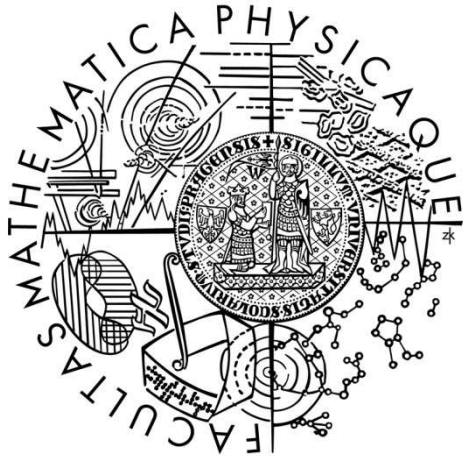
Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Martin Černý

Comparing reactive techniques to classical planning for intelligent virtual agents

Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Jakub Gemrot

Study programme: Computer science

Specialization: Non-procedural Programming and Artificial Intelligence

Prague 2012

I would like to thank all the members of the AMIS research group for excellent collaboration and friendly attitude, namely to supervisor of this work, Jakub Gemrot and to the leader of the group, Cyril Brom for continuous encouragement and helpful criticism.

Great thanks belong to my parents, who showed unending support and understanding during the whole period of my studies.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In ..... date ..... signature

Název práce: Srovnání reaktivních technik vůči klasickému plánování pro účely inteligentních virtuálních agentů

Autor: Martin Černý

Katedra / Ústav: Kabinet software a výuky informatiky

Vedoucí diplomové práce: Mgr. Jakub Gemrot

Abstrakt:

Mnoho současných počítačových her lze popsat jako dynamické simulace běžící v reálném čase obývané autonomními inteligentními virtuálními agenty (IVA), kde většina struktury prostředí je neměnná a jednou z nejběžnějších činností je pohyb. Ačkoliv se techniky plánování zdají být ideálním přístupem pro ovládání takovýchto agentů, plánování je v současných hrách používáno jen zřídka. Tato práce se snaží zodpovědět, zda současné akademické techniky plánování jsou připraveny pro použití v existujících hrách a za jakých podmínek. Práce porovnává reaktivní techniky s klasickým plánováním pro řešení problému „co udělat teď“ u IVA v herním prostředí. Několik hotových klasických plánovačů, které obsadily přední příčky v International Planning Competition bylo připojeno k virtuálnímu prostředí Unreal Development Kit pomocí knihovny Pogamut. Výkonnost IVA využívajících tyto plánovače a IVA s reaktivní architekturou byla změřena na třídě zkušebních prostředí vycházejících z počítačových her pod různou úrovní externích zásahů. Ukázalo se, že agenti využívající klasické plánování překonají reaktivní agenty pouze když velikost plánovacího problému je malá, nebo když změny v prostředí jsou buď nepřátelské k agentu nebo nepříliš časté.

Klíčová slova: inteligentní virtuální agenti, specifikace chování, klasické plánování, reaktivní techniky

Title: Comparing reactive techniques to classical planning for intelligent virtual agents

Author: Martin Černý

Department: Department of Software and Computer Science Education

Supervisor of the master thesis: Mgr. Jakub Gemrot

Abstract:

Many contemporary computer games can be described as dynamic real-time simulations inhabited by autonomous intelligent virtual agents (IVAs) where most of the environment structure is immutable and navigating through the environment is one of the most common activities. Though controlling the behaviour of such agents seems perfectly suited for action planning techniques, planning is not widely adopted in existing games. This paper attempts to answer the question whether the current academic planning technology is ready for integration to existing games and under which conditions. The paper compares reactive techniques to classical planning in handling the action selection problem for IVAs in game-like environments. Several existing classical planners that occupied top positions in the International Planning Competition were connected to the virtual environment of Unreal Development Kit via the Pogamut platform. Performance of IVAs employing those planners and IVAs with reactive architecture was measured on a class of game like test environments under different levels of external interference. It was shown that agents employing classical planning outperform reactive agents only if the size of the planning problem is small or if the environment changes are either hostile to the agent or not very frequent.

Keywords: intelligent virtual agents, behvaior specification comparism, classical planning, reactive techniques

Contents

1 Introduction

Substantial amount of artificial intelligence research is focused on the design of *intelligent agents*. Intelligent agent is an autonomous entity, which observes through sensors and acts upon an environment using actuators and directs its activity towards achieving its goals. [1] In this thesis only software agents inhabiting a virtual environment are considered. Such agents are usually referred to as *intelligent virtual agents* (IVAs). Computer-controlled IVAs have achieved reasonable success in environments that either do not change at all (e. g. sudoku solving) or change synchronously to agent actions and in a limited number of ways (e. g. chess).

In order to get closer to real world applications, IVAs need to act in *dynamic* and *continuous* environments. A dynamic environment may change at any time independently of agent actions. A continuous environment cannot be described by a set of discrete variables, but needs real-valued variables to capture its state. In most cases, this also means that there are infinitely many options for the agent, since its actions may take real-valued parameters. Presently, computer games – especially first person role-playing (RPG) and shooters (FPS) – provide problem domains that have many attributes of the real world, while maintaining a reasonable level of abstraction. This allows the developer of an IVA to focus on the high-level intelligence and elude lower-level problems such as processing sensory input. Thus computer game environments are suitable for IVA research.

One of the fundamental problems faced by an IVA is the action selection problem – what to do next? Since computer game environments have many properties of the real world, choosing the correct action shows to be very difficult even when the desired behaviour is very simple (from human point of view). It is still not uncommon that computer controlled agents in games have trouble with simple navigation through the environment. In dynamic, continuous environments, humans still prevail over computers in many tasks.

Because of this inherent difficulty, IVAs in computer games in the most cases exhibit only quite simple behaviour and are controlled by some kind of reactive technique, the most common being behaviour trees [2] and finite state machines [3]. Although those techniques handle the dynamic aspects of the world well, they have many limitations: their plans are fixed and cannot be altered during runtime and they require large amount of authoring work as the world gets more complex. There is however a complementary approach to solve the action selection problem – AI planning, which has a history of over 40 years of academic research. In this

thesis we focus on the longest studied approach – classical planning as solved by STRIPS. [4]

In classical planning the world is described as a set of predicates that may or may not hold in a particular state of the world and actions that change the state of the world. [1] Each action has a set of *positive preconditions* (predicates that must hold true) and a set of *negative preconditions* (predicates that must hold false) for the action to be applicable. The action has a set of *positive effects* (predicates becoming true after the action is executed) and *negative effects* (predicates becoming false after the action is executed). A *planner* finds a set of actions that the agent should execute to reach one of desired goal states from a specified initial state.

Ideally, classical planning would allow the agent to behave rationally even in very complex worlds and ease the work of the designer at the same time. The designer only needs to state the agent goals and how the world changes with agent actions and does not need to foresee every possible situation and provide a ready-made solution. The effort needed to create reactive plans grows rapidly with growing environment complexity. In fact, a complete reactive description of a complex behaviour must have exponential size unless the polynomial hierarchy of complexity classes collapses. [5]

Unfortunately, the gap between game AI and planning communities is still huge and only a few attempts were made to employ classical planning for controlling IVAs in dynamic environments. There are also numerous issues to be addressed for successful application of planning in complex domains: planning is intractable[1] in many cases, and the formulation of planning problem implicitly relies on the world being discrete and static[2]. More discussion on difficulties of planning in complex environments may be found in [6].

On the other hand, problems in game domains might be efficiently solvable by classical planning, given the recent advent in both computing power and planning techniques. Unfortunately only few attempts were made to employ classical planning for controlling IVAs in dynamic environments. The only published paper on planning implementation in a commercial game is the work of Orkin on F.E.A.R. [7] [8] that date back to 2004-2006. The planning system from F.E.A.R. called GOAP was reportedly used in other games [9] and it is likely that some other planning systems for games were created. However, no papers were published yet. There is also lack

[1] PSPACE-complete for both deciding plan existence and finding optimal plan [51]

[2] There are planners that are not limited to STRIPS-style planning that accommodate for dynamic worlds. They are however not considered in this thesis.

of performance comparison of planning algorithms and reactive agent architectures.

Since classical planning is computationally expensive, simplifications of the planning problem were proposed. One such simplification is to force actions to be irreversible. Irreversible actions may always be modelled as actions without negative effects[3]. If this is the case, the planning problem becomes polynomial[4]. [10] Planning domains without actions with negative effects will be referred to as *relaxed domains*. Since such domains do actually correspond to problems present in computer games, they are considered in the thesis as a special case. One example of such problem is finding a plan to complete a set of quests that depend on each other in a role-playing game – completing a quest is usually an irreversible action.

1. 1 Thesis Goals

First goal of the thesis is to connect several existing classical planners and planners for relaxed domains to Pogamut [11], a platform for development of IVAs for virtual environment of Unreal Engine [12]. Second goal is to design a suitable class of game-like test environments that would allow for comparison of planners' performance in solving tasks of varying difficulty under different levels of external interference. Subclass of the environments will be represented by relaxed domains. The thesis will then analyse the performance of IVAs employing classical planners, IVAs with reactive architecture and IVAs that employ planners for relaxed domains (when applicable) in those environments.

There are several important questions that should be answered by the analysis:

1. How big problem instances can contemporary planners solve fast enough for real-time decision making?
2. Under which interference conditions do planners perform better than reactive agents?
3. Is the relative performance of planners to reactive techniques better when the interference is friendly or hostile?
4. Are planner-controlled agents more vulnerable to frequent small changes or infrequent large changes of the world state?
5. Do planners for relaxed domains bring benefit over classical planners and/or reactive techniques for solving problems in real-time?

[3] Possible negative effects (and negative preconditions) of actions are modelled as positive effects (and positive preconditions) by creating new atoms prefixed with "not_".

[4] PTIME to find satisficing (satisfying & sufficient) solution, finding optimal plan is NP-hard

1. 2 Related Work

As noted in the introduction, the application of classical planning techniques to dynamic real-time domains is not well studied. Still there are several works that are related to this area. Those can be broadly categorized in three groups: 1) those that employ STRIPS-style planning, 2) those that work with hierarchical task networks (HTN) planning and 3) other works of interest in context of planning in dynamic worlds or using planning techniques in computer games and simulations that do not belong to either of the two categories.

1. 2. 1 STRIPS-style Planning in Real-Time Applications

The only planning system that is known to be implemented in a commercial game is GOAP by Jeff Orkin [7]. In general it is based on STRIPS formalism [1], but instead of Boolean predicates it employs discrete state variables that can take multiple possible values. The formalism is further enhanced by action costs and procedural preconditions (preconditions that are checked by running a procedure). The plan is found using the A* search algorithm [1].

An agent running GOAP has a set of goals with associated priorities. The priorities may change dynamically in response to changes in the world state (e. g. if the agent was hurt, it assigns higher priority to a healing goal). The agent always plans for the goal with highest priority. Whenever the currently executed plan is found invalid (it no longer reaches the goal state from the current state of the environment) or a different goal gains higher priority, the agent replans.

The commercial success of F.E.A.R. (the first game to implement GOAP) and positive reviews of the game AI prove that this kind of technique is valuable to AI design in games. There are also a few scientific works on GOAP by other authors than Orkin: D. Pittman implemented GOAP for Unreal Tournament environment in his master thesis [13]. While several scenarios are presented that show the advantages of GOAP, no actual performance evaluation was presented. E. Long ran a set of matches between bots controlled with reactive architecture (finite state machines) and bots controlled with GOAP in Unreal Tournament. [14] The work concludes that bots controlled with GOAP win the matches more often, but no fine-grained statistical analysis is done. P. Peikidis implemented GOAP for real-time strategy game Starcraft, but provided no exact evaluation [15].

Vassos and Papakonstantinou test BlackBox [16] and Fast Forward [17] planners on a domain representing a FPS game [18]. The results show that the planners are able to plan in sub-second time for reasonably sized problems. The planning component is however not connected to any real simulation.

Thompson and Levine compare performance of an agent employing a classical planner on several runs in static and dynamic versions of the same environment [19]. The paper is however focused on the agent architecture and the performance comparison is very brief.

Another of the few works directly related to STRIPS-style planning in dynamic environmnets is the AI Live system by Fernández et. al. [20]. The IPSS planner [21] is connected to a game scenario similar to the Sims game, but no performance analysis is provided.

Much earlier on, Armano et. al. proposed a layered agent architecture for computer games. [22] Layers are ordered from the low-level ones to the most abstract; the more abstract layers can control and/or inhibit the lower-level ones. Each layer has its own planner instance that operates on a layer-specific domain written in PDDL [23]. However, only simple implementation is mentioned and no evaluation was reported.

1. 2. 2 HTN Planning in Real-Time Applications

In HTN planning, the problem is not to find a series of action that leads from start state to goal state, but rather to find decompositions of abstract actions into less abstract ones until only actions that can be directly executed remain. HTN planning has been argued to provide a great performance advantage through hierarchical decomposition. [24] For this reason it was studied for use in real-time applications.

Hawes [25] presents a planning system based on HTN that is able to control processor time dedicated to planning based on current situation (dangerous situations require fast response even if it is not the optimal one). In order to achieve this it has a mechanism to extract the best solution available at any moment of the planner run. This property is called anytime planning. The system is evaluated in the environment of Unreal Tournament but it is compared only to a non-anytime planning system and not to a reactive architecture.

HTN has been also implemented to control groups of agents. Obst and Boedecker created HTN planner to control a team of robots playing robotic soccer. [26] Their system supports anytime planning and they allow the agents to interleave action execution and planning. However, no performance comparison is given. On a similar note, Gorniak and Davis [27] used HTN for controlling the behaviour of a squad of agents, but the performance is not evaluated and the system offers only limited failure handling.

In [28], Hoang et. al. run a set of „capture the flag“ matches between reactive bots and bots controlled with HTN. They conclude that HTN planning brought relative advantage to the agents.

There are also less obvious applications of planning in game domains. In the MIST interactive storytelling system [29] Paul et. al. take advantage of HTN planning in order to generate plots and dynamically repair them when changes in the world preclude their successful completion. The technique is evaluated in a real-time simulation where up to three agents not controlled by the storytelling engine distract the world state in pursuing their own goals. The planning system is however capable of degrading gracefully with more interference going on and still bring a reasonable number of plots to their climax.

1. 2. 3 Other Applications of Planning in Real-Time Environments

An interesting member of the family of planning approaches is to represent the problem of acting in an uncertain environment as a Markov Decision Process (MDP). Balla and Fern use Monte-Carlo based MDP solver called UTC to control behaviour of a group of units in a real-time strategy game. [30] The goal of the units is to perform a coordinated and effective assault on opponent troops. The technique is evaluated on a set of scenarios and the planner clearly outperforms simple reactive behaviour and is better or comparable to a human player.

Nguyen et. al. present CAPIR [31], a system based on MDP for controlling agent in non-deterministic dynamic environment that is trying to infer human player goals and aid him in achieving them. The MDP represents both the changes in the environment and the uncertain actions of the human user. A test was conducted where human players acted in a simple environment, with an aid from another agent. An agent controlled by CAPIR was compared to an agent controlled by another human. The computer agent was shown to be similar to a human controlled one both in absolute performance and perceived helpfulness.

The works cited so far implement online planning directly during the execution of the game. However, planning may also be performed offline. In [32] Kelly et. al. implement an offline planning system based on HTN that automatically generates reactive plans (scripts) to control non-player characters from abstract description of the game world. The system was tested by creating plans for characters in the The Elder Scrolls IV: Oblivion game.

1. 3 Chapter Summary

The rest of the thesis is organized as follows:

Chapter 2 describes the design of experiments that were performed to answer the questions raised in the goal statement.

Chapter 3 discusses results of preliminary experiments and points out specific hypotheses formulated from the data gathered.

Chapter 4 presents results of the final experiment runs and their statistical analysis.

Chapter 5 summarizes conclusions from experimental results, interprets them and suggests future work.

2 Methodology

This chapter discusses the design of experiments that were carried out in order to compare performance of agents employing planners and agents with reactive architecture and to answer questions poised in section 1. 1. First the focus area of thesis is made clear and general properties of environment dynamics are discussed, then the test environment design is introduced based on this discussion. The chapter continues with description of the agent architecture and metrics chosen for the experiments. The final part of the chapter deals with technical details of the experiment.

2. 1 Choosing Focus

Since the area of planning in dynamic domains is not well studied, it is important to focus on a well-defined problem with limited number of parameters. It is far beyond the scope of this work to consider every possible aspect of planning in computer game environments as there are simply too many. Taking too many options into account would also obscure the final results since it would be hard to determine, which factor is crucial. From the thesis point of view, the most important aspect is the dynamicity itself. Thus dynamic aspect will be highlighted as much as possible, while all the other factors will be either left out completely or kept as simple as possible.

2. 2 Dynamicity and Interference

It is important to design the experiments so that the results are not strongly connected to a particular domain and may be generalized to any agent planning in a dynamic environment. There are however many ways how dynamicity may be achieved. Thus it may be useful to investigate the nature of dynamicity present in computer games.

In most game-like environments the changes are continuous, while planning, as other symbolic AI approaches, is discrete by nature. A natural way to discretize the dynamics is to consider only “important” changes i.e. changes that would affect a chosen discrete representation of the world. On a very abstract level, discrete dynamics may be considered an *interference* to the initially static state of the (symbolic) world. Interference may be broadly categorized with three general parameters:

- *delay* – mean delay between two successive changes;
- *impact* – the scope of the impact of a single change to the state of the environment; and
- *attitude* – whether hostile or friendly changes are dominant. Hostile changes interfere with agent's goals, while friendly changes open new possibilities for the agent to reach its goals.

Table 1 summarizes a few game situations with respect to the above parameters. However keep in mind that such summary necessarily involves a large amount of subjective interpretation and therefore is by no way definitive.

Situation	Delay	Impact	Attitude
FPS shootout	0.5 – 2s	Small	Hostile
Quest in a RPG, no combat	> 5s	Medium	Balanced
Getting food in The Sims (from agent perspective)	1 – 5s	Small	Friendly
Navigating through a spaceship falling apart	1 – 3 s	Large	Hostile

Table 1: Comparison of game situations by their interference profile

It is beneficial if the test environment covers the complete spectrum of interference parameters, because such an environment may be considered as an abstract model of a whole class of games. To provide a fair comparison it is reasonable if the interference is completely random and independent of agent actions.

2. 3 General Environment Considerations

While most of the previous work in this area focused on performing matches between two classes of agents, we let the agents in our work to solve a common problem individually. This allows for independent interference and mitigates the influence of implementation details of the agents on overall result trends. It is also important that the problem is not overly complicated, so that there is not much room for improvement of reactive techniques by fine-tuning of the reactive plans by hand.

To keep the focus area small, several further restrictions are placed on the environment. First restriction is that the world is fully observable, i.e. the agent knows the exact state of the environment at all times. While this is certainly not the case in reality, it is easy to achieve in a virtual world. Furthermore, in an unobservable environment the result of the experiments would greatly depend on the algorithm implemented to handle agent's beliefs about the world. Several

alternative approaches would have to be compared, which is far beyond the scope of this work and open space for future work. Also note that handling environment dynamicity handles partial observability as well, albeit on a very simple level: the agent plans with its beliefs about the world. If the state of the world does not correspond to agent beliefs it will trigger a sudden change in the agent beliefs and such may be treated the same as an external interference.

Second restriction is that the agent actions are deterministic and the world is known. I.e. the agent knows all its available actions, knows how exactly they change the world and the changes are the same every time they are executed. This once again lets us focus only on the dynamic aspect of the world. Similarly to partial observability, non-determinism may be (to a certain extent) handled the same way the dynamicity is handled: action yielding a different result than expected may be treated as if an external change modified the state of the environment just after the action was executed.

The second restriction however does not imply that the agent should not check the outcome of actions it carries out. Since the environment is dynamic and real-time, it is inevitable that actions may actually fail when executed. This will be the case when a conflicting interference happens just after the agent has started an action and before the action was completed. Note that this kind of uncertainty is inherent (imposed by the environment dynamicity) and cannot be removed. On the other hand it can be handled easily on the agent side by simply checking whether the action has yielded the expected result. For this reason, agents will not explicitly take this possibility into account when deliberating.

2. 4 Test Environment

As was already mentioned, it is important to keep the experiments simple so that the results can be generalized. On the other hand, the world should be complex enough to require, or at least reward, non-trivial, goal-oriented action selection mechanism – e. g. it should not be likely to be solved by choosing actions at random. The environment should be able to represent wide range of interference parameters. And since the main areas of application of IVA research are computer games and simulations, the environment should also be inspired by those domains. Taking all the aforementioned requirements into account a simple yet flexible domain was designed, loosely inspired by the 1984 computer game Spy vs. Spy [33].

2. 4. 1 Environment Overview

The environment consists of rooms on a grid that are connected by corridors. In the middle of each corridor, there is a door. On both ends of the corridor, there is a

button. A button may open one or more doors or close one or more doors. It may open and close doors at the same time. The amount of doors changed by one button is not limited, neither is the distance from button to the affected door. A button may also not change any door at all. Initially, all doors are closed. The agent starts at a predefined spot and has a goal location to reach. See Figure 1 for an example scenario in such environment.

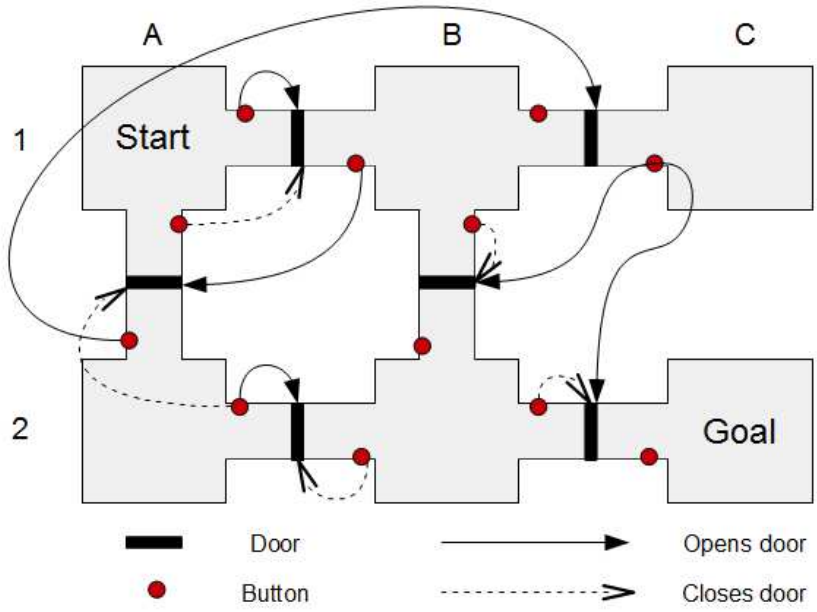


Figure 1: An example map

On the provided example map, the shortest solution to go from A1 to C2 is to:

1. Push the east button at room A1 to open door to B1
2. Go to B1
3. Push the west button at B1 to open door from A1 to A2
4. Go to A2
5. Push north button at A2 to open door from B1 to C1
6. Go to C1
7. Push west button at C1 to open door from B1 to B2 and from B2 to C2
8. Go to C2, which is the goal.

Note that, while this map is very small, it demonstrates that the problem at hand cannot be solved in the most straightforward way. The solution requires the agent twice to go away from the goal. Also there is a dead end: if the agent performs steps 1 – 4 as noted above and then pushes the east button at A2 to get to B2, he traps himself and is no longer able to reach the goal.

Also, the buttons were intentionally not implemented as switches. The reason is very simple: having buttons switch the door state would require the actions in

planning domain to have conditional effects (effects that depend on the state of the world), which are not part of the STRIPS formalism and not all planners directly support them. While it is possible to model them with unconditional actions, it may introduce exponential growth in the size of the problem. Since this is a first experiment of this kind, things were kept simple and switches were omitted.

