Masters Project - Adressing Problem Spreading in Agent Planning and Control

Author: Torben Jess, Advisor: Daniel D. Corkill & Victor R. Lesser

February 12, 2012

Abstract

This Masters project analyzes the influence that prediction of goal spreading has in comparison to just the prediction of the state of a specific goal in agent and Multi-Agent systems. A lot of environments like city fires or bacteria in a human have this spreading behavior. These are environments where one existing goal can cause another goal to start. This behavior should have an influence on the way different goals are evaluated and potential actions of agents are selected. So far the author is not aware of any work that specifically addresses the difference between spread prediction and no spread prediction.

We will analyze if and how prediction of this spreading effects agent decision making and show that predicting this kind of behavior can enable better decision and better performance of agent and Multi-Agent systems. The effects of spread prediction are analyzed for city fires in the RoboCup rescue Multi-Agent environment for burning buildings.

In addition to the analyzes of spread this master project therefore also offers some further interesting insights in the behavior of fire in RoboCup rescue and shows a good and efficient function to evaluate fires in RoboCup rescue.

1 Introduction

In this masters project I'm going to analyze the difference between goal prediction and goal-spread prediction and will empirically show that predicting the spread of goals to other goals can have great advantages. While usual goal prediction as is mainly focused on predicting one goal and it's possible state in some point in the future to make better decisions. The prediction of goal-spread prediction also tries to identify possible further goals that this initial goal can cause. This kind of behavior can be found in a lot of environments, but to the authors knowledge hasn't been addresses specifically for agents and Multi-Agent systems. Wewill analyze this two kind of prediction (spread and no-spread) for the case of building fires on the RoboCup Rescue fire simulator [1], where fires show spreading behavior to other buildings.

When fire emergency call-centers receive information about a fire they usually determine information like the size of the building, wind directions, content and material of the building, how close it is to other buildings and so on. This could be done by asking the person who is calling or by sending helicopters or first fire fighter to detect the situation. They probably also include information about the building from maps, danger reports from companies and weather reports. This knowledge is then used to generate some belief about the fire situation and decide which actions to take to extinguish the fire in the best way with the given resources of fire fighters. Intuitively they thereby use two slightly different kinds of prediction to identify the best actions. The first one is the behavior of the building itself, how fast it is burning and how much water would be needed to extinguish this fire. The second kind of prediction is about the spreading of a fire, is it going to ignite other buildings, when is this going to happen and how could it be prevented. For example if a giant storage hall is burning, the fire fighters are most likely sending more fire fighters with heavier equipment, then when a small family house is burning (for the direct building specific prediction). Or as another example, they would probably send more fire fighters if a fire is burning in the middle of town with a lot of other houses around then a fire in a house of similar structure burning in a suburb with a lot of space around each building (as an example for spreading prediction). Including this kind of knowledge obviously seems to improve the behavior of the real fire fighters. It is used in reality that is based on years of experience, a lot of science and common knowledge.

Considering extinguishing the fire in a building as a goal and the fire fighters as resources of an emergency center agent, then we are dealing with a scenario where delayed or incomplete satisfaction of one goal can lead to the creation of additional goals.

Similar applications where this spreading of goals can occur are bacteria. When one bacteria is a goal for the immune system, then the division of bacteria can cause more bacteria to cause damage and this can cause additional bacteria and so on. We have a scenario, where one goal can cause other goals to start. We define the initialization of a goal by another goal as spreading. The prediction of spreading is thereby the prediction of if, when and where a goal initializes other goals.

The knowledge of spreading has two advantages one just occurs limited resources the other one a potential for earlier reaction.

Assuming an environment with limited resources like we have it in many applications, this knowledge of goal spread can be very valuable. Like for the case of just one fire fighting resource and two fires in the same distance to the fire fighter one in the middle of the city and one in the outside of the city. Including the possibility and knowledge of spreading in the prediction of the fire fighter will probably cause the emergency center to choose the goal in the middle of the city and reduce the overall damage of buildings. The reason is that the building in the middle of the city is more likely to spread and cause more buildings to be on fire. In environments with a lot of resources the decision between different goals becomes unimportant, because all of them can be fulfilled successfully. For example if there are enough fire fighters to fight the fire in the city and in the suburbs appropriately we don't have to worry about where to send my resources. The problem is that this isn't always the case. In catastrophic scenarios the resources for fighting all possible fires are limited.

In the cases with limited and without limited resources, the knowledge of spreading or potential spreading can have one additional advantage, if it creates goals before they are even appearing. In our fire-fighting example this would be the case, when the fire station is already able to send resources to a building, because it knows that it is going to start burning.

This research is focused mainly on the first type of advantage where the resources are limited and have to be used in the right way. This considers specifically the empirical results and the developed algorithm. But the positive effects of recognizing goals earlier are partly addressed and considered. Therefore this research could be extended to the positive effect of spreading prediction where goals are recognized earlier and therefore fought more accurately.

We will show that this goal spread prediction, can make a big difference, by comparing non-spread goal prediction and spread goal prediction algorithm in the RoboCup Rescue domain. In this project we will apply this intuitive knowledge of spread that probably real emergency call center are using to the artificial RoboCup Rescue fire simulator and show the different behavior of agents in environments with goal spreading behavior. We will also analyze the agent behavior and reactions when they have knowledge about the spreading of fires and when they don't have this knowledge.

There is a lot of work on the prediction of different environments like [2] or on the prediction of the status of a specific goal in the future [3] [1]. But the consequences of spreading have not been addressed specifically in the literature.

The problem with spread prediction is that it isn't guaranteed to be accurate or just accurate in some specific number of cases. Therefore in some cases using spread prediction could actually perform worse then not using spread prediction, because it could cause wrong goal prediction and therefore wrong decisions. Part of this research therefore is to find out if additional computing time for prediction might be worth the gain of accuracy. Therefore some further questions that need to be answered are: How to predict the behavior of fires? How to predict the effects on other buildings? How accurate does the prediction have to be? And how much computation time does the prediction take for a specific outcome?

This work uses fixed implemented prediction functions for spread and no-spread, it is not focusing on learning fire development or fire spread, we assume that the learned function will reply in a similar way after some time steps that our accurate prediction functions respond. [2] But even for the case of learning the general results on considering spreading are the same. The case without any goal prediction is also not considered, because it wouldn't really be an accurate decision if we were not trying to decide how the fire would develop in some way.

[1] This is actually done almost all the time, when decisions between different goals need to be made.

[2] Using learning in prediction and no-prediction is a potential for future work.

Even just giving the actual fire state, to determine the number of resources, we would have to predict how much water we have to put on the building. Therefore making some well-informed decision about a fire always includes some predictive capacity about the development of the fire.

The expectation is that spread-prediction performs better in scenarios with spreading goals and lack of resources. This project will analyze this hypothesis by first analyze the theoretical effects of spreading and then develop a spread prediction algorithm for the RoboCup rescue domain and empirically analyze if this algorithm performs better then non-predictive algorithm. In addition to this analysis several spread predictive algorithm with different levels of accuracy will be compared considering their overall performance. We expect that more accurate prediction will perform better then less accurate prediction and we also expect to see a border of inaccuracy where predictive algorithms perform worth then non-predictive algorithms because of their inaccuracy.

The project is further structured as follows. First we overview related work. After that we describe some theoretical insights in the prediction of goal spreading. Then give a description of the specific RoboCup rescue domain and the OAA architecture [4] that we used for my implementation. In Section 5 we also describe the algorithm we used for the specific RoboCup rescue domain. In section 6 we present an empirical evaluation of the algorithms and compare them to my theoretical results, and in Section 7 we conclude with possible future work in this area.

2 Related Work

The literature on prediction in different domains in general is very large. Some papers like [5, 6] address the behavior of prediction and show that it can have advantages, but they are also not considering spread-prediction.

There are three kind of related work that is relevant for this research, the first kind of work is work on utility based goal evaluation. The second kind of work is on fires in general, prediction of fires and on fires in RoboCup rescue. Further work on prediction and especially spread prediction could be interesting. We want to give an overview about both kind of work.

2.1 Utility based goal evaluation

Many different approaches on goal evaluation exist like [25, 26, 27]. The idea of evaluating the utility of different plans for each goal is also used at different kinds of research for example [28, 29].

The problem with this different research in utility-based goal evaluation is that there exists no work that explicitly analyses the effects of including spread-prediction in the goal utility calculation. There is probably some work where potential knowledge of spreading is used to evaluate goals, but it is not stated specifically or the author is not aware of this work. The new research step we are planning to take in this research and make a connection between goal utility evaluation and spread prediction.

This work is important because we are including our prediction in the utility evaluation of different plans for a goal.

2.2 Fires, fire prediction and fires in RoboCup Rescue

There exists a lot of work on the whole RoboCup rescue domain like [7]. Some of them are actually further addressing the fire-fighting domain [8]. A few like [15] focus mainly on burning in the city. One of them [9] actually describes the RoboCup rescue domain and gives some insights why prediction can be helpful in this domain (see section 4.1 for further details on the fire simulator). None of them actually focuses on good reactions of agents towards this expected fire behavior.

Beside RoboCup rescue a few papers focus on the development and spreading of fires. But most of them like [16, 17] are mainly focused on land fires. A few are additionally focused on fires in cities like [18], but they are mainly about predicting fires in a more global way so that real fire fighters address the problem and not the interaction with the fire how it is required in Robocup rescue.

2.3 Spread prediction

Beside the goal and fire specific spreading there are other analyses of spreading in some other applications that are important and especially give some further theoretical insight. These domains are network science and analyses physical, chemical or biological behavior.

In network science there exist the theory about random graphs and epidemic spreading (or other kind of spreading like viruses) that addresses similar issues. Network science shows that in case of an infinite large number of goals, there either exist or doesn't exist some large component. In case of existence of a large component spreading of bacteria or other elements is much easier. Although this are mainly theoretical results, that don't always apply to other more complicated domains, it still explains in a god way, why and how fires or other things like bacteria are spreading and what are the key elements to prevent spreading or reduce the effects of spreading. There exist a lot of papers that address this problem; examples can be found in [19, 20].

The last domain applications in physical or biological process can be found in the computational analyzes of bacteria development [21] or the simulation of flower spreading over a field [22]. They are also mainly just theoretical results and can't directly apply for Robocup rescue because they are not considering different sorts of reaction to fires.

3 Benefit of Prediction in goal functions

We now present an analysis of when and why the prediction of spread performs better then prediction that doesn't consider spreading. We will first give a short formal description of spreading, because it hasn't been specifically defined for the case of goal spreading in agent systems. Then we explain why and in which cases spread prediction performs better then no-spread prediction and why spread can have such dramatic influences and why it is important to predict it.

3.1 Formal definition of spreading

Assume that we have n different goals g_i in $G = \{g_1, g_2, ..., g_n\}$ and an environment e(t) and a time step t with $t\epsilon\{1, ..., T\}$ and T is the number of time steps our evaluation is running. Let A_t denote the actions of different agents in time t. Then the spreading of a goal is when a goal g_k cause another goal in the environment and influence the environment, so that $e(t+1) = o(e(t), G, A)$ where o is the function of environmental updates. So that

$$G_t^{'} = G_t \setminus g_k$$

$$then\ G_{t+1}^{'} = f(o(e(t), G^{'}, A_t) \neq (G_{t+1} \setminus g_k)$$

$$and\ G_{t+1} = f(o(e(t), G, A_t)$$

$$and\ |G_{t+1}^{'}| < |G_{t+1}| + 1$$

Where f computes the goal in the environment. This means there has to be an other goal in G_{t+1} that is not in $G_{t+1}^{'}$ instead of g_k. If goal g_k wouldn't be in the environment we wouldn't have seen this additional goal to exist, therefore g_k spreads to g_{new} in time t.

The actual reason for this spreading is not included in this definition. It could be the lack of resources to work on this goal successfully in a specific time for example.

3.2 Possible advantage of spread prediction

We will analyze the advantage of spread prediction by first showing the advantages in a environment where perfect spread-prediction is possible in the the first part and then further take a look what happens if some of the strict assumptions int he first part are reduced.

3.2.1 Given completely accurate prediction

Assume that we have n different goals g_i in $G = \{g_1, g_2, ..., g_n\}$ over all time steps and for all goals we have m different potential plans $P = \{p_1, p_2, ..., p_m\}$. We also have environment e and the actions A_t . Let us further assume that we can define an algorithm that is able to accurately predict when, how and where one goal is going to spread to another goal if a specific plan is executed on this goal.

Then the perfect function decision function $H = f(G, P)$ that is using this algorithm should perform better then $H^{'} = f^{'}(G, P)$ that is not predictive. Where f is predicting the best development and spread of all goals in G to the end of our calculation T and is choosing Hout of P to achieve the best possible plans. $f^{'}$is doing the same but without considering any further goals that start from G. H is the selected plan that are executed, so that $A_t = getActionsOf(H, t)$.

We further know that for perfect prediction algorithm $S(H, G) \geq S(H^{'}, G)$ where S computes the overall system result that we can achieve on the goals, for a fire fighting scenario this result would be a limitation of overall building damage, because for no-spreading they perform identical and for spreading H is always more precise about the influences that a fire has over al time steps and therefore always makes the better decisions.

Therefore just if $S(H, G) = S(H^{'}, G)$ spread prediction has no advantage. The two possible reasons for this are they have either a non spreading environment or they make identical decisions even with spreading. Therefore three conditions have to be fulfilled so that spread-prediction has advantages. First there has to be a spreading environment because otherwise spread prediction doesn't make any sense. Second the decision between the two functions have to be different this means not all goals in the environment are allowed to spread in the same way and must have different future utility then they would have based on the non-predictive evaluation. Third there has to be a limited number of resources, so that not all goals can be satisfied immediately.

Therefore spread prediction performs better if the following three conditions are met:

1. Goal spreading exists
2. Time periods with limited resources
3. Different goal utility of goals for spreading and non-spreading

3.2.2 Given statistical accurate prediction

The problem with the previous results are that the assumptions are very strict. They either aren't given in most environments or their computation is too complex. [3]

[3]We will later see that both is the case in the RoboCup rescue domain I'm working in.

The two critical assumptions are the required accuracy of goal prediction and the accuracy towards the concrete building that we try to predict is burning.

The assumption of accurate goal prediction for the algorithm is very strict and pretty unrealistic in reality. But even for an algorithm that is just right a specific high enough percentage of time most of these information is very valuable. As potential future work there should be potential ways to proof that under the given conditions the average decision case should perform much better. [4]

The second critical assumption that is very strict is the requirement of accuracy towards the specific building that start burning. We probably won't be able to always say witch further potential goal would actually become a goal. The reasons are again possible environment or computational limitations. But even though this would strongly reduces or completely eliminates one possible advantage of prediction that is mentioned in section 1 the earlier recognition of goals. We can't always direct resources to goals we know we are going to get in the future. But the second prediction advantages the better use of resources on existing fires can still exist by just predicting the potential workload. We would still be able to decide between a building in the middle of the city and a building outside of the city. We might not be able to predict goals earlier but we could be able to detect additional workload that is going to be created by not fighting a goal accurately. This could still be a big advantage and produce much better results.[5]

Therefore even without these strict assumptions regarding accuracy spread prediction still has advantages and could improve the performance. Just the following additional conditions regarding the potential of the prediction function have to be fulfilled.

1. Prediction is statistically more accurate
2. Prediction of future workload is possible

We will develop a spread prediction algorithm that fulfill these conditions and show in the empirical results that is able to perform better then non-spread predictive algorithms in environments as described in 3.2.1.

3.3 Importance of spread detection

One important effect of spread beside the reasons where it can happen given in 3.2 is the possible exponential behavior of spread. If one goal spreads to another goal, which spreads to further goals, we have an exponential growth. If the possibility of spreading P to one of T possible target is

[4]We probably have to use $AVG(S(H,G)) \geq AVG(S(H^{'},G))$ for a specific assumed function but the results are the same.

[5]In this project we will focus on the prediction of workload and the positive effects that even just this prediction can have. But we will also describe how specific goal prediction could be included in the architecture and our predictive algorithm.

$$P_{Spread} \times T_{Spread} > 1$$

We have exponential growth given a uniform distribution of neighboring goals per goal of T (which could be assumed for city fires for example where each building almost always has the same amount of neighbors on each side).[6] It is know from network science [11] that we then could have a giant epidemic or similar a giant fire or other kind of giant influences. Therefore fighting spreading fires and reducing P and T would stop this effect.

4 Description of RoboCup Rescue fire simulator and the OAA architecture

RoboCup rescue is a research competition for catastrophic scenarios. It was initialized as a consequence of the Kobe earthquake and includes competitions for real robots and a simulator competition for different agents interacting in a Multi-Agent environment [1]. The simulator can be found in [12], it offers an environment with different rescue agents and simulators for fires, building collapses, roadblocks or civilians health. They all affect each other. Agents like call centers, fire fighter, ambulances and policemen should work together to achieve less fire destruction and less dead or injured people [13]. There is an annual competition between different teams to see which of them does the best job in reducing the damaged buildings and rescuing the most civilians [14]. We used this environment because of the goal spreading behavior of the fire simulator. In section 3 this is identified as one of the required environmental features.

The organizationally adept agent architecture is a Multi-Agent systems architecture that enables organizational control with organizational guidelines and makes it possible to adapt to these guidelines [30]. It also offers a structure of beliefs, goals and plans that is very useful for our case and is the reason why we chose to work with this basic architecture. In this project we used the special application of this architecture to the RoboCup Rescue fire simulator environment.

In the following subsections we will introduce the fire-simulator of RoboCup Rescue and give a basic overview about the way the fire simulator works. It should give the reader and overview about the environment we are working in to better understand how the RoboCup rescue specific algorithms we implemented work. In addition we will give an overview about the work on organizationally adept agents and the OAA architecture that was used as the basic structure for predictions. Both are important to understand the prediction algorithm and the empirical results presented in the next sections.

