).

Execution of each individual experiment was performed on MetaCentrum. MetaCentrum is an virtual organization that unites and shares computers of academic and research facilities. Their grid offers distribution of workload among the accessible systems. Thanks to that we had the opportunity to run more and much longer experiments than it would be possible on a single computer. The computers are mostly running a Debian 8 version of Linux. When queuing the experiments we can dedicate a number of CPUs and define an amount of memory to be used during the experiment. We have used the setting where 16 CPUs and maximum of 20GB memory can be used.

Execution

The experiments are operated by two BASH scripts. First script queues the experiments into MetaCentrum. It generates combinations of specified parameters a queues these experiments in the line for execution. The second one is then run when one of the grid computers is assigned to this task. This second script copies data needed for the experiments, defines JAVA version, size of JAVA heap to 8GB and runs the experiment with specified parameters. Each experiment is run with the 15 hour limit and 10 iterations. By running the same experiment with exactly the same parameters we can reduce the randomness of the algorithm results. After the experiment finishes we gather the output files and we can use them to gain mean values for each of the completed runs.

To run a single experiment we have to run the jar file of the desired planner and to specify several parameters.

java − Xmx8G − jar DIP.jar \$map \$uav \$enemies \$samples \$par \$depth \$mode
\$discount \$threads \$steps

The parameters are used to change the environment map, alter settings of MCTS algorithm, set the depth of simulation and reward discount factor. Additionaly we can also change the number of threads used in the simulation and number of steps after which we end the planning.

6.1 Parameter selection

First experiment that belongs to this category is the experiment for optimization of MCTS parameters. As we have no understanding how each parameter affects the global reward, we have to make a few experiments to see which one have greater impact and which values are associated with best performance of this algorithm. The indicated parameters that should be tested are number of samples, depth of simulation and discount factor for the reward function. These parameters affect how the algorithm searched the state space and how much it prefers immediate gain of rewards. The tested parameters were combinations of following settings

- Samples = 500, 1000, 5000, 10000, 50000
- Depth = 20, 40, 60
- Discount factor = 0.5, 0.7, 0.9

If we look at the results (fig. 6.1) of this experiment we can see how the discount factor affects the obtained reward. As we can see the discount factor 0.7 provides biggest gains of reward. Other settings have approximately the same dispersion of values, however both the highest mean and the highest absolute reward were acquired with this settings.

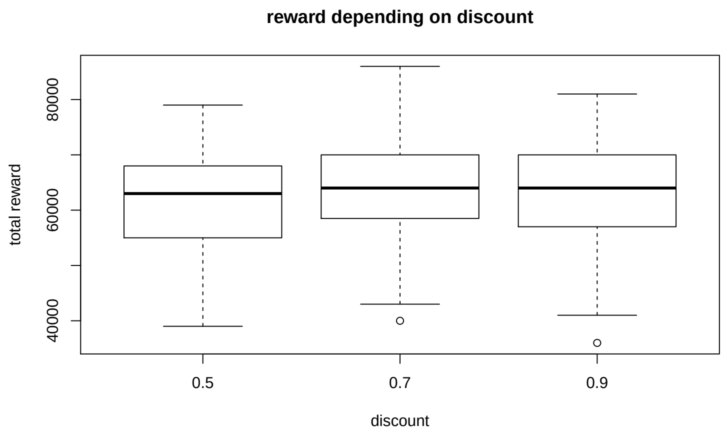

Figure 6.1: The effects of discount factor on obtained rewards.

The next graph (fig. 6.2) shows us how the reward depends on number of selections and simulations done by the algorithm. As we can see with higher number of samples the reward stays around the same values, but the variance is lower. The results for 50000 samples can be a little misleading. Since the computation time is dependent on number of samples we only managed to acquire two results for this setting. If we look at the graph we can expect that with increased number of samples we can expect more accurate values with stagnating or maybe slightly increasing mean values.

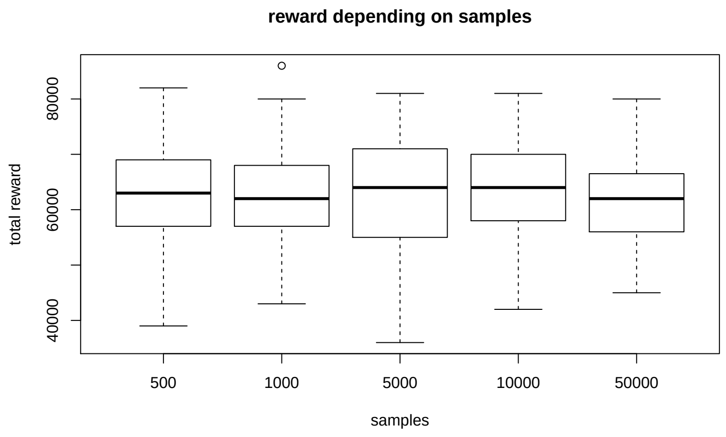

Figure 6.2: Effects of number of samples on rewards.

6.2 Quality

The experiments concluded to study the quality of obtained solution are all executed with the preset MCTS parameters gained from the previous experiment. The parameters are 10000 samples, the depth of each simulation run set to 40 actions and the discount factor set to 0.7. With these settings fixed we have selected our experiment variables and few levels for each of them. To receive results non-dependent on other factors we have to run the same experiments on multiple maps with different target routes and destinations. The layout of these maps can be seen in appendix B. We have used following combinations of these parameter settings

- 6 maps - size from 16x16 to 64x64 with 8-15 destination locations.
- N UAVs, where $N \in < 1, 3 >$.
- M targets, where $M \in < N, 2N >$.

We can further study the quality of presented solution with a comparison to results obtained by (baseline) centralized solution and the self-approximation solution. The same experiments were executed for all three versions and the data can be compared both in terms of quality, scalability and time demands. On (fig.6.3) you can see the average values gained by each of the compared solutions. The values are shown across all UAV, target and map combinations.

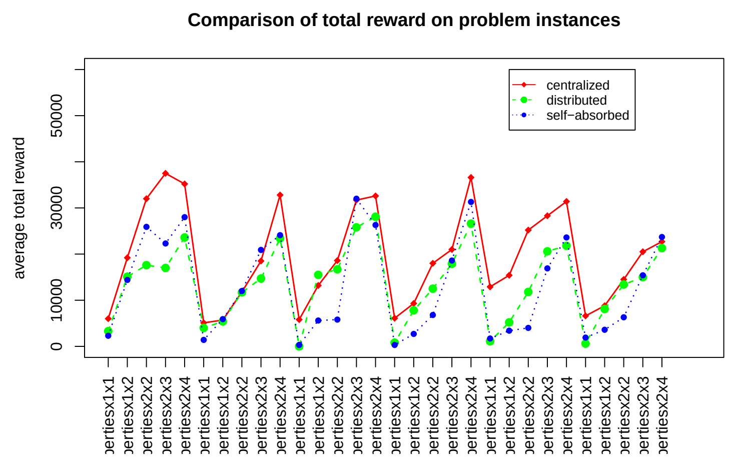

Figure 6.3: Comparison of obtained rewards.