2. 4. 2 Interference in the Environment

So far, only the static aspects of the environment were addressed. However, the extension to a dynamic world is straightforward. An easy and efficient way to introduce interference is to repeatedly choose a subset of the doors at random and alter their state. To achieve full independence of agent's actions, it is first decided (separately for each door), whether the door should be closed or opened and the door state is set regardless of the previous door state.

The interference parameters are implemented in a straightforward way: the impact is the fraction of the total door count that is affected (on average) by a single interference. The attitude is represented by the *friendliness* parameter, which is the probability that a single door is set to open state when it was chosen for interference.

A daemon agent responsible for the interference is introduced into the environment. To achieve full independence of agent actions, the daemon will decide first, whether the door should be closed or opened and then set the door state, whether it is the same as the actual state of the door or not. To prevent extreme volatility in the interference between different runs if the delay is long, the decision whether to open or close a chosen door is made for each door separately and not for all chosen doors together. This way the interference behaviour will change more smoothly with respect to those parameters.

The actual amount of door to be changed is drawn from an uniform distribution $U(0, 2 \times amount)$. The actual delay is taken from an uniform distribution $U(delay_{min}, 2 \times delay - delay_{min})$ where $delay_{min}$ is an empirically derived technical limit of the implementation for two successive interference to complete successfully. The value of $delay_{min}$ for the experimental setup was 30msec.

The randomness is realized through a built in random number generator of the Java language. To prevent unwanted noise in the experimental results, the generator is seeded with the same number for all experiments in a single round (experiments that differ only in the agent to be tested) so that the resulting interference is the same for all agents.

2. 4. 3 PDDL Domain Representation

The most widely accepted formalism for expressing classical planning domains is PDDL (Planning Domain Definition Language) [23] – the language used at the International Planning Competition (IPC) [34]. Because the number of doors affected by a single button is arbitrary, it would be difficult to describe a general action for pushing a button in PDDL. And since all of the planners chosen for experiments transform all domains to ground representations (no variables) internally, the planning domains for experiments were created as grounded from the start. This brings the benefit of controlling and pruning any possible symmetries that would otherwise arise when the planner would transform more general action descriptions to grounded ones. The PDDL abstraction does not take into account the precise location of an agent, but only the current room it is in. Thus the domain atoms may be divided into two groups:

- *At_Room_X* – true iff the agent is in the room X
- *Adjacent_X_Y* – true iff the door between rooms X and Y is open (if the rooms X and Y are not connected at all, the predicate is omitted)

Two classes of actions are available to the agent:

- *Move_X_Y* – moves the agent from room X to room Y. More precisely, the action preconditions are *At_Room_X* and *Adjacent_X_Y*, the positive effects are *At_Room_Y* and negative effects are *At_Room_X.*
- *Push_Button_X_At_Room_Y* – pushes the button X at room Y. More precisely the preconditions are *At_Room_Y* and the positive effects are *Adjacent_A_B* for all doors the button opens. Similarly, the negative effects contain predicates for all the doors that are closed.

The formulation of initial state and goal condition is then very straightforward:

Initial state:	*At_Room_X*	where X is the room the agent is in
	Adjacent_X_Y	for every open door
Goal condition:	*At_Room_Z*	where Z is the goal room

This domain representation will be referred to as *standard* in contrast to *relaxed* described in the next section. While the move action is likely to take reasonably longer than the push button action (about 1 sec to move, 0.2 sec to push a button in the actual environment implementation), action costs are not considered for the sake of simplicity and to allow usage of as many planners as possible. Furthermore, pushing buttons only takes place when it is necessary to open a door. The case when the same effect can be achieved either by pushing more buttons at closer locations or by pushing less buttons at more distant locations may be considered

rare (see discussion on random generation of maps in section 2. 4. 5). In this sense, optimizing plan length is nearly indistinguishable from optimizing plan execution time.

Note that since the grounding of predicates is done within the domain generation, only one atom of the pair *(Adjacent_X_Y, Adjacent_Y_X)* needs to be present in the domain because the domain generator may group the two together before sending the domain to the planner. This decreases the number of atoms and shortens effect lists of actions, thus simplifying the search space and reducing its dimension.

2. 4. 4 Relaxed Domains

One of the goals of this thesis is to separately evaluate planner performance in relaxed domains (domains with irreversible actions). At first glance this is simple – buttons will be allowed to open doors and not to close them. There is however one obstacle: the move action itself is reversible by its very definition. To get around this, the relaxed domains are expressed a little differently. The predicates for adjacency stay the same, so do actions for pushing buttons (except they are not permitted to close doors). But instead of explicitly reasoning about actual position, the agent reasons about the set of reachable locations – because buttons may not close doors, this set may only grow and thus adding a room to the set of reachable locations is an irreversible action.

The exact implementation of this concept in PDDL is a class of atoms *Reachable_Room_X* (representing the set of reachable rooms). Actions of the form *Reach_Room_X_From_Room_Y* are an equivalent to move actions. Their preconditions are *Adjacent_X_Y* and *Reachable_Room_X* and effects are *Reachable_Room_Y*. In the initial state, *Reachable_Room_X* holds only for the current location of the agent, the goal condition is *Reachable_Room_Z* where Z is the goal room.

The relaxed formalism was described in the terms of PDDL, but unlike the standard formalism, the plans that solve problems in relaxed formalism cannot be directly interpreted as actions the agent may execute in the environment. Instead, the reach actions are ignored and the push button actions are expanded to first move to the button location and then push the button. After successfully executing the plan, the agent's position is not defined, but the path to the goal room should be open, so the last step of the plan interpretation is to add move actions from the current agent position to the goal.

As mentioned above, planning in relaxed domains precludes reasoning about the agent's position. This means that it is also impossible to optimize agent movement

in such formalism. While the planner finds plans with minimal length, the plan length is only loosely related to plan execution time. While pressing less buttons and requiring fewer locations to be reachable should on average reduce travelling distances, this is definitely not the general case. For example, it is possible that the agent will press a button at location *A*, then press a button on location *B* and then return to *A* to press another button – different orderings of such actions are equal to the planner. Another issue is that the planner tries to minimize the number of reachable locations, because an action is needed for every location to be visited. So it prefers returning to already visited locations regardless of their distance. This is an example of a more general issue with relaxed domains: while they allow for faster and simpler solution algorithms, some important aspects of the problem may not be representable. It is of interest to what extent this will affect performance of agents using relaxed representation in practice.

2. 4. 5 Random Generation of Maps

Experiments require a reasonable number of different maps, some of them quite large. It would be tedious to design all those maps by hand and it would also introduce risk, that such maps would have some hidden exploitable feature in common that the author would not be aware of. Thus it is beneficial to generate them with a randomized algorithm.

The procedure of generating maps at random is the same for relaxed and standard maps. The parameters of the procedure are map size and the average number of doors opened by a single button and the average number of doors closed by a single button (zero for relaxed maps). The total number of doors opened and closed is not subject to randomness and is simply the product of the number of doors on the map and the respective average. The procedure is as follows:

1. Generate a random path (possibly with repeating locations) from start to end room.
2. Add door openings that ensure that every door on the generated path can be opened from some button in earlier part of the path. The button location on the path is chosen at random, but biased to prefer locations closer to the door - otherwise the majority of the active buttons would be at the beginning of the path.
3. Add other button effects choosing both button and affected door at random, until desired average number of open and close interactions is achieved, while protecting the solution created in step 2.

In maps created with the above procedure, most of the button interactions are completely random. Interactions introduced by the guaranteed solution path

(step 2) form a minor part of the total interactions. Because of that, the likelihood that two nearby buttons open the same door is not very large and drops quickly with growing map size. This supports the idea that optimizing plan length in standard domains is very similar to optimizing plan execution time as mentioned in section 2. 4. 3.

For standard domains the maps were created with average number of doors opened/closed by a single button varying from 0.2 to 2.5. For relaxed domains the number of doors opened by a single button varied between 0.2 and 1.7. The range for relaxed maps was lowered a little, because there is a simple relation between the number of doors opened and the difficulty of the map.

2. 5 Agent Development Platform

As noted in the introduction, computer games have become a popular AI research platform because they provide ready-made environments with many aspects of the real world and good looking visualisation. This thesis is not an exception. Among the possible game environments, Unreal Development Kit (UDK) [12] – a free version of the Unreal engine was chosen for several reasons:

1. It is free for non-commercial use.
2. It is primarily a first-person shooter engine, so that it provides a high level of detail in controlling the individual agents in the environment (contrasted to real-time strategy games and other game genres).
3. It is a general engine accompanied with good tools for creating and/or modifying every aspect of the game – most importantly, it can be modified to communicate with external process that handles the agent logic.
4. It has a long history of research usage (e. g. [20] [25] [28]).
5. Libraries for connecting external agent controllers to the engine are available.

Except for previous versions of the Unreal engine, there is no comparable game environment with respect to those qualities.

The Pogamut platform [11] is a framework for connecting agents to multiple game environments, most notably to several versions of the Unreal engine. It provides a high level agent programming interface written in the Java language. Since there is no other available platform with comparable level of abstraction, it was chosen for this thesis. Moreover, Java provides ever-growing wealth of libraries that simplify many programming tasks.

Pogamut was already connected to UDK, but this connection was not yet tested well, so one of the side goals of this work was to implement a more complex agent in Pogamut and fine-tune this connection. Another side goal was to connect several planners to Pogamut, so that further research on planning will be made possible.

2. 6 Agent Architecture

In the environment of Unreal, pushing a button or moving from one room to another is not an atomic action. Pushing a button consists of moving close to the button and then sending a "use trigger" command. To move correctly, the agent has to consider its position in the room and then plan a way by following navigation links present in the map. It also needs to check constantly whether it was stopped by an unexpected obstacle (e.g. door just closed). Similarly, the virtual environment does not explicitly represent rooms, doors or buttons. Those are simply three dimensional objects or sets of objects with certain physical properties. The activity of buttons and door movement are code fragments triggered by low-level events in the environment and have no declarative meaning. Thus the action selection mechanism needs methods to directly execute high level actions and access abstract environment state without working directly with the low-level representation of the virtual world. Although the Pogamut platform abstracts a lot of low-level issues away and provides ready-made code for navigation and other simple tasks, its atomic actions and sensing mechanisms are still too detailed. This section describes the implementation of an abstraction layer over Pogamut platform for the purpose of this work.

In accordance with assumptions laid out in section 2. 3, the agent has full access to the world state and knows the effects of pushing all the buttons beforehand. The agent is also immediately notified of any external change in the environment. In computer games terminology, a computer controlled agent in the game world is often referred to as *bot*. The terms bot and agent will be used interchangeably in the following text.

2. 6. 1 Architecture Overview

For the sake of simplicity, both reactive and planning agents are provided with the same abstraction layer and only differ in the action selection mechanism. The agent code is thus divided into two layers: the *mediator layer* responsible for the abstraction and the *deliberative layer* responsible for action selection. The two layers share an abstract structure called *world model*. The world model contains both static and dynamic content. The static content is the abstract navigation graph and the locations and effects of all buttons in the map. The vertices of the navigation graph are rooms and its edges are corridors connecting them. The

dynamic content is the current state of all doors. In addition, the world model maps all the abstract objects to their counterparts in the virtual world.

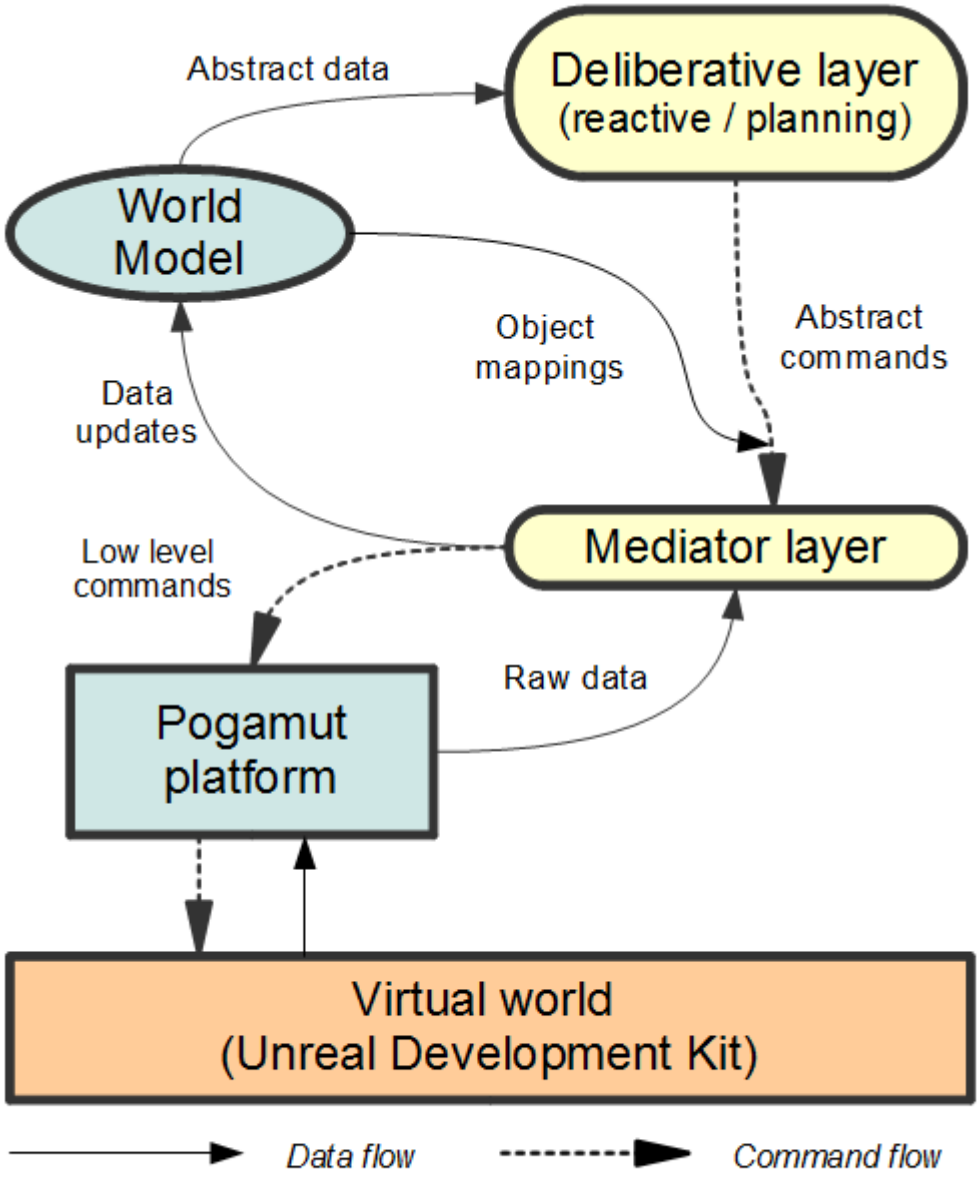


Figure 2: Agent architecture

The mediator layer communicates with the Pogamut platform, updates the world model with information received from the virtual world and keeps track of agent location. The mediator layer also encapsulates logic for pushing a nearby button and for movement to a different location (if reachable) and takes care of the technical details of such actions, including pathfinding and navigation in the virtual environment. The mediator layer and the world model implementations are the same among all tested agents.

The deliberative layer does not operate directly on the virtual world, it instead uses the data present in the world model to make decisions and send commands to the mediator layer, which executes them. I.e. the deliberative layer works directly with objects representing rooms, buttons and doors and does not care about the underlying technical details. The deliberative layer receives notifications from the world model, whenever a change in the state of the world happens.

The main agent logic is invoked by the Pogamut platform in 200-250ms intervals. The logic code then calls the mediator layer, which may send commands to Pogamut. Mediator layer in turn invokes the deliberative layer, which may send commands to mediator layer. External changes to the world state are processed asynchronously first by mediator layer, then they are sent to world model that notifies the deliberative layer.

2. 6. 2 Mediator Layer

The mediator layer provides abstraction for the deliberative layer at the level of actions. If no action is running, mediator calls deliberative layer, which may (and may not) start either a move action or a push button action. If there are no external events, the deliberative layer is not invoked again until the action is completed. In case of a change in the world state, the deliberative layer is notified asynchronously of the change and may either directly change the action executed or request to be questioned for the action in next logic iteration. If an action fails (e. g. closed door invalidates the current move action), the deliberative layer is notified of that event and new action is requested in next logic iteration.

Another important feature of the mediator layer is that it may execute heuristic actions without questioning the deliberative layer. There are two possible heuristics:

- If there is a clear path to the goal location, follow that path. (H1)
- If there is a button in the same room as the agent, that will open some unopened door and will not close any open door, push it. (H2)

The active heuristics are a parameter for the agent. In general, H1 heuristic was activated for all agents because it allowed the agent to be opportunistic when the environment was at least a little friendly. H2 was however more problematic and was switched on only for several reactive agents. See chapter 3 for further discussion.

2. 6. 3 Deliberative Layer – Planning Agents

The main agent logic resides in the deliberative layer, which is different for reactive and planning agents. For agents that employ planners, the deliberative

layer translates data from world model into PDDL and sends it to the planner. Until the planner responds, the deliberative layer initiates no action. After the plan is received, the deliberative layer translates it into actions and executes them in order while continuously checking, whether the plan is valid. I.e. whether its execution leads from current world state, including all door changes since the plan was created, to a goal state. If the plan is invalid, the planner is invoked again. If a heuristic action is triggered or if an action fails to execute, the current plan is discarded and the planner is called to yield a new one. Further details on the interface between the agent and the planner may be found in section 2. 9.

2. 6. 4 Deliberative Layer – Reactive Agents

For agents based on reactive decision making, the deliberative layer chooses an action to be performed whenever it is invoked. Three types of reactive agents were considered for comparison:

- Inactive – the deliberative layer initiates no action on its own. The agent performs only heuristic actions. This agent is included only as a baseline for comparison.
- Random – in every round, the agent chooses a button at random and pushes it. After the button has been pushed, it is added to a taboo list so that it is not pushed again before a certain timeout. Several ways to select such button were considered:
    - The button is chosen randomly among buttons in the nearest room that contains at least one non-tabooized button (Rand1).
    - The button is chosen randomly among all reachable non-tabooized buttons (Rand2).
    - A compromise between the two before – all non-tabooized buttons are considered, but the closer ones are preferred (Rand3).
- Greedy – if it is possible to move to a place closer to goal, the agent moves there. The bot does not push any buttons, unless it is a heuristic action.

In order to reduce the number of experiment runs needed, combinations of reactive agent types architectures and heuristics were compared first to each other to select the best to be included in the final runs. See section 3. 5 for results of those runs.

2. 6. 5 Further Notes on Agent Implementation

Here, yet another restriction is to be introduced so that an effective comparison of action selection mechanisms may be performed.

The restriction is that the agent is not aware about the dynamicity of the environment. Even though the changes in the environment are random, the agent theoretically could reason about the changes, e. g. it could decide to take a path that is longer, but brings more opportunities to accommodate a possible change in the world. This would however constrain the number of planners available and introduce a whole new parameter (the mechanism for handling uncertainty) into the comparison. Thus no of the agents were designed to take environment dynamics into consideration and the interference parameters are not available to them.

2. 7 Metrics

One of the most important decisions when performing an experiment is what exactly is going to be measured and how. This section deals with the choice of metrics and their interpretation.

For all experiments, a timeout was set. So the first metric is simply whether the agent managed to reach the goal before the time ran out. This will be referred to as *success rate*. Second, also very obvious metric is the time the agent spent solving the map, the *solution time*. Measuring and comparing the solution time makes sense only for the runs where agent actually reached goal. It is important to note, that in order to shorten the needed computing time for the experiments and to put a bigger pressure on the planning agents, the whole simulation was run three times faster than the UDK standard. However, all times, delays and so on in this text are presented in the actual real world time. This means that if the simulation would run in real time, similar performance of planning agents will be achieved on a machine three times slower than the one that ran experiments for this thesis.

When measuring the solution time, the startup time of the agent is not included. This is reasonable because interference daemon starts only after the bot has been fully initialized. While the startup phase does not usually take much time, there is a big increase in startup time of the first agent in an experiment batch and then a smaller increase for the first run of a particular agent type. This is due to the fact that the Java Virtual Machine compiles Java bytecode to machine code “just in time” i.e. the first time it is needed. The average startup time except the first run in the batch is about 20ms, since no agent does any complicated preprocessing. So not including it in the grand total should not make noticeable difference in the results.

Next metric, which is considered to be of secondary importance, is the distance travelled by bot while solving the map. The distance is measured as the number of transfers from one room to a neighbouring one and will be referred to as *rooms travelled*. Rooms travelled are only compared for the cases, when the agent did

reach the goal. It is introduced as an equivalent of consuming resources in the simulation world. It is possible, that some agents are more effective considering the solution time, but travel much more than others. However none of the agents presented does try to optimize this metric in any way.

Another resource to be considered is the computing power. This is of different kind than rooms travelled, since it is not a resource in the simulation world, but a resource that is important for the developer. The computing power needed is measured as the time spent deliberating. The experimental setup ensures that the action selection process will have the same system resources for all runs and the action selection is fully single threaded (including all planners). Thus the time spent deliberating should directly represent the amount of computing power required. It is however to be expected, that an agent that spends more time solving a map, will also spend more time deliberating simply because it runs longer. For this reason, the time is not measured absolutely but rather as a fraction of the total running time. The term *deliberation time* will refer to this fraction. Deliberation time is taken for both successful and unsuccessful runs. The deliberation time is by definition strongly dependent on the speed of the computer that runs the experiment.

Among the planning agents, average plan length is considered as the last metric. The average is first computed for a single run and then the statistical study is done with those. Treating every single planner run as a separate sample would put greater weight towards the runs where more replanning was needed, which might correlate with a different plan length. To compute average plan length, only the planner executions that were successful (returned a plan) are considered. Average plan length is taken for both successful and unsuccessful experiment runs, but only when at least one planner execution was succesful.

2. 8 Chosen Planners

Planners for comparison were chosen according to several criteria:

- Performance in the Sequential Satisficing Track of the International Planning Competition (IPC)
- Availability and ease of installation and use
- Popularity in the AI community (which is a very subjective matter)

Out of the four fastest planners at IPC 2011 (the most recent) three were based on Fast Downward platform [35], including the winner. The winning planner – LAMA 2011 [36] was chosen to represent this platform. LAMA 2011 also delivered the best overall plan quality and the Fast Downward platform occupied all of the first four places in quality ranking. LAMA planner is based on heuristic forward search and combines landmarks with other heuristics.

Second fastest planner at IPC 2011 was the Probe [37] and so it was chosen too. Probe performs forward heuristic search, it is based on Fast Forward [17]. Probe and LAMA 2011 represent the state of the art in classical planning.

Apart from the two very new planners, three older planners, which have earned reasonable respect in the past years and were employed in similar experiments in the past were chosen. The first is SGPlan 6 [38], which won IPC 2006. SGPlan is based on the idea of parallel decomposition, where the planning problem is divided into several subproblems that are partially independent. The subproblems are then solved by calling modified version of the Metric-FF planner [39].

The propositional only version of Metric-FF, the Fast Forward planner [17], a top performer at IPC 2002 was also chosen. Fast Forward performs a forward state space heuristic search. It was included in the comparison to represent a simpler approach to planning.

The last general planner incorporated is the BlackBox [16]. BlackBox constructs the planning graph for the problem and converts it into SAT problem, which is then solved by various solvers.

In addition to those five planners, a specialized planner for the relaxed domains was added – the ANA* planner [10]. ANA* also performs a heuristic state space search, but takes advantage of being tailored to relaxed domains. In contrast to the rest of the planners, which are written in C/C++, ANA* is written in Java.