[6]For other distributions similar results exist with slightly different but still constant values exist that might imply just spreading to a larger part of goals but with similar dramatic effects. (see [10] for details)

4.1 fire simulator

For this project we worked with the fire simulator of RoboCup Rescue. The reasons are first, that it is one example for an environment that has a goal spreading nature. One building is able to ignite other close buildings. That behavior is required to show the positive effects of spread prediction. Second it is predictable as already explained by the authors of the fire simulator [9]. This predictability is a basic requirement to test the effects and required accuracy of prediction and especially spread prediction. And third limiting the prediction to just fire fighters and fires makes it easier to observe the agents behavior and to get a better impressions and the different positive and negative effects of prediction. Adding other simulators in this environment and analyze the different behavior is part of possible future work (see section 7 for details).

We will give a brief overview about the simulator. A detailed description of the fire simulator in the RoboCup rescue domain can be found in this paper [9]. The simulator simulates two things the fire in a building and the possible spreading effects of fires. They are both dependent on each other because the spreading of fire to other buildings means that the actually burning building is loosing this energy. This happens for example by heating up the surrounding air or emitting radiation.

The fire simulator uses a four step simulation of these two aspects. The first and second one are mainly about the building itself the third one monitors the exchange of energy and the fourth one again is a building specific computation. Each of these four steps are executed in every time step of the simulation. [7]These four steps are:

1. burn()
2. cool()
3. exchangeBuilding()
4. cool()

The first step burn() simulates the actual burning and destruction of the building itself. The simulator assumes that buildings are made out of one material. They start burning because either the simulation tells it to burn for example by random selection [8] or because the temperature of the building is higher then the ignition temperature of the material[9]. This can happen possible because another close burning building heats it up and then actually spreads the fire to this building.

Each building has a specific amount of fuel depended on its size and material. This fuel is used to determine how much energy and heat a fire can produce and

[7]The length of the simulation can be defined in a specific configuration file for our experiments we set the length to 300 time steps.

[8]Necessary to create specific scenarios after an earthquake for example

[9]This variable can also be set in the configuration of the simulator. For our test we left it at the standard configuration

how much of a building is already burned down [10]. The temperature and the energy are directly related and the production of energy via burning directly causes a reduction of fuel. When a building runs out of fuel, its energy and therefore its temperature is decreasing. Once no more fuel is left a building is burned down.

The cool() steps which is used to calculate the effects of water that is put on a building reduces the energy of a building and thereby also the temperature of the building, which eventually can cause a building to be extinguished, if the effect is strong enough. The main influence on the cooling step has the amount of water put on a building in every time step by the fire fighters. Not all the water put on a building is necessarily used. Depended on the building there is only a limited amount of water that can be used to reduce the temperature in one time step, the rest is used in following time steps. Or in another case if a building is not burning then this water can later be used to cool a building that is heating up from the environment. When the temperature is below a specific material depended threshold temperature[11], it stops burning. This can happen by water that reduces the temperature or because all the fuel of a building is burned. The cool method is started twice in every time step, once after the burn method and one after the exchangeBuilding method.

The exchangeBuilding() method models two different effects that reduce or increase the energy of buildings dependent on the surrounding buildings, the environment and the air temperature. It can therefore cause a building to eventually start burning or getting extinguished. It models two different effects of energy exchange, the energy exchange by air and the energy exchange by radiation.

For the exchange of air it assumes a giant grid of different air cells over the whole environment. The squares of air are heated up or cooled down by the buildings in that cell. When a building has a high temperature then it increases the temperature of the air and cools down itself by loosing this energy. If a building has a lower temperature it heats up and cools the surrounding air. In every time step each cell also exchanges the temperature with the neighboring cells and therefore cause a transportation of hot air to cooler air regions. This warmer air can then heat up the buildings in neighboring cells. In addition to that each cell is also loosing some natural energy in every time step, for example by warm air rising [9, 12]

The exchange by radiation is modeled by considering the outside walls of each building as the source of radiation production. It is depended on the wall size and the building material. Because the calculation of angels and line of sight of all walls to each other is too complicated, the simulator uses a Monte-Carlo process at the beginning to calculate how much percentage of radiation energy of which building reaches another building [9]. He then computes the radiation energy of each building by specific physical formula and from this the simulator calculates how much energy of this radiation will reach the next

[10]The amount of building damage is a main value in the utility computation of the RoboCup rescue competition that can be found here [14] and also our main variable

[11]Can also be adjusted in the configuration file.

buildings and therefore increase the energy of these buildings. Each building looses this radiation energy to other buildings but also gets energy from the radiation of neighboring buildings, depending on the percentage calculated in the Monte-Carlo process, some of the radiation energy actually gets lost, because the Monte-Carlo simulated rays didn't hit any other building. This simulates the natural loss of radiation in different directions. At the current state different highest of buildings are not considered in the simulator yet [9].

Because we are not including any effect of water cooling on non burning buildings at this analyzes yet, we don't have to worry about predicting the amount of water on not burning buildings yet. By including this we would further analyze the second positive effect of spread prediction that is the ability to recognize goals earlier, which is not focus in this project.

Except for one small normal distributed random variable in the burn method and the Monte-Carlo process for radiation the calculation in the fire simulator is very deterministic [9]. All the values used to compute the fires in the simulator or good estimates of these values can be achieved for the call center agents. Therefore the fire development and possible spreads of fires are predictable in some relatively accurate way. In addition to the first criteria of a spreading the element of predictability also exist in this environment. It therefore offers a good test-base for this project.

We will use it to develop an algorithm that predicts the potential spreading of fire to make better decisions on the plans an agent should take. A lot of different approaches are possible for doing this prediction, for example statistical methods based on several test runs or learned approaches with several inputs. We decided to use an approach that uses a simplified calculation of the fire simulator. In comparison to a real fire scenario this means my approach tries to analyze a simplified physical model of the fire behavior and uses this to make predictions. This might be the most intuitive one, but that doesn't necessary mean that this approach make the most accurate predictions under all circumstances. There are probably other methods that are calculating the results faster and maybe better. The empirical results show that this approach works very accurate (see section 6 for empirical results). Overall the accuracy is mainly critical in relation to what spread prediction accuracy is actually required to show better total performance and less regarding the best possible accuracy, because our main focus is on showing the advantage of prediction.

Many of the values given in RoboCup rescue like building and environment temperature or building material for example are also available in real fire scenarios. That means that in the future there could be techniques that consider the effects of goal spreading in real fire fighting scenarios.

This is just a general overview about the fire simulator. Further details can be found in the actual algorithm description that go into some details on the actual fire simulator and describe how specific things are computed (see section 5). For more details on the formula, the physical background, the exact algorithm description and the different variables used see [9].

This description of the fire-simulator will be the basis of our spread-prediction and non-spread prediction algorithms that are defined in section 5.

4.2 OAA architecture

The OAA architecture is designed to enable organizational guidelines for agents, use them to direct their behavior and limit the search space of each agent [4, 30]. It especially makes it possible for the agents to evaluate these guidelines and decide weather to follow them and eventually define new guidelines when specific patterns in the behavior of agents are detected. It offers an architecture, with beliefs, goals and plans that are created. It also has specific algorithms to decide which of the possible candidate plans have to be selected later and how the plans are created [30]. All this has already been applied for the RoboCup Rescue fire domain with different fires as goals and a sequence of actions from the different agents as possible plans. Therefore this architecture offers a good structure for our experiments. It makes it easier to focus mainly on the effects of prediction and the different advantages that prediction can have on different goals. It is also just focused on decisions about different fires, so that we can focus on spread prediction and non-spread prediction, how it is used for goal evaluation and the agents reactions towards prediction. We therefore don't change the basic structure of the OAA architecture and just adepted the calculation for goals and plans considering spread prediction or no-spread prediction there. To make sure that organizational effects do not influence our prediction evaluation, we didn't use any of the guidelines of the architecture and were therefore able just to study the behavior of agents with and without spread prediction.

In the OAA architecture for the RoboCup rescue domain there are two types of agents from the original RoboCup rescue domain, the first one are fire fighters and the second one call centers. The call centers are controlling the fire fighters and are telling them what to do. They decide which goal and plan to follow and therefore make the important decision. The fire fighters are therefore the resources of the call center agents [30]. Our spread prediction and no-spread prediction is included in the call center, because of the utility calculation and decision process happening there.

The OAA architecture computes the utility for different fires on buildings that we define as our goals the following way. For every building the call center agent has a specific building belief. If this building belief changes to building on fire, a goal for this building is created. [12]For every goal a series of possible plan templates is created. Each plan template p for goal g requires a different amount of resources. In every time step each existing plan for every goal is evaluated according to the following utility function.

$$U(p,g) = U_R(p,g) \times w_R + U_O(p,g) \times w_O + U_S(p,g) \times w_S \tag{1}$$

Where $U_R(p,g)$, $U_O(p,g)$ and $U_S(p,g)$ represent the self, request and organizational utility of different plan and goal combinations. Self is the utility that an agent gets for following his own local perspective; it considers things like time to get to the fire with his own resources for example. Request is for

[12]Buildings that are not on fire aren't a target yet, but here we could include possible buildings that are heating up as potential targets as future work.

the case, when someone requests another call center for resources. The organizational utility is the utility provided for following a specific organizational behavior given by guidelines. In the case without guidelines that we are using in our example $U_O(p,g) = 0$ because there are no guidelines for which following could provide a utility.

For the RoboCup rescue case there are two types of plan templates. The first ones are RequestTemplates and the second ones are ExtinguishTemplates. The requests are send to other call centers to get some of his resources. The ExtinguishTemplates are for fires that each call centers fights on his own with his own resources. For Request $U_S(p,g) = 0$ because we don't have any own utility when we fulfill the requests of other agents and for extinguish templates $U_R(p,g) = 0$ because we don't get any utility from the requesting agent for doing our own goals.[13]

The w_R, w_O and w_S represent the percentage of each utility that counts towards the total utility $U(p,g)$. It can be used to control the architecture towards more importance of the organization or towards a specific behavior, so that it is more oriented towards his own view or the view of other agents when he requests them.

After this utility calculation the goal with the highest utility for a specific plan template is selected in the following greedy algorithm and the specific resources are reserved. (See algorithm 1 and description here [30])

It is executed again, till none of the potential plan templates has enough resources left to be executed, all plan templates just generate a negative utility or there are no more possible goals. Each selected plan template and goal combination is then started as a specific plan with the necessary actions by the call center that is sending the fire fighters.

After giving an overview about RoboCup Rescue, its fire simulator and the OAA architecture that is used for this project, we now what to introduce the algorithm used for the spread prediction and not spread prediction analysis.

A combination of both types of plan templates such as partial lending of resources isn't included in the current state of the architecture [30]. A further description of the idea can be found in [4] and further information about the architecture can be found in [30].

After a description of different the basic environment that we used we now want to give a description of the two algorithms that we used for my empirical analyzes.

5 Algorithm description

In this chapter we want to give a basic algorithm description of the algorithm for the RoboCup rescue domains. For readers of this project documentation who are mainly interested in the algorithms that could be used for RoboCup

[13]The values for U(p,g) are simplified because of our decision to work without guidelines and just analyze the spread prediction effects. For other cases the OAA architecture offers a more complex structure, where different utilities are combined to compute U.

Algorithm 1 Algorithm in the OAA architecture for deciding which plans templates to execute

Input: Goals, PlanTemplates, Resources

```
committedPlans = null
committedGoals = null
reservedResources = null
possibleGoals = Goals
freeResources = Resources
repeat
    bestGoal = null
    templatesOfBestGoal = null
    bestGoalPlanResources = null
    eservedResources = null
    maxU = 0
    for all g in possibleGoals
        templates = getFeasiblePlanTemplates(PlanTemplates, freeResources, g)
        if templates.size() > 0
            for all pt in templates
                ptResources = computeBestResources(pt, freeResources)
                # Here we execute the utility calculation in the way explained later
                u = calculateUtility(g, pt)
                if u > maxU
                    maxU = u
                    bestGoal = g
                    templatesOfBestGoal = pt
                    bestGoalResources = ptResources
        else
            possibleGoals.remove(g)
    committedPlans.add(templatesOfBestGoal)
    committedGoals.add(bestGoal)
    reservedresources.add(bestGoalPlanResources)
    freeResources.remove(bestGoalPlanResources
until bestGoal == null or maxU < 0 or freeresources == empty
return commitedPlans, commitedGoals, reservedResources
```

Output: CommittedPlans, CommitedGoals, ReservedResources rescue this section and section 6 are probably the most important one. Readers that just care about the advantage of prediction might just take a short look at this section. They are probably mainly interested in section 6 would probably be more interesting for them.

To show the advantage of spread prediction in the RoboCup rescue domain and potentially in a lot of other domains we developed two different algorithms. The first type of no-spread prediction algorithms optimizes his activity due to the evaluation of the goals at the current state and reduces the total amount of building damage at a particular goal (building). It predicts the expected state of a building when a specific plan is executed on this building. The second spread prediction algorithm tries to predict if and how the building is spreading in the future and includes this prediction knowledge in the utility calculation of these goals.

Based on the algorithm utility calculation of goals in the OAA architecture our algorithms have to be able to evaluate the following values $U_R(p, g)$ and $U_S(p, g)$. We will show how both algorithm calculate this values for a specific plan p and a specific goal g, because this is the only step in the OAA architecture that is mainly changed by our algorithm.

5.1 No spread prediction algorithm

The no prediction algorithm just analyzes the goal (or building) it doesn't consider any further effects that the fire possibly has on other buildings and just calculates the expected damage of this particular goal for a specific plan. The algorithm calculates U_R and U_S based on the given utility function structure in the OAA architecture [30] in the following way.

$$U_S(p, g) = v(g) \times S(p, g) - c(p, g)$$

For U_R it is calculated in a similar way.[14] $v(g)$ is the value of goal g, $S(p, g)$ computes the satisfaction degree for goal g given the specific plan template p and $c(p, g)$ is the cost of the plan [4]. The main goal of our prediction is to minimize the total building damage of the RoboCup rescue domain. Therefore we set

$$v(g) = GroundAreaOfBuilding \times NumberOfFloors$$

The reason for that is that the building damage score in RoboCup rescue we try to minimize for our case[15] is defined as

$$BuildingDamageScore = \frac{BuildingsDamaged}{TotalAreaOfAllBuildings}$$

[14]Details on why these two values are enough for the specific OAA architecture can be found in section 4.

[15]Using additional goal functions to optimize the agents behavior wasn't the main concern in this project, therefore we focused on the building damage, because it can be measured in a relatively easy way and the computation of a satisfaction degree and related costs is also easily possible.

16
Where

$$BuildingsDamaged = \sum_{i=0}^{\#of Building} B_i.TotalArea() * B_i.percentageDamage()$$

We can see that if a larger buildings is burned with a specific percentage the score is reduced more then if a smaller buildings is burned. Therefore giving goals with a larger building size a higher utility is a good strategy to get a better building damage score.

$S(p, g)$ as the satisfaction degree in the OAA architecture [30] is the expected state that building has in percentage of damage if plan p is executed on goal g . We will calculate this value by using knowledge about the physical processes happening in the fire. In RCR the fire simulator models these physical processes. . We want to use this knowledge to model the physical processes that would happen in a burning building for a specific plan and then try to determine the percentage of damage for each plan. The expected damage is used as a value for the satisfaction degree.

The main task of our algorithm is in finding a good no-spread prediction to calculate an accurate value for $S(p, g)$. In this physical model $S(p, g)$ mainly depends on the amount of fuel at the end. In addition the time t it takes to execute a plan is important to calculate opportunity costs $c(p, g)$.

Theses three things are used to compute the utility for each plan and goal combination. Over all goals and the potential plans for each goal we try to find the one with the highest overall utility using a greedy algorithm (as described in section 4.2).

We will do the calculation for $S(p, g)$ and t with one algorithm. It will calculate the expected fuel that is left at the end and the amount of time it takes till the fire is out. Assuming the fire simulator describes the actual behavior of the fire [17]. Our model calculates fuel after extinguishing and time for a specific plan by using the knowledge from this simulator.

As we have seen in section 4.1, the main steps are burn, cool and exchange-Buidling. To make accurate predictions about the fire our algorithm will approximate the main steps of this simulator. There are only a few exceptions in accuracy that our algorithm makes are for special cases beside the once already mentioned above.

The main difference to the real simulator is that we are doing this calculation not for all buildings but just for our specific building that is burning. We will compute each goal independent of other fires in the environment (with an exception in the spread-prediction algorithms named burn that consider some

[16]In RoboCup rescue the building damage is defined as a discrete function with three steps that set the damage of the building according to the configuration of the simulator. The different steps depend on the amount of building fuel that is already burned.

[17]Although it is also just a simulation of real fire, because large city fires are very hard to describe in reality. See [9] for further details.

surrounding buildings) and not consider their physical process, because otherwise the computation would get to complicated. This makes our calculation faster and goal specific. It also enables us to do this computation for all plan templates and goals. But this also makes it a less accurate especially in more complex situations with a lot of fires in the surrounding environment. The neighboring building could start burning for example and cause our fire to get stronger too. Our algorithms wouldn't cover these cases. Empirical data regarding the influence towards accuracy can be found in section 6.1.

We will now first explain how we calculate the cost $c(p, g)$ and then how we recompute each of these main steps in our no spread prediction algorithm to compute $S(p, g)$ and t for $c(p, g)$. First we explain how we calculate the cost, then we show how we initialized the values for our model, then we show the three main steps and finally how we are also compute the time for a plan and the achieved satisfaction degree.