The scatterplots (fig.6.5) depict the comparison of obtained rewards by all three solutions. As we can see the centralized version yields the best results. Unfortunately the results of solution proposed in this thesis does not outperform the self-absorbed solution. In many instance results of both solutions are similar and in general on this experiment sample it seems that the UAV simulation does not provide deemed benefits. This means that further study of the simulation and extensive testing will be necessary. On the other hand a minor

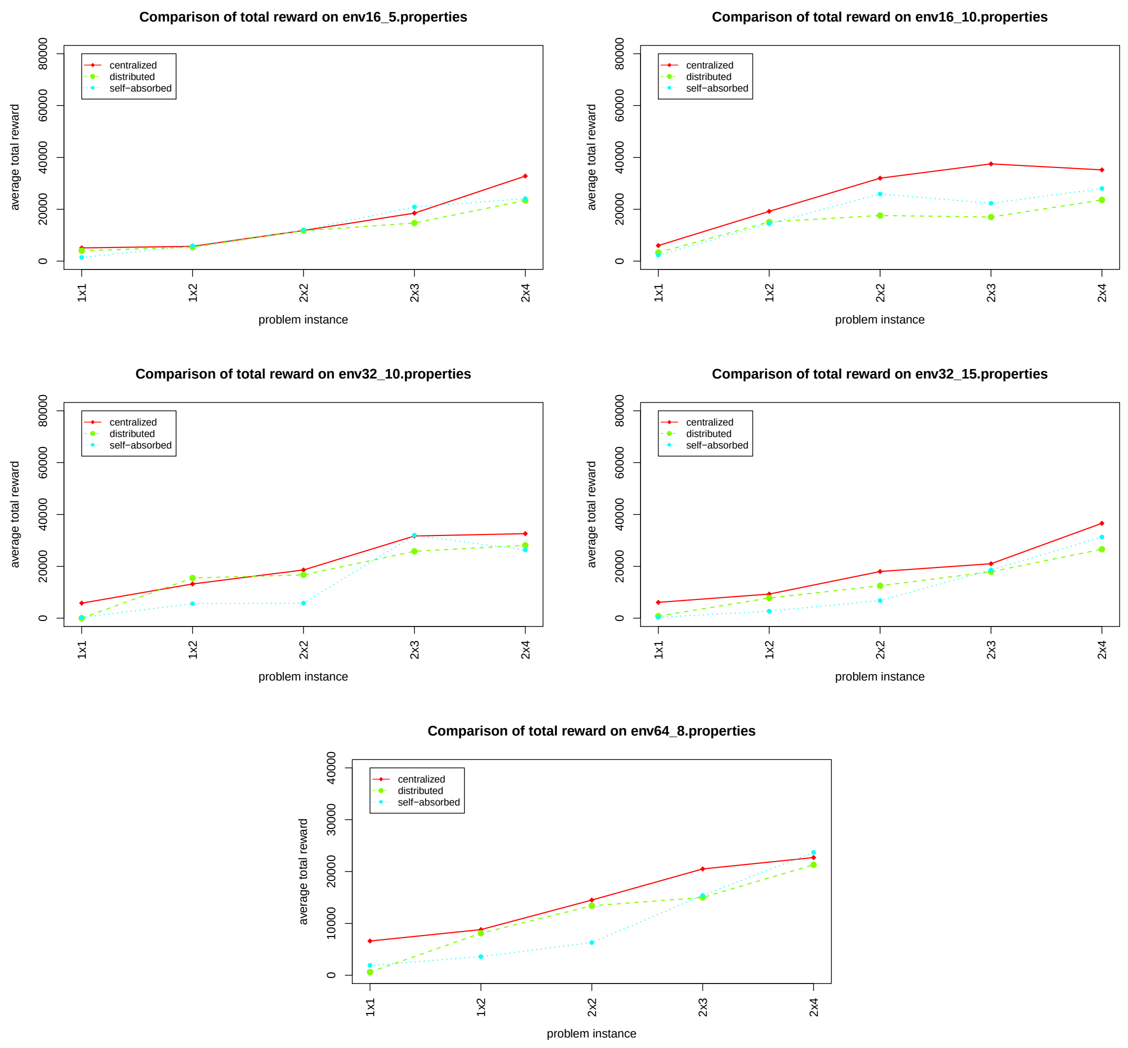

Figure 6.4: Comparison of different maps.

positive thing is that there are more instances where the proposed solution surpassed the reward gained by the centralized solution.

The graphs that compare global reward (fig.6.6) and rewards gained by individual UAVs (fig.6.5) show that as expected the centralized version has a little bit higher rewards. Despite of that there are several exceptions where the decentralized solution scored analogous and even few with slightly better results. From this comparison we can state that the rewards are slightly higher in the centralized version as anticipated from their nature.

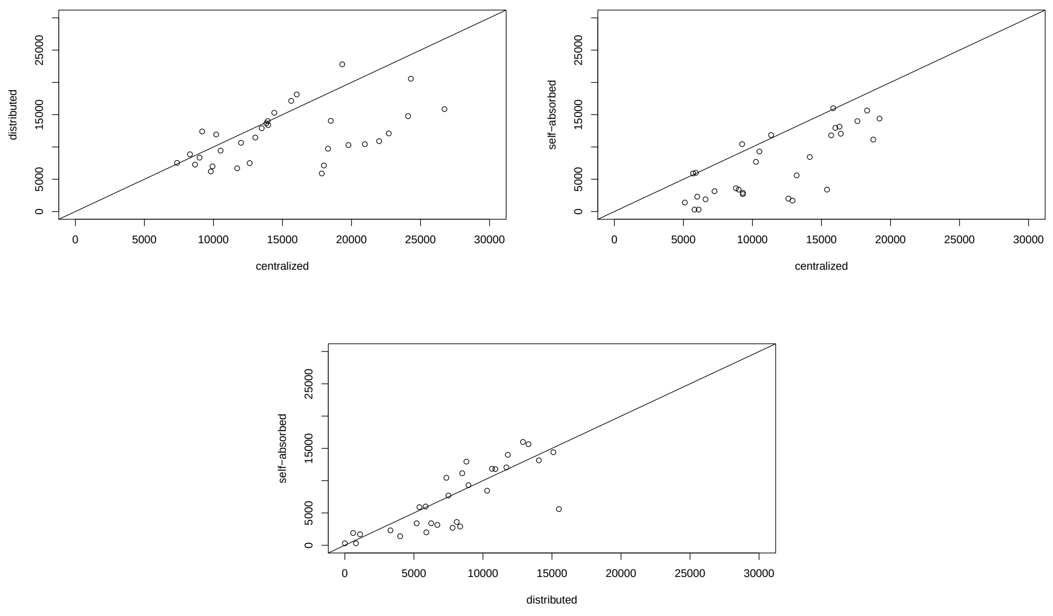

Figure 6.5: Values of agent rewards in compared solutions

In order to eliminate effects of the environment, we compare gained rewards on the same problem in a series of experiments on different maps with two possible destination location sets. On the graphs (fig.6.4) you can see that the maps and number of possible destinations for targets have a slight effect on the obtained reward. The structure of the map and number of destinations influence how probabilities of target's incidence are spread on the map in time. This is the reason why we also compare individual results on given maps.