Another specialized planner for relaxed domains - RelaxPlan [10] was considered for addition. RelaxPlan is written in Sicstus Prolog, it transforms the problem to a constraint satisfaction problem and then uses a solver (part of the Sicstus Prolog distribution) to solve it. RelaxPlan was successfully connected to the experimental environment under Windows, but I was not able to make it run on the computing server which ran the final experiments. For this reason, RelaxPlan was not included.

In relaxed maps, planners for standard domains are tested both with the standard representation and the relaxed representation to ensure that the actual difference in planner performance is not introduced by some aspect of the representation.

2. 9 Connecting Planners

In order to connect the planners to Java, a separate subproject called Planning4J [40] was started to provide an universal API to connect classical planners to Java. The code to connect IPC planners was obtained by massively refactoring parts of the ItSimple project [41] codebase. The problems for the IPC planners are written to files, which are then read by the planners. The output of the planners is either

collected from an output file or from standard output of the planning process. The only exception is the ANA* planner, which is fully Java, so the Java objects that represent individual PDDL elements are directly translated into objects that ANA* operates on. However, the benefit of direct data transfer is marginal, because the mean time spent in I/O operations is below 0.1% of total running time and the maximal value is slightly above 1%.

2. 10 Hardware & Software

Several preliminary experiments mentioned in chapter 3 were run one at a time on a computer with Intel T2050 processor (2 cores at 1,6GHz, 32bit) with 2GB RAM, running Microsoft Windows XP SP 3. The rest of the preliminary experiments and all the final experiments were run on a dedicated computing server with two AMD Opteron 2431 processors (6 cores each, 2.4GHz, 64bit) and 32GB RAM, running CentOS (Linux core version 2.6).

In the server environment, five experiments at a time were run. This setup did allow each planner instance and each environment simulation to have its own core to run on. While the RAM was shared, the planners were all compiled in 32bit mode, so they could not occupy more than 4GB of RAM, but with most domains they stay well below 512MB. Each experiment process was run in separate Java Virtual Machine and with separate UDK server, both occupying up to 1 GB RAM in peaks and 250 to 700 MB on average. So even considering the hypothetical maximum of memory usage, there are still 2GB free for system and other miscellaneous operations.

In order to make UDK server work on Linux, it was run on top of Wine abstraction layer [42], version 1.5. While the Wine abstraction slows the process down, UDK server did not demand more than 70% of a single core computing time in peaks and about 50% on average. This leaves a big margin for the Java Virtual Machine running the agent logic and the Pogamut platform, which never rose above 30% of a single core processing time and was way below 20% on average. Furthermore there are two idle cores available at all times to accommodate any possible peaks. Thus the experiments running in parallel may be considered independent and did not compete for system resources.

3 Preliminary Results and Their Interpretation

In preparation for the final experiment runs, several batches of preliminary experiments were performed. This chapter starts with a description of experimental setup and continues discussion of the observations made on results of those experiments. Next part of the chapter deals with adjustments to experiment design and choice of particular parameter values for final experiment runs. The last part introduces concrete hypotheses to be tested by the final experiment runs.

The data was examined using the R software [43]. Plots were created with ggplot2 package [44]. Whenever a boxplot is presented, its middle line is at the median, the box stretches from the first to the third quartile and the whiskers reach to further 1.5 interquartile distance or to the maximal/minimal value, whichever is closer to median. If boxplots in the same diagram represent uneven number of samples, the number of samples for each category is indicated in the box.

3. 1 Preliminary Experiment Parameters

For the preliminary experiments, three different values for all the main interference parameters were considered. See Table 2 for exact values. The delay values are a bit unusual, because they were initially chosen for runs without the three time speed multiplier set for the experiments. An important reference is the time that the bot needs to travel from one room to the next, which is 1s. The small value of interference impact was chosen so that even on smaller maps, at least one door will be affected on average by a single interference.

Parameter	**Values**		
Interference delay	0.333s	1.667s	5s
Interference impact	0.05	0.1	0.2
Interference friendliness	0.3	0.5	0.7

Table 2: Parameters for preliminary experiments

The most thoroughly tested planner in preliminary runs was the BlackBox, the main reason being that it runs also on Windows.

3. 2 General Observations

Several important observations were made, that have impact on results of all the other experiments, they will be presented prior to discussion of other topics.

3. 2. 1 Result Fragility

The first thing to mention is that the experimental results are quite fragile, although the random seeds are fixed. It is quite common that upon running the same experiment batch twice on the same hardware, about 10% of experiments have significant difference in the solution time compared to the other run. Further inquiries showed that this is solely due to minor desynchronizations between the process running agent logic and the process running the virtual world. The difference arises, when the agent moves through a corridor that is just being closed by interference. If the agent is unsuccessful, the solution time grows significantly, especially, if the interference delay and/or impact are low. Since the time needed to travel from one room to a neighbouring one is about 1sec and the agent logic is invoked 4-5 times per second, the location of the bot at the time of interference may differ up to 1/4 of the corridor length between successive runs. The smallest interference delay is 1/3s, so the chance that a race condition arises while passing through a single corridor is large enough to explain the observed number of discrepancies.

3. 2. 2 Metrics Normality

Another important observation is that both solution time and rooms travelled distributions are not normal, unless a log transform is applied to them. In subsequent text, all hypotheses about solution time and rooms travelled will always speak about their respective logarithms. The other metrics can be considered reasonably normal without transformation.

3. 2. 3 LAMA Planner Issues

In general the LAMA 2011 planner (IPC 2011 winner) performed very poorly. This corresponds to a warning at the Fast Downward website:

"The portfolio configurations are intimately tied to the competition time limit of 30 minutes. If you use a different time limit, you cannot use these planner configurations out of the box." [45]

Inspecting the planner output reveals that while the planner comes up with a solution quite quickly it then spends a lot of time searching for better solutions. In order to accommodate for that, a modification was introduced that uses exactly the same search procedure but returns the first solution found. Still the planning time is often unreasonably large. Further inquiries pointed at the preprocessing phase – the Fast Downward platform translates PDDL input to a different formalism called SAS+, which is based on state variables. This translation may take from several seconds to tens of seconds, which is a big performance hit, considering the

interference delays. Due to experiment time constraints and its very poor performance in preliminary experiments, LAMA 2011 was left out of the comparison in favour of more experiment variants. Agents with any planner based on Fast Downward platform are unlikely to perform well in dynamic environments unless the preprocessing phase is left out (i.e. if the problem was described in SAS+ formalism from the very beginning).

3. 3 Overall Planner versus Reactive Performance

A first series of experiments was run on two medium and two large standard maps with BlackBox and random agents, performing three runs for each combination of agent, map and interference parameters. BlackBox and Rand1 were run also for two medium and two large relaxed maps.

In general, those results showed that planning agents gain significant advantage over reactive ones in both success rate and solution time – however, this has not yet included Greedy agent, which was added later. See Figure 3 for details. Note however, that they represent uneven number of runs – BlackBox and Rand1 were run more times than Rand2. This advantage is more significant when the environment is hostile or if the dynamicity is low. See Figure 4 and Figure 5 for detailed results.

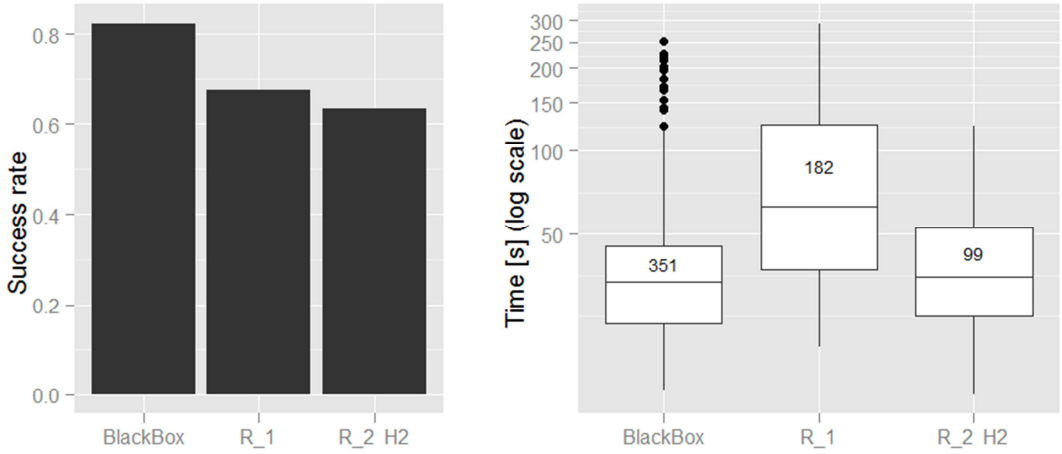


Figure 3: Success rate and solution time of agents in preliminary runs

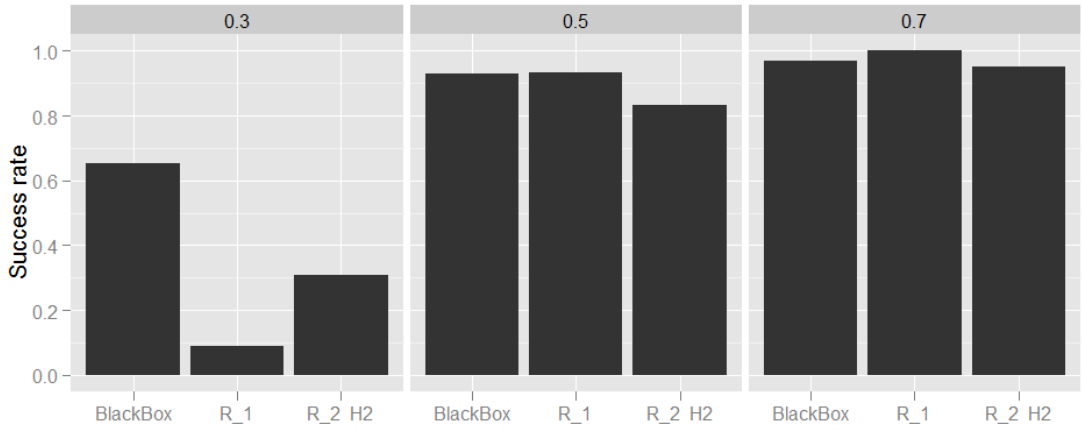


Figure 4: Success rate of agents in preliminary runs depending on friendliness

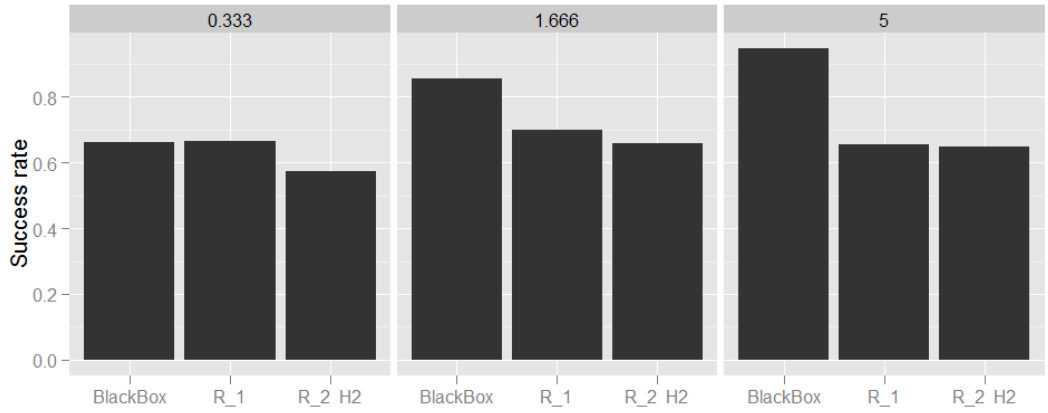


Figure 5: Success rate of agents in preliminary runs depending on interference delay

3. 4 Choosing Map Size

While running the preliminary experiments it was shown, that on maps consisting of 5x5 rooms (referred to as small maps), the planning is very fast: FF and BlackBox both plan faster than 50ms including I/O operations on such domains. Since the logic is run in 250ms cycles, this is close to negligible and smaller maps were thus not considered. For large maps (10x10 rooms), the planning time of FF and BlackBox grows to 400 – 1500ms, which should introduce significant latency in the planning agents' responses. To provide a greater margin, maps of size 13x13 were tested as well and medium (7x7) maps were added to smooth the scale.

The size of respective planning domains is presented in Table 3. The domain sizes are the same for both relaxed and standard domains. While the number of predicates is fixed with map size (predicates correspond to rooms and corridors) the number of operators is variable, because not all buttons in a level have to be active.

Map size	Predicates	Operators
Small (5x5)	65	90 – 160
Medium (7x7)	133	190 – 336
Large (10x10)	280	390 – 720
13x13	481	650 - 1248

Table 3: Planning domain sizes

3. 5 Choosing Best Reactive Approach

Since there are quite a few reactive approaches laid out (see section 2. 6. 4), not all of them could be included in the final comparison due to computing time constraints. This section deals with experiments that tried to determine, which reactive approach is the best and thus has to be included.

The Inactive agent will be included in all comparisons, because it is an important baseline. If it should function as a baseline, it does not make sense both to run the Inactive bot without any heuristic (it would not do anything at all) and to include heuristic H2 (the bot pushes buttons that open something but do not close anything, if they are nearby). The Inactive bot will hence be part of the final comparison, running heuristic H1 (if there is a path to goal, follow it).

It does not make sense to run any of the other reactive bots without heuristic H1 – for Greedy bot it is irrelevant and the random bots would be very weak otherwise. Since Rand1 presses the closest buttons only, its behaviour would change only very slightly with H2 and thus it will not be run with H2. Rand2 (considers all reachable buttons for random selection), Rand3 (combination of Rand1 and Rand2) and Greedy will all be tried with and without H2.

3. 5. 1 Standard Domains

The different reactive agents were run on 4 medium maps and 3 large maps. For every set of parameters and map, every agent was run twice resulting in 54 experiments per bot and map. To analyse the success rate, the results were first tested with χ^2 test for heterogeneity to determine, whether there is a difference between individual bots. Then multiple comparison of means with Tukey contrasts [46] over an ANOVA fit with a first order generalized linear model was performed to compare pairs of agent types. One way between-subject ANOVA is used to compare solution time and Tukey's test is run to determine, which agents have better solution times.

For medium maps the null hypothesis that different reactive approaches yield the same results can safely be rejected ($p = 6.10^{-5}$). The Greedy agent with H2

heuristic performs over the average (see Figure 6). Indeed, the null hypothesis that Greedy agent with H2 heuristic performs the same as other agents can safely be rejected ($p = 10^{-6}$). There are no other statistically significant differences – the random agents and Greedy agent with H1 heuristic have the same success rate with p-value 0.29.

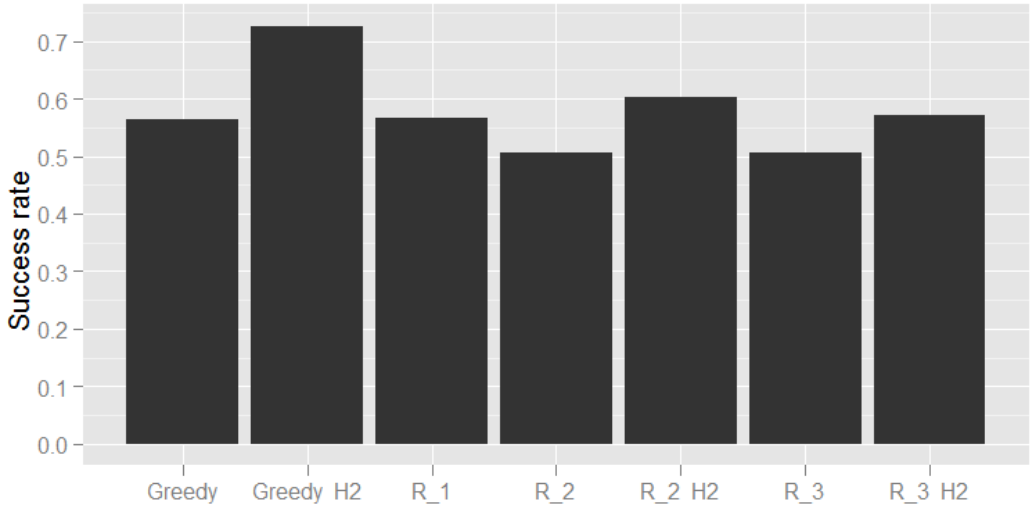


Figure 6: Success rate of reactive agents on medium maps with standard domains in preliminary runs

There are also significant time differences: the null hypothesis that all the reactive approaches have the same solution time can be rejected (p = 0.01). Tukey's test reveals that the only significant difference is that between Rand1 and Greedy with H2 (p = 0.006).

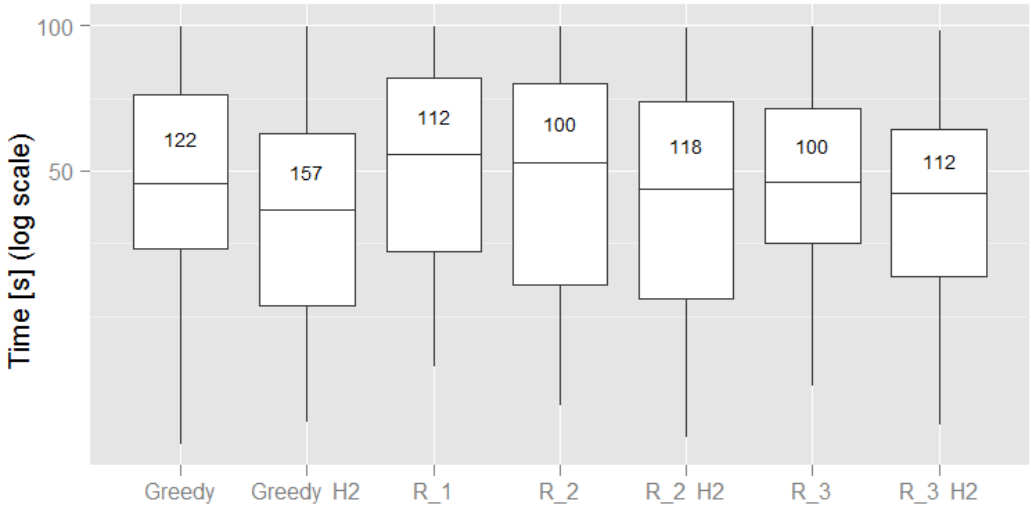


Figure 7: Solution time of reactive agents on medium maps with standard domains in preliminary runs

For large maps the null hypothesis that different reactive approaches yield the same results can safely be rejected ($p = 10^{-10}$). The Greedy agent both with and without H2 heuristic performs over the average (see Figure 8). Indeed, the null hypothesis that both Greedy agents have the same success rate as the rest can safely be rejected ($p = 3 \cdot 10^{-12}$). There are no other statistically significant differences – all of the random agents have the same success rate ($p = 0.064$) and the Greedy bot with and without H2 heuristic has the same success rate ($p = 0.34$).

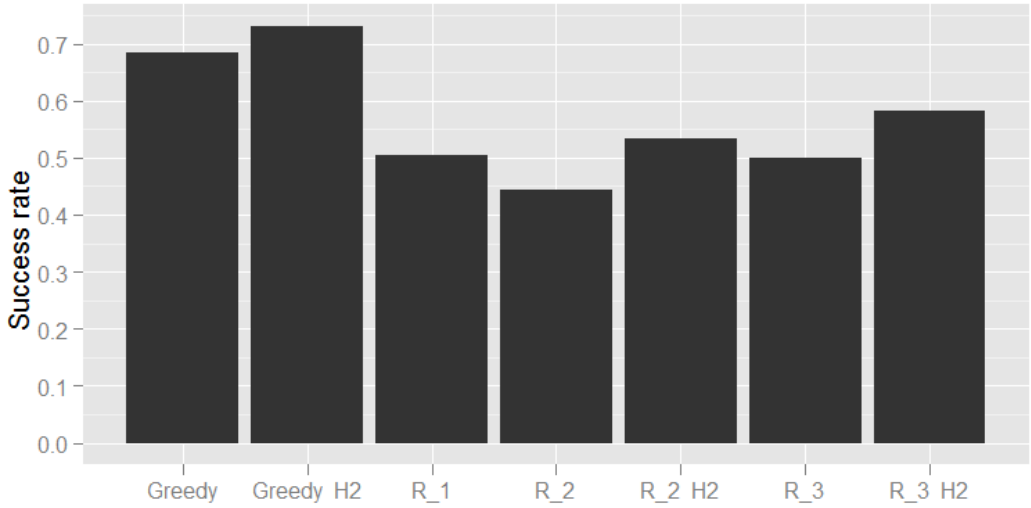


Figure 8: Success rate of reactive agents on large maps with standard domains in preliminary runs

There are no significant differences in solution time for large maps – (all $p > 0.5$). See Figure 9 for the actual results.

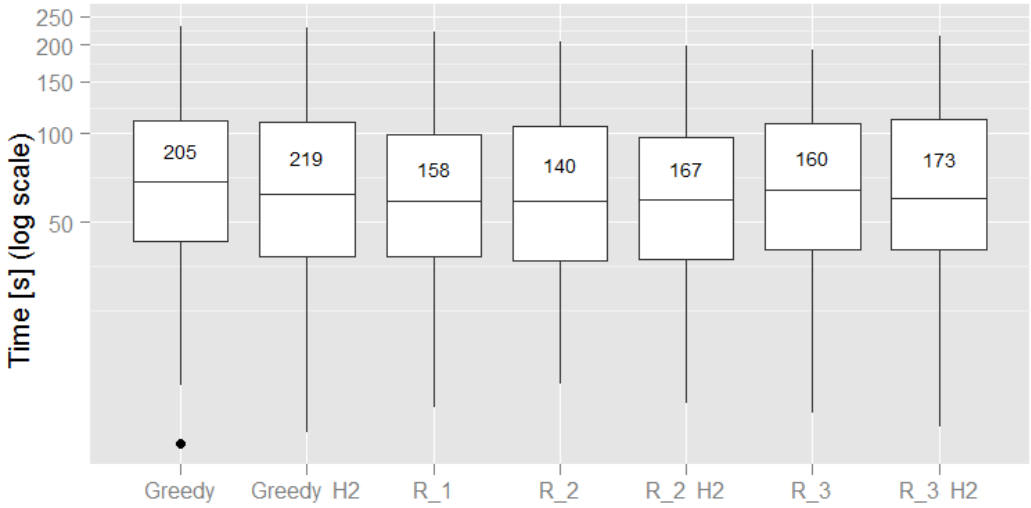


Figure 9: Solution time of reactive agents on large maps with standard domains in preliminary runs

3. 5. 2 Relaxed Domains

All reactive agents were tested on two large and two medium maps. For every set of parameters and map, every agent was run twice. Statistical methods are the same as in preceding subsection.

For medium maps and success rate, the only significant difference is that Greedy without H2 is worse than all other except Rand2 without H2 (all $p < 0.001$) and that Greedy with H2 heuristic is better than Rand2 without H2 ($p = 0.007$). For large maps Greedy with H2 is significantly better than all other (all $p < 0.003$), but other differences are not significant.