5.1.1 Cost calculation

The cost of the utility function $c(p, g)$ for this algorithm are opportunity cost[18]. The opportunity costs are the costs for not being able to use the resources in the next time steps. If another fire starts then we can't use our already used resources. Therefore in the non-spread predictive case $c(p, g)$ is the possibility of another fire to start over the time frame where we need our resources multiplied with the building area we could safe with this resources if such a fire starts. Beside the knowledge of future possible fire arrival these costs are therefore mainly dependent on the amount of time we need to extinguish the building with a specific plan. Because time can be computed with the satisfaction degree as we will see in the next subsections, the main problem is to calculate the expected future arrival rate of new fires.

Given this description one function for the opportunity cost could be the following one:

$$c(p, g) = \sum_{t=1}^{t(p,g)} \sum_{m \epsilon M} \sum_{k \epsilon K_m} \psi(\Gamma(p), m, k, t)\Gamma(p)\delta_{k,m}(t, r)$$

With $\delta_{k,m}(t, r)$ being the cumulative distribution function of the probability that k building of size m start burning till time t that we can't fight with my other current resources. r is an array with the expected number of free resources over the future step. $t(p, g)$ is depended on the specific goal g and the specific plan p and we are summing over all t. The costs are summed over all possible number of buildings $k \epsilon K_m$ with $K_m = \{\# \, of \, building \, of \, size \, m\}$

[18]Like the RoboCup rescue simulator doesn't include costs for damaged fire fighters or other potential costs like fuel for the cars or the actual man fire fighter in its evaluation function, we also don't include this in our algorithm at the moment. This is potential future work for making the environment more complex and not specifically relevant for our analysis because we want to focus on the analyzes of spread of goals and the effects of this and not the analyzes of potential values to include into a utility function.

and all possible building sizes $m \epsilon M$ with $M = \{All\, kind\, of\, building\, sizes\}$. $\Gamma(p)$ is the number of resources that would be reserved for our plan p to fight this fire g and $\psi(\Gamma(p), m, k, t)$ the area of the k buildings of size m that we could safe with these resources till time t. The number of resources $\Gamma(p)$ is already in the plan, $\psi(\Gamma(p), m, k, t)$ could be estimated by empirical test or by using knowledge about how the simulator work. The values of building size and number of buildings can also be computed from the given knowledge. The hard part is defining the probability distribution. $\delta_{k,m}(t, r)$ for a given time t. [19]This cumulative distribution has to include a lot of knowledge would have to use the number of free resources r and the further expectation and probability distribution of future fires to be calculated accurately.

Therefore to simplify the problem we assume the arrival in the past for some limited time step will be similar to the arrival in future and that fires arrive with a Poisson distribution for one specific resource. This is also done in the OAA architecture [4]. It doesn't necessary has to be the case; therefore one direction of future work is to improve this calculation (see section 7). Important is that this cost function doesn't include any kind of spreading knowledge, otherwise we would mix the effects of spreading knowledge in the following functions and no-spreading knowledge in this function. We would otherwise mix these two kind of effects and it would be able to clearly say which behavior caused which specific advantage. Maybe the spreading knowledge in the cost function improved our original no-spreading algorithm, so that it performed much better then without his spreading knowledge.

After explaining the calculation for our cost we now show how we compute $S(p, g)$ as the satisfaction degree and $t(p, g)$ for our cost.

5.1.2 Initialization

Before starting the calculation of these three methods, we need the starting energy level e_t and the fuel f_t of each building. To calculate the initial energy level e_0 we use the getTemperature() function of a building. But because this function can just be computed[20] as an integer approximation of the real value in the fire simulator so that $getTemperature() = floor(TempInFireSimulator)$. This might occur because inaccuracy in the measurement of the real temperature that exists in a building. We have to add 0.5 to this value to minimize the overall expected error and use this approximation for further calculation. With this specific temperature we are then able to approximate the energy of a building by

$$energy = temperature \times V_{Building} \times Capacity_{Building}$$

with the capacity being a material dependent constant.

Usually the initial fuel can be calculated very easy with

[19]Which as we will later see that $t(p, g)$ can be computed together with the satisfaction degree.

[20]Because Robocup rescue is limiting the information it is giving to the call centers artificially to make their simulation more accurate

$$f_0 = V_{Building} \times fuelDensity_{Building}$$

With fuelDensity being a material dependent constant [21]and $V_{Building}$ being the volume of a building. So that

$$V_{Building} = NumberOfFloors_{Building} \times GroundArea_{Building} \times RoomHeight$$

with the room height being 3 and identical for all buildings.

The problem for the InitialFuel are cases where the building was already burned before and now starts burning again or where the building is already burning for some time, before we start our analyzes. In this cases some fuel is already burned and we have to adjust the fuel that our algorithm starts with. This can be done by using an approximation that uses the same physical knowledge. By monitoring the temperature of each building for the previous time step we are able to analyze the change of temperature and therefore a change of the energy level in a building. Assuming that just the burning process happened and not considering any change in energy by air, we are able to approximate the fuel of a building at the beginning.

Beside this computation of initial fuel, energy and temperature we also compute two values that we need for the exchange with other buildings. The first one is an Air Grid that is used to monitor the energy exchange between the different buildings with the air and is also used to simulate the heat transportation via air.

In addition we also compute the markov process described earlier, where each building computes the building that receives some of its energy and the specific percentage of energy that this building is receiving. But instead of doing is for all buildings we just do it for our specific building we are analyzing and that is loosing energy. This makes the calculation much easier and faster.

Both initialization for radiation and air transport leave us with some neighboring buildings, that get some of our energy. We will use this transport of energy to our neighbors to compute if their are starting to burn or not and analyze their spreading in 5.2.

After the initialization we start a loop where we repeatedly compute the following three functions in the order described in section 3.1. The loop stops if all fuel is burned or the temperature of a fire is below a material specific threshold for each building, then we stop the this iteration and have our result for the final fuel.

5.1.3 burn()

For the burn step we use the initial fuel f_0, temperature $temp_t$ and energy of the building to compute the amount of fuel burned in the next time step. This burned fuel is called consumed and

[21] Defined in the configuration file of RoboCup rescue.

$$consumed_t = \frac{f_{t-1}}{f_0} * \frac{temp_{t-1}}{1000} * burnRate$$

The burnRate is a normal distributed random variable where the values for variance and expected value are defined in the configuration of Robocup rescue. The consumed fuel is then added to the energy variable and subtracted from the fuel. By changing the energy it is also changing the temperature because

$$temperature = \frac{energy}{Volume_{Building} * Capacity}$$

With the capacity being strictly material dependent and defined for each building. This computation is exactly how the computation works in the physical simulator with the only difference that we have a slightly different random-seed for our burnRate because of two time steps difference to the simulator and our temperatures and fuel are just approximations of the values in the simulator. This whole computation is also just done for just one building and not for all building.

Whenever a specific threshold of the initial fuel percentage is burned our algorithm uses this threshold to calculate the percentage of damage in a building (details in subsection 5.1.6). The results for U_R and U_S for this method are identical.

5.1.4 cool()

To calculate the cooling effect, we try to approximate the amount of water that is put on the building in each time step by the resources of this specific plan p. Given the amount of water the calculation is exactly like it is in the simulator just with the approximated values. In every time step the amount of water we expect our fire fighters to put on a building is added to the total water quantity of a building. The water quantity is then multiplied by a specific water coefficient to calculate the effect of water. If the effect is bigger then the energy of a building then the following computation is done:

$$pc = 1 - (\frac{effect - energy}{energy})$$

$$effect = effect * pc$$

and the new values for the effect are used. Otherwise the effect of water directly reduces the energy level and the water quantity of a building. Note that it is possible that not all the water for one time step is used in this time step, it therefore can happen that the amount of water considered to calculate the effect is bigger then the amount of water we put on the building in this step.

To calculate the amount of water that we can to put on a building in each time step, we need the time to travel to the building the time we can actually fight the fire, the time to refill and come back to continue fighting the fire and the time we can fight the fire after we refilled our tank. The reason, why the first fighting time before refill and the second fighting time after refill are different is that the fire brigade could already have used some of his water for another fire and therefore isn't full. [22]

The time we can put water on a building can be calculated by the amount of water in the firefighter divided by the amount of water we put on a building in each time step. The time for putting water on a building with a full tank could be calculated by dividing the capacity of the tank by the amount of water we can put on a building. To calculate the time to get to the building and the time to get to the refill station to refill the tank of the firefighter we have to find a good approximation. We just know the resources of a plan and the building, but not the exact time the resources take to actually travel to a building. To calculate the travel time we use that most of the maps in RoboCup rescue use a Manhattan like structure. Calculating the actual distance would require a search of possible path for every plan, for every firefighter to every building and from the building to the next refill station. This operation is too complicated to do it in this simulation. Therefore we assume Manhattan distance and use the time that it takes to travel to the building. The following graphs show that the Manhattan distance is a good measure for our algorithm (see Figure 1).

They show the actual time of travel relative to the calculated Manhattan distances. The results are computed by selecting buildings and the related locations of roads of three maps (Kobe, Berlin and Test) at random and then plotting there Manhattan distance and the actual travel time. Just in rare cases the time difference between the actual time and the estimated time was really big relative to the total time to estimate. The high correlation indicates that there is a very strong connection between these two values especially for the Kobe map, that we are using for our tests. Therefore we can use the following function to calculate the traveling time of our agents to the fire and to the refill station and the Kobe map.

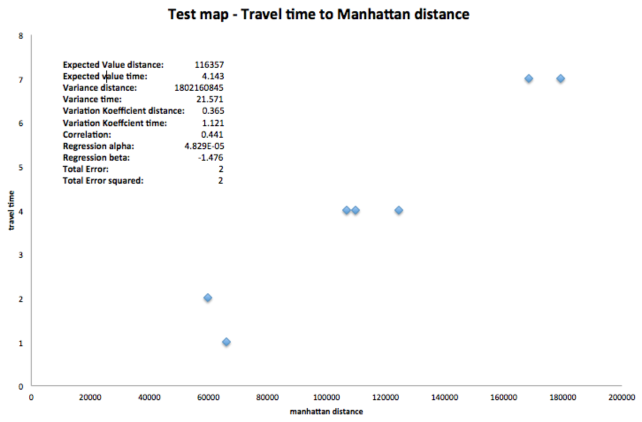
$$travelTime = \begin{cases} ceil(\alpha \times ManhattanDistance + \beta) & if\, value\, after\, .\, is\, >\, 0.7 \\ floor(\alpha \times ManhattanDistance + \beta) & otw \end{cases}$$

Where the values for $\alpha = 0.000027179890590584$ and $\beta = 0.615934309$ are coming from our recursion on the experimental data. An the value of 0.7 as the threshold between floor or ceil from empirical reduction of the total error by trying different values.

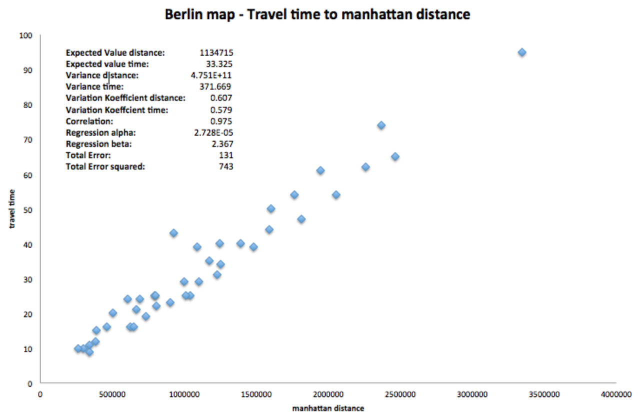
Using all this to approximate the travel times, we can calculate the amount of water we are able to put on the building in each time step in algorithm 2.

[22] At the current state we are not trying to further optimize the plan by deciding weather or not the fire fighter should might not totally fill his tank or should fill up first before going to the fire. This could make sense in some special cases. In general the advantage of getting to the fire with some water at the beginning seems to be higher then refilling first, considering the increasing nature of the energy level and the several runs with the simulator. Not totally refilling is also not really promising in general as long as we don't have many refilling stations because the travel time is therefore relatively high. For the purpose of simplicity and because the general case seems to be working pretty good so far we leave this additional complexity in this step.

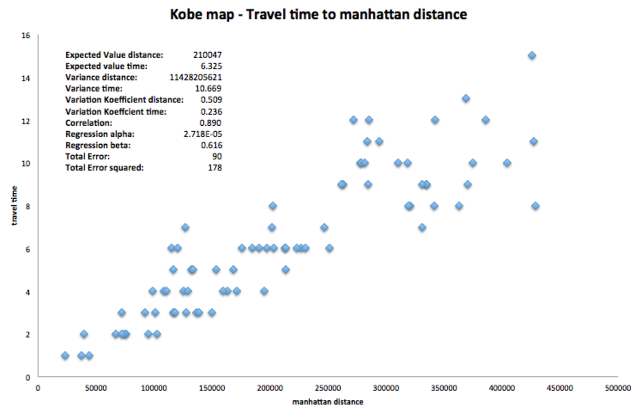
Figure 1: Relation between the manhattan distance and the actual travel time for the Test, Kobe and Berlin map



Expected Value distance:	116357
Expected value time:	4.143
Variance distance:	1802160845
Variance time:	21.571
Variation Koefficient distance:	0.365
Variation Koeffcient time:	1.121
Correlation:	0.441
Regression alpha:	4.829E-05
Regression beta:	-1.476
Total Error:	2
Total Error squared:	2



Expected Value distance:	1134715
Expected value time:	33.325
Variance distance:	4.751E+11
Variance time:	371.669
Variation Koefficient distance:	0.607
Variation Koeffcient time:	0.579
Correlation:	0.975
Regression alpha:	2.728E-05
Regression beta:	2.367
Total Error:	131
Total Error squared:	743



Expected Value distance:	210047
Expected value time:	6.325
Variance distance:	11428205621
Variance time:	10.669
Variation Koefficient distance:	0.509
Variation Koeffcient time:	0.236
Correlation:	0.890
Regression alpha:	2.718E-05
Regression beta:	0.616
Total Error:	90
Total Error squared:	178

Algorithm 2 Computation of water is put on a building per time step

Input: numberOfFirebrigade, TravelTimePerFirebrigade, RefillTimePerFirebrigade, FireFightingTimePerFirebrigade, FreschFightingTimePerFriebrigade

```
WaterPerTimeStep = new Array[NUMBER_OF_TIME_STEPS]
for all i in numberOfFirebrigade
    counter=TravelTimePerFirebrigade[i]
    while counter < TravelTimePerFirebrigade[i]+FightingTimePerFirebrigade[i]
        WaterPerTimeStep[counter] = WaterPerTimeStep[counter] + WATER_PER_STEP
        counter++
        if counter>NUMBER_OF_TIME_STEPS
            end while loop
for all i in numberOfFirebrigade
    counter=TravelTimePerFirebrigade[i]+FightingTimePerFirebrigade[i]
    while counter < NUMBER_OF_TIME_STEPS
        counter = counter + RefillTimePerFirebrigade[i]
        secondCounter = 0
        if j>NUMBER_OF_TIME_STEPS
            end while loop
        while secondCounter < FreshFightingTimePerFirebrigade[i]
            secondCounter++
            counter++
         WaterPerTimeStep[counter] = WaterPerTimeStep[counter] + WATER_PER_STEP
            if counter>NUMBER_OF_TIME_STEPS
                end while loop
return WaterPerTimeStepArray
```

Output: WaterPerTimeStep

The resultingvariable is then used in our algorithm to get the amount of water per step. This is then used in our algorithm in every step to calculate the effect of the water. It is computed in the initialization and then used in the cool method of our approximated adepted algorithm.

The calculation above is for U_S. For U_R the calculation is almost identical to the calculation used for $U_S(p, g)$. The problems for requests are that we have to estimate the location of fire fighters in another region, because we don't know there exact locations. Therefore we have to use a good estimate for there location to calculate the time it takes to get to our building. To do this we assume an uniform distribution of firefighters in the area of the requested agent and calculate their travel time based on this estimate.

5.1.5 exchangeBuilding()

Computing the exchangeBuilding() method is the most difficult. The problem is that we are just trying to get the development of one building for a specific plan. Calculating for all buildings would be too complicated. Especially for several plan templates that would have to be considered separately and considering that we still wouldn't be able to measure everything accurately, we have to simplify our computation in comparison to the physical processes in the RCR fire simulator. The exchangeBuilding() method is different from the cool() and the burn() method, in the fact that it is not just considering one building, but several other buildings in the environment. It is therefore also the reason for fires to spread to other buildings. There are many different ways for this step and we compared these different ways where we tested different version that just test some of these steps and just ignore the other for a lot of different test cases.

The first step of the exchangeBuilding method involves all Grid areas sharing the temperature of the air with the neighboring grid areas. The change in the temperature of the air is the air to air coefficient multiplied by, the length of each time step and the delta temperature between each cell and the average over all its neighboring cells. The delta temperature is the temperature in a grid element subtracted by the average temperature of all neighboring grids.

$$\Delta Temp = (cellTemp - AVG_{Neighbors}) \times T \times \Omega$$

T the length of a time step and the Ω air to air coefficient are given by the fire simulator configuration file. For the average of all neighbors, it just includes the addition of at most eight neighboring temperatures and one division by 8[23],

There are two ways to implement this, one is to go through all 400 to 5000 grid elements and do this computation (dependent on the configuration of the building) or just limit it to a specific amount of neighboring buildings, both ways have been tested for our empirical results in 6.1 (see Algorithm 1 and 2).

[23] In a Grid structure with equal sized rectangles each grid element has 8 neighbors when it is not on the border of the grid. Otherwise it has less then 8

After computing the exchange between the different grids we now compute the exchange of heat between air and building. It is the exchange of heat of buildings with the surrounding area.