The results on different maps show that the size of the map has an effects on the obtained reward (fig.6.8 and 6.4) . At smaller maps the difference is marginal, but at the largest one, even when the number of target destination is lower, the reward is smaller. The influence of the number of target destinations cannot be evaluated alone. It is bound with the location of target's destination which subsequently influences the quantity of used road segments.

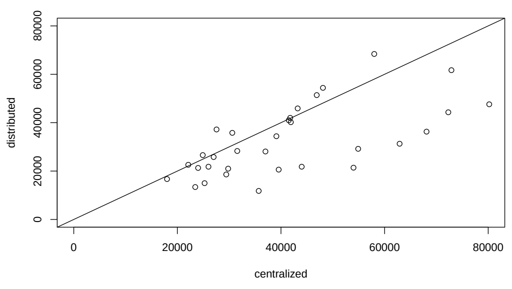

Figure 6.6: Comparison of obtained global reward by centralized and distributed solutions.

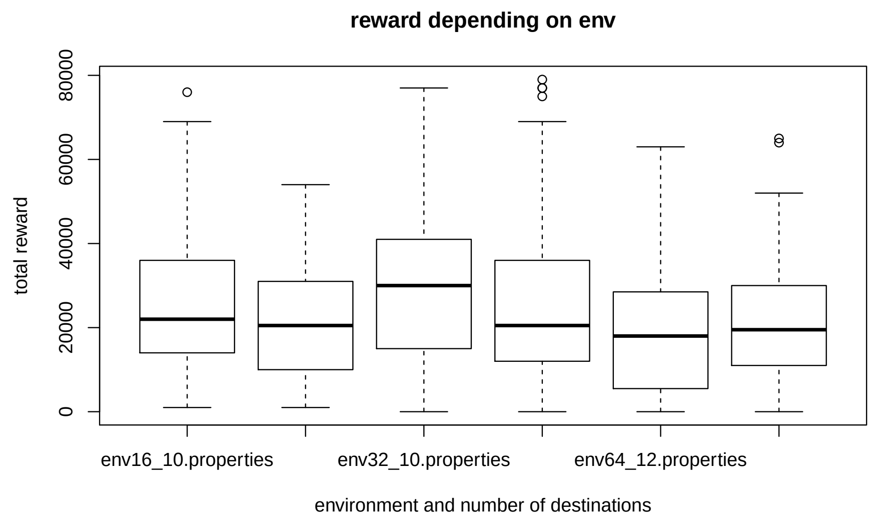

Figure 6.7: Reward depending on environment.

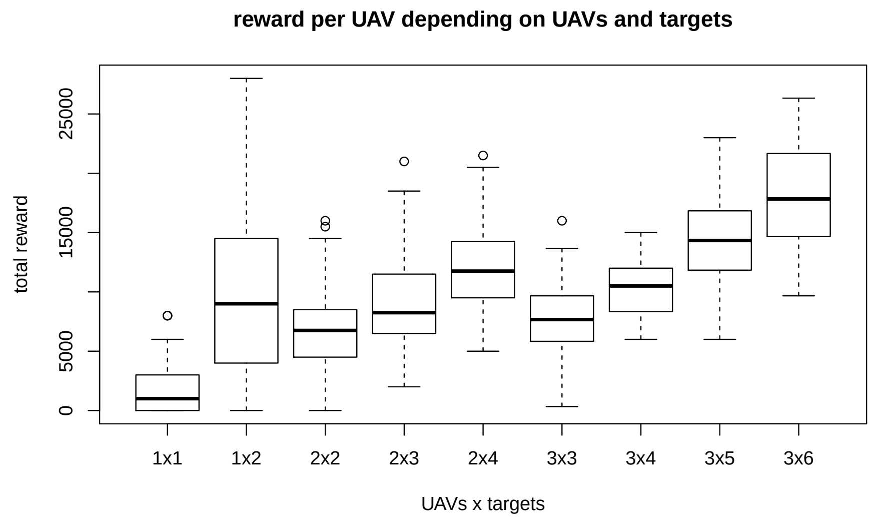

Figure 6.8: Reward per UAV dependances.

At figure 6.8 you can see how reward is dependent on number of UAVs and the number of targets present in the problem instance. Furthermore figure 6.9 shows the histograms of obtained reward in problems with 3 UAVs. You can see the obtained reward values for both 3 target and 6 target instances.

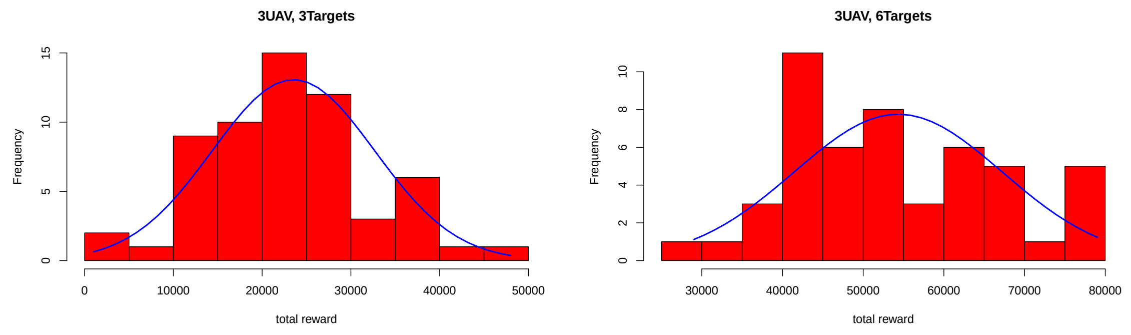

Figure 6.9: Histograms of obtained reward.

Interesting information can be found, if we take a look at the figure 6.10. It shows the increase of reward earned per single UAV in situation when we decrease the total number of UAVs by one. In this example a 3 UAV, 3 target scenario is compared with a 2 UAV, 3 target one. The figure 6.11 then shows how reward is affected both by the number of UAVs and the number of targets.