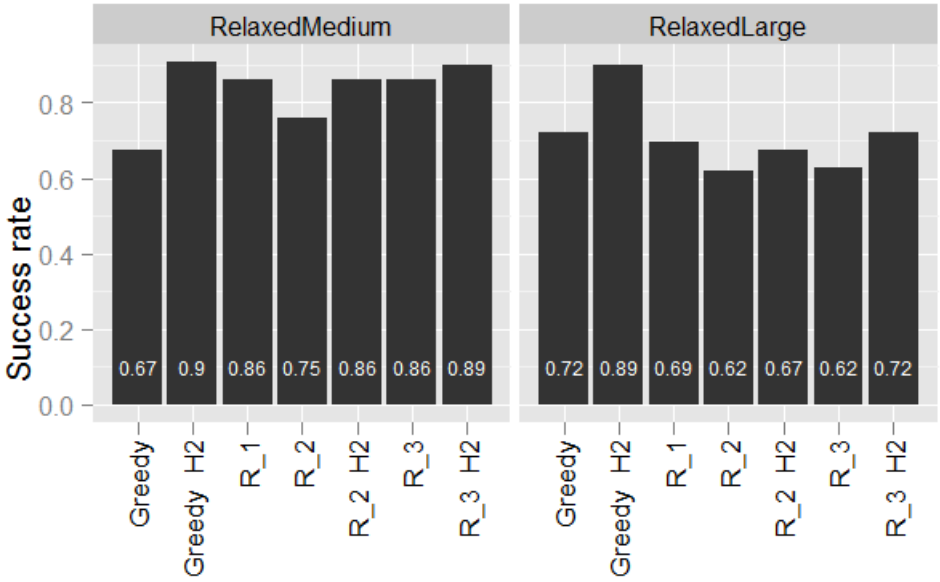


Figure 10: Success rate of reactive agents on relaxed maps

Considering solution time and medium maps, Greedy with H2 is significantly better than Rand2 without H2 ($p < 10^{-6}$); Rand3 with H2 is better than Rand2 without H2 ($p < 10^{-6}$) and Greedy without H2 is worse than all the other except Rand2 without H2 (all $p < 0.001$). For large maps, Greedy with H2 is better than all the other agents (all $p < 0.04$) and Rand3 with H2 is better than Rand2 without H2. Other differences are not significant.

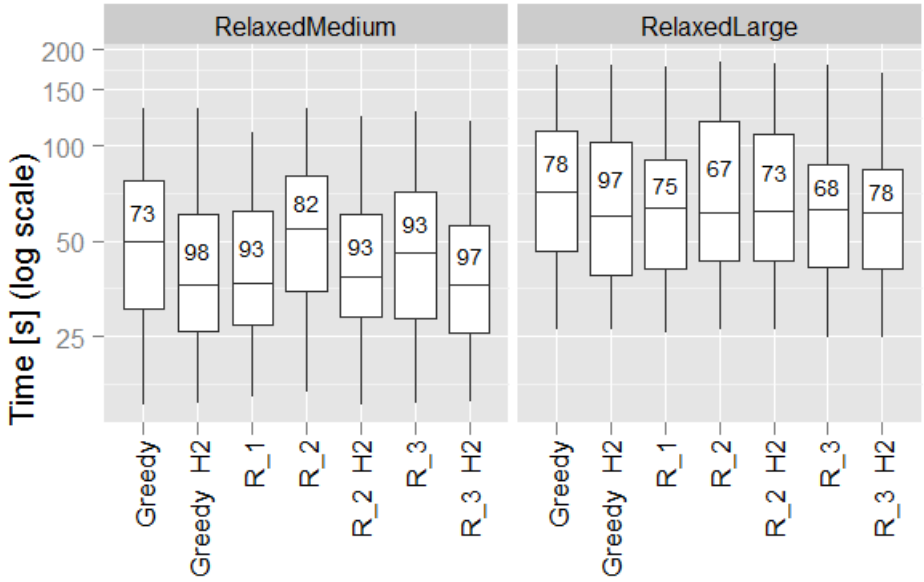


Figure 11: Solution time of reactive agents on relaxed maps

3. 5. 3 Conclusion

Greedy with H2 heuristic is obviously the best reactive approach and will definitely be included in the final runs. However, it would be beneficial to employ the best of random agents as another baseline in the experiments. Random variants without H2 heuristic can be ruled out, because they performed the weakest, but the differences between the remaining three random agents are not statistically significant. Taking the insignificant differences into account, Rand3 with H2 is better than Rand2 with H2 which is better than Rand1. But Rand3 is the most complicated of the three and relies on a user-chosen constant to fine-tune its behaviour, so the second best, Rand2 with H2, was chosen for final runs because it is simpler.

3. 6 Determining Final Experiment Parameters

The results from preliminary experiments suggest that the overall range of selected interference parameters values is large enough, in fact a little too large – with the longest delay and smallest impact, the planning bots require no replanning in most cases, even if the environment is quite hostile (friendliness = 0.3). On the other hand, the shortest delay and greatest impact result in the map being either nearly unsolvable or too easy, depending on friendliness of the environment. For this reason the parameter values need to be changed, so that the results gain more variability.

Among the three parameters, best candidate for change is the interference delay, because its values are the most arbitrary. The actual delays were selected so

that there are more combinations that yield the same mean number of door changes per second, thus we can better decide, whether it is the delay or the impact that has greater impact on the agent performance.

In the first batches of experiments, Greedy agent was not yet included, resulting in optimism about the planner performance. But upon including Greedy agent, it was soon clear that it outperforms planning agents in most cases. Unsurprisingly, the planning agents were better in experiments with lower friendliness, but even with friendliness 0.3, planners were worse in experiments with shorter delay. To investigate, whether there is a transition point in the friendliness spectrum that clearly benefits planning agents, two new friendliness values (0 and 0.15) were introduced. More friendliness values were not added on the opposite end of the spectrum because, even for friendliness 0.7 the success rate of all agents was 100% or only slightly worse. Adding values on the lower end of friendliness spectrum should bring the planning agents on a par with Greedy or better.

Parameter	**Values**				
Interference delay	0.5s	1.5s	3s		
Interference impact	0.05	0.1	0.2		
Interference friendliness	0	0.15	0.3	0.5	0.7

Table 4: Parameters for final experiments

3. 7 Planners and Heuristics

To assess the usability of heuristics for planners, agents employing BlackBox and FF were run with no heuristics, H1 heuristic only and both heuristics. The agents were run on one standard and one relaxed map of three different sizes (Medium, Large and 13x13) three iterations were run, with parameters for the final experiments, except that the two friendliness values below 0.3 were left out.

While the success rate of BlackBox and FF with the same heuristic cannot be considered different (p-value 0.38 for both heuristics case, 0.94 for H1 case and 0.99 for no heuristics case), the planners with H1 heuristics gain significant advantage over planners without H1 (all $p < 10^{-3}$). Adding H2 heuristics degrades the performance in most cases (13x13 maps and FF is the only exception). Figure 12 shows the results graphically.

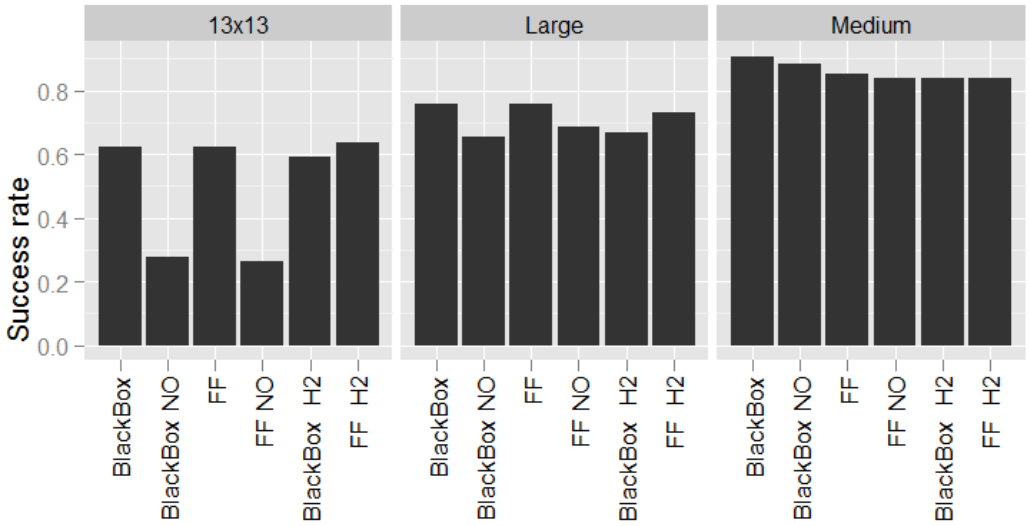


Figure 12: Success rates of planning agents with different heuristics grouped by map size.

Considering solution time, the results are less clear, with almost all of the differences on Large and Medium maps not being statistically significant. On 13x13 maps on the other hand, most of the differences are significant (all $p < 0.03$) except for BlackBox and BlackBox H2 ($p = 0.99$), BlackBox NO and BlackBox ($p = 0.11$) and FF and BlackBox NO ($p = 0.99$). Thus FF on 13x13 maps is the only exception where adding H2 heuristic reduces solution time. See Figure 13 for graphical results.

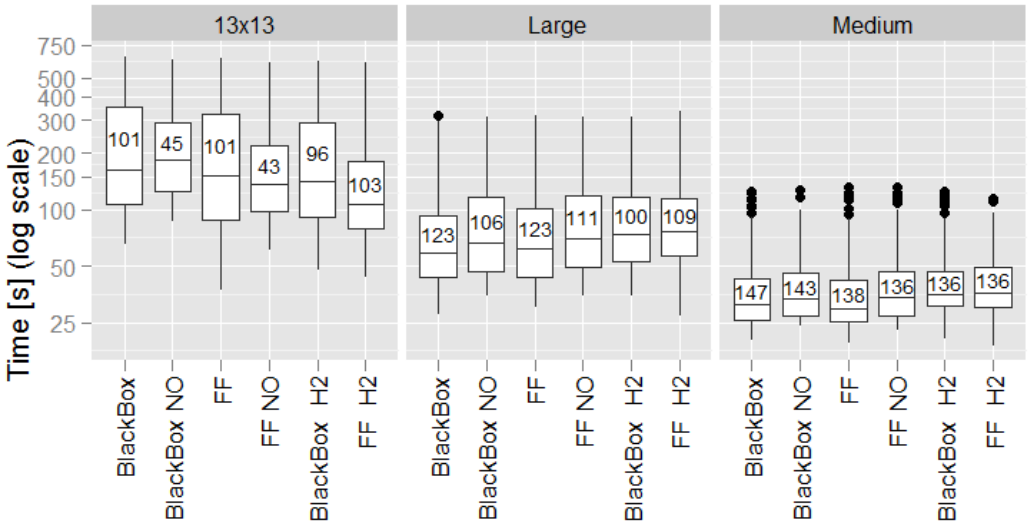


Figure 13: Solution time of planning agents with different heuristics grouped by map size.

As noted above, H1 heuristics brings significant raise in success rate of planning agents, but adding H2 degrades, or (for large maps) does not raise success rate. Adding H2 also does not in general improve solution time. For this reason, all planners were run with H1 heuristic only in the final runs.

3. 8 Experiment Scale

The preliminary experiments showed that the simulations are very time consuming. It was therefore necessary to find a balance between the quality of results (more experiments on more maps bring better results) and computing time needed. Since the random factor of interference has severe impact on agent performance, at least three runs with the same agent, parameters and map and different random seed are needed to reduce noise. This further limits the number of maps that may be tried, especially for relaxed domains, where there are more than twice as much agents as in the standard domains (each planning agents with both domain representations and the ANA planner). In the end trying 9 different maps for standard domains and 6 for relaxed domains seemed a reasonable trade-off considering the amount of computing time available. The single exception are 13x13 standard maps, where only four maps were included due to time constraints (the larger the map, the more expensive the experiments are) A list of map classes is presented in Table 5 along with timeouts for the respective map type.

To determine appropriate timeouts for different map sizes, planning agents were run on all standard maps without interference. The timeouts for final runs were chosen as 5 times the average solution time of planners on standard maps of respective sizes. This way there is sufficient margin for at least five replannings, which should be enough to show how the planner copes with the dynamicity of the environment. The timeouts are the same for standard and relaxed maps to allow for comparison.

Map class	Size	Number of maps	Timeout [s]
Small	5x5	9	135
Medium	7x7	9	165
Large	10x10	9	250
13x13	13x13	4	900
Relaxed small	5x5	6	135
Relaxed medium	7x7	6	165
Relaxed large	10x10	6	250
Relaxed 13x13	13x13	6	900

Table 5: List of map classes with timeouts

3. 9 Hypotheses for Final Run

Based on the preliminary results, several hypotheses were formulated. Those are to be tested by the final experiment runs:

Hypothesis 1: The solution time and success rate of all agents is significantly affected by all interference parameters (impact, delay, friendliness)

Hypothesis 2: Agents with planners will outperform reactive agents in success rate, solution time and rooms travelled. I.e. a pairwise comparison of each planning and each reactive agent will favour the planning agent (for all map sizes separately).

Hypothesis 3: The relative advantage of planning agents over reactive ones will be greater when the friendliness is lower and the delay greater. I.e. the interactions agent type – friendliness and agent type – delay will be significant for both success rate and solution time.

Hypothesis 4: Greedy bot will outperform Random bot and Inactive bot in success rate, solution time and rooms travelled (for all map sizes separately).

Hypothesis 5: There will be significant differences in different planner performances in solution time and success rate (for all map sizes separately).

Hypothesis 6: All relaxed planner variants will be outperformed by their standard counterpart on all relaxed maps in solution time, success rate and rooms travelled (for all map sizes separately).

4 Results

This chapter presents the results of the final experiment runs. It starts with discussion of several issues encountered during the final runs. Then results for standard and relaxed domains are given separately with both informal and rigorous statistical analysis provided, the hypotheses introduced in section 3. 9 are tested and basic conclusions are drawn. The final part of this chapter summarizes the results of hypothesis testing

Results for standard and relaxed domains are analysed separately, because some agents were run only for relaxed domains and it was not clear how to incorporate that into a sound analysis involving all agents. A side by side comparison of performance on relaxed and standard domains would be needed anyway. Moreover separating domain types leaves more focus on interactions of different metrics which is of greater interest. Last but not least, analysing the data from both domain types together did not reveal any trends that would not be present in separate analysis.

For each combination of map (see section 3. 8 for a full list), agent and interference parameters (see section 3. 6 for details) three experiments were run with different random seeds for interferences. This led to a total of 68 175 experiment runs taking over 145 days of computing time.

The data was examined using the R software [43]. Plots were created with ggplot2 [44] and scatterplot3d [47] packages. Whenever a boxplot is presented, its middle line is at the median, the box stretches from the first to the third quartile and the whiskers reach to further 1.5 interquartile distance or to the maximal/minimal value, whichever is closer to median. If boxplots in the same diagram represent uneven number of samples, the number of samples for each category is indicated in the box.

4. 1 General Issues

Two issues were identified when analysing the final results. First is that on rare occasions, SGPlan, FF and Probe failed with an error (or without even signalling an error on some occasions) and did not return a value. Most of the failures occurred in standard domains and only few in relaxed domains. Such failures were always reproducible – the same situation on the same map failed always. However, those situations were extremely rare compared to the total number of planner invocations (less than 1 failure in 10 000 invocations) and the planner always started to plan correctly after next interference. While the impact on experiment

results is small, it still means that academic planners are not stable enough for immediate production use.

Second issue is quite high amount of button usage failures – over 0.5 failure per run. Analysis revealed that the cause is the speed of agent movement. The agent decides to start the push button action after receiving information that its close enough. But the agent keeps moving towards button location and may rarely move too far in that direction and get out of button activation distance before the push button action is started in the virtual environment. It is likely, that this kind of failures will have greater impact on planning agents, because upon action failure the agent always replans, but there is no simple way to compensate the agent for such failure and thus the failures will only contribute to noise in the results.

The button failure issue is a fine example of the obstacles faced when acting in a dynamic and continuous environment.

The statistical methods adopted for the analysis of numeric metrics (solution time and rooms travelled) assume that the data are close to normal distribution. Histograms showing that the data can be considered normal are found on the attached DVD.

4. 2 Standard Domains

In this section, the results for standard domains are provided accompanied with a series of plots that demonstrate patterns in the gathered data. In each subsection an informal discussion of the results is followed by rigorous statistical analysis and discussion of hypotheses introduced in section 3. 9.

4. 2. 1 Overall Results

Inspecting the data as whole, it is not possible to directly compare solution time or rooms travelled, because different map sizes are involved. However the plot of succes rates of different bots shows some interesting results (see Figure 14 and Figure 15). Most importantly, the Greedy bot has succes rate very close to planning agents, even outperforming Probe. Thus planning did not bring the agents large advantage. In general, the success rate is suprprisingly high – this shows that actually finishing the task was not that difficult.

Another interesting point is that the deliberation time (see Figure 15) is very similar among planning agents. More specifically: SGPlan spent on average 30% of time deliberating, FF spent 32% of time deliberating, Probe and BlackBox spent on average 35% of time deliberating and their deliberation times cannot be considered significantly different ($p > 0.99$), while all other differences are significant (all

$p < 10^{-6}$). The p-values were given by Tukey's HSD method with ANOVA fit over a linear model.

The most important factor for success rate seems to be environment friendliness. In a friendly environment, nearly all experiment runs ended successfully. Even balanced friendliness setting (0.5) allowed most of the agents to reach the goal (see Figure 16). Interference delay and impact however do not seem to have significant effect on the success rate (see Figure 17 and Figure 18).

Analysing results formally, a generalized linear model for success rate shows that all interference parameters are significant with $p < 10^{-15}$ and that friendliness has the highest effect size. A linear model for logarithm of solution time shows similar values, except that the interference impact level of 0.1 is significant only at $p = 0.0046$. Thus all interference parameters are significant as stated by Hypothesis 1 which thus holds for standard domains[5].

A second order generalized linear model with interaction for success rate shows that the interaction of interference delay and agent type (reactive or planning) is highly significant (all $p < 10^{-15}$) The interaction of interference friendliness and agent type is not significant (all $p > 0.09$) for most combinations thereof, the only exception is at friendliness level 0.5 where the interaction is significant with $p < 10^{-15}$. Thus the latter interaction cannot be considered significant in general which contradicts Hypothesis 3. Although noticed post-hoc, the interactions of interference delay and impact and of interference friendliness and impact are both highly significant (all $p \leq 0.005$).

A second order linear model for logarithm of solution time showed that the interactions of agent type with both delay and friendliness are significant (all $p < 10^{-7}$) in accord with Hypothesis 3. Similarly to success rate, the interaction of interference delay and impact was noticed post-hoc to be significant (all $p < 10^{-5}$), but the interaction of interference friendliness and impact is significant only for some combinations thereof.

[5] See section 3. 9 for a full list of hypotheses.

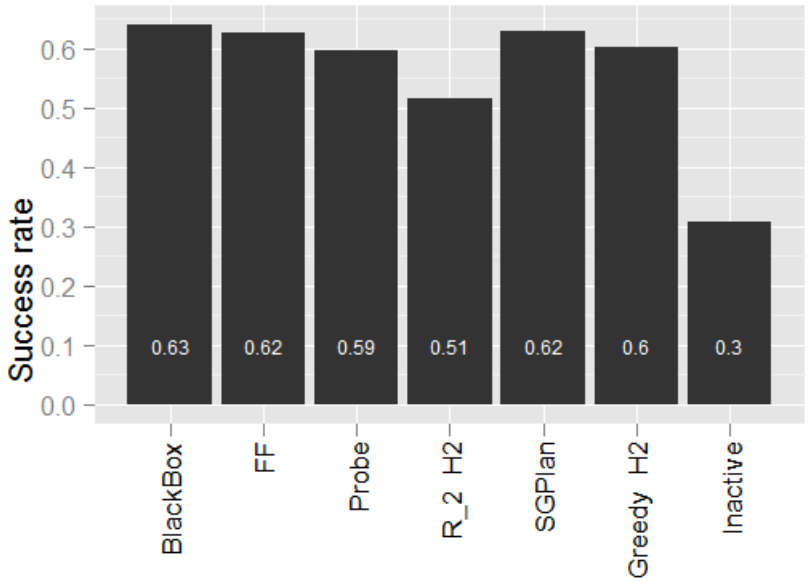


Figure 14: Success rates of all agents in standard domains

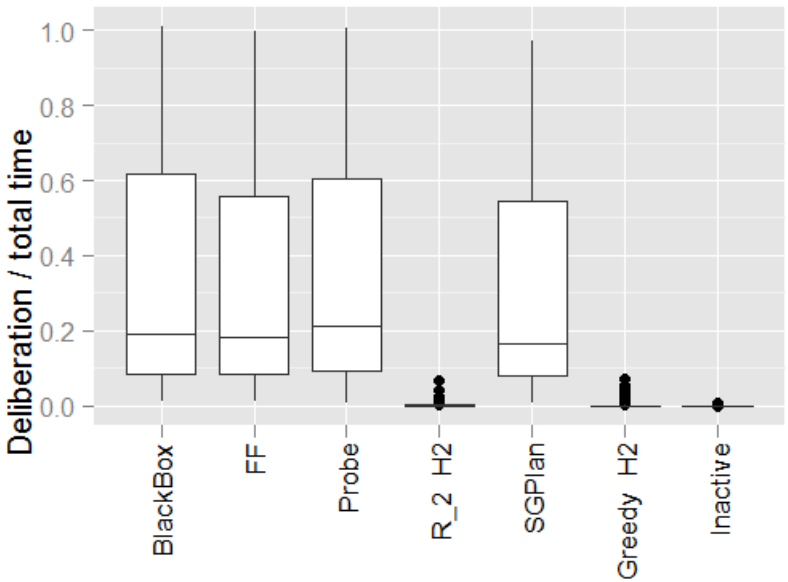


Figure 15: Deliberation time of all agents in standard domains

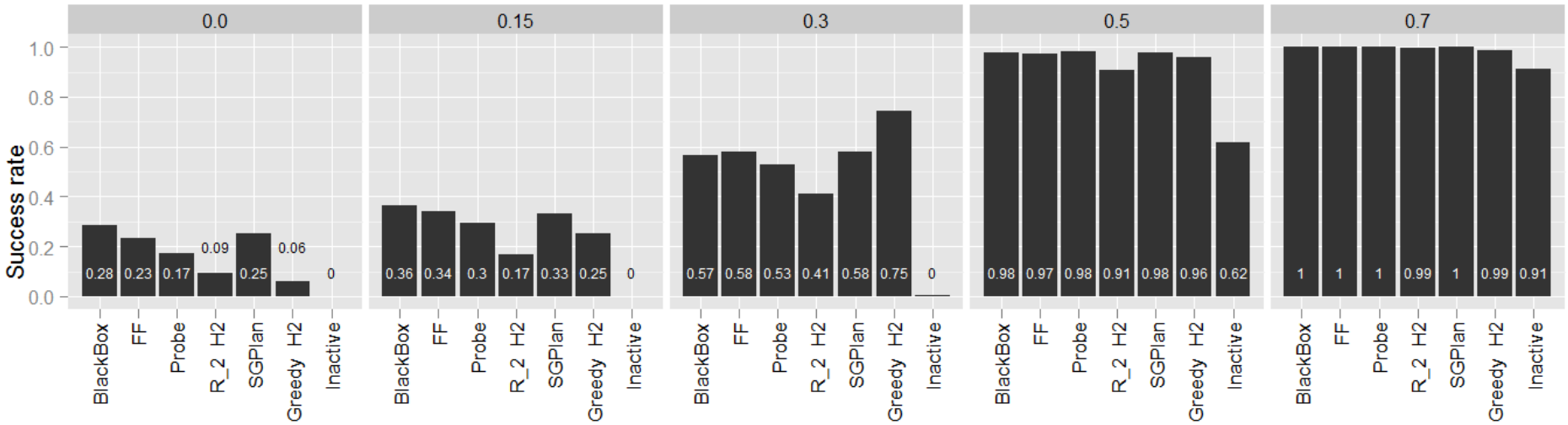


Figure 16: Success rate of all agents in standard domains depending on interference friendliness