For one building this step is pretty simple. We just have to do the same computation that the simulator is doing for our cell. Not considering all the other buildings and their cell makes the computation also much faster. The simulator is doing the following computation for each building and all the cells the building is in:

$$EnergyTransfer = (cellTemp - Temp_{Building}) \times \phi \times cellCover_{Building} \times A$$

$$newCellTemp = oldCellTemp - EnergyTransfer/\varrho$$

Where ϱ the Air cell heat capacity, ϕ the Air to building flow and A the cell size are from the configuration file and can be adjusted. The cell Temp starts at 0 and is updated in each time step. The temperature of a building is given in our computation and the cell cover for a building is computed in the initialization. It says how much percentage of a building are covered by a cell, so that each cells get a specific percentage of the building energy. This computation is the same as in the simulator, but limited to our initial building. In the spread prediction we will later include this calculation for a limited number of other buildings. We basically assume that all other buildings are not burning and therefore don't heat up the air. We will see that this assumptions works pretty good even if some neighboring buildings are burning but gets less accurate once we have a lot of neighboring buildings burning. In the empirical results we included a version where we tested the algorithm once with just considering our building and the exchange with air and once also considering a specific number of neighboring buildings, by just computing the exchange of energy for this building and not actually compute their burning (see section 6.1 for algorithm 3 and 4). For the spread prediction we also include an algorithm that computes the actual burning of neighboring buildings. All results can be found in the empirical results of section 6.1.

The third step to compute the exchangeBuilding method after computing the exchange of air heat between cells and the exchange of heat between buildings and the air, is the exchange of radiation energy between buildings. Given the temperature of our building we can calculate its radiation energy the following way.

$$radiationEnergy = TotalWallAreaOfBuilding \times \sigma \times \epsilon \times Temp^4$$

With σ being the Stefan Boltzmann constant and σ being the material specific degree of emission [9]. Because the simulator temperature is measured in celsius but this physics formula requires Kelvin we have to do the following computation $Temp = getTemperature()_{OfABuilding} + 273$. This is the same way it is in the simulator [9].

This radiation energy first reduces the energy of our building by

$$NewEnergy_{Building} = OldEnergy_{Building} - radiationEnergy$$

In addition it is also heating up other buildings and other buildings are heating up our building that we are considering. Including this additional information requires some further computation, therefore two approaches are possible. The first one just involves our building loosing his radiation energy and not computing the Monte-Carlo process or an estimate for determining the connected buildings.

The second approach computes the exchange of radiation between different buildings and also the computation of buildings to the initial building. To do this we have to first calculate the connection values that the simulator is computing by a Monte-Carlo process. The process works like this depending on the radiation energy of a specific wall it is sending a couple of random rays and when it hits a building that this is part of the connected buildings. The number of time a connected building is hit by such an random ray determines the connected value which is the number of hits divided by the total number of rays send of our initial building. We then calculate the energy arriving at our building and the energy going to another building with the this value.

We are doing the following algorithm for our initial building:

$$NE_{InitialBuilding} = OE_{InitialBuilding} + \sum_{k\varepsilon\{AllConnectedBuildings\}} RE_k * CV_k$$

$$NE = NewEnergy$$

$$OE = OldEnergy$$

$$RE = RadiationEnergy$$

$$CV = ConnectionValue$$

This is the same calculation that the simulator is doing and then we are doing the following calculation for each connected building:

$$NE_{ConnectedBuilding} = OE_{ConnectedBuilding} + RE_{InitialBuilding} \times CV_{ConnectedBuilding}$$

We assume that the connection value of our building to another building are identical. This can be done because they usually have similar results for their connection values as we tested for five buildings and 20 runs. We therefore assume they are the same, although this isn't exactly the case in a random process like it is used in the RCR simulator. The exchange of energy between all connected buildings with other buildings then the initial building is also not considered, because the computation effort would be too much do this for all connected buildings. (See section 6.1 at algorithm 3 and 4 that consider the exchange with other neighboring buildings.). This values are also considered for 5.2 to calculate the spread of fires to other buildings.

For the exchange building we mainly change the energy of the building that is later used to compute the fuel they burn and determine the destruction of the building. After describing the different steps of our algorithm we now want to give a further analyzes of the evaluation of our results in the satisfaction degree and the time calculation.

5.1.6 Time and satisfaction degree calculation

After doing all this calculation, we will now show how this calculation can be used to determine the two important values time and satisfaction degree.

For the time we have to consider that our calculation is executing each of the physical process approximation steps similar to the fire simulator. Our calculation does the steps (burn, cool, exchange building) described in the previous subsection. To produce the time we computed those steps in a loop where the time was the counter, when the temperature of the building was smaller then a specific threshold the loop stopped and the counter can be seen as the result for the time, because our plan would end when the building is burned down or extinguished. We just have to adjust the time by adding two time steps to this counter, because this is the time our algorithm and the OAA architecture take to detect that a building is on fire, develop possible plan templates calculate all specific plan and goal combinations and initialize the actions of the plan like starting to send the fire fighter to the fire. This means when we evaluate our plan based on the time , when we start to execute the plan, what is also the time we are blocking the resources.

Therefore t as a result is the number of loops that our algorithm takes for a specific goal and plan combination till the fire is extinguished or the building is burned down. It therefore represents the time our resources would be blocked and therefore is really good for our specific calculation of opportunity cost.

For the satisfaction degree we have to use the fuel that is left at the end of our calculation. In our calculation with the steps described above, the energy is changed in all these steps and used to calculate the temperature and the fuel. The temperature indicates when the building stops burning and determines the time for this to happen as described above and the fuel is used in the simulator to the determine the amount of damage that happened to a building.

The actual calculation for this is the same for our algorithm as it is for RoboCup rescue. Whenever the fuel is going under a threshold percentage of the initial fuel at the beginning of the simulation it is going to the next state. RoboCup rescue has four of these states. The first one is reached when $fuel < 0.66 \times InitialFuel$, the second one is reached, when $fuel < 0.33 \times InitialFuel$ and the last one when $fuel$ is 0.

The first state represents a light fire, the second state a heavier fire, the third state a really strong fire and the fourth state that a building is burned down.

These states are then used to calculate the amount of damage that happens to a building. The specific damage can be given in the configuration file. But they are usually set to 66 percent for the first 33 for the second and 0 for the third and fourth, while the building is burning and 0.75 for the first, 0.50 for the second, 0.25 for the third and 0 for the fourth. Therefore extinguishing a building actually reduces the damage of this building.

The results for the time and fuel are then used to compute $S(p, g)$ and $c(p, g)$. It is important to notice that this algorithm tries to predict the possible state that the building has after a plan is executed, but he is not trying to predict any kind of spreading to the neighboring buildings. He is therefore analyzing the current state of fires and not any potential spreads of the fires.

The empirical results in section 6.1 show the accuracy of our algorithm for the no-spread predictive and spread prediction case.

We will now take a closer look at the algorithm for spread prediction and how it is different from our algorithm without spread prediction.

5.2 Algorithm with spread prediction

The predictive algorithm is similar in the case that it also uses the physical properties of the fire simulator[24], but in addition to just analyze what the possible results for the current fires are it tries to predict the influence of the current fires to other buildings. By doing that it should be able to perform better then the algorithm, without prediction because it included the potential of fires to spread to close bigger buildings for example. We will explain different algorithm that we used later to empirically analyze the effects of prediction.

This it might give different utility to buildings that could for example inflame other buildings. The basic structure for utility calculation is identical to the calculation for the no-prediction algorithm. We also have $U_R(p, g)$ and $U_S(p, g)$ and use them in a similar way. The value for $v(g)$ and the calculation for the amount of water for every time step is also identical. Therefore we can also use the same and already existing plan and goal selection algorithm (as described in section 4.2 and mentioned in section 5.1).

The main differences are in $S(p, g)$ and $v(g)$. To calculate these values we have to identify the buildings that we predict are going to start burning when we use this plan. We will use the same basic algorithm for analyzing each goal, so that we can compare the effects of prediction and no-prediction and don't have influence from different goal evaluation in our results. To do that we will use the knowledge of how many energy is already going to a specific number of close buildings, for example because they are in the same air cell or they exchange radiation energy with our building.

The exchange for a specific close building is described and done as in 5.1.5. The next question is how we select the number of close buildings, [25] and how we decide weather they start burning or not, because there is no clear rule which

[24]It could also use the real physical properties that exist for a normal fire in a building.

[25]Close buildings are defined as buildings that could be ignited by our building.

buildings to consider and when these start burning. The critical problem is considering more buildings means more complexity. Selecting all buildings for this wouldn't work because it would take too much time. There are different ways we could do this and we will present an overview on the different ways in 5.2.2.

Another question we have to address is whether we just see what buildings we are igniting or if we also compute their burning behavior. That means if we should also consider that this building is heating up further because it is burning and has further influence on other buildings. This would cause more computation but would make our prediction possibly more accurate. We will explain how we computed this for some of my experiments in 6.2.3. Finally we will show how our computation for opportunity cost and satisfaction degree change due to this additional knowledge in section 5.2.4.

5.2.1 Selecting close buildings

After showing how each of the building could exchange energy with our building in 5.1.5. But instead of just limiting these effects to the buildings for air in the same grid and for radiation the results of the Monte-Carlo process we choose the buildings as described in here, which can be very similar to the way already described.

We now give an overview over possible ways to select the building. We then just have to compute the energy of this buildings and go through some number of buildings and check their temperature in each time step.

To get a good impression on how accurate the prediction has to be and what values should be considered we developed the following fife selection algorithms. That decide which building to consider as close buildings and when to consider them ignited or burning, so that we can say our fire spreads to this building. Three of these algorithms consider the temperature of a specific number of buildings. When this temperature is above a material dependent threshold we define this building as burning like it is done in the simulator. In addition we also developed two algorithms that don't directly include the temperature knowledge although this is the main criteria that should be considered in every algorithm, because it is determining if a building is igniting or not in the real simulator. They are used as empirical benchmark for the existing algorithms.

The following list contains each selection method with a short description on how it works:

1. Air predict: This selects all buildings, that are in the same grid elements then the burning building and ignites them when there temperature is higher then the ignition temperature for the material they are made of

2. Radiation predict: This selects all buildings that share radiation energy with out building and has the same ignition criteria then air predict

3. Air and radiation predict: This considers both buildings from air predict and radiation predict and ignites them also with the ignition temperature criterium

4. Random predict: This algorithm randomly selects one of the buildings from air and radiation predict and ignites it with probability 0.1. Where the probability was selected because over a number of runs over several buildings this was the average number of buildings ignited in each time step. It is mainly used as benchmark for 1-3.

5. Random select predict: This algorithm always takes a random building out of the buildings for air and radiation predict and marks it as ignited, when one building starts burning in a specific time step. For example if we have building A, B and C and by temperature building C starts burning, then this algorithm just selects one of A,B or C and ignites it at random. And if A,B start burning then it selects two. It is much less accurate regarding the building that is burning, but it is still containing the knowledge if another building starts burning and therefore contains some knowledge of spread.

Empirical results for all this five algorithms can be found in 6.3.2. After we showed how building

5.2.2 Considering the burning of other buildings

If we know that one building is ignited the questions is what are we doing with this knowledge and how could we further use this information. One way is just use the knowledge that this building is burning. A further way would be to compute this is building to and analyze it's influence. It would also be interesting to see if the further evaluation of another goal has influence on our goal as we analyzed in section 3.1 already. Therefore we developed the following algorithm to further predict the burning of already ignited building. The problem is that this computation is much more complicated because it can drastically increase the things we have to compute. But it might also strongly improve the accuracy and performance of prediction. This effects and especially the effect of additional influence on the initial fire, should be analyzed with this algorithm.

We do this computation for each of the first 10 ignited building we know that start burning, because for more then 10 the computation starts to get too complicated. This already shows that this approach is very limited and can't be applied to all ignited fires. Because we already do the computation of exchange with air and we also do some kind of radiation exchange in with our initial building computing the Monte Carlo process for all new ignited buildings would be too complicated, we don't do this steps for the new ignited buildings. We also assume that there is no water put on this building, because if we would assume so, we would have to predict further plan templates or use other kind of approximations. Because it usually always take some time till new fire fighters arrive at a building and the goal we want to evaluate just burns for a limited time we can make this assumption. Therefore we also don't include this cool calculation for new ignited buildings. The only step we have to do for all new burning buildings is the burn step of the simulator. The good thing about this simplification is, that the computation becomes much easier then if we would consider all buildings and can therefore be done without much additional complexity. It is also just focused on the burn step and therefore just on the one ignited building. The radiation would include considering further buildings and an additional process to determine the connection values.

The calculation for the burn step is identical to the calculation for the burn step of our initial burning building. Details on the computation can be found in section 5.1.3 and section 4.1. It requires the temperature (or energy), the fuel and some building specific information like capacity, area, number of floors and material. The last one can be computed given the building and the first one is already computed for 5.2.2. So by adding one more variable for every building like the fuel we are able do this calculation. Because the calculation of initial fuel wouldn't have a huge effect for each building and would be too complicated, we assume that each building didn't burn before to avoid the problem with previous buildings mentioned in 5.1.1.

It is also important to notice that by computing this step for all buildings that have a certain temperature and by setting the specific temperature of neighboring buildings who already burn higher then the threshold value, we automatically also compute the burning of neighboring buildings that have already been on fire. This includes even more information in our algorithm and makes him potentially more precise without significantly changing the simulation. We thereby also analyze the positive effect of this is that we also analyze the effects of potential other local goals on our spread prediction.

The rest of the first algorithm stays the same way and is not changed further. We just add the computation loop where each of the ignited building is burned like we already defined with the burn function. As we will see in the empirical results is that calculations that are only just considering the other buildings to heat up via radiation and air heat transportation already performs worse then algorithm that don't consider this. The reason is that this buildings already contain much more heat then the real buildings, because they don't loose heat to other buildings via radiation and especially via air.

We therefore don't use this additional burn information to influence our initial building and just try to use this information to predict further ignition of buildings.

Details on the empirical results for this algorithm can be found in section 6.1.2.

5.2.3 Satisfaction degree calculation

After using these 5 algorithms to identify possible burning buildings we use this to calculate the satisfaction degree and the cost. But this time we have to include the additional information for the fires that start burning in the calculation. For the satisfaction degree we are doing the following computation:

$$S_{Spread}(g,p) = S_{NoSpread}(g,p) + \frac{\sum_{k \varepsilon \{ConsideredBuilding\}} BuildingSize_k \times \gamma}{BuildingSize_g}$$

The $S_{NoSpread}(g,p)$ is given by section 5.1. We know the considered buildings and therefore their total area from the previous described algorithm.

For the cost the calculation for our initial building is identical but in addition we have the costs for buildings that are starting to burn. which are:

$$c_{Spread}(p,g) = c_{NoSpread}(p,g) + \sum_{k \varepsilon \{IgnitedBuilding\}} BuildingSize_k \times \gamma$$

We multiply each buildings size by degree γ because we assume that we will extinguish it very fast. In the given configuration of the simulator damage calculation this is the percentage of building that is not considered damaged in the best case.

$$\gamma = (1 - DamageExtinguishedBuildingInStateOne)$$

The value for the damaged extinguish building in state one is given in the configuration file. It is usually the value 0.75 and the best possible state a building we know is going to ignite could be in. We could also assume the worst, set the value to 1 and assume that the whole building is definitely going to burn down. To make a better calculation for this we would actually consider the number of other fires currently burning and the number of free resources. Analyzing this as a further and probably more accurate prediction of the future state of a building given resources and other goals. But for a first analyzes of prediction this calculation is sufficient and very conservative, that means it is underestimating the possible positive effects of prediction.

By multiplying γ with the building area we have the minimum safe area of the considered, that this plan would cause for a specific goal. We then divide the possible damage of the building by the area of our goal building, because when we multiply this with $v(g)$ we get:

$$v(g) \times S_{Spread}(g,p) = S_{NoSpread}(g,p) \times BS_g + \sum_{k \varepsilon \{CB\}} BS_k \times \gamma$$

$$BS = BuildingSize$$

$$CB = ConsideredBuilding$$

Which is the minimum total building damage that a plan would cause and therefore the identical utility equivalent to the no spread prediction case, where we just analyze the building and the destruction for each building.

6 Empirical results

We will analyze the spread prediction and no-spread prediction algorithms described in the previous section based on three values: accuracy, time consumption and buildings damage prevented.

First we will analyze what kind of accuracy my prediction and my spread prediction algorithms were able to generate. This important to get an impression of what accuracy level is necessary to get better performance with spread prediction. The second one is the time of these two different type of algorithms take. Spread prediction usually means additional computation and therefore additional time consumption we want to analyze what is the actual computation time difference between these algorithms. The last result analyzed is the total building damage, because this is the main value that should be reduced at least for the utility functions as we set them for this project.[26]

6.1 Required accuracy

For the accuracy we will first analyze the accuracy of the non-spread prediction algorithm. We will analyze different algorithms defined in section 5. After that we will analyze the spread prediction algorithm and show how accurate the different algorithms are in defining how, where and when a fire is spreading. In the last section we will analyze the effects of the algorithms where we further computed the burning behavior of an already ignited building.

6.1.1 Non spread prediction accuracy

In section 5.1 we showed that the steps in the algorithm for the burn and the cool method of the RoboCup rescue fire simulator are relatively simple, therefore it can also be computed easily without further considering if all these computational steps are really necessary.

An algorithm without the burn step wouldn't work at all and an algorithm that is not considering an algorithm that is not considering the cool step would deliver the same result for all plans. These two steps are therefore absolute essential for an algorithm that tries to compute the physical processes of a fire[27]. We therefore consider these steps in all six algorithms.

The only critical step was the exchange fire method. In section 5.1.5 we showed the different versions for this method, that should be tested towards their accuracy. We therefore will test the following 6 algorithms and check how accurate they are:

1. Algorithm 1: Considers the radiation and the exchange of energy with the air. It also computes the global air exchange.
2. Algorithm 2: Considers the radiation and the exchange of energy with the air. It just computes the local exchange of air with direct neighboring cells.
3. Algorithm 3: Considers the radiation and the exchange of energy with the air. It also computes the radiation energy and the temperature exchange of neighboring buildings.It also computes the global air exchange.