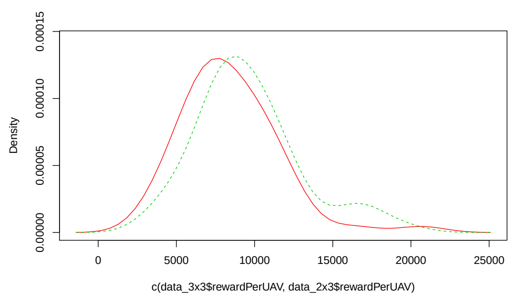

Figure 6.10: Reward per UAV.

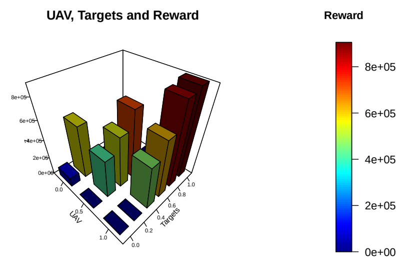

Figure 6.11: UAV, Target, Reward dependence.

6.3 Scalability

In this part we talk about the scalability of our solution. The concluded experiments focus on how certain parameters influence the computation time. We want to observe how the growing number of UAVs influence the complexity. Moreover we have identified more variables that we want to monitor and learn about their effect on the performance while computing a plan. For example they are number of targets. We are aware that these results can be dependent on concrete road network structure and this problem would require more extensive testing. But for the sake of the solution comparison it is sufficient.

First we would like to present some data gained from the parameter setting experiments that also provide information about scalability. The experiments with different settings of the MCTS parameters. All the parameter combinations can be found in section 6.1.

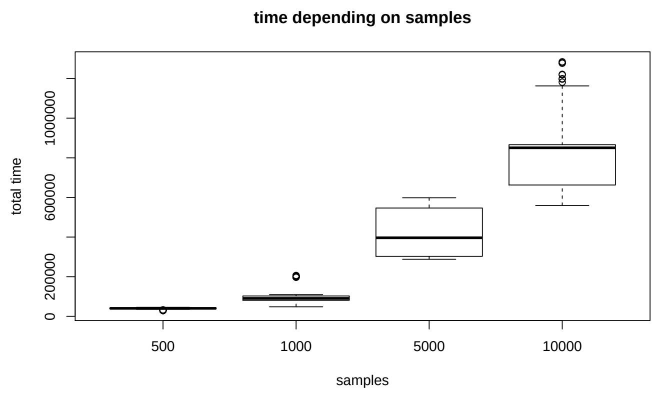

Figure 6.12: Detailed time dependence on the number of samples.

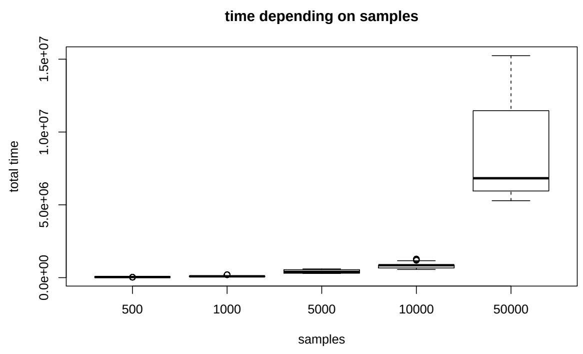

Figure 6.13: Time dependence up to 50000 samples.

What we can see from the results (fig.6.12) gained across all the different settings is that the time needed to solve the tracking problem is dependent on number of the samples. The experiments with number of samples up to 10 000 finished in under 30 minutes. In case of the 50 000 samples, we can see the computation times growing to somewhere around 3 or 4 hours.

The next experiment was focused especially on the scalability. Again we have used the MCTS parameters settings 10000 samples, 40 depth of simulation and 0.7 reward discount factor gained in previous experiment (section 6.1). Unlike in the last experiment we have also locked the map selection to an unchanging middle sized environment.

This experiment was designed to provide information on scalability of proposed solution when more UAVs are present. We want to survey if the computation time changes when we add more UAVs. Also the important question is how the computation times change when more target entities with the need to select an action at each time step are present. The selected levels for our parameters are as follows

- Number of UAVs $< 1, 10 >$.
- Number of targets = 5, 10.

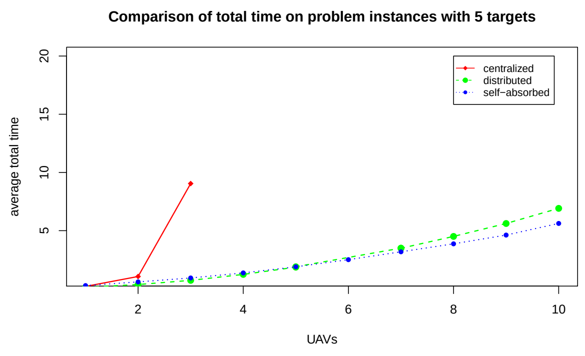

Figure 6.14: Time complexity dependence on the number of UAVs.

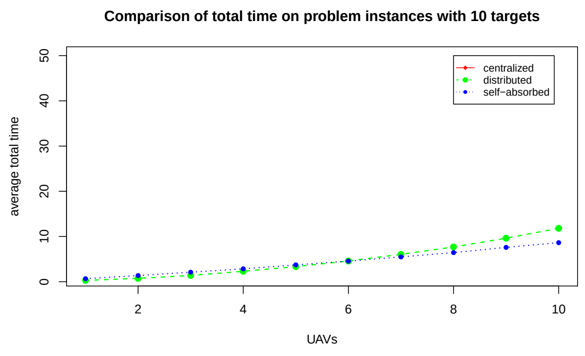

Figure 6.15: Time complexity with increasing number of UAVs and 10 targets.

The results of scalability experiments can be seen at figure 6.14 and 6.15. As we have anticipated the centralized solution is the most time consuming. The growth between instances with 3 and 4 UAVs is much greater compared to the increase seen in both the distributed and self-absorbed solutions. The time complexity is not the only problem of the centralized version.

As the figure 6.14 depicts, the instances with greater number of UAVs did not finish due to their memory demands. In the centralized solution the number of states is growing rapidly with both the number of UAVs and the number of targets. The state space size grows with both the number of combinations of UAV's actions and all the possible outcomes of the search for targets. This is the reason why we have obtained even less results for the 10 enemy instance.

The comparison between the distributed solution presented in this thesis and the self-absorbed solution offers more complex information. Both figure 6.14 and 6.15 depicts that the computation time of the decentralized solution is higher, but the increase is not that steep. We can state that the increase of time caused by the addition of the UAV simulation is reasonable. Also both the decentralized and self-absorbed solutions are more scalable in regards to both the time and memory requirements.