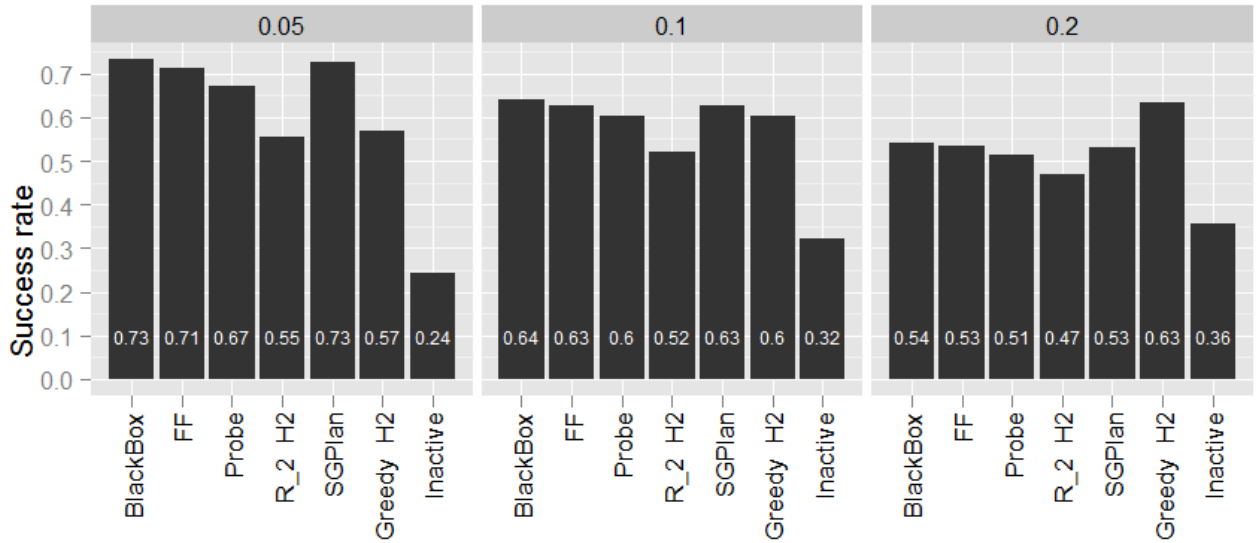


Figure 17: Success rate of all agents in standard domains depending on interference impact

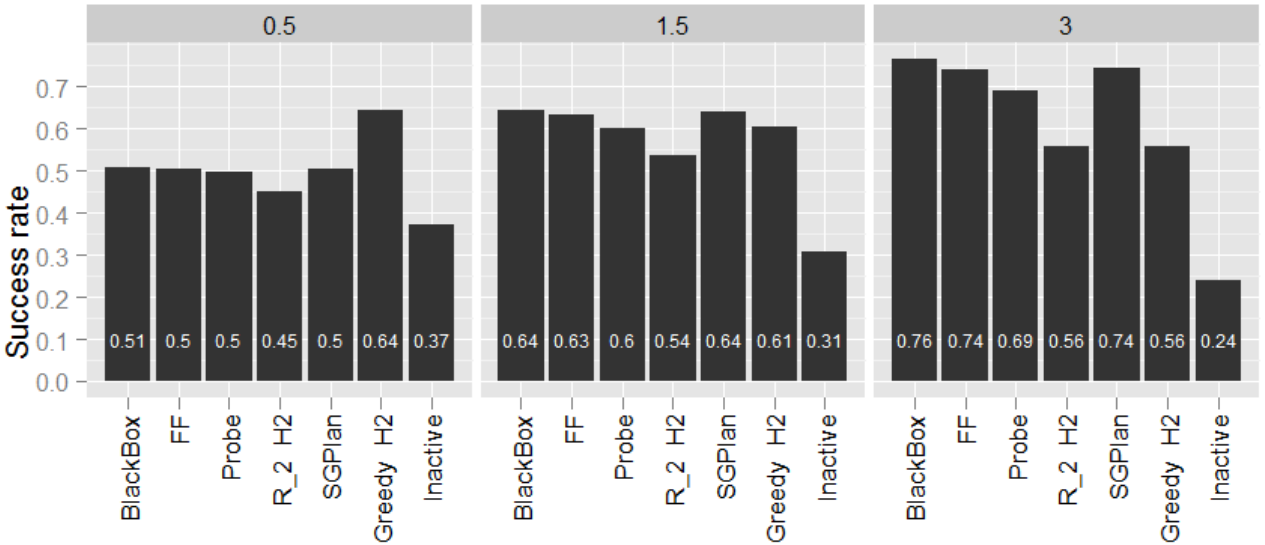


Figure 18: Success rate of all agents in standard domains depending on interference delay

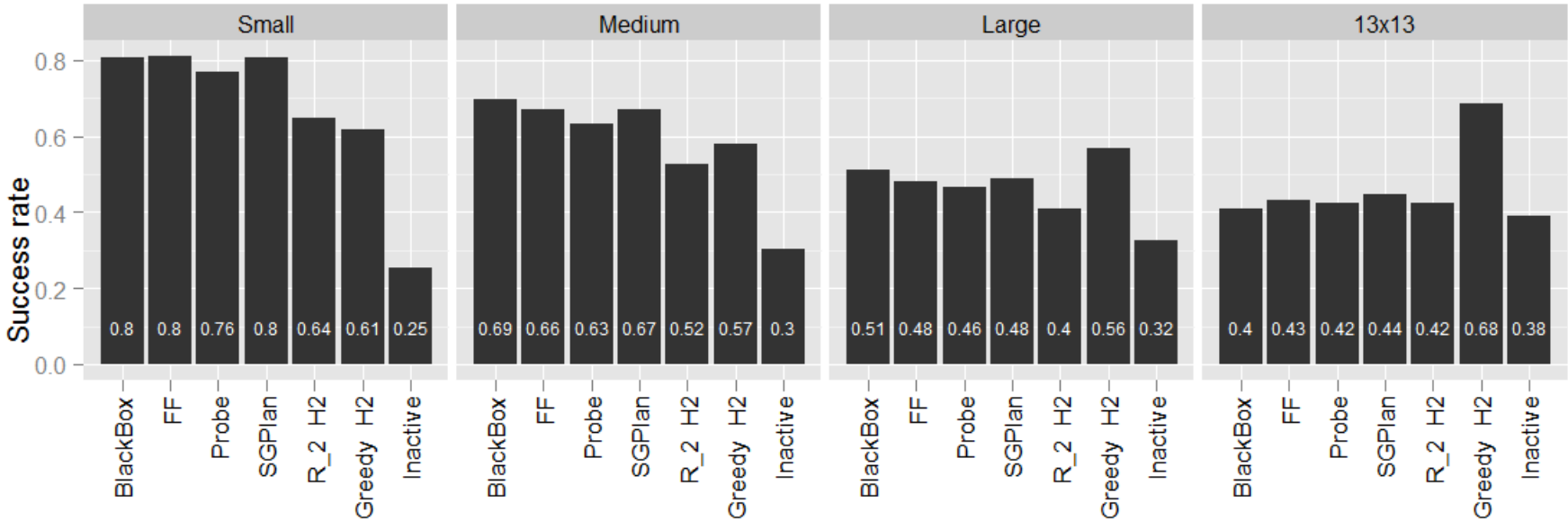


Figure 19: Success rate of all agents in standard domain depending on map size

4. 2. 2 Success Rate and Map Size

The success rate of agents at different map sizes is shown graphically in Figure 19 and summarized numerically in Table 6. It shows that planning agents greatly outperform all reactive agents in small maps, but their performance degrades with growing map size, yielding lead position to Greedy agent for large maps.

While the success rate of planning agents decreases with the growing map size, the success rate of reactive agents behaves differently. This is due to the different timeout values – the growth of timeout was designed to be proportional to degrading planner performance, but reactive agents degrade slower. Thus the success rate of reactive agents may grow with growing map size.

Statistical results for success rates are assessed using multiple comparison of means with Tukey contrasts [46] over an ANOVA fit with a first order generalized linear model.

In total results the differences between Fast Forward (FF), BlackBox, SGPlan 6 and Greedy are not significant (all $p > 0.88$), while all the other differences are (all $p < 10^{-3}$).

On small maps, differences among planners are not significant, while all other differences are (all $p < 10^{-3}$).

Although the actual results differ, similar p-values hold for medium and large maps except that Probe – BlackBox difference becomes significant ($p < 10^{-3}$).

On 13x13 maps, Greedy is significantly better than the rest (all $p < 10^{-3}$) and SGPlan with FF are better than Inactive ($p < 10^{-3}$ and $p = 0.01$ respectively).

Other differences are not significant. The Inactive baseline bot showed that in many cases no smart acting is required to complete a map.

Map	BlackBox	FF	Probe	SGPlan	Greedy	Rand2	Inactive
Small	***0.80***	***0.80***	0.76	***0.80***	0.61	0.64	0.25
Medium	***0.69***	0.66	0.63	0.67	0.57	0.52	0.30
Large	0.51	0.48	0.46	0.48	***0.56***	0.40	0.32
13x13	0.40	0.43	0.42	0.44	***0.68***	0.42	0.38
Total	**0.60**	**0.59**	**0.57**	**0.60**	***0.61***	**0.50**	**0.31**

Table 6: Success rate of agents in standard domains with differing map size

Contrasting those results to hypotheses from section 3. 9, we can see that planning agents did not beat reactive agents at all map sizes as was predicted by Hypothesis 2. More precisely, the hypothesis holds only for small and medium

maps. For 13x13 maps even Random agent becomes indistinguishable from planners, and Probe along with BlackBox is not significantly better than Inactive.

Hypothesis 4 predicted that Greedy will outperform both Random and Inactive bots, which does hold except for small maps, where Random is better than Greedy. This exception is likely due to the fact that on small maps, even when choosing randomly, it is difficult to choose a path that leads away from goal.

Finally, let us consider Hypothesis 5 which says that there will be significant differences among different planners. Examining the results shows the contrary – the differences are not significant except for Probe – BlackBox pair in medium and large maps.

4. 2. 3 Success Rate and Interference Parameters

The interaction of success rate and interference parameters is of prime interest of this thesis. While the actual numbers are not so interesting, the overall trends are important. Figure 20 shows a plane fitted through the average success rates of SGPlan and Greedy bots, depending on the environment friendliness and the interference delay. It shows the principal difference between the reactive and planning approaches in handling dynamicity. While the success rate of the Greedy agent grows with shorter interference delays, the success rate of SGPlan decreases quite steadily. There is a minor exception to this rule at the friendliness level 0, because in such a setting the environment dynamics cannot bring the reactive agent any new opportunity. Note that the Inactive agent has similar properties to Greedy, while Random agent is similar to planning agents (see Figure 21).

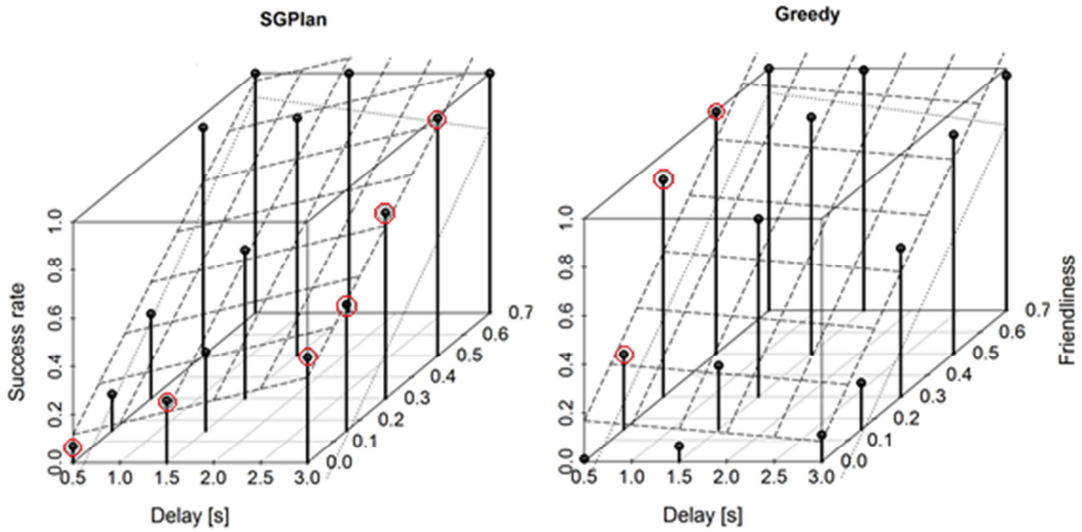


Figure 20: Success rate of SGPlan and Greedy bot in different dynamic conditions.
The dotted lines show a plane fitted to the results of the Inactive bot. Planes are fitted to the averaged results and they are intended only as a visual cue. Red circles mark points where the respective agent is significantly better than the other (all $p < 0.01$).

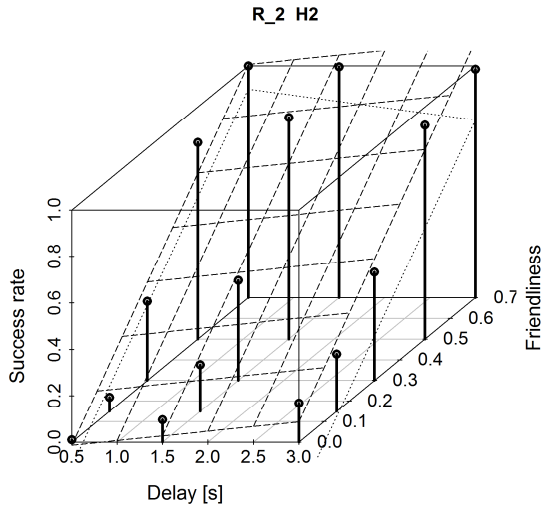


Figure 21: Success rate of Rand2 bot in different dynamic conditions.

This data informally supports Hypothesis 3, that planning agents gain more advantage with greater delay and lower friendliness.

The last remaining question is whether the interference delay or the interference impact has greater effect. To compare this, we have two pairs of parameters that give identical mean number of door changes per second as seen in Table 7.

Delay [s]	Impact	Door changes per second
1.5	0.10	0.15
3.0	0.05	0.15
1.5	0.20	0.30
3.0	0.10	0.30

Table 7: Pairs of interference parameters with identical mean number of door changes per second.

Let us take SGPlan as a representative of planning agents (we have seen that planners do not differ much in performance). Greedy and Rand2 represent the reactive approaches. We compare their performance on the aforementioned pairs of interference parameters. Clearly SGPlan acts much better with longer delay and larger impact, while Greedy acts better with shorter delay and smaller impact. In accord with previous observations, Rand2 reacts to changing interference similarly to planning agents. See Table 8 for actual values.

This is not a big surprise: the main bottleneck in planner performance is the amount of replanning which takes the same time regardless how much the world has changed, but shorter delay means more replanning. On the other hand, Greedy

depends on opening of doors by chance. Shorter delays albeit lower impact let him steadily advance one or two rooms a time.

Agent	1.5s x 0.1	3.0s x 0.05	1.5s x 0.2	3.0s x 0.1
SGPlan	0.63	***0.83***	0.52	***0.75***
Greedy	***0.58***	0.50	***0.66***	0.56
Rand2	0.53	***0.58***	0.50	***0.57***

Table 8: Comparing the importance of interference impact and delay for success rate in standard domains

4. 2. 4 Solution Time

The mean solution times and their standard deviations are presented numerically in Table 9 and graphically in Figure 22.

Map	BlackBox	FF	Probe	SGPlan	Greedy	Rand2	Inactive
Small	***23.3 (13)***	28.2 (19)	28.1 (15)	24.7 (14)	32.4 (19)	33.2 (18)	44.9 (20)
Medium	***42.7 (27)***	46.0 (30)	50.2 (31)	46.2 (30)	58.6 (38)	61.1 (36)	70.3 (39)
Large	***72.1 (46)***	75.7 (49)	85.6 (51)	78.5 (49)	96.5 (58)	94.8 (53)	103 (61)
13x13	214 (188)	***167 (144)***	206 (167)	181 (172)	253 (218)	230 (199)	255 (187)

Table 9: Average solution times [s] of agents in standard domains with standard deviation (in brackets). Best results in each row are highlighted.

To analyse the solution time, a linear model is fitted to the data with solution time log transformed to be closer to normal distribution, and Tukey's HSD test [48] is performed to reveal significant differences between agent pairs.

On small maps, all differences are significant (all $p < 10^{-3}$) except for SGPlan - BlackBox ($p = 0.2$), Probe – FF ($p = 0.99$) and Greedy – Rand2 ($p = 0.99$).

On both large and medium maps, all planners beat all reactive agents and the differences among reactive agents are significant (all $p < 10^{-5}$), while the only significant difference among the planners is Probe – BlackBox ($p < 0.01$, other $p > 0.14$).

On 13x13 maps, all differences except for FF – SGPlan, BlackBox – Greedy and Inactive – Rand2 are significant (all $p < 2 \cdot 10^{-5}$).

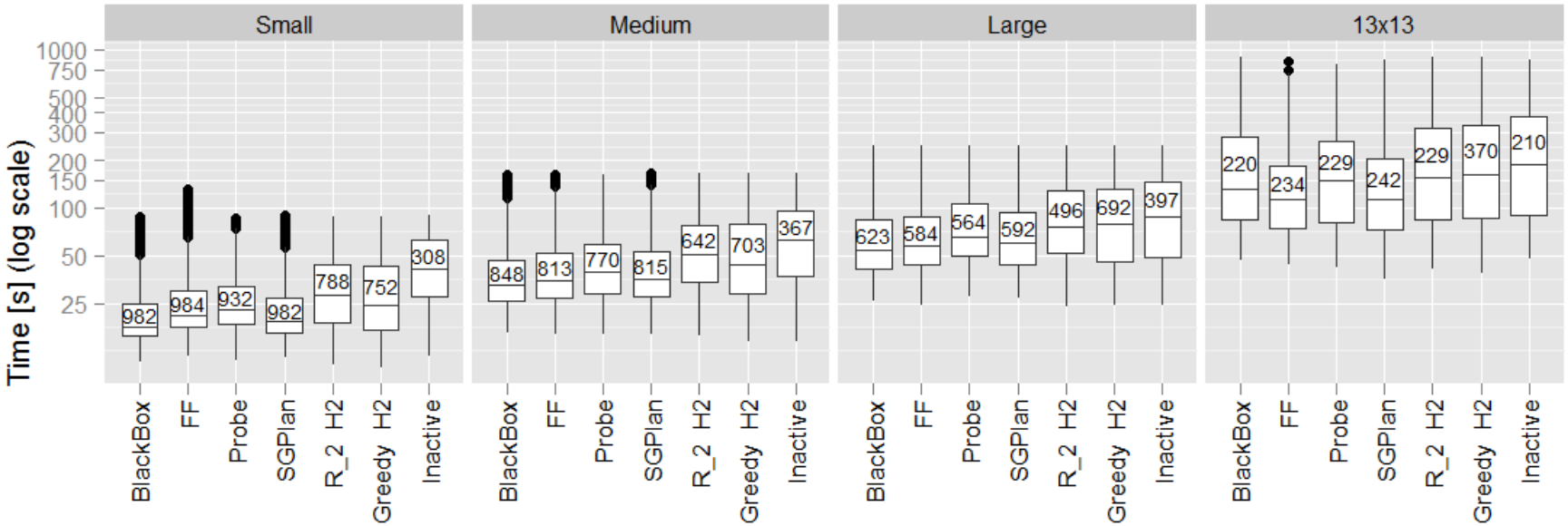


Figure 22: Solution times of agents in standard domains

While the results for solution time are favourable to the planners, they should be interpreted with caution, since the number of successful runs is very different among agents. Thus the longest times – the ones where the agent failed to reach the goal before timeout – are effectively not included. To remedy this, right-censored accelerated failure-time survival model [49] was fitted to the data using R package survival [50]. The error distribution was lognormal and multiple comparison was performed with Tukey contrasts [46]. The survival model takes all runs into account and is aware of the fact, that unsuccessful runs are to be interpreted as "time greater by unknown amount than timeout". The null hypothesis is that different agents would have the same mean solution time had the timeout been infinite. It however does not produce numbers that could be presented directly and only allows for partial ordering of agents.

For small maps, survival analysis reports relations very close to those reported by previous analysis. For medium maps, survival analysis also yields results similar to those above, but points out that all differences between agents are significant (all $p < 0.01$) except for SGPlan – FF.

For large maps, things are getting more interesting. According to survival model, Greedy is not significantly different than BlackBox, FF and SGPlan (all $p > 0.1$) while BlackBox is significantly better than all the other agents except Greedy (all $p \leq 0.001$). Probe is significantly worse than all the other planners and Greedy, but is still better than Rand2 and Inactive (all $p < 0.001$). Thus it is possible to say that although Greedy performed better in success rate than BlackBox, BlackBox made up for it in lower solution times.

For 13x13 maps, Greedy is significantly better than all other agents (all $p < 0.001$) and Inactive is significantly worse than other agents (all $p < 0.01$). FF and SGPlan are not significantly different and are better than all other except Greedy (all $p < 0.001$). Probe, BlackBox and Rand2 cannot be considered significantly different (all $p > 0.2$).

The results above can be summarized by assigning ranks to various agents as shown in Table 10.

Map	BlackBox	FF	Probe	SGPlan	Greedy	Rand2	Inactive
Small	***1***	3	3	***1***	5	5	7
Medium	***1***	2	4	2	5	6	7
Large	***1***	3	5	3	***1***	6	7
13x13	4	2	4	2	***1***	4	7

Table 10: Ordering of agents in standard domains according to survival model of solution time

The above results once again disprove Hypothesis 2 which states that planning agents will outperform all reactive agents – this only holds for smaller maps. But Hypothesis 4 (that Greedy is best among reactive agents) holds for solution time.

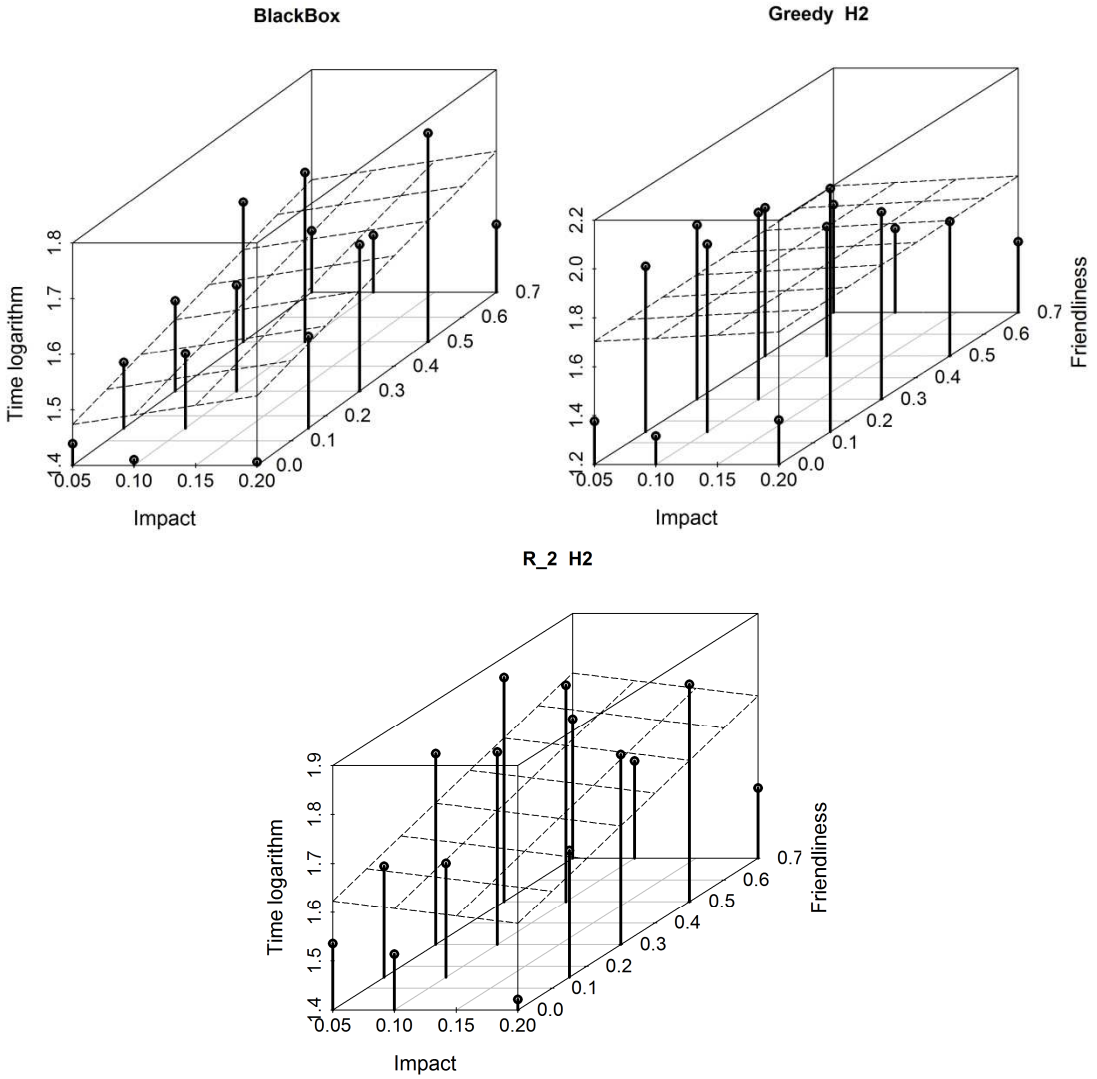


Figure 23: Mean solution time of agents for standard large maps under different dynamic conditions.
Planes are fitted to the averaged results and they are intended only as a visual cue.