[26]As already indicated further work might include other values and therefore might consider other evaluations of their functions.
[27]Which are also simulated by the fire simulator of RoboCup Rescue.

Table 1: Values of the used test cases.

test cases	AVG time	AVG fuel	AVGstate	Variance time	Variance fuel	Variance state	# cases neighbors burn
50 cases	20.63	1017370	1.51	15.68	637119	0.691	23
100 cases	19.7	642855	1.42	14.41	270837	0.7	43

4. Algorithm 4: Considers the radiation and the exchange of energy with the air. It also computes the radiation energy and the temperature exchange of neighboring buildings.It also computes the local air exchange.

5. Algorithm 5: Considers the exchange of energy with the air, but not the radiation. It also computes the global air exchange.

6. Algorithm 6: Considers the radiation, but not the exchange with air. It also computes the global air exchange.

Fore these algorithms we developed 50 test cases for these scenarios for all algorithms and then further tested the best four of them on 50 additional cases. (Overview about the test cases can be found in Table 1). In the results we distinguished between cases were the neighbor was burning and the neighbor wasn't burning. The reason is that we limited our computation for each plan to just one building. This makes cases were neighbors burn harder to compute accurately, they should therefore be analyzed seperately. The algorithms are evaluated based on their accuracy regarding fuel and state a building has after a fire is extinguished and the time it took to extinguish the fire. It is important to notice that state and fuel strongly influence each other. The state determines the actual result of the algorithms but as we explained in section 5.1.6 therefore a correct prediction of it is important. But because the state is computed based on the fuel (see section 5.1.6 for details) and accuracy for this value also means more accurate states. Especially because the state can just have 4 values and is reducing the values for fuel to values from 1 to 4 an accurate fuel prediction is a very important. As we will see in section 6.2 algorithm 2 also works faster then algorithm 5 and 6 and should therefore be preferred also for time critical situations.

For the 50 test cases table 2 shows the correlation between the actual results and the estimated results of our non-spread prediction algorithms. Table 3 shows the error in the prediction.

The graphs that showing the graphical relation for actual and estimated results can be found in attachment 1 (Figures 5 to 11).

We see that the algorithm 5 and 6 are less accurate results, then algorithm 1 to 4 regarding almost all kinds of error. They are also a little worse in the correlation values but not then clear. When we look at the graphs in attachment 1 this is getting clearer. Regarding the fuel algorithm 5 and 6 tend to overestimate

Table 2: Correlation of our analyzed algorithms with the real values for 50 test cases

Algorithms	total			no neighbors burn			neighbors burn		
	time	fuel	state	time	fuel	state	time	fuel	state
Algorithm 1:	0.874	0.915	0.777	0.978	0.991	0.742	0.880	0.872	0.787
Algorithm 2:	0.932	0.914	0.834	0.972	0.981	0.742	0.925	0.883	0.844
Algorithm 3:	0.840	0.695	0.660	0.974	0.959	0.408	0.843	0.499	0.699
Algorithm 4:	0.837	0.812	0.668	0.976	0.964	0.593	0.808	0.718	0.680
Algorithm 5:	0.801	0.740	0.718	0.925	0.738	0.637	0.834	0.738	0.825
Algorithm 6:	0.792	0.693	0.497	0.899	0.636	0.350	0.831	0.784	0.609

Table 3: Average error to the real state for the 50 test cases

Algorithms	total			no neighbors burn			neighbors burn		
	time	fuel	state	time	fuel	state	time	fuel	state
Algorithm 1:	4.54	60103	0.22	1.59	24250	0.074	8	102191	0.391
Algorithm 2:	4	73020	0.18	2.07	35291	0.074	6.48	117310	0.304
Algorithm 3:	6.35	114766	0.33	2.72	51233	0.16	10.3	183822	0.522
Algorithm 4:	5.98	101253	0.28	2.56	48256	0.111	10	163467	0.478
algorithm 5:	8.88	238750	0.62	8.85	232940	0.703	8.91	245570	0.522
Algorithm 6:	9.9	284907	0.78	9.93	287406	0.889	9.87	281974	0.652

the burning process while 1-4 are pretty accurate but sometimes underestimate it a bit. The reason is that 5 and 6 are not considering the complete exchange of energy with the environment. Therefore some energy that is actually leaving the building is not computed in our simulation. This also explains why they don't perform worse regarding the fuel for the case when the neighbors are burning, because they are then indirectly capturing the additional heat that actually comes from neighboring buildings but is not considered in algorithm 1-4.

The time is usually underestimated especially for longer burning processes by algorithm 1-4 and overestimated for algorithm 5 and 6. Again 1-4 perform worse with burning neighbors and 5 and 6 perform almost identical for burning and not burning. The reasons are the same as for the fuel. Algorithm 1-4 are considering not neighboring all influences and 5 and 6 are not loosing enough energy to the environment. Similar results can also be found considering the state. With 5 and 6 tend to overestimate the state and the damage that is going to happen in both case with and without neighbors burning and 1-4 tend to underestimate the damage for the case of neighboring buildings burning.

Overall algorithm 5 and 6 are not performing very good especially for the more common cases of fires that are burning shorter. [28]We also try to prevent

[28]The reason for this is the more common type of smaller buildings in the Kobe RCR map and also in a real city, therefore predicting them more accurately is important. It doesn't happen every day that a sky scraper is burning and different techniques could be applied for this but smaller buildings can't be treated separately in a city and a good algorithm for them is more important.

Table 4: Correlation of further analyzed algorithms with the real values for 100 test cases

Algorithms	total			no neighbors burn			just neighbors burn		
	time	fuel	state	time	fuel	state	time	fuel	state
Algorithm 1:	0.915	0.906	0.899	0.984	0.961	0.957	0.9	0.825	0.825
Algorithm 2:	0.945	0.736	0.831	0.977	0.853	0.791	0.93	0.606	0.883
Algorithm 3:	0.868	0.402	0.844	0.95	0.459	0.922	0.873	0.331	0.762
Algorithm 4:	0.872	0.436	0.850	0.951	0.479	0.936	0.861	0.379	0.758

Table 5: Average error in comparison to the real value for 100 test cases

Algorithms	total			no neighbors burn			just neighbors burn		
	time	fuel	state	time	fuel	state	time	fuel	state
Algorithm 1:	3.58	128732	0.15	1.63	79809	6.16	6.16	193582	0.256
Algorithm 2:	4.07	219561	0.18	2.28	143129	6.44	6.44	320878	0.233
Algorithm 3:	5.89	395301	0.316	3.29	338825	9.21	9.21	467536	0.442
Algorithm 4:	5.73	369881	0.28	3.21	326196	9.07	9.07	427787	0.419

the case were we have to consider a lot of neighboring fires and exponential fire growing. It therefore should be happen less often then the case that no neighbors are burning. Therefore accuracy for the no-burning environment is a bit more relevant.

Therefore we are After eliminating algorithm 5 and 6 and further comparing algorithm 1-4 with additional 50 cases. We get the empirical results regarding error and correlation in table 4 and 5.

Graphs of this 100 cases can be found in attachment 2. Overall the results are in some way similar to the results for the 50 cases, but it is getting clearer that algorithm 1 and 2 are performing best and algorithm 3 and 4 not that accurate. Algorithm 1 seems to outperform algorithm 2 a bit.

Regarding fuel all four algorithm perform very good in general. Especially for the cases were no neighbors are burning. If the neighbors are burning they all tend to underestimate, which makes sense because they are all not considering neighboring influences due to computational complexity. Algorithm 3-4 are underestimating more then algorithm 2 and 1. The reason for this is that they don't perform so good in cases were neighbors are burning they have more and bigger errors. Between algorithm 1 and 2, algorithm 1 performs best regarding the fuel, because it makes less and smaller errors as you can see in the graphs. The results for the state are similar to the results for fuel.

Regarding the time all algorithm tend to underestimate, especially for longer cases and when the neighbors are burning. Again algorithm 3 and 4 are a less accurate and algorithm 1 seems to perform a bit better then algorithm 2, because it makes less errors.

Overall we can make the following order regarding accuracy of the for algorithm:

1. Algorithm 1
2. Algorithm 2
3. Algorithm 4
4. Algorithm 3
5. Algorithm 5
6. Algorithm 6

Of course and if we would consider things like computation time algorithm 2 might perform better then algorithm 1 (see section 6.2).

Surprising is that algorithm 1 and 2 perform better then algorithm 3 and 4, because they also compute the burning process of neighbors. The reason probably is that they are too inaccurate in computing this process in this buildings, because considering more details would cause more computational time.

In the next step we will analyze the accuracy of the different spread predictive algorithms.

6.1.2 Spread prediction accuracy

For the prediction of spread we developed 8 algorithm and 45 test cases. The algorithms are the following:

1. One: Checks all buildings that are in the same area f they have heated up enough to burn
2. Two: Checks all buildings that receive radiation energy from initial building if they have heated up enough to burn
3. Three: Checks all buildings that receive radiation energy from initial building and are in the same area if they have heated up enough to burn
4. Four: Randomly selects a building to ignite out of the when one other building when one building would start according to spread prediction algorithm three
5. Five: Randomly selects a building with probability 0.1 and marks it as ignited
6. Burn one: Spread prediction algorithm one but with the additional steps of computing the burning the other building, when it would burn.
7. Burn two: Spread prediction algorithm two but with the additional steps of computing the burning the other building, when it would burn.
8. Burn three: Spread prediction algorithm three but with the additional steps of computing the burning the other building, when it would burn.

Table 6: Empirical comparison between the eight spread predictive algorithms

	One	Two	Three	Four	Five	Burn one	Burn two	Burn three
Correct predicted buildings	210	452	444	450	153	219	461	460
Wrong predicted buildings	2	143	142	136	152	8	132	132
Predicted ignited buildings	212	595	586	586	305	227	593	592
Real ignited buildings	1120	1120	1120	1120	1120	1120	1120	1120
Percentage of correct predicted buildings	18.9%	53.1%	52.3%	51.5%	27.2%	20.3%	52.9%	52.8%
Correlation between predicted and actual number of ignitions	0.649	0.652	0.7	0.7	0.448	0597	0.736	0.737
Correlation between predicted starting time and actual starting time	0.566	0.531	0.547	0.486	-0.118	0.595	0.545	0.55

(For further details on this algorithms see section 5.2).

Based on the 45 test cases where the fire of a building was spreading to other buildings we measured to critical values. The first value is the predicted number of buildings that we expect are going to burn in comparison to the actual number of buildings that start burning. This is giving an impression about the accuracy regarding the accuracy about the effect a fire could potentially have, by predicting the number of buildings that are starting we are doing a workload prediction and are comparing the algorithms using this prediction. If we would want to use the second predictive advantage as it is described in section 1 and 3 we would also have to measure if the correct buildings are predicted in the correct order.

The second comparison is the time the algorithms predict a building is going to start burning. This is just done on the correct predicted buildings. So for all these correct predictions we are comparing the predicted and the actual time.

Both values influence the actual decision described in section 5 and are therefore important measurements for determining the accuracy of each algorithm. Empirical results comparing the eight algorithms based on these two values can be found in table 6.

Graphs that are showing the relation between number of predicted ignitions and actual ignitions can be found in attachment 3. The graphs that show the relation between the actual and the predicted ignition time are in attachment 4.

Regarding the accuracy in the number of fires the overall performance of the algorithms doesn't look very good in comparison to the results from 6.1.1, but relative to the complexity of the problem still seem to be pretty good. The main problem is that some fires grow so fast, that our algorithms can't cover this growth. Most of them are therefore getting more inaccurate with growing number of total buildings that start burning, while they are more accurate with smaller number of starting buildings. Due to the usual size of all buildings in the Kobe map and in the usual city the fires mainly don't tend to grow that strong so that the problem appears less in reality. One isn't performing very good and strongly underestimates the number of buildings that start, while two and three show better accuracy performance. This shows that including the radiation buildings has much stronger influence on how good we can identify if a building would be ignited. Two and three also have problems with longer fires, because environmental influences get more complex and harder to compute. The algorithm three and four are identical regarding the number, because four uses three to identify when it should ignite a random building. Five as the completely random algorithm obviously often choses the wrong building. Burn one is a bit better regarding the number of correct predicted buildings but the overall correlation is also not very good. Like one it is also underestimating many buildings. Burn two and burn three show good performance and are more accurate regarding the number of correct predictions per building. This is another indicator that shows how important the exchange of radiation energy is in the fire simulator. They also show a higher correlation between the actual and the predicted number of ignitions. Burn two and burn three also show some tendency to underestimate.

Surprisingly is the accuracy regarding the actual buildings that are predicted is more accurate for four then for three, also three chooses the buildings randomly. This probably shows again the importance of radiation prediction, which is implicit in this algorithm. Because a lot of radiated buildings are selected to start igniting the algorithm selects a lot of these buildings even with random selection and is therefore very accurate. The reason for the little higher accuracy could then just have been luck regarding the selection of a view houses.

Good is that most algorithm are more accurate in the smaller number of actual ignitions, because these cases are more common in our test map and also in most cities, were most of the buildings are smaller.

Regarding the actual and predicted ignition time of correct spread predictions algorithm the overall performance doesn't seem very good but has to been seen with the knowledge about the complexity of this problem. One and burn one seem to be the most accurate one but they just tend to recognize smaller fires which are then also easier to predict regarding the time. Algorithm four is not so accurate and especially five is really bad. In five we clearly see the random behavior. Four shows still a pretty good performance, the reason is probably the same as it was for the exactness regarding the numbers. The importance of radiation prediction gets clear in this.

The most accurate once are two, three and burn two, burn three. They clearly show the highest performance, but the burn algorithms are a bit more accurate.

When it comes to the overall accuracy of workload prediction the following order regarding the performance of the algorithms seems therefore to be appropriate. [29]

1. Burn three
2. Burn two
3. Three
4. Two
5. Four
6. Burn one
7. One
8. Five

But this is just considering the workload prediction which is mainly relevant for our case. For the prediction of possible further targets algorithm one would be great because it has a high success rate.

6.2 Time consumptions

After we analyzed the algorithm considering their accuracy we also want to analyze the time used by the algorithm to compute the prediction, because one of the main disadvantages of prediction is the time it takes to calculate it.

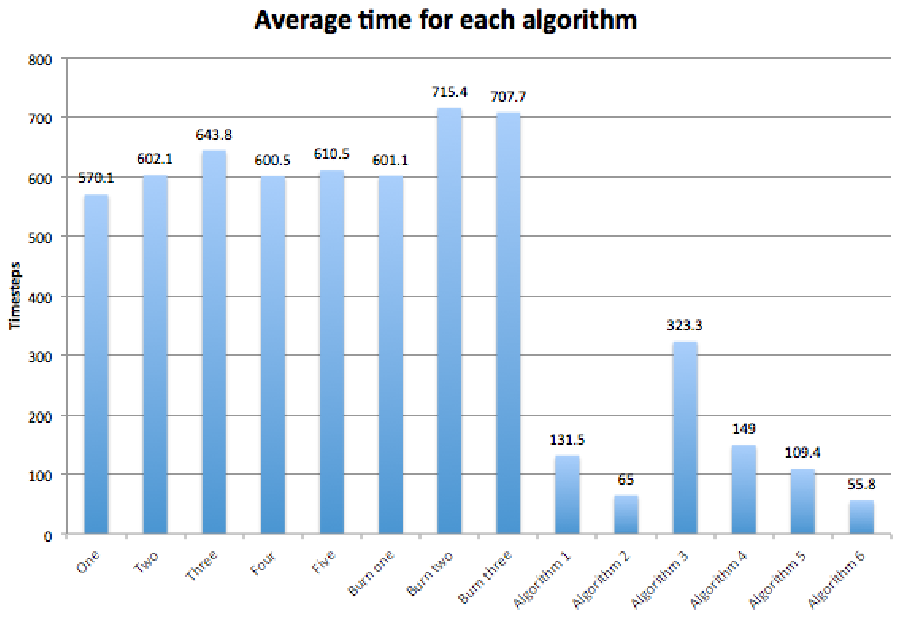
The graph in figure 2 shows the time results for the algorithms of 6.1.

We see what we already expected during the algorithm description in section 5. The spread predictive algorithms take much more time steps then the not spread predictive algorithms. Among the spread predictive algorithms the algorithm *burn one*, *burn two* and *burn three* also take more time then the algorithms that are not consider additional buildings to burn. Because as we have seen in 6.1.2 *burn two* and *burn three* are the most accurate spread prediction algorithms and consider most buildings they also require the most time. We see that using either *three* or *burn three* as our predictive algorithm isn't much worse then the other algorithms in relation to the overall time they take to the othter algorithms and the better performance they show.

For the no-spread prediction algorithms we see that algorithms that consider more influences like *Algorithm 3* for example which considers the heat of other buildings and the global air take much more time then algorithms that are just considering the building itself. We also see that the computation of the global air takes more then the double time in *Algorithm 2* and *Algorithm 1*. Considering the small gain in accuracy for *Algorithm 1* in comparison to *Algorithm 2* and the bad performance of the other algorithms, overall *Algorithm 2* shows the best performance in relation of time to accuracy of these six algorithms. But for short time or higher accuracy *Algorithm 6* and *Algorithm 2* are still good

[29]This is more a qualitative then a strict quantitive order given the empirical results and our expectations about the environment.

Figure 2: Average running time of all algorithms tested in the accuracy section



options. As future work some improvements in the consideration of the burning of other buildings in *Algorithm 3* and *Algorithm 4* as described in 6.1.1 could make them potential candidates if more accuracy are required.

Algorithm 2 still shows a very good performance for RoboCup rescue and seems already seems to be a very promising algorithm for this domain, although potential further improvements as described in 7.2 are always possible.

Overall we clearly see that spread prediction requires much more time then just prediction. One of the reason is definitely the strong increase in things that have to be considered. It also shows that this prediction can't be done for many additional steps. In the next subsection we will analyze if this additional computational effort generates better results, then no-spread prediction.