Chapter 7

Conclusion

We have studied the concepts of multi-agent planning, the foundations of both the classical and the probabilistic planning. With the use of the recommended literature and a number of additional books and articles, we have familiarized ourselves with a number of models based on Markov decision processes, algorithms used in planning for computing both the optimal and approximated policies, and effective approximations.

In this thesis we have focused on the subclass of planning problems called multi-agent planning for decoupled tasks. We have studied the unique features of this problem, researched solutions and state of the art methods used for solving similar problems. We have judged the usability and benefits of these methods in connection with our problem. The real-world applications of the decoupled tasks were also considered and the multi-agent multi-target tracking problem was selected as the ideal representant to demonstrate our solution.

As the first step of our solution we have described the general problem of multi-agent probabilistic planning for decoupled tasks. Following this description we have formulated a theoretical model with specific properties rising from the substance of this problem subclass. We have proposed a solution that exploits the problem characteristics and incorporates use of problem decomposition, self-absorbed approximation and simulation of other agents' actions by means of the alteration of the obtained reward. This general model and proposed solution was then used as the starting point for a model specifically tailored for the real-world problem of multi-agent multi-target tracking. This second model is more specific and provides definitions of the environment, UAVs and tracked targets.

The planner and simulation environment was then built on the basis of this second model. The implementation illustrates the solution for multi-agent multi-target tracking problem and enables the evaluation by experiments. It is based on the decentralized planning where each UAV uses Monte Carlo Tree Search algorithm for acquisition of their next action. The implementation also builds on the UCT metric and additional improvements for tree expansion and probabilistic planning. The simulation part of the MCTS then incorporates an innovative algorithm for simulation and evaluation of presence of other UAVs by means of reward altering.

The thesis also presents results of experiments used to evaluate characteristics of proposed solution. These experiments are focused on setting the algorithm parameters, quality of the presented solution and the scalability in real-world applications. The results of these experiments were confronted with results obtained by (baseline) centralized and the self-absorbed solutions. As expected the centralized solutions outperforms both the decentralized and self-absorbed solutions. If we look at the comparison of obtained reward between the presented decentralized solution and the self-absorbed one, we can see that in many scenarios the results are similar. As opposed to what we have expected the results of solution proposed in this thesis compared to the self-absorbed solution did not present the anticipated increase in utility.

We have several ideas what may cause this outcome. It can be a result of insufficient effect of the reward alteration based on other UAVs location. This can be solved by introducing a parameter that will enable to control it's effects. Another possibility is that the process of simulating the surrounding UAVs is too simple. The randomness of the simulation may cancel out most of the effect of their presence. In each decomposition the main UAV follows a much more sophisticated model of behavior than in the simulation. In the future we suggest adding multiple models of their behavior and compare them in extensive testing. The complexity of these new models must balance it's benefits with the time demands to preserve the scalability of presented solution. Also the model in current use does not guarantee localization of target and it can happen that all surrounding UAVs decide not to locate the target and rely on others to do so.

The experiments also provide comparison of scalability of proposed solution. As we have expected the advantage of the decentralized solution presented in this thesis was proven far more scalable than the centralized solution. The computation times were significantly shorter than for the centralized version and moreover the experiments have pointed out that the memory demands are also much smaller. Compared to the self-absorbed solution the increase of time consumption of proposed solution by adding the simulation and evaluation of nearby UAVs in solution presented in this work is reasonable.

This thesis and the solution proposed by it shows, that the specifics of decoupled tasks present an opportunity for their exploitation in order to make the planing more scalable. The decoupled tasks and their usage to model large real-world problems should be further researched. We believe that the scalability and quality of the obtained solution can be further improved on.

7.1 Future work

Future work and additional improvements can be done in regards of both the proposed general model in this thesis, or the solution of the multi-agent multi-target tracking problem. We present a short list of possible areas of additional research:

- Improve UAV simulation
  One of the possibilities is to improve the simulation of surrounding UAVs and their effect on reward by adding multiple models of their behavior and perform a set of comparing experiments. Also the addition of control parameter for the reward alteration caused by other UAVs in close proximity can be introduced and possibly combined with a machine learning algorithm for its setting.
- Additional real-world problems
  Another follow-up to this work could be the definition of other real-world problem as the multi-agent planning for decoupled tasks. This could also provide additional ideas for improvement of the general model for solving decoupled tasks that is presented in this thesis.
- Expanding the presented problem
  The current problem of multi-agent multi-target tracking can be further enhanced to employ fuel capacity and refueling actions, movement restrictions or for example a thread in a form of anti-air defense or unwanted radar tracking which must be evaded.

Bibliography

[1] AMATO, C. The Dec-POMDP Page, 2016. Dostupné z: <http://rbr.cs.umass.edu/camato/decpomdp/index.html>.

[2] BERNARDINI, S. et al. Leveraging Probabilistic Reasoning in Deterministic Planning for Large-Scale Autonomous Search-and-Tracking. In COLES, A. J. et al. (Ed.) *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, s. 47–55. AAAI Press, 2016. Dostupné z: <http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13178>.

[3] BERNSTEIN, D. S. et al. The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.* November 2002, 27, 4, s. 819–840. ISSN 0364-765X. doi: 10.1287/moor.27.4.819.297. Dostupné z: <http://dx.doi.org/10.1287/moor.27.4.819.297>.

[4] BOUTILIER, C. Planning, Learning and Coordination in Multiagent Decision Processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '96, s. 195–210, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. Dostupné z: <http://dl.acm.org/citation.cfm?id=1029693.1029710>. ISBN 1-55860-417-9.

[5] BRAZIUNAS, D. POMDP Solution Methods. Technical report, Department of Computer Science University of Toronto, 2003.

[6] BROWNE, C. et al. A survey of Monte Carlo tree search methods. *IEEE TRANSACTIONS ON COMPUTATIONAL INTELLIGENCE AND AI*. 2012.

[7] CAPITAN, J. – MERINO, L. – OLLERO, A. Decentralized cooperation of multiple uas for multi-target surveillance under uncertainties. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, s. 1196–1202. IEEE, 2014.

[8] CARLIN, A. – ZILBERSTEIN, S. Value of Communication in Decentralized POMDPs. In *Proceedings of the AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains*, Budapest, Hungary, 2009. Dostupné z: <http://rbr.cs.umass.edu/shlomo/papers/CZmsdm09.html>.

[9] CHANEL, C. P. C. – TEICHTEIL-KöNIGSBUCH, F. – LESIRE, C. Multi-Target Detection and Recognition by UAVs Using Online POMDPs. In DESJARDINS, M. – LITTMAN, M. L. (Ed.) *AAAI*. AAAI Press, 2013. ISBN 978-1-57735-615-8.

[10] CHASLOT, G. M. – WINANDS, M. H. – HERIK, J. H. Parallel Monte-Carlo Tree Search. In *Proceedings of the 6th International Conference on Computer and Games*. Springer, 2008.