4. 2. 5 Solution Time and Interference Parameters

The relation of solution time and interference parameters is not so clear as with success rate. However, some trends are visible. Figure 23 shows solution times for BlackBox, Greedy and Rand2 in large maps under different dynamic conditions. There is a change in trend between reactive and planning agents: while BlackBox has longer solution time with growing impact, Rand2 has shorter. Greedy agent stays somewhere in between with no clear trend visible. It is interesting, that for both BlackBox and Greedy, mean solution time grows with growing friendliness. This is most likely due to fact that unsuccessful runs are not accounted for. Thus

values for higher friendliness contain more runs, and the runs that were "added" in contrast to lower friendliness are likely those that are close to timeout, because if they had large margin from the timeout, they would likely finish even with lower friendliness. It is to be noted that plots for other map sizes look roughly the same.

There are also weaker, but similar trends for interference delay. However, the result provide no strong evidence that Hypothesis 3 holds, i.e. that the relative advantage of planning agents grows with growing friendliness and smaller delays.

4. 2. 6 Rooms Travelled

The average number of rooms travelled (see Table 11) strongly favours Inactive bot. This is to be expected, since Inactive bot is programmed not to travel much. Actually, for smaller maps, Inactive bot rarely walks more than the minimal path from start to goal. Since this is a secondary metric and the results are already quite skewed because only successful runs are included, the results will not be discussed in more detail. But it is to be noted that most of the differences are statistically significant. The statistics to analyse rooms travelled are the same as for solution time, refer to preceding section for details.

Map	BlackBox	FF	Probe	SGPlan	Greedy	Rand2	Inactive
Small	9.2 (4)	9.7 (4)	12.3 (5)	9.7 (4)	10.5 (6)	17.0 (10)	***8.2 (2)***
Medium	17.4 (7)	18.4 (8)	21.8 (9)	18.8 (8)	16.9 (6)	32.4 (18)	***13.4 (3)***
Large	28.3 (11)	29.5 (12)	35.1 (14)	30.0 (12)	26.8 (9)	50.2 (28)	***21.2 (5)***
13x13	37.1 (14)	46.5 (21)	56.6 (32)	53.2 (28)	37.2 (10)	92.1 (69)	***31.8 (8)***

Table 11: Average rooms travelled of agents in standard domains with standard deviation (in brackets). Best results in each row are highlighted.

To accommodate for unsuccessful runs, survival analysis is performed. To stay closer to reality, data are modified so that unsuccessful runs are always considered as travelling through at least as much doors as is the minimal length of a path from start to goal. For the sake of brevity, only final ranking is shown as summarized in Table 12. If two agents have different rank, it means that they are significantly different at 0.05 level.

Taking survival model into account has brought results that are much more intuitive to understand. Inactive bot is still very good, but has lost its dominant position to Greedy and for small maps, even to planners. But take caution. While taking only successful runs into account favoured Inactive bot, survival model is very unfavourable to Inactive, especially for smaller maps. As seen in Table 11, Inactive frequently walks just the minimal distance, but the survival model treats unfinished

run as “something more than minimal distance”, which is likely to be unfair to Inactive in most cases.

Once again the Hypothesis 2 that planning agents will prevail was disproved, but Hypothesis 4 (that Greedy will outperform Inactive and Rand 2) holds.

Map	BlackBox	FF	Probe	SGPlan	Greedy	Rand2	Inactive
Small	***1***	***1***	5	***1***	4	7	5
Medium	***1***	***1***	6	***1***	***1***	7	***1***
Large	3	4	6	4	***1***	7	***1***
13x13	2	4	4	4	***1***	7	2

Table 12: Ordering of agents in standard domains according to survival model of rooms travelled

4. 2. 7 Planner Comparison

The differences among planning agents in deliberation time are not very large, except for 13x13 maps where the size of planning problem starts to be too large for BlackBox – this was already visible in the analysis of success rate and solution time where BlackBox loses its dominant position among planners for 13x13 maps. Indeed, for small maps, the only significant difference is that FF is worse than BlackBox ($p = 10^{-4}$) and SGPlan ($p = 0.007$). Among medium maps the only significant difference is that Probe is worse than BlackBox ($p = 0.002$). For large maps the differences start to be sharper: while SGPlan, BlackBox and FF are not significantly different (all $p > 0.95$), Probe is significantly worse than all of them (all $p < 10^{-4}$). And for 13x13 maps, all differences are significant (all $p < 10^{-6}$). See Table 13 and Figure 24 for actual results.

Thus for all but the largest maps, Hypothesis 5 (different planners have different performance) does not hold.

Map Size	BlackBox	FF	Probe	SGPlan
Small	***0.21 (0.19)***	0.24 (0.21)	0.23 (0.21)	0.22 (0.20)
Medium	***0.25 (0.24)***	0.26 (0.26)	0.27 (0.25)	0.27 (0.25)
Large	***0.37 (0.31)***	0.38 (0.30)	0.41 (0.30)	***0.37 (0.31)***
13x13	0.78 (0.28)	0.50 (0.36)	0.62 (0.33)	***0.43 (0.36)***

Table 13: Deliberation time of planning agents in standard domains with their standard deviation (in brackets). Best results for each map size are highlighted.

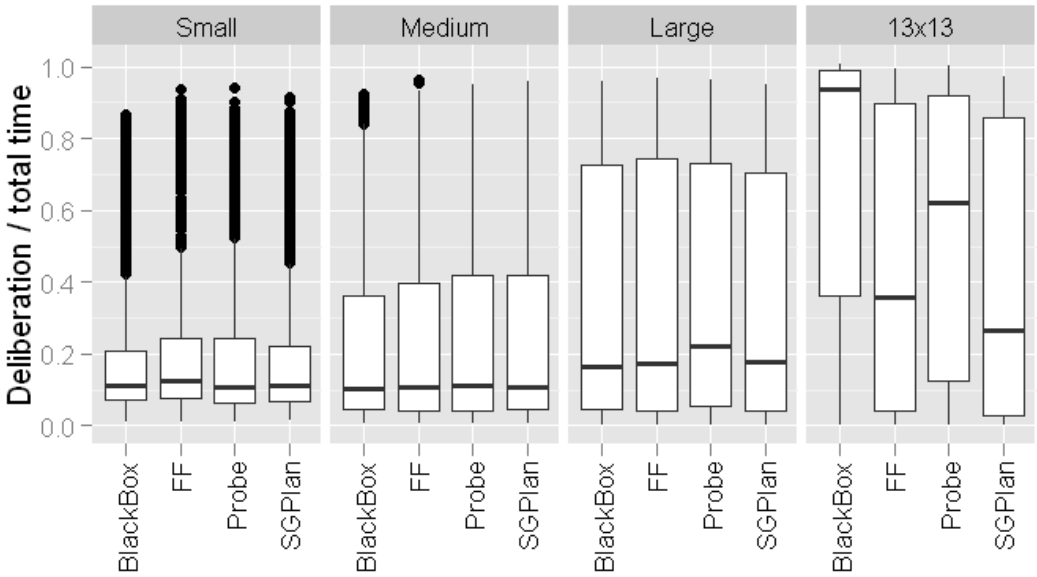


Figure 24: Deliberation time of planning agents in standard domains grouped by map size

Average plan length is an image of quality of plans generated by different planners. For all map sizes the statistical situation is the same: all differences are significant (all $p < 0.005$) except for SGPlan – FF which performed similarly for all map sizes (all $p > 0.7$). The similarity between SGPlan and FF might follow from the fact that SGPlan is built upon FF. This is the only metric where there are large differences between different planners. Note that BlackBox keeps the shortest plan length in all map sizes, although it spends too much time generating them in 13x13 maps.

Map Size	BlackBox	FF	Probe	SGPlan
Small	***13.2 (2.8)***	13.7 (3.0)	16.4 (4.7)	13.8 (3.0)
Medium	***20.3 (5.6)***	22.6 (8.0)	27.7 (12.1)	22.5 (7.8)
Large	***28.1 (6.8)***	31.9 (8.0)	42.7 (14.8)	32.3 (8.2)
13x13	***52.2 (21.5)***	65.4 (23.0)	93.4 (47.8)	64.6 (23.6)

Table 14: Average plan length in standard domains with their standard deviation (in brackets). Best results for each map size are highlighted.

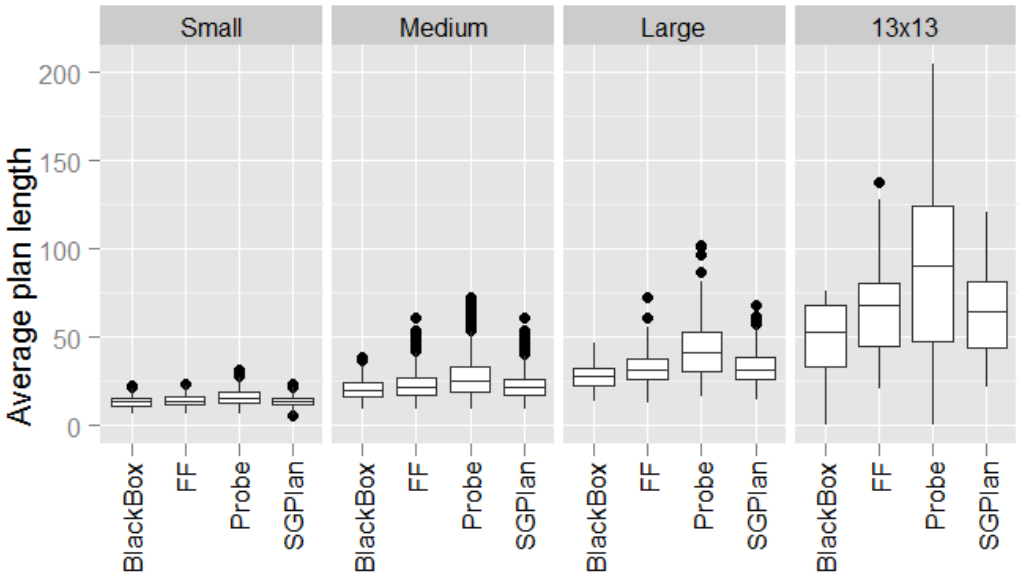


Figure 25: Average plan length of planning agents in standard domains

Overall there are surprisingly small differences in planner performance. SGPlan and FF stayed the closest, but BlackBox was close to them in all but the largest maps. Probe consistently took last place among planners in nearly all metrics, although it performed very well in the International Planning Competition 2011. The winner of the competition, LAMA 2011, was not even included in the final runs, because its response time was too long. This obviously follows from the fact that the latest planning competition was not designed to favour quick response for simple problems, but focused instead on difficult problems with response time in minutes.

4. 3 Relaxed Domains

In this section, the results for relaxed domains are provided accompanied with a series of plots that demonstrate patterns in the gathered data. In each subsection an informal discussion of the results is followed by rigorous statistical analysis and discussion of hypotheses introduced in section 3. 9.

The statistical methods are exactly the same as in corresponding analysis in section 4. 2, so see there for further statistical details.

4. 3. 1 Overall Results

Examining all the results from experiments in relaxed domains together shows that Greedy has the best success rate of all (see Figure 26). More formally Inactive is significantly worse than all other agents (all $p < 0.01$) and ANA* is worse than all other except Inactive (all $p < 0.01$). Greedy is better than all other agents (all

$p < 0.01$). Most of the other differences are not significant, except that Rand2 is significantly worse than FF, BlackBox and SGPlan.

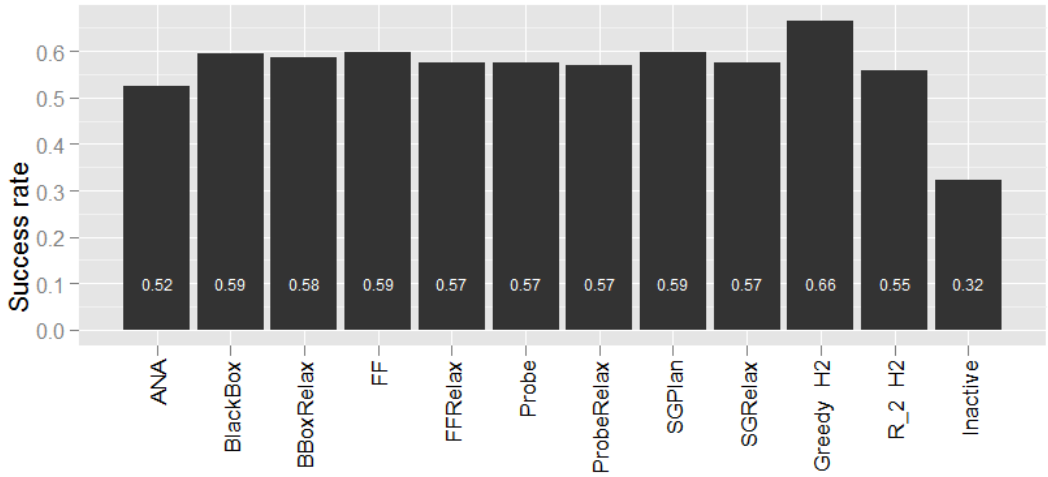


Figure 26: Success rate of agents in relaxed domains

The situation with deliberation time (see Figure 27) is more complicated than that in relaxed domains. ANA* is clearly the worst with average deliberation time of 0.57, followed by BlackBox (0.45), Probe (0.44) and ProbeRelax (0.43) all other planning agents had average deliberation time close to 0.35. Of course the deliberation time is dominated by reactive agents. The differences among BlackBox, Probe and ProbeRelax are not significant, as are the differences among BlackBoxRelax, FF, FFRelax, SGPlan and SGPlanRelax. The differences among reactive agents are not significant as well. All other differences are significant (all $p < 10^{-6}$).

Note that the deliberation time of planning agents is actually greater than in standard domains. Either the relaxed domains were for some reason more difficult or the solution times have decreased faster than the time spent deliberating.

The effect of individual interference parameters is similar to standard domains. A generalized linear model for success rate shows that all interference parameters are significant with $p < 10^{-15}$ and that friendliness has the highest effect size. A linear model for logarithm of solution time shows similar values, except that the interference friendliness level of 0.5 is significant only at $p = 0.003$ and the effect size of interference delay is only slightly lower than that of interference impact. Thus all interference parameters are significant as stated by Hypothesis 1 which thus holds for relaxed domains.

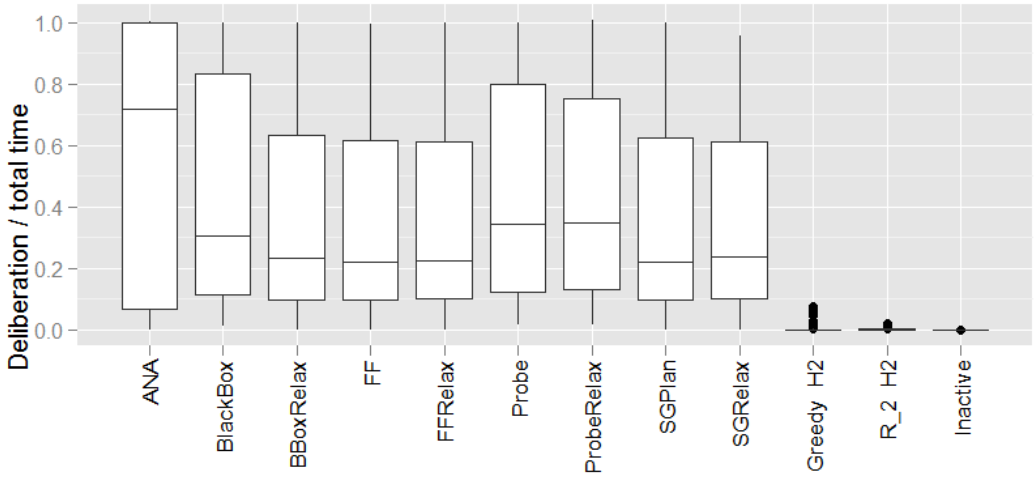


Figure 27: Deliberation time of agents in relaxed domains

A second order generalized linear model with interaction for success rate shows that the interaction of interference delay and agent type (reactive or planning) is significant (all $p < 0.018$) The interaction of interference friendliness and agent type is significant only for friendliness 0.3 and 0.5 and not for Inactive agent (all $p < 0.009$) However the latter interaction cannot be considered significant in general which contradicts Hypothesis 3.

A second order linear model for logarithm of solution time showed that the interactions of agent type with interference delay is significant except for Greedy agent (all $p < 0.03$). But the interaction agent type – friendliness is not significant, which contradicts Hypothesis 3.

4. 3. 2 Success Rate and Map Size

The success rate of agents at different map sizes is shown graphically in Figure 28 and summarized numerically in Table 15. Similarly to standard domains, the planning agents greatly outperform all reactive agents in small maps, but their performance degrades with growing map size, yielding lead position to Greedy agent for large maps. Most notably the performance of ANA* degrades very quickly. There are two possible reasons for this behaviour:

a) ANA* is written in Java and this might be a serious performance hit compared to other planners written in C or
b) the trouble is in the algorithm – while ANA* is specialized for relaxed domains and has better theoretical bound on complexity, it is possible that problems in relaxed domains (both in relaxed and standard formalism) happen to be simple for standard planners and thus they exhibit similar performance in practice.

Map size	ANA*	BlackBox	BlackBox Relax	FF	FF Relax	Probe	Probe Relax	SGPlan	SGPlan Relax	Greedy	Rand2	Inactive
Small	0.86	***0.87***	0.83	***0.87***	0.83	0.83	0.83	0.86	0.83	0.80	0.82	0.31
Medium	0.49	***0.70***	0.66	0.69	0.64	0.64	0.64	0.67	0.65	0.60	0.59	0.30
Large	0.36	0.42	0.43	0.42	0.39	0.40	0.40	0.43	0.39	***0.53***	0.39	0.28
13x13	0.37	0.38	0.41	0.41	0.41	0.40	0.40	0.42	0.41	***0.70***	0.42	0.37
Total	**0.52**	**0.59**	**0.58**	**0.59**	**0.57**	**0.57**	**0.57**	**0.59**	**0.57**	***0.66***	**0.55**	**0.32**

Table 15: Success rate of agents in relaxed domains with differing map size

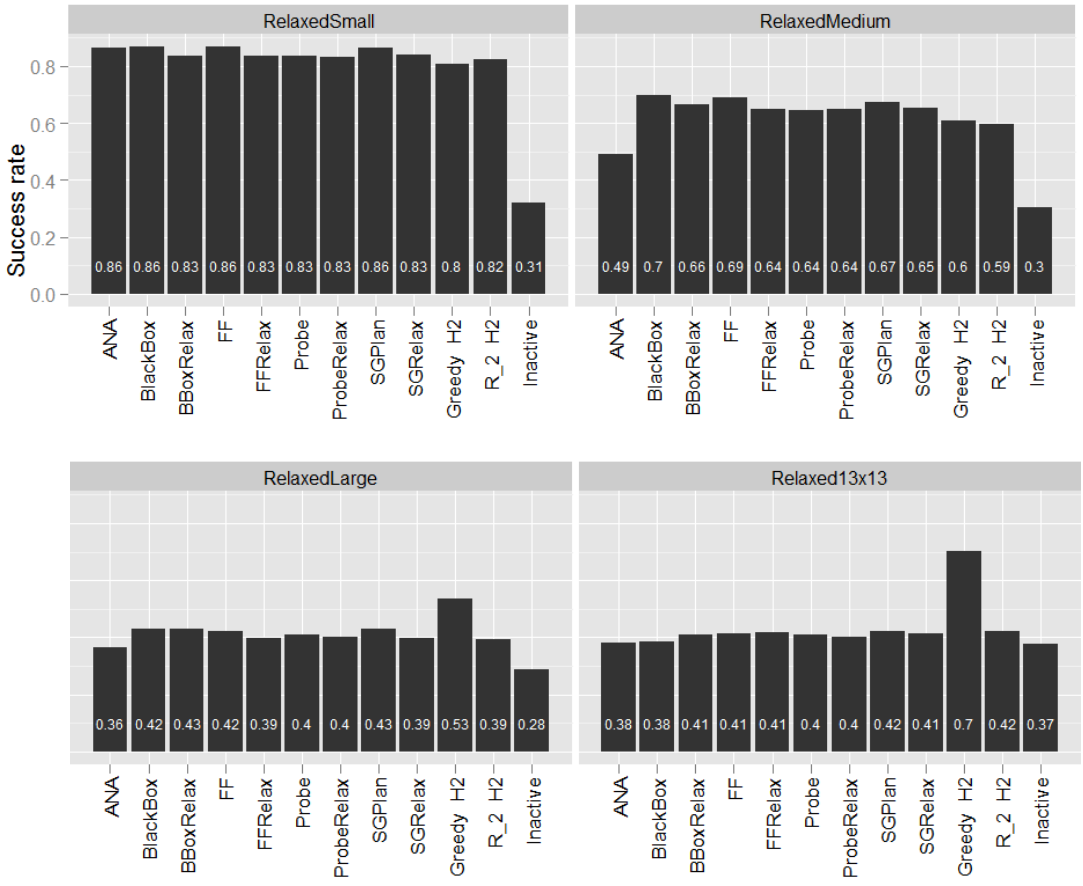


Figure 28: Success rate of agents in relaxed domains with differing map size

The relaxed variants of standard planners almost never beat their standard counterparts. But in the case of BlackBoxRelax and FFRelax their performance degrades slower with growing map size, and at the same time they have lower mean deliberation time than their standard counterparts. This hints us that BlackBox and FF planners solve the relaxed problems faster, but are slowed down by the difficulties of interpreting a relaxed plan, as described in section 2. 4. 4.

While the success rate of planning agents decreases with the growing map size, the success rate of reactive agents behaves differently. This is due to the different timeout values – the growth of timeout was designed to be proportional to degrading planner performance, but reactive agents degrade slower and thus their success rate may grow.

On small maps Greedy is significantly worse than ANA*, BlackBox, FF and SGPlan (all $p < 0.01$) and Inactive is significantly worse than all other agents (all $p < 0.01$). Other differences are not significant.