6.3 Building damage

While the 6.1 and 6.2 are mainly worried about the performance of my specific algorithms. These section analyzes the effects that a spread predictive algorithm has in comparison to non-spread predictive algorithm regarding performance. It therefore analyzes the potential positive effects of prediction. To do this is requires the most complicated environmental settings[30]. We developed three

[30]An environmental setting is hereby defined as some arrangement of the environment, for example more buildings burning in areas with higher building density. It is not a defined configuration where we already know which buildings is burning when this term is defined as a scenario for this project.

kind of different environmental settings where we want to test our algorithm and compare the performance of algorithms with spread prediction to the algorithms without spread prediction.

These three environmental settings are:

1. Setting with enough resources for all goals
2. Setting with lack of resources, but small difference in the evaluation of goals
3. Environment with enough resources and strong difference in goal utility evaluation

The settings are oriented on the theoretical results we already developed in section 3, where we identified settings with low resources and a lot of difference in the spreading behavior of buildings as the one where spreading prediction has the most positive effect. This would be equivalent to setting 3.

The setting one and two analyze the effect of spread prediction in settings where we theoretically expect the effect of spread prediction to have a lower advantage. The first one represents the one where there are too many resources and the decision for a specific goal is less critical. The third one represents the setting where all goals have a very similar spreading behavior and therefore the difference between a spread-predictive and non-spread prediction in the evaluation of goals is small and their behavior and results really similar.

For each environmental setting we developed 2 specific scenarios, where we implemented this setting. Each of these scenarios has a different number of fire fighters and OAA agents fire stations that control the fire fighters. The first scenario has always 6 agents and just one OAA agent fire station agent. The second scenario has 10 fire fighting agents and two OAA agent fire station. The guidelines usually used in the OAA architecture are not used for our scenarios. We chose this settings because we wanted to test the prediction algorithms in a complex environment and show where it performs better. We also wanted to show that it works for the case with and without requests and without request in the specific environment we used and is therefore also applicable for Multi-Agent systems.

The empirical results for these six test cases can be found in table 7 (setting 1), table 8 (setting 2) and table 9 (setting 3) and attachment 5. The algorithms are compared based on the overall damage they can prevent to happen to the city.

About setting 1 we see as expected that all fires are extinguished really fast and that there doesn't exist any negative exponential spreading effect that has dramatic influences on the performance of any of these results. We also see that in one case the performance of our spread-prediction algorithms is better and in the other one the performance of the no-spread predictive algorithms are better.

In setting 2 we see a similar situation but with two difference. First the difference between the algorithms is much bigger. Therefore resource critical means also more difference in overall system performance of different algorithms.

Table 7: Empirical results for damage test cases comparison (Setting 1)

Setting 1 - Test case 1			Setting 1 - Test case 2		
Rank	Algorithm	Undamaged area at the end in %	Rank	Algorithm	Undamaged area at the end in %
1	Burn three	79.4	1	Algorithm 2	86.1
2	Three	79.2	2	Algorithm 1	85.3
3	Burn one	78.8	3	Algorithm 4	85.0
4	One	78.4	4	Algorithm 3	84.6
5	Burn two	77.1	5	Burn three	83.2
6	Algorithm 1	76.7	6	Burn two	82.9
7	Algorithm 4	76.4	7	One	82.4
8	Algorithm 2	76.2	8	Algorithm 6	82.4
9	Two	76.0	9	Four	82.0
10	Algorithm 3	75.6	10	Algorithm 5	81.3
11	Algorithm 5	74.5	11	Burn one	81.2
12	Four	74.4	12	Two	81.0
13	Algorithm 6	74.1	13	Three	80.7
14	Five	73.4	14	Five	79.1

Table 8: Empirical results for damage test cases comparison (Setting 2)

Setting 2 - Test case 1			Setting 2 - Test case 2		
Rank	Algorithm	Undamaged area at the end in %	Rank	Algorithm	Undamaged area at the end in %
1	Burn three	57.2	1	Algorithm 1	74.4
2	Burn two	55.1	2	Algorithm 2	74.0
3	Three	53.9	3	Burn two	72.7
4	Algorithm 1	53.4	4	Burn three	71.6
5	Two	52.8	5	Algorithm 3	70.9
6	Algorithm 2	52.2	6	Burn one	70.3
7	Four	48.3	7	Algorithm 4	58.4
8	Burn one	48.2	8	One	56.9
9	One	46.4	9	Four	56.9
10	Algorithm 4	45.1	10	Algorithm 5	54.2
11	Algorithm 3	42.9	11	Three	53.9
12	Algorithm 5	18.5	12	Two	52.8
13	Algorithm 6	15.2	13	Algorithm 6	52.3
14	Five	13.5	14	Five	13.5

Table 9: Empirical results for damage test cases comparison (Setting 2

Setting 3 - Test case 1			Setting 3 - Test case 2		
Rank	Algorithm	Undamaged area at the end in %	Rank	Algorithm	Undamaged area at the end in %
1	Burn three	62.1	1	Burn three	78.5
2	Burn two	57.9	2	Burn two	77.5
3	Algorithm 2	26.7	3	Three	75.3
4	Algorithm 1	25.5	4	Two	73.4
5	Algorithm 4	25.4	5	Four	72.9
6	Algorithm 3	24.3	6	Algorithm 1	46.7
7	Three	23.4	7	Algorithm 4	46.4
8	Four	21.9	8	Algorithm 2	46.2
9	Two	21.4	9	Algorithm 3	45.6
10	Algorithm 6	19.8	10	One	45.3
11	Burn one	18.9	11	Burn One	44.2
12	Algorithm 5	18.5	12	Algorithm 5	34.5
13	One	17.8	13	Five	34.3
14	Five	16.8	14	Algorithm 6	33.5

Second in both cases we see this negative exponential spreading behavior, where a few algorithms perform very bad and much worse than the other algorithms.

Setting 3 shows a very strong performance of spread-predictive algorithms in comparison to algorithms that don't consider spreading. It also shows a much stronger tendency to the negative exponential spreading effect we have seen in setting 2.

There seems to some kind of required algorithm accuracy so that spread-prediction performs better.

Overall three kind of settings we can see a strong tendency that more accurate algorithms perform better then less accurate algorithms. We don't have enough cases to say something about influence of accuracy in general. But overall there could be some relation that needs further analysis. Spread-prediction is mainly an advantage in the last kind of setting while it isn't really an advantage in the first and second kind of setting.

7 Conclusion and Future Work

After this empirical evaluation we will show the conclusions that can be drawn from this project and show possible future work that could be done in this field.

7.1 Conclusion

In the beginning we defined that the project had to main goals analyze the effect of spread-prediction and define better functions for the fire-fighter goal evaluation in RoboCup rescue.

7.1.1 Spread vs. no-spread prediction

The main result of this project is that the prediction of spread can be possible and have possible effects especially when it is able to prevent an exponential growth of the number of goals (for example fires like we have seen it in the results of 6.3). The prediction of spread should be considered in these domains. The environment has to fulfill the three criteria difference in goal evaluation, a spreading environment and limited resources to make spread-prediction better then usual predictive functions. In section 6.3 we saw that spread-prediction doesn't always show a better performance if these criteria don't exist in the environment. Our hypothesis that spread-prediction performs better then non-spread prediction is therefore just true if these conditions are fulfilled. The reason is that these conditions can cause exponential behavior and make it possible for spread-prediction algorithms to detect them. Spread-prediction algorithms are therefore capable to prevent this exponential growth better then other algorithms.

Beside this general result that spread-prediction can have advantages the issue of required accuracy is also important. In section 6.3 we saw that the algorithms Burn three and Burn two have been the most accurate of them and showed the best performance. They have been the only algorithms that have been able to stop this exponential effect twice in the second and the first case. But also less accurate algorithms like Three Two and Four have been able to do this in at least one case. There probably isn't a strict limit what kind of accuracy is required in the predictive algorithms. It always depends on the actual situation. But there definitely is a relation between more challenging environment that fulfills our requirements for spread-prediction and more accuracy required to prevent exponential spreading and disastrous behavior. Further theoretical and empirical work is required to get a better understanding of this relation.

Interesting is to notice that the random algorithm Four that mainly predicts the workload still can show a great performance. This means that the actual accuracy in workload prediction has been the main effect to get a better performance.

The completely random and therefore really bad algorithm Five showed the overall worst performance, therefore at least some accuracy in the spread-prediction is required.

We addition we have seen that the computation for spread prediction has two closely related problem. First is limited, it can't be done till forever because the number of possible cases and especially potential goals can grow very fast too. Second the analysis of the effect that two goals can have on spreading to another goal showed that this prediction can get less accurate when just the spreading of one goal is considered. Both problems make goal specific computation harder and less accurate. But although this problems exist spread prediction can still deliver good performance.

Overall spread prediction should be considered in applications, where one goal can cause other goals, to potentially improve the agents performance. To check if to use them or not the criteria to the environment are given in section 3. But as we saw it is important to make sure that the spread-prediction algorithms used are accurate enough, while the specific level of accuracy probably depends on the actual application

All these results are limited to the spread-prediction that focusses on workload prediction. The advantage of spread-prediction that are based on earlier recognition of goals are just briefly considered in this project.

7.1.2 Results regarding RoboCup rescue

In addition to this spread-predictive results that should be considered in developing algorithms in spreading environments there are some additional results to consider for RoboCup rescue.

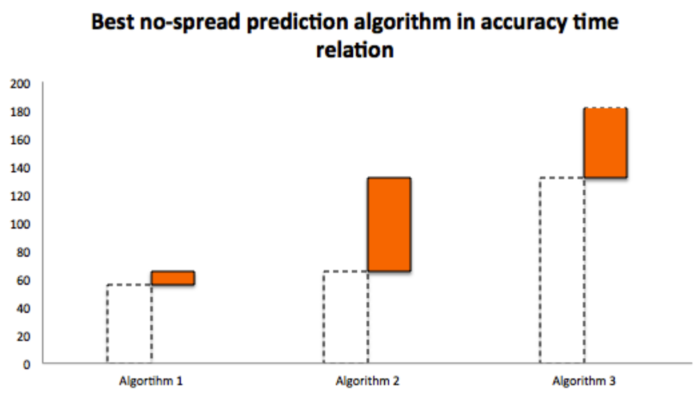
My approach for defining a good function for RoboCup rescue was mainly focussed on using the physical knowledge included in the fire-simulator and try to define good functions that are able to work with these functions. As we could see in our empirical results of 6.1 and 6.3. There is a close connection between the accuracy of an algorithm and it's overall performance. Smarter and better evaluation of goals (or fires) improve the overall performance of the system. We therefore recommend using algorithms that are as accurate as possible and in the best case also include spread-predictive algorithms in RoboCup rescue because they show a better performance. But potential better results from techniques in machine learning or improvement of the described algorithms are definitely possible and a direction of future work can be found in section 7.2.

In RoboCup rescue it sometimes very critical that the algorithms to evaluate goals perform fast enough. The importance of this critical time performance is already included in the RoboCup rescue competition for example [14]. The exact evaluation depends on a lot of different factors and possible adjustments of the existing algorithms are possible (see future work section 7.2). For the algorithms we evaluated figure 3 for no-spread prediction and figure 4 for spread-prediction give a good overview on the algorithm performance in relation to the required time. It could be used as a guidance on which type of algorithm to use in which situation.

7.2 Future Work

There are various fields for possible future work. They can be divided in two basic directions. The first one is the development of better algorithms for the fire simulator and improvements of the algorithms developed in this paper. The second possible direction is in the direction of spread prediction. For both types we listed possible developments that we think could be done in the future.

Figure 3: Accuracy-time relation for no-spread prediction



Algorithm	Required available time till when it performs best
Algorithm 1	55.8
Algorithm 2	65
Algorithm 3	131.5

For the first type the following improvements are:

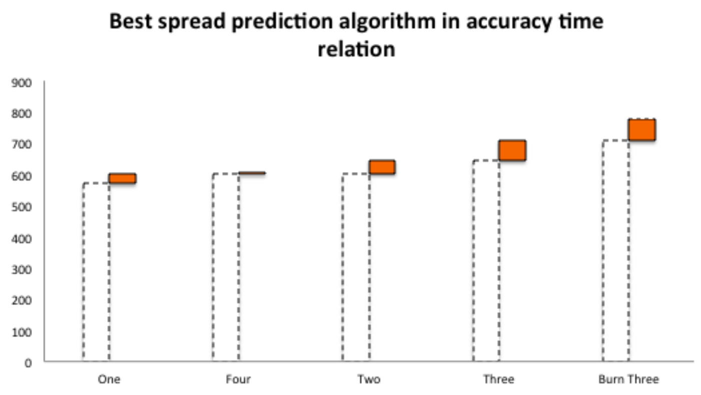
- Include a more sophisticated cost function, that is including more effects and considering other influences especially for a more advanced interaction with the OAA architecture.

- Test the effects of more complex computation. In this project we mainly wanted to show the positive effects that prediction can have and give some insights in the fire simulator or RoboCup rescue in the future more empirical work could be done to find better algorithms that include more advanced calculations and other effects without increasing the computational time to much. Possible things are to include some constant increasing or decreasing factors where we know our simulator gets inaccurate, to try to include more accurate burning of other buildings or to estimate the exchange between the buildings. This requires a lot of different experiments with various functions and could generate some further progress in the accuracy of the predictions.

- Try different prediction approaches like learning or statistical models with the same input that possible have faster computational time and see if they can compete with the algorithm developed in this paper.

- Make the calculations for the algorithms more accurate, so that could perform better in a RooCup rescue competition. For example experiments

Figure 4: accuracy-time relation for spread prediction



Algorithm	Required available time till when it performs best
One	570.1
Four	600.5
Two	602.1
Three	643.6
Burn three	707.7

with algorithm 3 and 4 could

- Include more simulators to see how prediction and especially spread prediction can perform in more advance environments.
- Include the possible positive effects of considering buildings that are heating up as goals or already consider buildings where we know that the current plan will not successfully fight them. That would make the reacton faster and improve the overall performance of the fire fighters. (Using the second possitive effect of spread-prediction).

For the second type of possible future work that range is much more broader and various topics are possible.

- Further theoretical work in the directions that are already suggested. So far the analysis is focussed on one simple example in the future more sophisticated theoretical analysis could be done to analyze when the spreading has catastrophic exponential effects and when spreading stays locally because the spread is converging to a smaller local area. The field of network science and some of it's mathematical results can also be very helpful for this analysis.
- Application in different environments and in other applications. This would help showing that spread prediction has advantages in various domains. A few possible domains have already been suggested. One application that seems interesting in combination of further empirical work is the application in an simple POMDP . It would make it possible to analysze the basic effects on a simpler level without many environmental influences.
- Include spread learning, where the agents try to learn when and where fires are going to sprad to another building for example. This would help to show that spread prediction can be applied in other broader fields and is also interesting for learning.

Acknolegement

This material is based in part upon work supported by the National Science Foundation under Award No. IIS-0964590. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Attachement 1

Figure 5: Algorithm accuracy for fuel (50 cases)

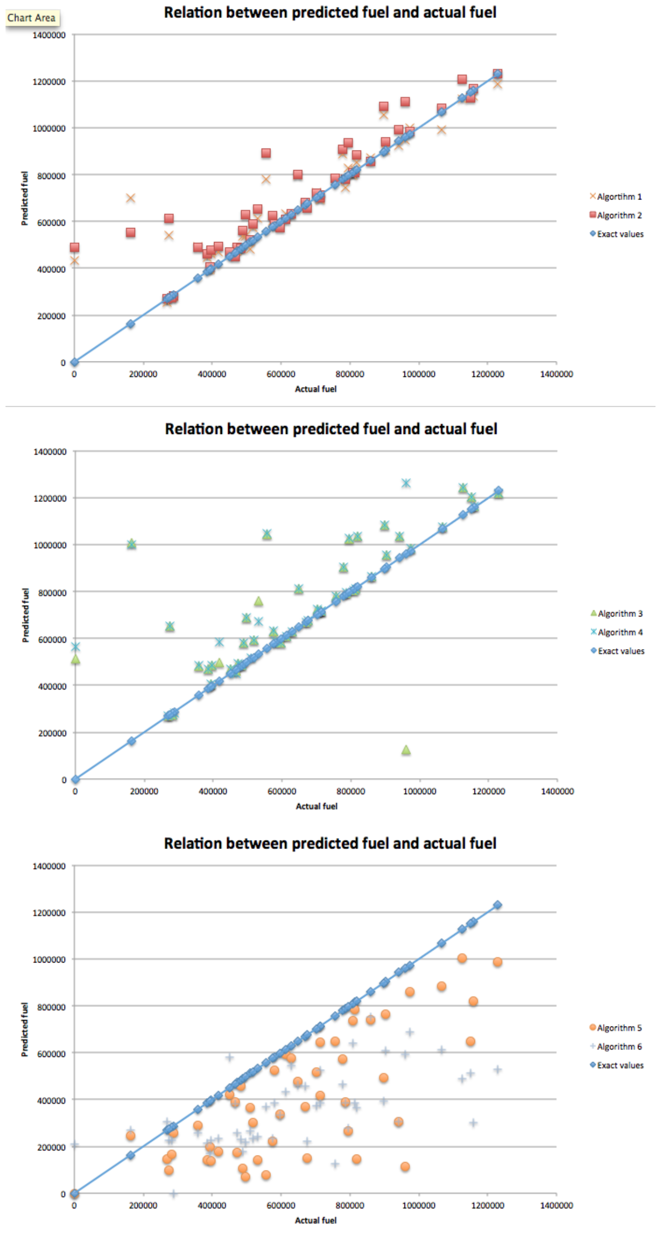
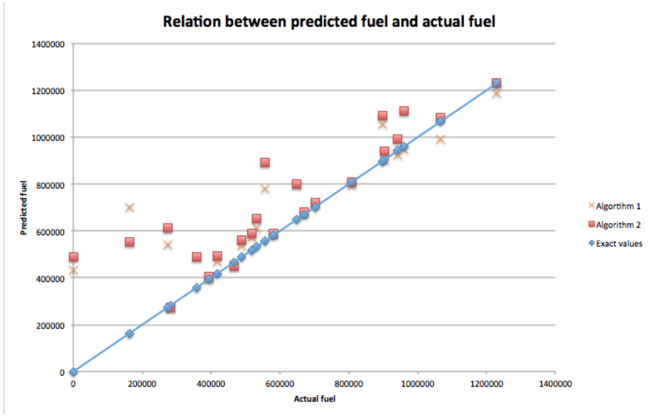
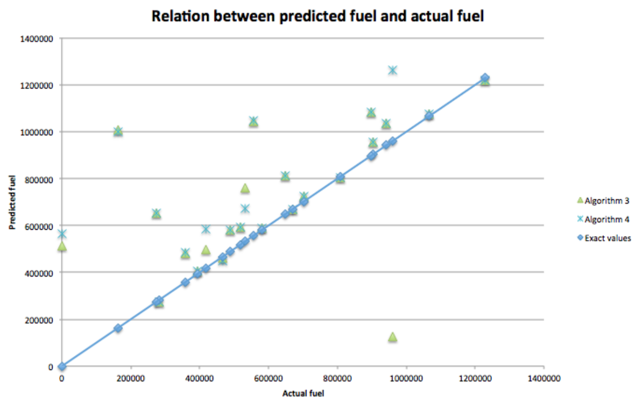