[11] CLAES, D. et al. Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, s. 881–890, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems. Dostupné z: <http://dl.acm.org/citation.cfm?id=2772879.2773265>. ISBN 978-1-4503-3413-6.

[12] FIKES, R. E. – NILSSON, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*. 1971, 2, 3-4, s. 189–208.

[13] HAY, D. – SHIRAZIPOURAZAD, S. – SEN, A. Optimal Tracking of Multiple Targets Using UAVs. In *Combinatorial Optimization and Applications - 8th International Conference, COCOA 2014, Wailea, Maui, HI, USA, December 19-21, 2014, Proceedings*, s. 750–763, 2014.

[14] HOWARD, R. A. Dynamic programming and Markov processes, 1960. Dostupné z: <http://opac.inria.fr/record=b1117485>.

[15] KAELBLING, L. P. – LITTMAN, M. L. – CASSANDRA, A. R. Planning and Acting in Partially Observable Stochastic Domains. *Artif. Intell.* May 1998, 101, 1-2, s. 99–134. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X. Dostupné z: <http://dx.doi.org/10.1016/S0004-3702(98)00023-X>.

[16] KOCSIS, L. – SZEPESVáRI, C. Bandit based Monte-Carlo Planning. In *In: ECML-06. Number 4212 in LNCS*, s. 282–293. Springer, 2006.

[17] MCDERMOTT, D. et al. PDDL-the planning domain definition language. 1998.

[18] OLIEHOEK, F. A. Decentralized POMDPs. In WIERING, M. – OTTERLO, M. (Ed.) *Reinforcement Learning: State of the Art*, 12 / *Adaptation, Learning, and Optimization*. Berlin, Germany: Springer Berlin Heidelberg, 2012. s. 471–503.

[19] OLIEHOEK, F. A. – SPAAN, M. T. J. – VLASSIS, N. Dec-POMDPs with delayed communication. May 2007.

[20] POOLE, D. – MACKWORTH, A. *Artificial Intelligence: Foundations of Computational Agents*. New York, NY, USA : Cambridge University Press, 2010. ISBN 0521519004, 9780521519007.

[21] ROTH, M. – SIMMONS, R. – VELOSO, M. Decentralized Communication Strategies for Coordinated Multi-Agent Policies. In *Multi-Robot Systems: From Swarms to Intelligent Automata, Volume III*, 2005.

[22] RUSSELL, S. J. – NORVIG, P. *Artificial Intelligence: A Modern Approach*. New Jersey : Pearson Education, 2 edition, 2003. ISBN 0137903952.

[23] SILVER, D. – VENESS, J. Monte-Carlo Planning in Large POMDPs. In *In Advances in Neural Information Processing Systems 23*, s. 2164–2172, 2010.

[24] SPAAN, M. – AMATO, C. – ZILBERSTEIN, S. *Decision Making in Multiagent Settings: Team Decision Making* [online]. 2011. Dostupné z: <`http://users.isr.ist.utl.pt/~mtjspaan/tutorialDMMS/tutorialAAMAS11.pdf`>.

Appendix A

List of abbreviations

UAV	Unmanned aerial vehicle
SaT	Search and tracking
MCTS	Monte Carlo Tree Search
SPATAP	Spacial Task Allocation problem
MDP	Markov decision process
POMDP	Partially observable Markov decision process
Dec-POMDP	Decentralized partially observable Markov decision process

⋮

Appendix B

Environment maps used in experiments

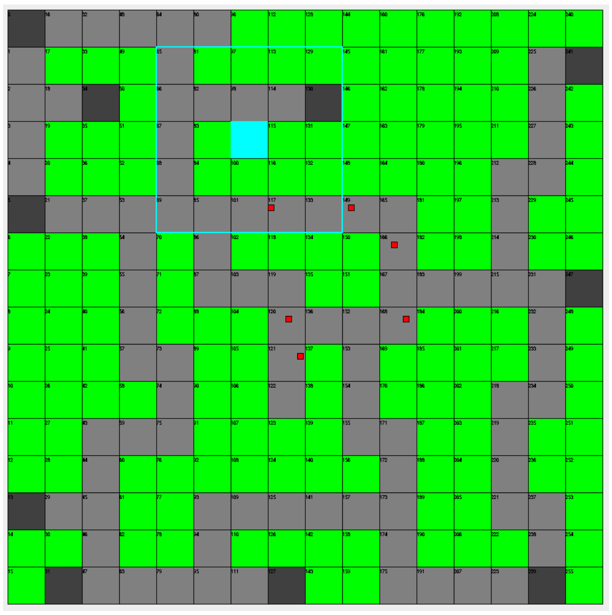

Figure B.1: 16x16 map with 10 destinations

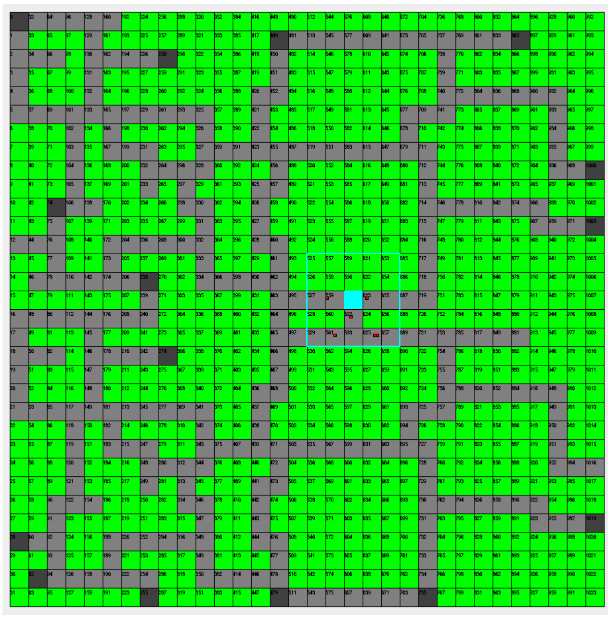
Figure B.2: 32x32 map with 15 destinations