On medium maps Inactive is significantly worse than all other agents (all $p < 0.01$) and ANA* is worse than all other except Inactive (all $p < 0.01$). Greedy and Rand2 are significantly worse than BlackBox, BlackBoxRelax, FF and SGPlan (all

$p < 0.048$), in addition Rand2 is worse than FFRelax and SGPlanRelax (both $p < 0.047$), but for Greedy this difference is not significant.

On large maps Inactive is significantly worse than all other agents (all $p < 0.01$) and Greedy is significantly better than all other agents (all $p < 0.01$). Moreover ANA* is worse than BlackBox, BlackBoxRelax, FF, Probe and SGPlan (all $p < 0.017$). Other differences are not significant.

On 13x13 maps Greedy is better than all other agents (all $p < 0.01$) and Inactive is worse than BlackBoxRelax, FF, FFRelax, SGPlan, SGPlanRelax, Probe and Rand2 (all $p < 0.039$). ANA* and BlackBox are both worse than Rand2, SGPlan and FFRelax (all $p < 0.015$). ANA* is worse than FF and SGPlanRelax (both $p < 0.039$).

The results obtained are very similar to those from standard domains. Planning agents perform better than reactive only for small and medium maps. But there is an exception: Rand2 is among the top performing agents in small maps. This is probably due to fact that on relaxed maps, pushing any button can only open ways to reach goal and on small maps there are not so many buttons so the chance of randomly pushing the correct ones is high.

Except for ANA* there are also nearly no significant differences in planner performance, including differences between relaxed and standard variants of planners, which does not prove Hypothesis 5 and Hypothesis 6.

Hypothesis 4 (Greedy will outperform both Random and Inactive bots), does hold except for small maps, where Random is insignificantly better than Greedy.

Comparing the actual numbers to those for standard domains (see page 44), shows several interesting points:

1. The greatest difference is in small maps.
2. Inactive agent has the same performance in all maps – this is not a surprise, it only waits for interference which acts the same in both map types.
3. For medium maps and larger, agents have only slightly better or even worse performance in relaxed maps. This hints that relaxed maps are not much simpler for planners to handle and simultaneously the H2 heuristic does not bring significantly better advantage in relaxed maps. But it is also possible, that relaxed maps were more difficult by chance (random generation) alone, or that the different parameters of random generation for relaxed maps had some unpredicted impact.

4. 3. 3 Success Rate and Interference Parameters

The overall trends that affect success rate are similar to those in standard domains – planning agents along with Rand2 perform better with growing delay, while Greedy and Inactive perform worse. See Figure 29 and Figure 30.

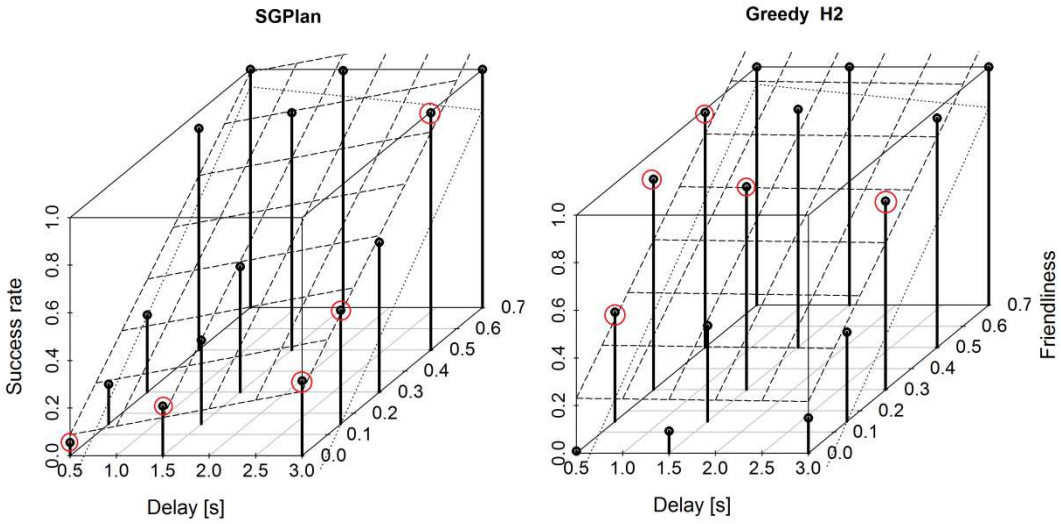


Figure 29: Success rate of SGPlan and Greedy bot in different dynamic conditions in relaxed domains.
The dotted lines show a plane fitted to the results of the Inactive bot. Planes are fitted to the averaged results and they are intended only as a visual cue. Red circles mark points where the respective agent is significantly better than the other (all $p < 0.05$).

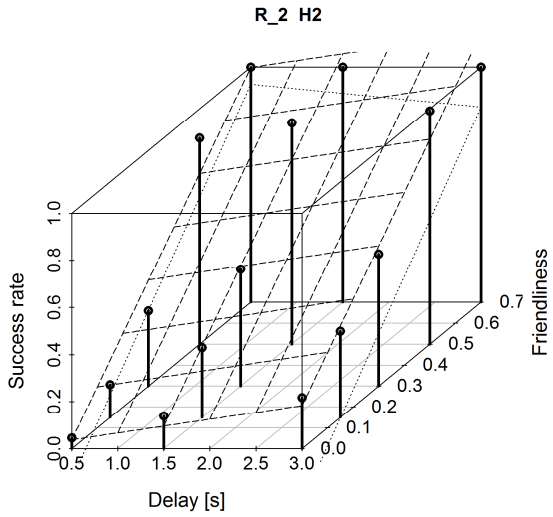


Figure 30: Success rate of Rand2 bot in different dynamic conditions in relaxed domains.

This data informally supports Hypothesis 3, that planning agents gain more advantage with greater delay and lower friendliness.

Comparing performance of planning agents (SGPlan was again taken as a representative) on pairs of impact and delay values that lead the same mean door changes per second shows similar results to standard domains (see page 47): SGPlan acts much better with longer delay and larger impact, while Greedy acts better with shorter delay and smaller impact. In accord with previous observations, Rand2 reacts to changing interference similarly to planning agents. See Table 16 for actual values.

Agent	1.5s x 0.1	3.0s x 0.05	1.5s x 0.2	3.0s x 0.1
SGPlan	0.63	***0.75***	0.53	***0.68***
Greedy	***0.66***	0.61	***0.69***	0.64
Rand2	0.56	***0.67***	0.51	***0.62***

Table 16: Comparing the importance of interference impact and delay in relaxed domains

4. 3. 4 Solution Time

The mean solution times and their standard deviations are presented numerically in Table 17 and graphically in Figure 31.

For small maps, Inactive is significantly worse than all other agents (all $p < 10^{-6}$). Rand2 and Greedy are significantly worse than all planning agents (all $p < 10^{-6}$). BlackBox, FF, SGPlan and ANA* are all better than BlackBoxRelax, ProbeRelax, FFRelax and SGPlanRelax (all $p < 0.004$). In addition BlackBox and FF are better than Probe (both $p < 0.004$) and Probe is better than SGRelax ($p = 0.03$).

For medium maps all differences among FF, BlackBox and SGPlan are not significant (all $p > 0.88$) as well as all differences among SGPlanRelax, Probe, ProbeRelax, BlackBoxRelax and FFRelax (all $p > 0.7$). Also Rand2 is not significantly different from both Greedy and ANA* (both $p > 0.2$). But all other differences are significant (all $p < 0.002$, except SGPlan – Probe where $p = 0.02$).

For large maps all differences between a reactive agent and a planning agent are significant (all $p < 10^{-4}$). In addition all differences among reactive agents are significant as well and ANA* is significantly different from all other planning agents (all $p \leq 10^{-4}$). Among the rest of the planners, only ProbeRelax is significantly different from SGPlan, FF and BlackBox (all $p < 0.015$).

For 13x13 maps, Inactive is worse than all other agents (all $p < 0.004$). SGPlan, FF, FFRelax, SGRelax and BlackBoxRelax are all significantly better than the rest of the agents (all $p < 10^{-6}$), while the differences among them are not (all $p > 0.11$). ProbeRelax and Greedy are not significantly different from each other, but are better than ANA*, BlackBox and Rand2. The last significant difference is that Probe is better than ANA* and Rand2 (all $p < 10^{-4}$).

Map size	ANA*	BlackBox	BlackBox Relax	FF	FF Relax	Probe	Probe Relax	SGPlan	SGPlan Relax	Greedy	Rand2	Inactive
Small	25.6 (18)	24.6 (18)	27.3 (20)	***24.5 (18)***	30.0 (21)	27.5 (19)	29.5 (20)	24.8 (19)	30.1 (20)	29.7 (24)	30.5 (21)	56.9 (32)
Medium	65.2 (35)	***43.9 (26)***	52.2 (31)	45.7 (28)	54.3 (31)	52.9 (29)	56.4 (32)	45.3 (27)	51.9 (29)	58.2 (36)	58.7 (36)	76.0 (37)
Large	98.9 (58)	***77.8 (47)***	85.8 (49)	79.2 (46)	90.1 (50)	87.7 (51)	95.4 (49)	82.0 (49)	89.1 (50)	99.0 (56)	93.9 (52)	107.8 (57)
13x13	272 (217)	251 (195)	201 (176)	186 (174)	206 (184)	253 (207)	250 (197)	***180 (158)***	213 (188)	265 (214)	227 (207)	273 (215)

Table 17: Average solution times [s] of agents in relaxed domains with standard deviation (in brackets). Best results in each row are highlighted.

Map size	ANA*	BlackBox	BlackBox Relax	FF	FF Relax	Probe	Probe Relax	SGPlan	SGPlan Relax	Greedy	Rand2	Inactive
Small	***1***	***1***	5	***1***	5	5	5	***1***	5	10	10	12
Medium	11	***1***	4	***1***	4	4	4	***1***	4	9	9	12
Large	9	2	2	2	6	6	9	2	6	***1***	9	12
13x13	-	-	-	2	4	-	-	2	4	***1***	-	-

Table 18: Ordering of agents in relaxed domains according to survival model of solution time

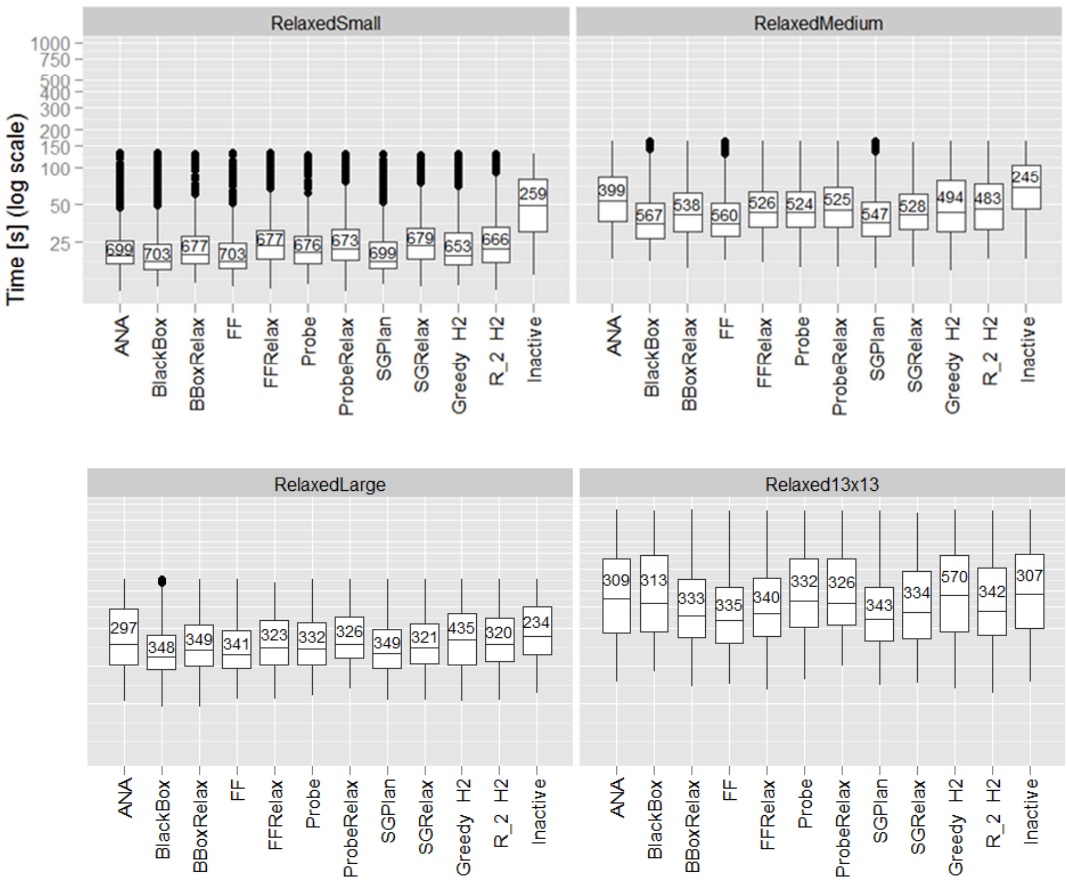


Figure 31: Solution time of agents in relaxed domains with differing map size

Those results favour standard planners, but relaxed planners are only slightly better or even worse (ProbeRelax) than Rand2. Interestingly enough, Rand2 is better than Greedy. But as in standard domains, those results must be interpreted cautiously. Survival analysis should give better answers – its results follow.

For small maps Inactive is worse than all other agents (all $p < 0.01$). FF, ANA*, BlackBox and SGPlan are better than all other (all $p < 0.016$). Differences among the rest of the planners are not significant (all $p > 0.27$) but all the planners are significantly better than Greedy and Rand2 (all $p < 0.19$), while the difference between Greedy and Rand2 is not significant ($p > 0.99$). This is in accord with both success rate and solution time results.

For medium maps the results are almost the same as for small maps, except that ANA* is now worse than all other except Inactive (all $p < 0.01$).

For large maps the Inactive agent is once again worse than all other agents (all $p < 0.01$). Greedy is significantly better than all other agents (all $p < 0.01$) except BlackBox ($p = 0.08$). BlackBox is not significantly different than BlackBoxRelaxed, FF

and SGPlan (all $p > 0.74$) but is better than the rest of the agents except Greedy (all $p < 0.01$). SGPlan and FF are both better than SGPlanRelax, ANA*, FFRelax, SGPlanRelax, Rand2 and ProbeRelax (all $p < 0.01$) but are not significantly different from each other and from BlackBoxRelaxed (all $p > 0.99$). In addition SGPlan is better than Probe ($p = 0.03$). BlackBoxRelaxed is better than ProbeRelax, SGPlanRelax, Rand2, ANA* and FFRelax (all $p \leq 0.01$). The only remaining significant differences are that ANA* is worse than FFRelax, Probe and SGRelax (all $p \leq 0.01$). Since there are many overlapping sets of agents that are not significantly different from each other, ordering the agent performance based on this results is questionable.

For 13x13 maps the analysis showed some counter-intuitive results. For example it put ANA*, BlackBox and Inactive among the top agents – but those had both worse mean solution time and success rate. A closer look at the data showed a possible cause. While the solution times for smaller maps are reasonably normal, ANA*, BlackBox, Probe, ProbeRelaxed and Rand2 and Inactive agents show slightly bimodal characteristics for 13x13 maps. However those agents were by no means top performers, so the analysis was repeated and problematic agents were left out. Among the rest of the agents, Greedy is significantly better than all other agents (all $p < 0.003$) and SGPlan and FF are better than both FFRelax and SGRelax (all $p < 0.001$).

The results of survival analysis can be summarized by assigning ranks to agents as shown in Table 18.

It is of interest, why the solution times are bimodal. If it happened for planning agents only, it would be possible, that due to extreme costs of planning in such a large domain, no more than two replannings could be done before the timeout in most cases. Thus there would emerge two large groups with significantly different mean solution time – those that required one replanning and those that required two. Such hypothesis has indeed large support in the data: BlackBox for example had two thirds of successful runs with at most two planner executions. Why such thing happened with Inacitve or Rand2 is however unclear.

The results of survival analysis are very similar to those from standard domains – while planners on large maps have much lower success rate than Greedy, they remedy most of it by shorter solution times.

The above results once again disprove Hypothesis 2 which states that planning agents will outperform all reactive agents – this only holds for smaller maps. But Hypothesis 4 (Greedy is best among reactive agents) holds for solution time. Also the Hypothesis 6 (relaxed variants of planners are worse than their standard counterparts) holds for most of the cases.

Comparing the actual values of solution time to those from standard domains shows similar properties as the comparison made for success rate – the absolute performance of agents in relaxed domains is only slightly better or quite often worse. This strengthens the suspicion that relaxed maps were by chance or different setting of random generation actually more difficult than the standard ones.

4. 3. 5 Solution Time and Interference Parameters

Unlike in standard domains, the trends in relaxed domains for solution time in combination with interference parameters are the same for all types of agents: the performance is generally worse with growing delay. This might be due to fact that in relaxed domains H2 heuristic starts to be quite goal oriented, or it might be the result of the fact, that only successful runs are included, which could have skewed the results. See Figure 23.

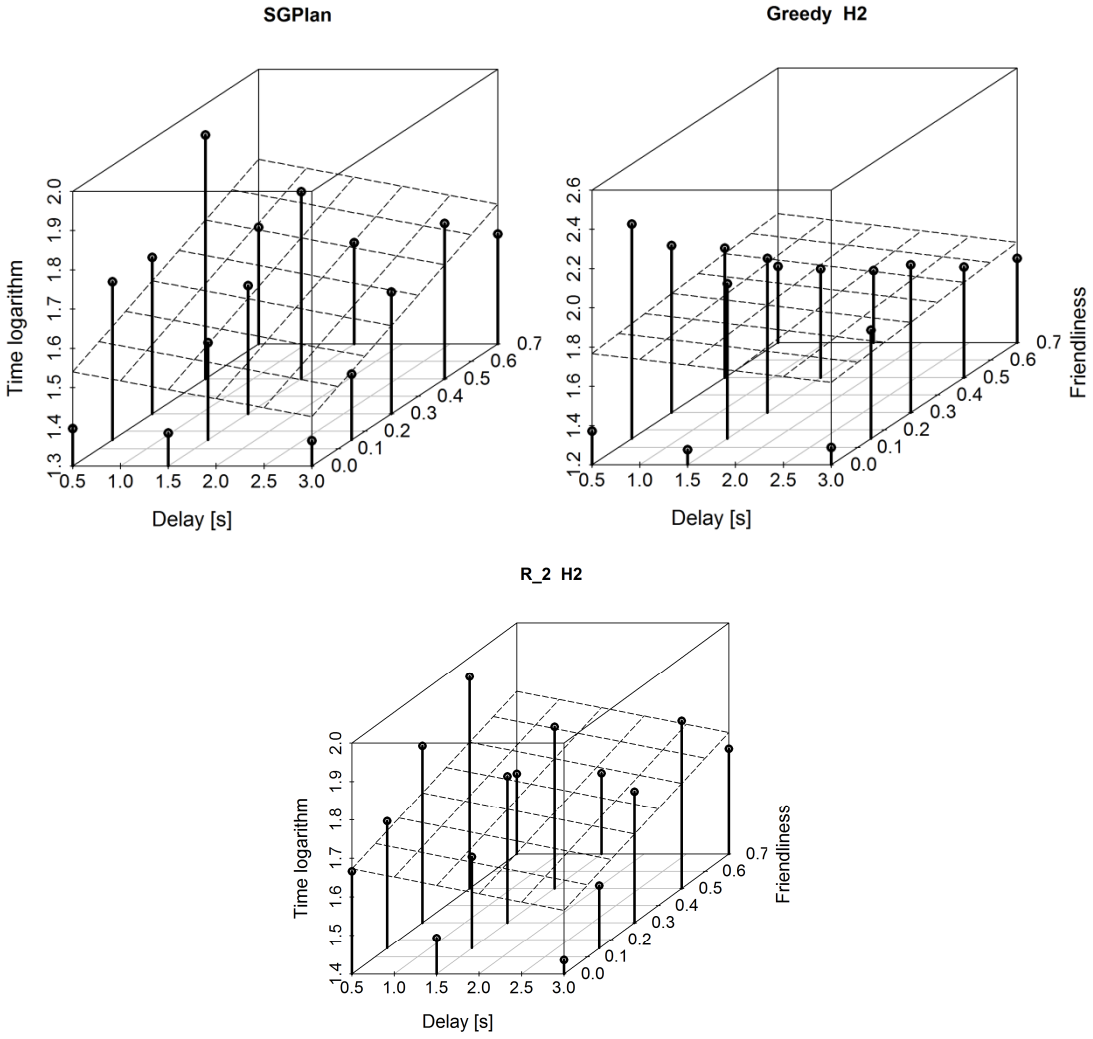


Figure 32: Mean solution time of agents for standard large maps under different dynamic conditions in relaxed domains.
Planes are fitted to the averaged results and they are intended only as a visual cue.

There are also weaker, but similar trends for interference impact. The results provide no evidence that Hypothesis 3 holds, i.e. that the relative advantage of planning agents grows with growing friendliness and smaller delays.

4. 3. 6 Rooms Travelled

Quite unsurprisingly, the average number of rooms travelled (Table 19) favours Inactive agent – similarly to standard domains. For the sake of brevity note, that most of the differences (basically all greater than 2) are statistically significant. The performance of ANA* converges to that of Inactive with growing map size – this is because with growing deliberation time, the ANA* is more and more controlled solely by heuristic H1.

The results of survival analysis of rooms travelled are shown in Table 20. Different rank of two agents indicates that they are significantly different at 0.05 level.

The results support Hypothesis 6 – relaxed variants of planners are significantly different than their standard counterparts.

Map size	ANA*	BlackBox	BlackBox Relax	FF	FF Relax	Probe	Probe Relax	SGPlan	SGPlan Relax	Greedy	Rand2	Inactive
Small	11.2 (4)	9.1 (3)	10.0 (4)	8.9 (3)	12.3 (5)	11.0 (4)	12.7 (4)	9.0 (3)	12.3 (5)	9.4 (6)	13.9 (11)	***8.2 (2)***
Medium	17.7 (7)	17.6 (6)	21.4 (9)	18.1 (7)	24.7 (10)	23.9 (10)	23.0 (9)	18.7 (7)	23.2 (9)	16.3 (6)	25.7 (14)	***13.1 (3)***
Large	25.4 (9)	30.6 (13)	34.4 (16)	32.0 (14)	40.8 (20)	37.2 (16)	35.5 (17)	32.3 (14)	39.6 (18)	26.3 (8)	45.1 (24)	***21.3 (4)***
13x13	31.9 (8)	36.5 (16)	59.0 (39)	47.6 (26)	67.8 (46)	48.0 (31)	52.2 (35)	51.3 (29)	69.3 (50)	37.7 (12)	80.1 (61)	***31.6 (9)***

Table 19: Average rooms travelled of agents in relaxed domains with standard deviation (in brackets). Best results in each row are highlighted.

Map size	ANA*	BlackBox	BlackBox Relax	FF	FF Relax	Probe	Probe Relax	SGPlan	SGPlan Relax	Greedy	Rand2	Inactive
Small	6	***1***	5	***1***	8	6	8	***1***	8	***1***	12	8
Medium	***1***	***1***	7	***1***	8	8	8	***1***	8	***1***	12	***1***
Large	***1***	4	4	4	8	8	8	4	8	***1***	12	***1***
13x13	2	2	5	5	10	5	5	5	10	***1***	12	2

Table 20: Ordering of agents in relaxed domains according to survival model of rooms travelled

4. 3. 7 Planner Comparison

The deliberation time of planning agents in relaxed domains in various map sizes is shown in Figure 33 and Table 21. Interestingly enough, ANA* showed clearly the best performance in small maps, but degrades very quickly. For larger maps the relaxed variants of FF and BlackBox gain advantage over their standard variants, however this does not seem to apply to SGRelax and Probe. This might be due to fact, that BlackBox and FF use a very straightforward planning approach, while SGPlan and Probe are more sophisticated, which could have allowed them to exploit the relaxed nature of the problem even in standard formalism. Significance analysis follows.