Figure 6: Algorithm accuracy for fuel just cases with neighbors burning (50 cases)





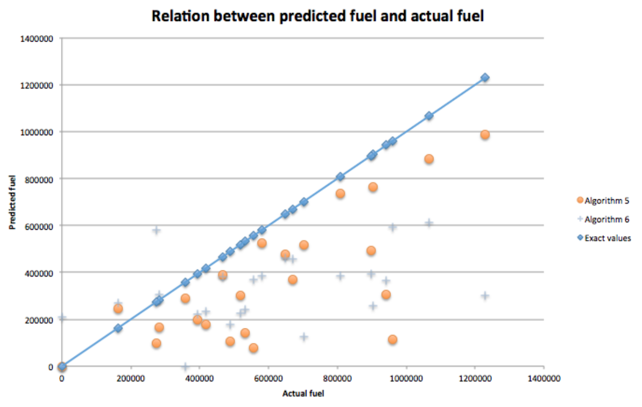
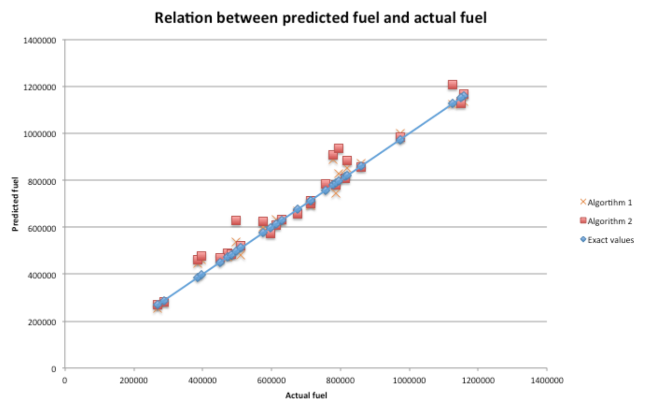
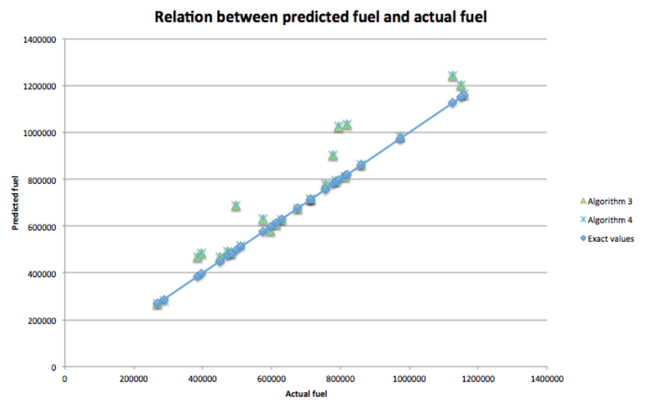


Figure 7: Algorithm accuracy for fuel just cases with no neighbors burning (50 cases)





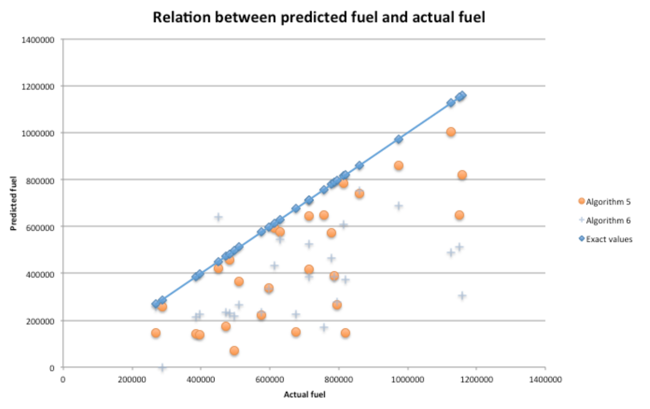


Figure 8: Algorithm accuracy for time (50 cases)

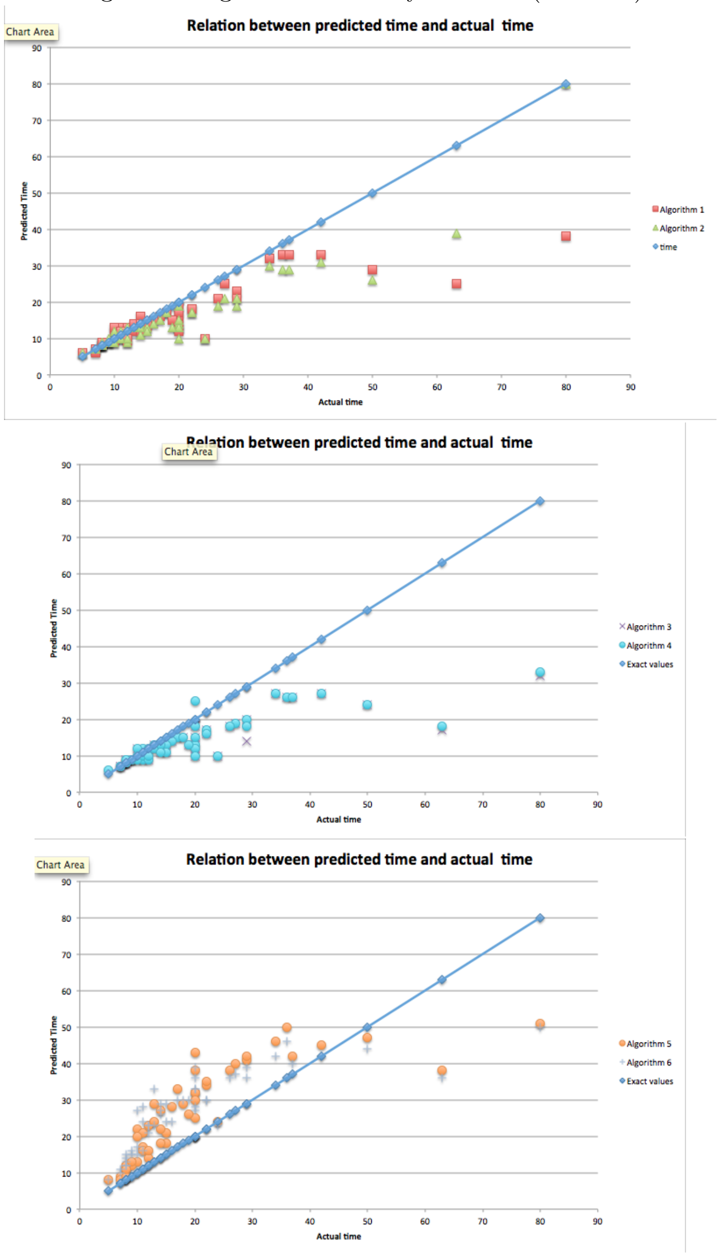
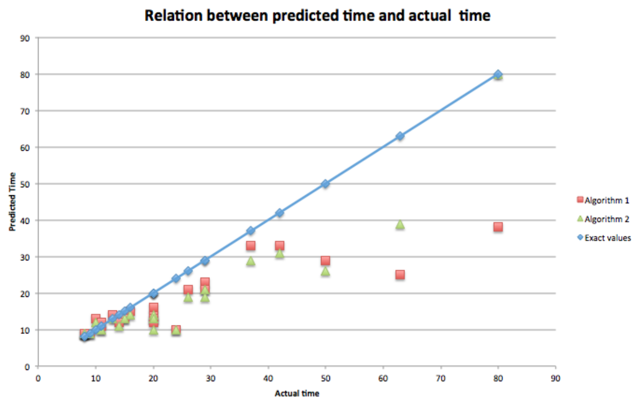
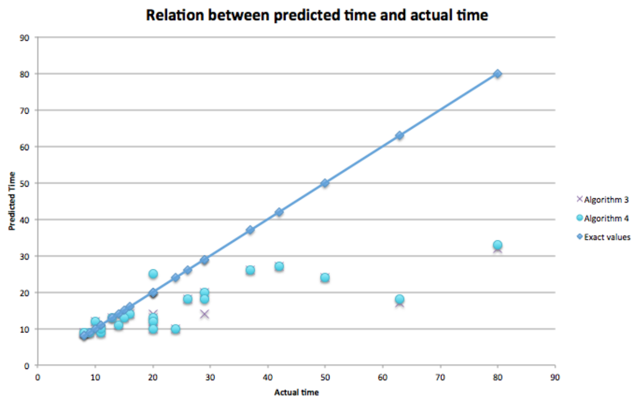


Figure 9: Algorithm accuracy for time just cases with neighbors burning (50 cases)





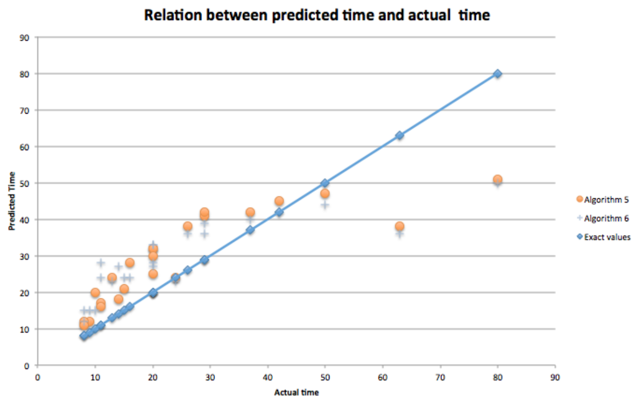


Figure 10: Algorithm accuracy for time just cases with no neighbors burning (50 cases)

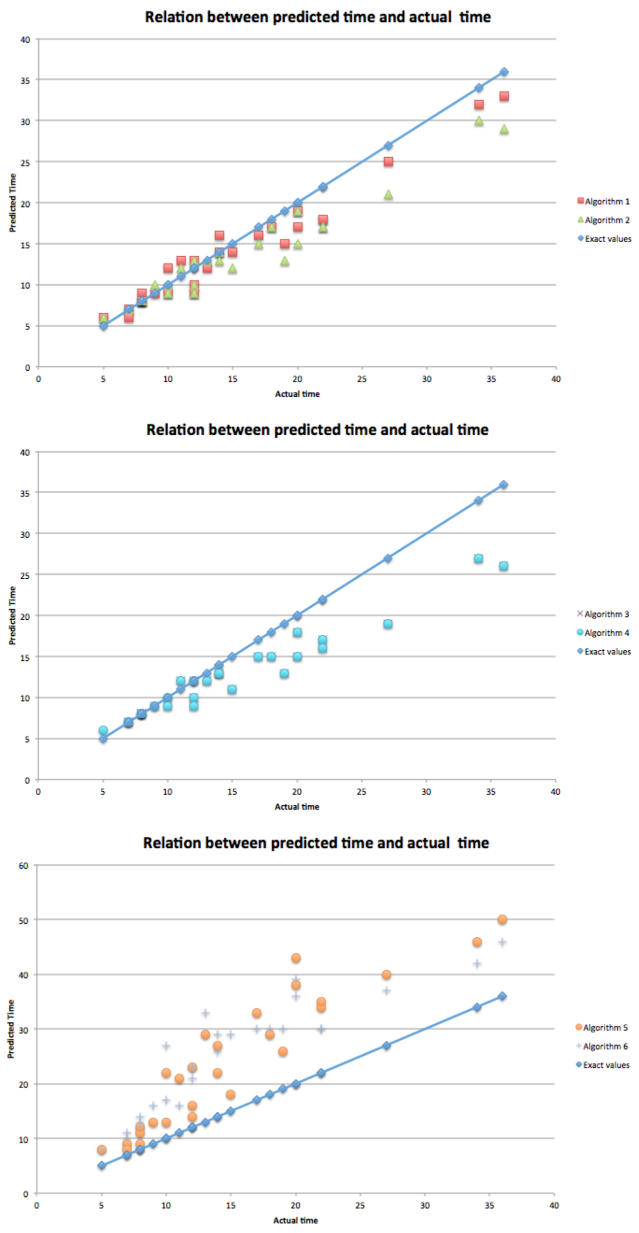
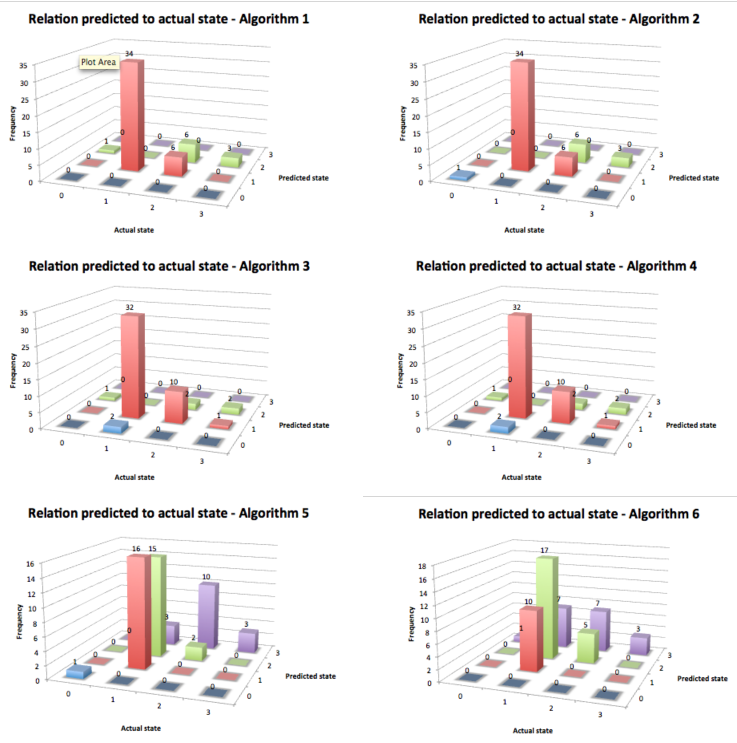


Figure 11: Algorithm accuracy for state (50 cases)



Attachement 2

Figure 12: Algorithm accuracy for fuel (100 cases)

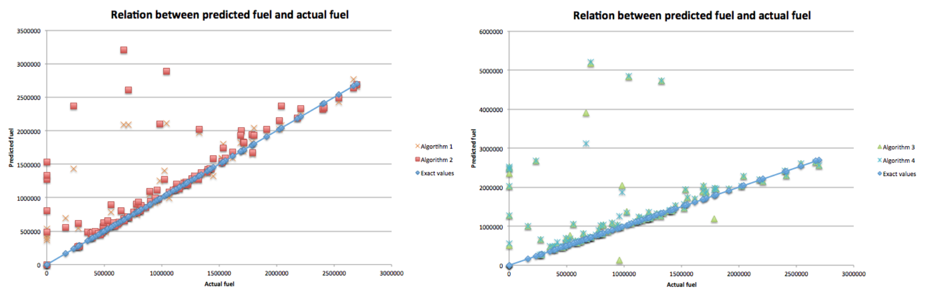


Figure 13: Algorithm accuracy for fuel just cases with neighbors burning (100 cases)

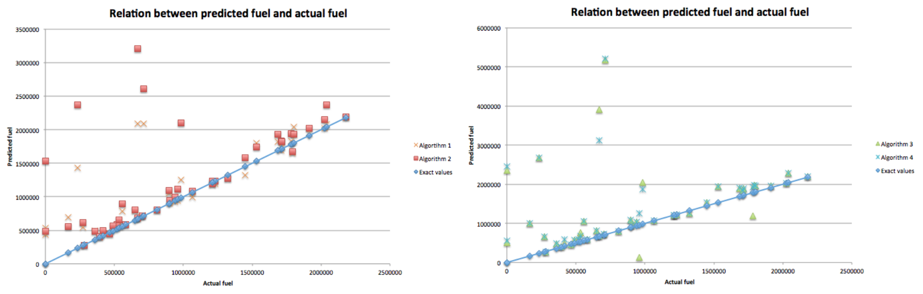


Figure 14: Algorithm accuracy for fuel just cases with no neihgbours burning (100 cases)

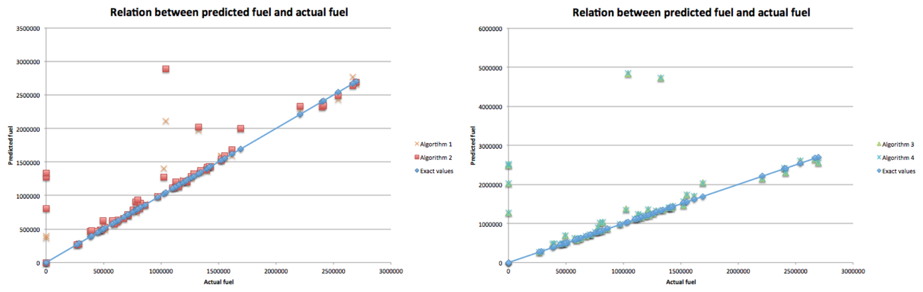


Figure 15: Algorithm accuracy for time (100 cases)