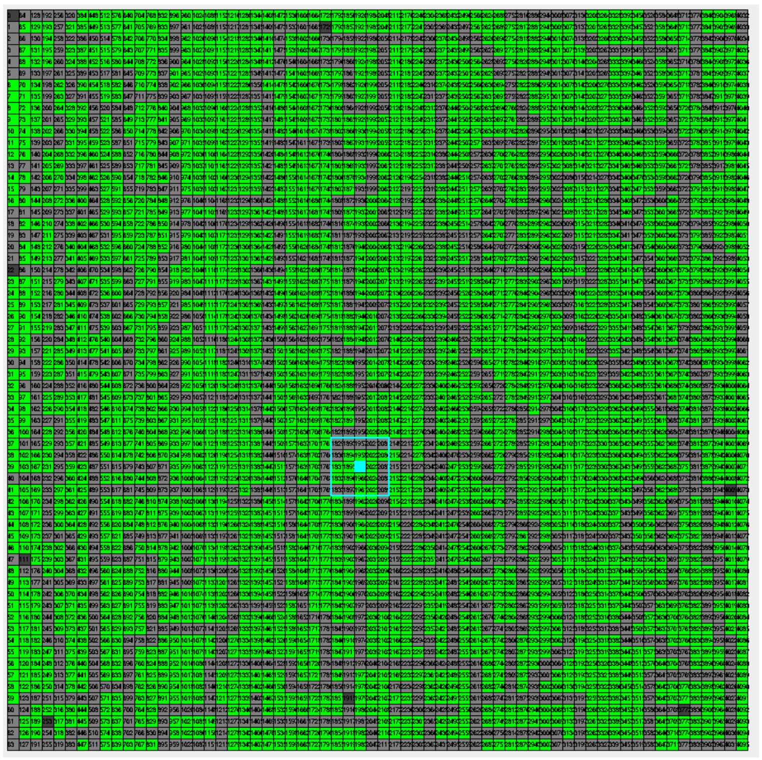
Figure B.3: 16x16 map with 8 destinations

Appendix C

Experiment results preview

	uavs	targets	samples	par	depth	discount	avgTotalReward	avgTotalTime	avgRewardPerTime
1	3	6	500	1	20	0.50	66600.00	0.66	101903.91
2	3	6	500	1	20	0.70	62600.00	0.65	96949.72
3	3	6	500	1	20	0.90	63300.00	0.50	126314.72
4	3	6	500	1	40	0.50	57300.00	0.66	86530.63
5	3	6	500	1	40	0.70	67200.00	0.67	100312.39
6	3	6	500	1	40	0.90	64200.00	0.67	96347.55
7	3	6	500	1	60	0.50	56800.00	0.73	77708.55
8	3	6	500	1	60	0.70	61100.00	0.73	83682.08
9	3	6	500	1	60	0.90	62600.00	0.67	92983.44
10	3	6	1000	1	20	0.50	61000.00	0.97	63277.01
11	3	6	1000	1	20	0.70	60800.00	0.95	64410.10
12	3	6	1000	1	20	0.90	63100.00	1.35	46656.65
13	3	6	1000	1	40	0.50	59000.00	3.35	17611.29
14	3	6	1000	1	40	0.70	65300.00	1.51	43180.10
15	3	6	1000	1	40	0.90	65800.00	1.52	43194.55
16	3	6	1000	1	60	0.50	60500.00	1.59	38376.77
17	3	6	1000	1	60	0.70	65500.00	1.59	41423.54
18	3	6	1000	1	60	0.90	65300.00	1.74	37601.14
19	3	6	5000	1	20	0.50	61800.00	5.00	12376.42
20	3	6	5000	1	20	0.70	66700.00	4.96	13446.45
21	3	6	5000	1	20	0.90	63300.00	5.05	12543.33
22	3	6	5000	1	40	0.50	61600.00	6.55	9408.28
23	3	6	5000	1	40	0.70	63500.00	9.78	6490.02
24	3	6	5000	1	40	0.90	60900.00	8.82	6905.57
25	3	6	5000	1	60	0.50	65500.00	9.46	6933.58
26	3	6	5000	1	60	0.70	59100.00	6.53	9058.15
27	3	6	5000	1	60	0.90	62600.00	7.87	7980.74
28	3	6	10000	1	20	0.50	59500.00	10.79	5594.26
29	3	6	10000	1	20	0.70	67700.00	10.88	6224.36
30	3	6	10000	1	20	0.90	64700.00	10.90	5939.08
31	3	6	10000	1	40	0.50	63600.00	14.46	4401.46
32	3	6	10000	1	40	0.70	67300.00	14.49	4648.24
33	3	6	10000	1	40	0.90	66000.00	13.45	4912.98
34	3	6	10000	1	60	0.50	60600.00	14.19	4272.67
35	3	6	10000	1	60	0.70	60800.00	14.39	4225.69
36	3	6	10000	1	60	0.90	64900.00	20.50	3171.69
37	3	6	50000	1	20	0.50	61444.44	90.15	682.32
38	3	6	50000	1	20	0.70	62777.78	99.84	629.08
39	3	6	50000	1	20	0.90	65125.00	101.26	643.24
40	3	6	50000	1	40	0.50	57250.00	190.54	303.58
41	3	6	50000	1	40	0.70	66142.86	125.93	525.02
42	3	6	50000	1	40	0.90	63250.00	185.79	340.77
43	3	6	50000	1	60	0.50	60000.00	212.46	282.05
44	3	6	50000	1	60	0.70	61666.67	247.88	248.50
45	3	6	50000	1	60	0.90	56500.00	192.16	294.19

Table C.1: Average values obtained in parameter setting experiment

Appendix D

User manual

In order to run the planner you have to install JAVA runtime environment first. Any version of JAVA 7 or above should suffice. The software was tested on version `JAVA 1.7.0_67-b01`. No further software installation is required.

The source code can be found on the included CD or on the Bitbucket repositories `https://bitbucket.org/starouscz/dip_janstary` .

The planner can be either run by a script or from the command-line. It is necessary that the folder `.\resources` is present in the location of main jar file of the application. This folder contains necessary map files. To run the planner from the command-line use

```
java -jar DIP.jar $map $uav $enemies $samples $threads $depth 1
$discount $threads $steps
```

for example with parameters `env16_5.properties,2,3,10000,1,40,1,0.7,1,20` . To make further experiments or to try other settings the map file must be edited. It can be opened in a ordinary text editor. Inside the map file you can specifies size of the map, layout of roads, starting positions of targets and UAVs, fuel of the UAVs and possible and true destinations for targets. In order to run a problem with a certain number of UAVs and targets their starting positions have to be specified inside the map file.