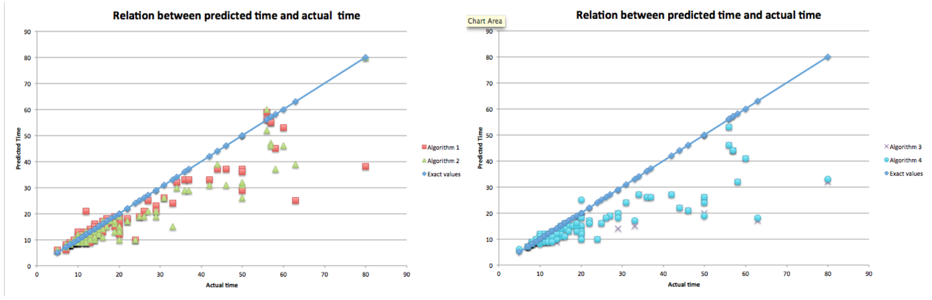


Figure 16: Algorithm accuracy for time just cases with neighours burning (100 cases)

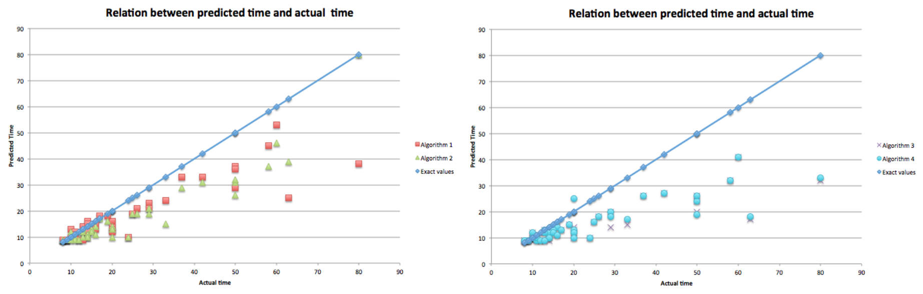


Figure 17: Algorthm accuracy for time just cases with no neighbors burning (100 cases)

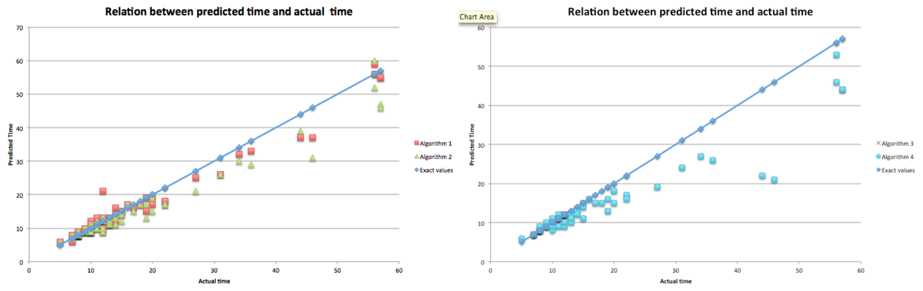
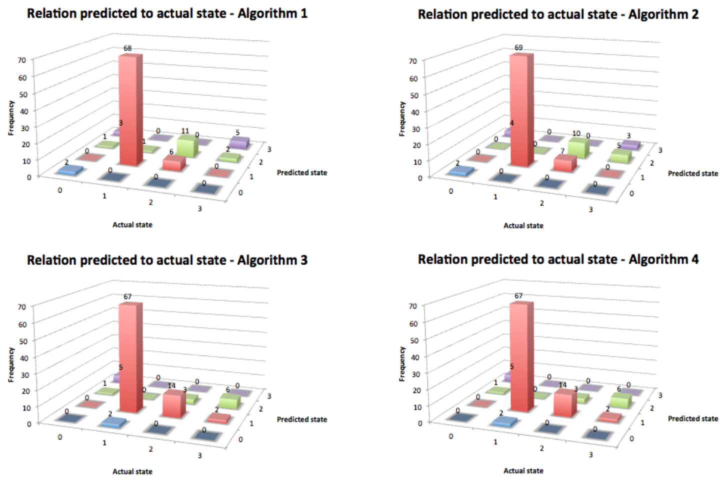


Figure 18: Algorithm accuracy for state (100 cases)



Attachement 3

Figure 19: Relation between predicted and actual number of starting buildings for one, two, three and four

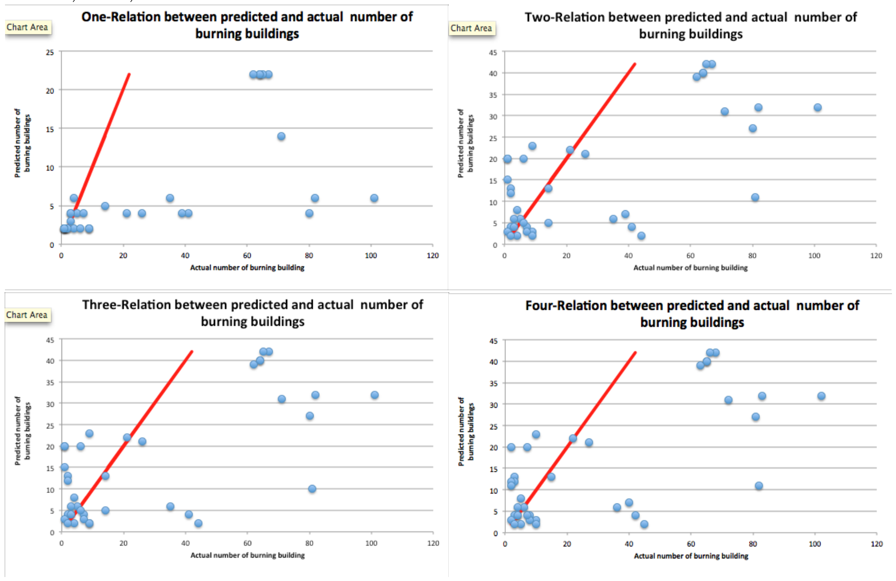
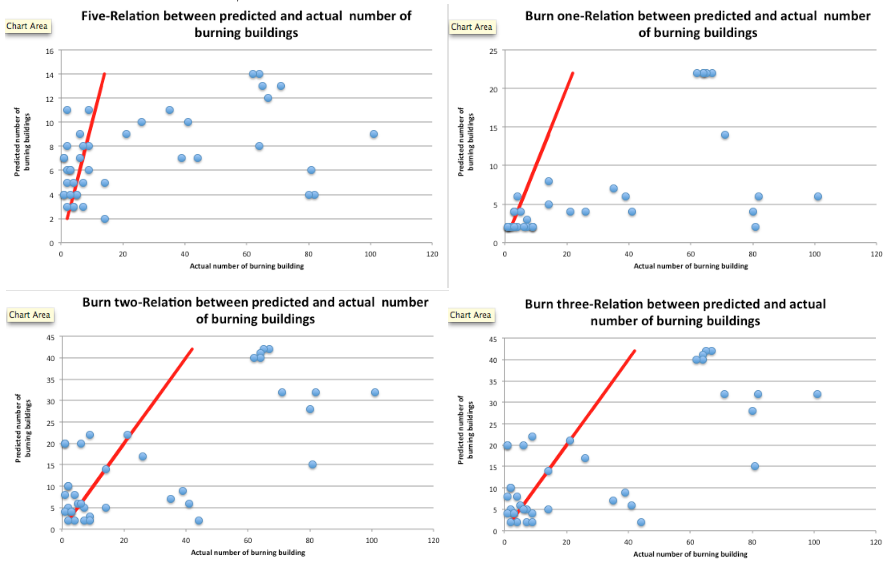


Figure 20: Relation between predicted and actual number of starting buildings for five and burn one, two and three



Attachement 4

Figure 21: Relation between predicted and actual ignition time for algorithm one, two, three and four

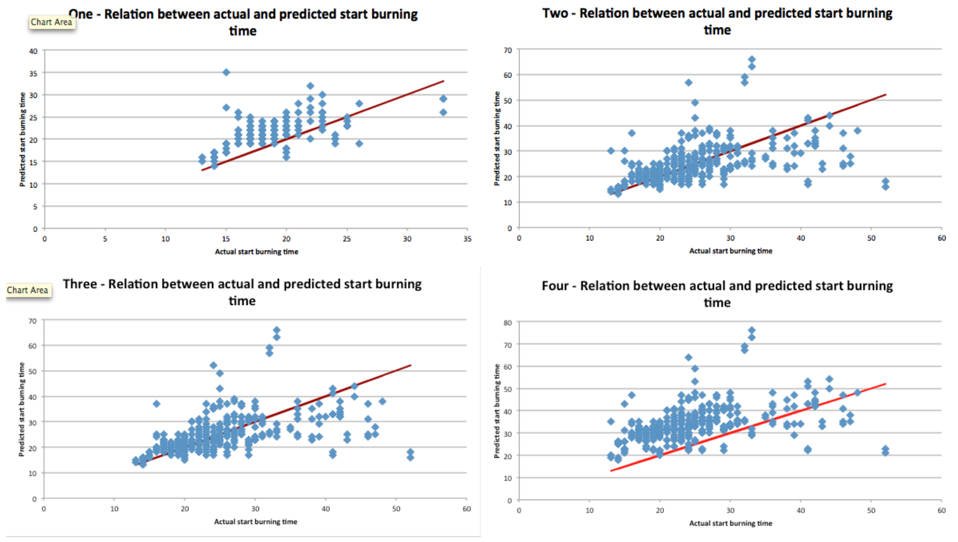
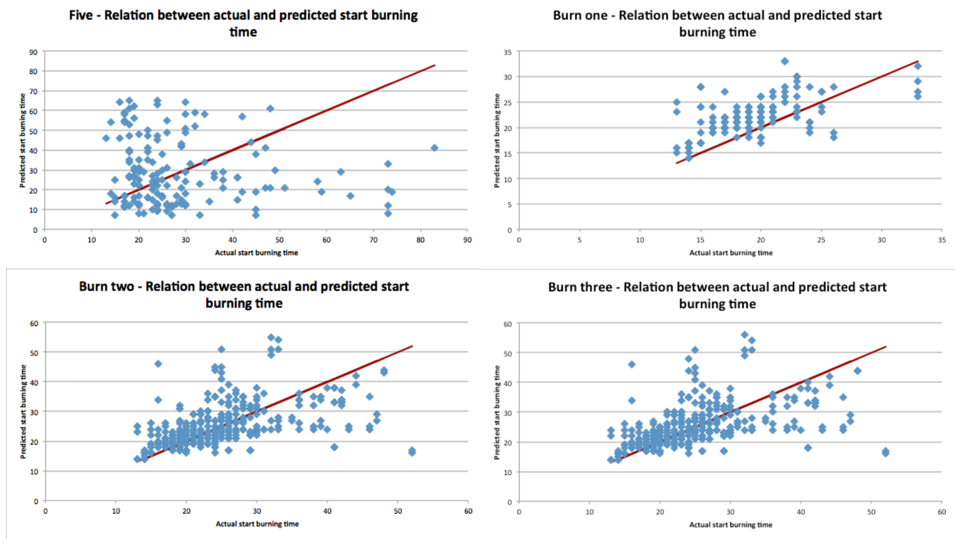


Figure 22: Relation between predicted and actual ignition time for algorithm five and burn one, two and three



References

[1] H. Kitano, S. Tadokor. RoboCup Rescue - A Grand Challenge for Multiagent and Intelligent System. AI Magazine, Vol. 22, No. 1, p. 39-52 , 2001

[2] F. Bousqueta, C. Le Page. Multi-agent simulations and ecosystem management: a review. Ecological Modelling, Vol. 176, Issues 3-4, p. 313-332, September, 2004

[3] L. Beaudoin. Goal processing in autonomous agents. PhD Thesis, School of Computer Science, University of Birmingha,, August, 1994

[4] Daniel D. Corkill, Edmund H. Durfee, Victor R. Lesser, Huzaifa Zafar, Chongjie Zhang. Organizationally Adept Agents. COINS2011 Workshop at Tenth International Conference on Autonomous Agents and Multiagent Systems (AAMAS-11), Taipei, Taiwan, May 2011

[5] H. Van Dyke Parunak et al. Real-time agent characterization and prediction. AAMAS '07 Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems, No. 8, Article 278, Honolulu, Hawaii, 2007

[6] H. Van Dyke Parunak et al. Characterizing and Predicting Agents via Multi-Agent Evolution. US Patent, Application number: 13/079,766, Publication number: US 2011/0178978, A1 Filing date: Apr 4, 2011

[7] Andy Song , Lin Padgham , Lawrence Cavendon. Prediction in Dynamic Environment:Robocup Rescue Exploration. Adaptive Learning Agents Workshop 2007 in AAMAS07, p. 22-27, 2007

[8] Frans Oliehoek, Arnoud Visser, A hierarchical model for decentralized fighting of large scale urban fires. AAMAS'06 Workshop on Hierarchical Autonomous Agents and Multi-Agent Systems, p. 14-21, May, 2006

[9] Timo A. Nüssle, A. Kleiner, M. Brenne. Approaching Urban Disaster Reality: The ResQ Firesimulator. Proceedings of the International RoboCup Symposium '04, 2004

[10] M. E. J. Newman. Networks: An Introduction. Oxford University Press, chapter 12-17, March 2010

[11] Kenah E., Robins J.M. Second look at the spread of epidemics on networks. Physical review, Vol. 76, Issue 3, p. 1-12, September, 2007

[12] Robocup Rescue Simulation Project. `http://sourceforge.net/projects/roborescue/`. state: 12/01/2011

[13] Tadokoro, S. et al. The RoboCup-Rescue project: a robotic approach to the disaster mitigation problem. Vol. 4, p. 4089-4094, San Francisco, CA, USA, August, 2000

[14] RoboCup Rescue Simulation League Agent Competition 2010 Rules and Setup (preliminary). February, 2010

[15] Masayuki Ohta, Tomoichi Takahashi, Hiroaki Kitano. Robocup-Rescue Simulation: in case of Fire Fighting Planning. In Proceedings of RoboCup, Volume 2019/2001, p. 351-356, 2000

[16] Ronald G. Rehm, Anthony Hamins, Howard R. Baum, Kevin B. McGrattan, David D. Evans. Community-scale fire spread. In Proceedings of the California's 2001 Wildfire Conference: 10 Years After the 1991 East Bay Fire, July, 2002

[17] Stephen G. Berjal, John W. Hearne. An improved cellular automaton model for simulating fire in a spatially heterogeneous savanna system. Ecological Modelling, Vol. 148, No. 2, p. 133-151(19), February, 2002,

[18] Keisuke Himoto and Takeyoshi Tanaka. Preliminary Model for Urban Fire Spread: Building Fire Behavior Under the Influence of External Heat and Wind. In Fifteenth Meeting Of The UNJR Panel On fire research and safety, Vol. 2, p. 309-319, San Antonio, TX, November, 2000

[19] Duncan S. Callaway, M. E. J. Newman, Steven H. Strogatz, Duncan J. Watts. Network Robustness and Fragility: Percolation on Random Graphs. Vol. 85, p. 5468–5471, Issue 25, December, 2001

[20] Bollobás, Béla. The Evolution of Random Graphs—The Giant Component. Cambridge University Press, Cambridge studies in advanced mathematics, 73 (2nd ed.), pp. 130–159, 2001

[21] Mini Ghosh, Peeyush Chandra, Prawal Sinha, J.B. Shukla. Modelling the spread of bacterial disease: effect of service providers from an environmentally degraded region. Applied Mathematics and Computation, Vol. 160, p. 615–647, 2005

[22] Jeger M.J., Pautasso M., Holdenrieder O., Shaw M. W. . Modelling disease spread and control in networks: implications for plant science. The New Phytologist Vol. 174, No. 2, p-279-297, January, 2007

[23] Alberto Talamo, Yousry Goha. Deterministic and Monte Carlo Modeling and Analyses of Yalina-Thermal Subcritical Assembly. Nuclear Engineering Division, Technical Report, Published at http://www.osti.gov/bridge/, July, 2010

[24] A. Farinelli, G. Grisetti, L. Iocchi, S. Lo Cascio, D. Nardi. RoboCup Rescue simulation: Methodologies tools and evaluation for practical applications. Robocup 2003 Robot Soccer World Cup Vii, Vol. 3020, p. 645-653, 2004

[25] Peter Haddawy, Steve Hanks. Utility Models for Goal-Directed, Decision-Theoretic Planners. Computational Intelligence, Vol 14(3), p 392–429, August 1998

[26] Xin Li, Leen-Kiat Soh. Applications of Decision and Utility Theory in Multi-Agent System. University of Nebraska–Lincoln, Computer Science and Engineering Technical Report TR-UNL-CSE-2004-0014, Issued September 2004

[27] J. R. Marden, A. Wierman."Overcoming the Limitations of Utility Design for Multiagent Systems" submitted for journal publication, 2011.

[28] Lon Padgham, Michael Winikoff. Developing Intelligent Agent systems. John Wiley & Sons, ISBN 0-470-86120-7, chapter 2 & 9, 2004

[29] Michael H. Bowling and Rune M. Jensen and Manuela M. Veloso. Multiagent Planning in the Presence of Multiple Goals. Booktitle: Planning in Intelligent Systems: Aspects, Motivations and Methods, John Wiley & Sons, Inc., 2005.

[30] Chongjie Zhang et al. Adapting the Behavior of Organizationally Adept Agents by Using Annotated Guidelines. submitted for publication, November, 2011