Appendix E

Content of the enclosed CD

```
.
|-- binaries                                        #planner executables
|   |-- DIP.jar
|   |-- DIP_noGUI.jar
|   |-- resources                                   #map folder
|   |   |-- env16_10.properties
|   |   |-- env16_5.properties
|   |   |-- env32_10.properties
|   |   |-- env32_15.properties
|   |   |-- env64_12.properties
|   |   |-- env64_8.properties
|   |   `-- env_scale.properties
|   |-- runCmd.txt
|   `-- run.sh                                      #example of run script
|-- experiments                                     #folder with the experiments
|   |-- compared                                    #solutions used for comaprison
|   |   |-- centralized
|   |   |   |-- binary
|   |   |   |   `-- DIPC2a.jar                      #the centralized solution
|   |   |   `-- source
|   |   |       `-- src.zip
|   |   `-- self-absorbed
|   |       |-- binary
|   |       |   `-- DIPSA.jar                       #the self-absorbed solution
|   |       `-- source
|   |           `-- src.zip
|   |-- out_centralized1.csv                        #experiment results
|   |-- out_centralized2.csv
|   |-- out_centralized3.csv
|   |-- out_distributed1.csv
|   |-- out_distributed2.csv
|   |-- out_distributed2new.csv
|   |-- out_parameters_15h.csv
|   |-- out_parameters_2.csv
|   |-- out_parameters_3.csv
|   |-- out_scalability_c2a.csv
|   |-- out_scalability_c2.csv
|   |-- out_scalability_c3.csv
|   |-- out_scalability_c4.csv
|   |-- out_scalability_d2.csv
|   |-- out_scalability_d3.csv
|   |-- out_scalability_d4.csv
|   |-- out_scalability_d.csv
|   |-- out_scalability_sa2.csv
|   |-- out_scalability_sa3.csv
|   |-- out_scalability_sa4.csv
|   |-- out_scalability_sa.csv
|   |-- out_self_absorbed2.csv
|   |-- out_self_absorbed.csv
|   |-- out.zip
|   |-- results_compare.csv
|   |-- results_distributed_avg.csv
|   |-- results_parameters_avg.csv
|   |-- results_scalability.csv
|   `-- RunScripts                                     #MetsCentrum run scripts
|       |-- deployall.sh
|       |-- deploy.sh
|       |-- enqueue_experiments_15h.sh
|       |-- enqueue_experiments_centralized3.sh
|       |-- enqueue_experiments_centralized.sh
|       |-- enqueue_experiments_distributed2.sh
|       |-- enqueue_experiments_long.sh
|       |-- enqueue_experiments_parameters2.sh
|       |-- enqueue_experiments_parameters3.sh
|       |-- enqueue_experiments_scalability_fix.sh
|       |-- enqueue_experiments_scalability_fix_small.sh
|       |-- enqueue_experiments_scalability.sh
|       |-- enqueue_experiments_self.sh
|       |-- enqueue_test.sh
|       |-- join.sh
|       |-- kill_all.sh
|       |-- pull.sh
|       |-- run_experiment_centralized_a.sh
|       |-- run_experiment_centralized_b.sh
|       |-- run_experiment_centralized.sh
|       |-- run_experiment_self.sh
|       |-- run_experiment.sh
|       |-- test_centralized.sh
|       |-- test_distributed.sh
|       `-- test_self_absorbed.sh
|-- readme.txt
|-- sources                                 #java source codes
|   |-- java
|   |   |-- resources
|   |   |   |-- env16_10.properties
|   |   |   |-- env16_5.properties
|   |   |   |-- env32_10.properties
|   |   |   |-- env32_15.properties
|   |   |   |-- env64_12.properties
|   |   |   |-- env64_8.properties
|   |   |   `-- env_scale.properties
|   |   `-- src
|   |       |-- environment
|   |       |   |-- AntiAir.java
|   |       |   |-- Area.java
|   |       |   |-- Environment.java
|   |       |   |-- EnvLoader.java
|   |       |   |-- Paths.java
|   |       |   |-- Sector.java
|   |       |   `-- Unit.java
|   |       |-- fleet
|   |       |   |-- ActionGenerator.java
|   |       |   |-- Action.java
|   |       |   |-- EvasiveAction.java
|   |       |   |-- Fleet.java
|   |       |   |-- MoveAction.java
|   |       |   |-- MoveTargetAction.java
|   |       |   |-- RefuelAction.java
|   |       |   `-- SearchAction.java
|   |       |-- graphics
|   |       |   |-- EnemyMarker.java
|   |       |   |-- KeyboardListener.java
|   |       |   |-- MyMouseListener.java
|   |       |   |-- MyMouseMotionAdapter.java
|   |       |   |-- Square.java
|   |       |   |-- TestPane.java
|   |       |   `-- Window.java
|   |       |-- planner
|   |       |   |-- Decider.java
|   |       |   |-- Main.java
|   |       |   |-- Mcts.java
|   |       |   |-- Node.java
|   |       |   |-- Planner.java
|   |       |   |-- ProbDistributor.java
|   |       |   |-- Simulator.java
|   |       |   |-- State.java
|   |       |   |-- UnobservedNode.java
|   |       |   `-- UnobservedTargetNode.java
|   |       `-- test
|   |           `-- Reality.java
|   `-- text                                            #text source code
|       |-- csplainnat.bst
|       |-- DIP_zadani.PDF
|       |-- figures
|       |   |-- LogoCVUT.eps
|       |   |-- LogoCVUT.pdf
|       |   |-- seznamcd.eps
|       |   `-- seznamcd.pdf
|       |-- hyphen.tex
|       |-- img                                         #images
|       |   |-- 131_cropped.pdf
|       |   |-- 16.png
|       |   |-- 32.png
|       |   |-- 64.png
|       |   |-- bin_cropped.pdf
|       |   |-- comp_avgrew.pdf
|       |   |-- comp_avgrewxuav_d.pdf
|       |   |-- comp_avgrewxuav.pdf
|       |   |-- comp_avgrewxuav_sa.pdf
|       |   |-- comp_env16_10.pdf
|       |   |-- comp_env16_5.pdf
|       |   |-- comp_env32_10.pdf
|       |   |-- comp_env32_15.pdf
|       |   |-- comp_env64_8.pdf
|       |   |-- comp_rew.pdf
|       |   |-- comp_rewxtime.pdf
|       |   |-- comp_sc_line10.pdf
|       |   |-- comp_sc_line5.pdf
|       |   |-- dipu2.png
|       |   |-- mcts_nodes1.pdf
|       |   |-- mcts_nodes2.pdf
|       |   |-- ovr_cropped.pdf
|       |   |-- q1.pdf
|       |   |-- q2.pdf
|       |   |-- q3.pdf
|       |   |-- q4.pdf
|       |   |-- q5.pdf
|       |   |-- q6.pdf
|       |   |-- q7.pdf
|       |   |-- q8.pdf
|       |   |-- reward-discount.pdf
|       |   |-- reward-sel.pdf
|       |   |-- reward-time_.pdf
|       |   |-- reward-time.pdf
|       |   |-- roates_cropped.pdf
|       |   |-- roulette_cropped.pdf
|       |   `-- sec_cropped.pdf
|       |-- k336_thesis_macros.sty
|       |-- reference.bib
|       |-- Rscripts                    #graph scripts
|       |   |-- process_centralized.r
|       |   |-- process_compare.r
|       |   |-- process_distributed.r
|       |   |-- process_parameters.r
|       |   `-- process_scalability.r
|       `-- Stary-thesis-2017.tex       #thesis source
`-- text
    `-- Stary-thesis-2017.pdf           #thesis text

25 directories, 162 files
```