In small maps ANA* is better than all other planners (all $p < 10^{-6}$) and ProbeRelax is significantly better than SGPlanRelax and Probe (both $p < 0.04$). Other differences are not significant.

For medium maps, ANA* is worst of all (all $p < 10^{-6}$) and ProbeRelax is worse than BlackBox, FF and SGPlan (all $p < 0.003$). Other differences are not significant.

In large maps, ANA* is worst of all (all $p < 10^{-6}$) and ProbeRelax is second worst (all $p < 0.014$). The last significant difference is that BlackBoxRelax is better than Probe and BlackBox (both $p < 0.002$).

For 13x13 maps, ANA* and BlackBox are worse than the rest of agents (all $p < 10^{-6}$), but not significantly different from each other. Most of the other differences are significant as well ($p < 0.012$) except that FFRelax is not different from SGRelax and SGPlan, SGRelax is not different from SGPlan and BlackBoxRelax, and BlackBoxRelax is not different from SGPlan and FF.

Thus for all but the largest maps, Hypothesis 5 (different planners have different performance) does not hold.

Comparing average plan length (see Table 22 and Figure 34) shows that ProbeRelax consistently finds the shortest plan. Also planners based on Fast Forward (all but BlackBox) find shorter plans in relaxed formalism than they do in standard formalism. This is probably because visiting a location twice does not incur longer plan in relaxed formalism, while it does so in standard formalism. It is uncertain, why BlackBox was not able to exploit this.

Among standard variants of planners, BlackBox finds the shortest plans (similarly to standard domains).

Most of the differences (basically all greater than 10%) are significant at 0.05 level.

Map size	ANA*	BlackBox	BlackBox Relax	FF	FF Relax	Probe	Probe Relax	SGPlan	SGPlan Relax
Small	***0.05 (0.06)***	0.19 (0.18)	0.20 (0.19)	0.19 (0.18)	0.20 (0.18)	0.21 (0.19)	0.18 (0.18)	0.20 (0.19)	0.20 (0.18)
Medium	0.72 (0.29)	***0.27 (0.23)***	0.29 (0.24)	0.28 (0.23)	0.29 (0.24)	0.29 (0.25)	0.31 (0.24)	0.28 (24)	0.29 (0.24)
Large	0.62 (0.41)	0.46 (0.31)	***0.41 (0.30)***	0.43 (0.37)	0.43 (0.33)	0.45 (0.30)	0.49 (0.28)	0.43 (0.30)	0.44 (0.30)
13x13	0.89 (0.18)	0.87 (0.24)	0.50 (0.33)	0.51 (0.37)	***0.46 (0.33)***	0.80 (0.25)	0.74 (0.26)	0.48 (0.34)	0.47 (0.33)

Table 21: Deliberation time of planning agents in relaxed domains with their standard deviation (in brackets). Best results for each map size are highlighted.

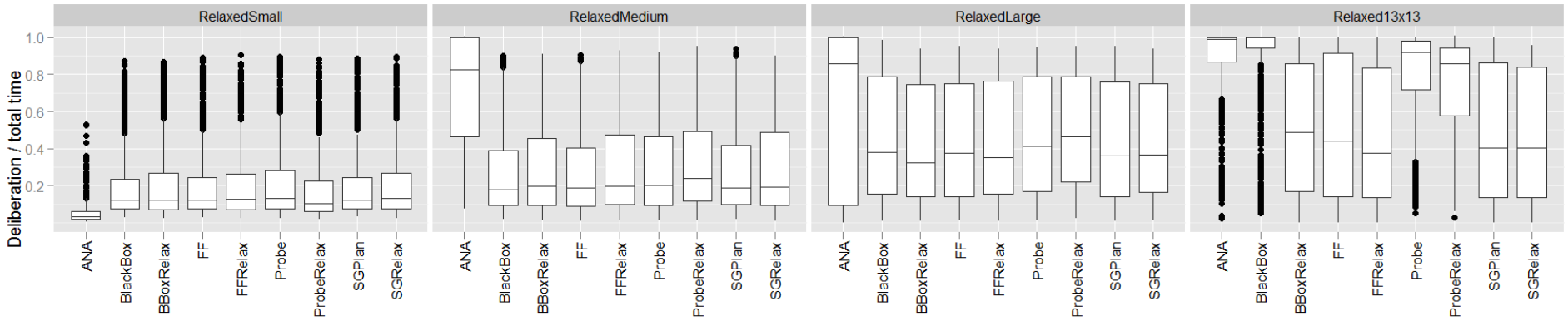


Figure 33: Deliberation time of planning agents in relaxed domains with different map size.

Map size	ANA*	BlackBox	BlackBox Relax	FF	FF Relax	Probe	Probe Relax	SGPlan	SGPlan Relax
Small	11.6 (2.4)	12.2 (2.2)	12.8 (2.5)	12.4 (2.2)	12.2 (2.3)	14.6 (3.9)	***11.5 (2.4)***	12.4 (2.2)	12.0 (2.4)
Medium	18.7 (5.3)	20.2 (7.2)	20.3 (5.4)	21.5 (8.3)	18.5 (4.5)	27.3 (11.0)	***18.2 (4.7)***	21.9 (2.2)	18.4 (2.4)
Large	32.1 (7.3)	33.1 (8.9)	33.0 (8.2)	37.3 (10.0)	31.4 (8.0)	51.2 (18.2)	***30.0 (8.0)***	38.2 (11.7)	31.4 (8.0)
13x13	55.6 (15.4)	54.2 (21.8)	55.3 (19.3)	78.9 (36.2)	53.0 (20.4)	114.7 (57.6)	***51.2 (19.5)***	78.6 (37.7)	52.5 (19.6)

Table 22: Average plan length in relaxed domains with their standard deviation (in brackets). Best results for each map size are highlighted.

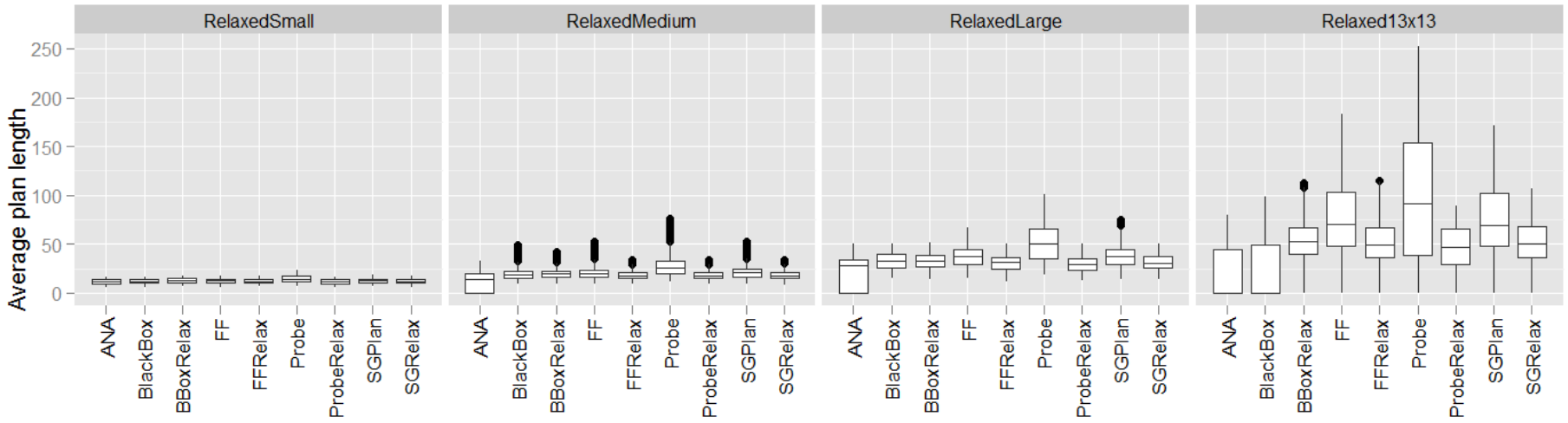


Figure 34: Average plan length of planning agents in relaxed domains

4. 4 Hypothesis Summary

In section 3. 9, six hypotheses were formulated. This section summarizes the results in this chapter with regard to the hypotheses.

Hypothesis 1 stated that all interference parameters are significant for agent performance. This hypothesis holds.

Hypothesis 2 states that planning agents will outperform reactive ones. For success rate this is true only for small and medium maps. Planning agents are better in solution time for all map sizes, but when survival analysis is introduced to remedy for unsuccessful runs, they are better only for small and medium maps. In rooms travelled, Inactive agent is the best. If survival analysis is introduced, planners are comparable (but not better) for small and medium maps.

Hypothesis 3 states that relative advantage of planning agents will be greater with lower friendliness and greater delay. This was partly confirmed for success rate, but not for solution time. Some of the respective interactions in statistical models were not significant.

Hypothesis 4 states that Greedy will be the best reactive agent. This hypothesis holds.

Hypothesis 5 states that planners will have different performance. This hypothesis can be rejected. The only metric where planners do differ is average plan length. The exception is ANA* that performed significantly different on most maps and BlackBox that performed under the average for 13x13 maps.

Hypothesis 6 stated that relaxed variants of planners will perform worse than their standard counterparts. Hypothesis holds only for solution time and rooms travelled, the differences are not significant for success rate.

5 Conclusions and Future Work

This chapter presents conclusions drawn from experimental results and suggests future work.

5. 1 Conclusions

Overall, all thesis goals were met and sufficient data was gathered to answer the main questions from section 1. 1. Following text summarizes those results.

In order to connect planners to Pogamut platform, Planning4J library was created. Planning4J allows for easy connection of IPC planners to Java and provides a flexible API for any classical planner in general.

A class of flexible test environments running in UDK was created along with tools too randomly generate new instances. Different planners were then thoroughly tested on those environments under different dynamic conditions.

5. 1. 1 Planner Performance

The most important conclusion is that in small or hostile or less dynamic domains, the contemporary planning algorithms are fast enough to provide advantage over the reactive approaches. This advantage grows with growing hostility, lower interference and smaller domain sizes. The perceived limits of real-time applicability (planning faster than 1s) of contemporary planners are somewhere above hundred atoms and two hundred ground actions.

While it is still improbable that AI in a commercial game would be allowed to consume a whole processor core, it is likely that given today's gaming devices, solving problems with tens of predicates and actions in real-time will be easily manageable. Performance could be improved by a tighter integration of the planning component. Moreover, all tested planners seek optimal (shortest) plans. In most game scenarios, suboptimal plans would be sufficient which could greatly speed the search process up.

On the other hand, the results also explain why planning is not the first choice in IVA design. Unless the environment is either changing slowly or in an extremely hostile way, even a simple reactive approach might prove reasonably efficient. While planning is most effective for smaller domains, it is also easier to write specialized reactive agents for such domains. This reduces the possible gain from implementing a planning algorithm. It is also useful to know that the planner performance depends more on the interference delay than on interference impact.

In general, most of the planners showed nearly equal performance, except for LAMA 2011 which had too low response time for small problems that it was excluded from final comparison. SGPlan showed the most stable performance with regard to domain size.

It was also shown that small frequent changes are harder for planners to tackle than large infrequent changes.

Unfortunately, the question which approach is better for IVA design remains open. Our results do not show that either approach would be better in general case.

5. 1. 2 Relaxed Domains

While planning in relaxed domains is theoretically easier than planning in general domains, our results do not suggest that relaxed domains would bring any advantage. ANA*, a specialized planner for relaxed domains, performed very poorly – although this might be due to its implementation details. The relaxed formalism did not bring any advantage to IPC planners in most cases, partly due to more difficult interpretation of plans in relaxed formalism.

Although there are hints that planning in relaxed domains might scale better for larger domains, this was not proven even with the largest domains tested (481 ground predicates and over 1000 ground actions) which are quite large for regular game scenario.

5. 1. 3 Limitations and Shortcomings

There are nevertheless some limitations to the applicability of results of this thesis to the general case. Despite all measures taken to the contrary, the environment is still quite specific. The design of interferences made waiting in front of a door until it opens by chance – which is an important part of Greedy agent operation – a viable choice. But this is not a typical feature of a game scenario.

It is also possible that the simplicity of the environment (only two kinds of actions, simple goals) altered the results in some way. The map generation procedure might also have influenced agent performance in either way.

Another limitation is that only square maps were considered, which might have favoured the reactive approaches, because those are simpler to solve by greedily reducing the distance to goal.

5. 1. 4 Connecting Planners

An important side part of work on this paper was to connect classical planners to Java and the Pogamut platform with one universal API through the development of Planning4J open source library [40]. We hope that such a tool would help other

researchers cross the gap between planning and IVAs. Also this thesis forms a tested connection between the virtual world of UDK and IPC planners which should enable further research in this area.

Among the planners connected to Planning4J is also the RelaxPlan – a specialized planner for relaxed domains written in Prolog. It was not included in the experiments, because the connection did not work on 64bit Linux, but it works on both 32bit Linux and Windows.

5. 2 Future Work

Multiple possibilities for future research are available. It would be interesting to see if the given results scale to more extreme parameter values, larger maps and more complex domains. Involving only partially observable environment, non-deterministic actions or adding multiple goal-oriented adversarial or cooperative agents would also bring further insights.

Another research direction is to tightly integrate the planner with the agent. Interleaving planning and execution as well as meta-reasoning about the planning process and explicit handling of uncertainty in the world might bring a significant performance boost. The simplest way to directly handle uncertainty might be to create heuristics that would govern plan execution and replanning, while the planning process might stay the same.

The ANA* planner already has internal anytime capabilities, but those are not exposed to the user. Exploiting those could bring advantage to the ANA* planner. It would also be interesting to directly express the domains in SAS+ formalism, so that planners from Fast Downward platform would be competitive.

To decide whether the idea of relaxed domains brings any true benefit to planning in games it would be necessary to implement a relaxed planner fully in C++ so that its performance becomes comparable to other IPC planners.

To decide which approach is better for IVA design it is also important, what are the costs associated with implementing either one. Is it easier to develop a reactive agent or to construct a PDDL model? This is a qualitatively different analysis which needs to be performed.

Bibliography

1. RUSSEL, S. and NORVIG, P. *Artificial Intelligence: A Modern Approach*. 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 2002

2. CHAMPANDARD, A. J. Understanding Behavior Trees. *AiGameDev.com* [online]. 2007 [accessed 2012-04-12]. Available at: http://aigamedev.com/open/article/bt-overview/

3. FU, D. and HOULETTE-STOTTLER, R. The Ultimate Guide to FSMs in Games. In: RABIN, S. *AI Game Programming Wisdom II*. Hingham, Massachusetts: Charles River Media, 2004, pp. 283-302. ISBN 1-58450-289-4.

4. FIKES, R. E. and NILSSON, N. J. Strips: A new approach to the application of theorem proving to problem solving. 1971, vol. 2. 189-208.

5. JONSSON, P. and BÄCKSTRÖM, C. On the Size of Reactive Plans. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, sv. 1. Menlo Park (California): The AAAI Press, 1996, pp. 1182-1187. ISBN 0-262-51091-X.

6. POLLACK, M. E. and HORTY, J. F. There's more to life than making plans. 1999, vol. 20(4). 71-83. ISSN 0738-4602.

7. ORKIN, J. Three States and a Plan: The AI of F.E.A.R. In: *Proceedings of the Game Developer's Conference*. 2006

8. ORKIN, J. Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference. In: YOUNG, M. and LAIRD, J. *Agent Architecture Considerations for Real-Time Planning*. Menlo Park (California): The AAAI Press, 2005, pp. 105-110. ISBN 1-57735-235-1.

9. ORKIN, J. Goal-Oriented Action Planning (GOAP). *Jeff Orkin personal page* [online]. [accessed 2012-3-25]. Available at: http://web.media.mit.edu/~jorkin/goap.html

10. BARTÁK, R., et al. When Planning Should Be Easy: On Solving Cumulative Planning Problems. In: *PlanSIG*. 2011

11. GEMROT, J., et al. Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents. In: *Agents for Games and Simulations*. Springer, 2009, pp. 1-15. LNCS 5920.

12. EPIC GAMES, INC. *Unreal Development Kit* [online]. [accessed 2012-3-17]. Available at: http://www.udk.com/

13. PITTMAN, D. *Practical Development of Goal-Oriented Action Planning AI*. Dallas. Master Thesis. Southern Methodist University.

14. LONG, E. *Enhanced NPC Behaviour using Goal Oriented Action Planning*. Master Thesis. University of Abertay Dundee, School of Computing and Advanced Technologies, Division of Software Engineering. Available at: http:// www.edmundlong.com/Projects/ Masters_EnhancedBehaviourGOAP_EddieLong.pdf

15. PEIKIDIS, P. *StarPlanner: Demonstrating the use of planning in a video game*. Final year project. University of Sheffield, University of Sheffield. Available at: http://pekalicious.com/starplanner/report/peikidis.pdf

16. KAUTZ, H. and SELMAN, B. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. In: *Working notes of the Workshop on Planning as Combinatorial Search*. Pittsburgh: 1998

17. HOFFMANN, J. and NEBEL, B. The FF Planning System: Fast Plan Generation Through Heuristic Search. 2001, vol. 14. 253-302.

18. VASSOS, S. and M., P. The SimpleFPS planning domain: A PDDL benchmark for proactive NPCs. In: *Workshops at the Sevent Artificial Intelligence and Interactive Digital Entertainment Conference*. 2011

19. THOMPSON, T. and J., L. Realtime execution of automated plnas using evolutionary robotics. In: *IEEE Symposium on Computational Intelligence and Game*. 2009, pp. 333-340

20. FERNÁNDEZ, S., et al. Planning for an AI based virtual agents game. In: *Working notes of the ICAPS'06 Workshop on AI Planning for Computer Games and Synthetic Characters*. 2006, pp. 14-20

21. RODRIGUEZ-MORENO, M. D., et al. IPSS: A Hybrid Approach to Planning and Scheduling Integration. 2006, vol. 12. 1681-1695.

22. ARMANO, G., CHERCHI, G. and VARGIU, E. An Agent Architecture for Planning in a Dynamic Environment. In: *AI*IA 2001: Advances in Artificial Intelligence*. Berlin: Springer, 2001, pp. 388-394. ISBN 978-3-540-42601-1.

23. FOX, M. and LONG, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. 2003, vol. 20. 61-124. ISBN 1076 - 9757.

24. KNOBLOCK, C. Search Reduction in Hierarchical Problem Solving. In: *Proceedings of the Ninth National Conference on Artificial Intelligence*. Menlo

Park, California: The AAAI Press, 1991, pp. 686-691. ISBN 0-262-51059-6.

25. HAWES, N. *Anytime deliberation for computer game agents*. Birmingham. PhD Thesis. University of Birmingham, School of Computer Science. Vedoucí práce Sloman. Available at: http://www.cs.bham.ac.uk/~nah/bibtex/papers/hawes04phd.pdf

26. OBST, O. and BOEDECKER, J. Flexible Coordination of Multiagent Team Behavior Using HTN Planning. In: *RoboCup 2005: Robot Soccer World Cup IX*. Berlin: Springer, 2006, pp. 521-528. ISBN 978-3-540-35437-6.

27. GORNIAK, P. and DAVIS, I. SquadSmart: Hierarchical Planning and Coordinated Plan Execution for Squads of Characters. In: *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference*. Menlo Park (California): The AAAI Press, 2007, pp. 14-19. ISBN 978-1-57735-325-6. Available at: http://petergorniak.org/papers/gorniak_aiide07.pdf

28. HOANG, H., LEE-URBAN, S. and MUÑOZ-AVILA, H. Hierarchical Plan Representations for Encoding Strategic Game AI. In: *Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*. Menlo Park (California): The AAAI Press, 2005, pp. 63-68. ISBN 1-57735-235-1.

29. PAUL, R., et al. Adaptive Storytelling and Story Repair. In: *Interactive Storytelling*. Berlin: Springer, 2011, pp. 128-139. ISBN 978-3-642-25288-4.

30. BALLA, R.K. and FERN, A. UCT for Tactical Assault Planning in Real-Time Strategy Games. In: *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence*. Pasadena (California): The AAAI Press, 2009. ISBN 1-5773-5429-X.

31. NGUYEN, T.H. D., et al. CAPIR: Collaborative Action Planning with Intention Recognition. In: *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Stanford, California: The AAAI Press 2011, 2011

32. KELLY, J.P., BOTEA, A. and KOENIG, S. Offline Planning with Hierarchical Task Networks in Video Games. In: *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*. Stanford (California): The AAAI Press, 2008. ISBN 978-1-57735-391-1.

33. Spy vs. Spy (1984 video game). *Wikipedia - the free encyclopedia* [online]., 18 March 2012 at 17:11 [accessed 2012-3-31]. Available at: http://en.wikipedia.org/wiki/Spy_vs._Spy_%28computer_game%29

34. ICAPS CONFERENCE. ICAPS Competitions. *ICAPS Conference* [online]. [accessed 2012-4-4]. Available at: http://ipc.icaps-conference.org/

35. HELMERT, M. The Fast Downward Planning System. 2006, vol. 26. 191-246. ISSN 1076-9757.

36. RICHTER, S., WESTPHAL, M. and HELMERT, M. LAMA 2008 and 2011. In: *Seventh International Planning Competition (IPC 2011), Deterministic Part*., pp. 50-54. Available at: http://www.informatik.uni-freiburg.de/~srichter/papers/richter-etal-ipc11.pdf

37. LIPOVETZKY, N. and GEFFNER, H. Searching for plans with carefully designed probes. In: *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 20111)*. Menlo Park, California: The AAAI Press, 2011, pp. 154-161. ISBN 978-1-57735-503-8.

38. HSU, C.W. and WAH, B. W. The SGPlan Planning System in IPC-6. In: *Proceedings of the Sixth International Planning Competition*. 2008, pp. 5-7. Available at: http://wah.cse.cuhk.edu.hk/wah/Wah/papers/C168/C168.pdf

39. HOFFMANN, J. The Metric-FF planning system: Translating "Ignoring Delete Lists" to numeric state variables. 2003, vol. 20. 291-341.

40. ČERNÝ, M. *Planning4J - Java API for AI planning* [online]. [accessed 2012-4-5]. Available at: http://code.google.com/p/planning4j/

41. VAQUERO, T.S., SILVA, J.R. and BECK, J.C. Analyzing Plans and Planners in itSIMPLE 3.1. In: *Proceedings of the Knowledge Engineering for Planning and Scheduling workshop. The 20th International Conference on Automated Planning & Scheduling (ICAPS 2010)*. Toronto: 2010

42. COMMUNITY, W. *Wine HQ* [online]. [accessed 2012-4-6]. Available at: http://www.winehq.org/

43. R DEVELOPMENT CORE TEAM. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2011. ISBN 3-900051-07-0. Available at: http://www.R-project.org/

44. WICKHAM, H. *ggplot2: elegant graphics for data analysis*. New York: Springer, 2009. ISBN 978-0-387-98140-6. Available at: http://had.co.nz/ggplot2/book

45. IPC Planners. *Fast Downward Web* [online]., 2011-11-16 14:02:34 [accessed 2012-04-11]. Available at: http://www.fast-downward.org/IpcPlanners

46. HORTHORN, T., BRETZ, F. and WESTFALL, P. Simultaneous Inference in General

Parametric Models. 2008, vol. 50(3). 346-363.

47. LIGGES, U. and MÄCHLER, M. Scatterplot3d - an R Package for Visualizing Multivariate Data. 2003, vol. 8(11). 1-20.

48. MCKILLUP, S. *Statistics Explained: An Introductory Guide for Life Scientists*. Cambridge: Cambridge University Press, 2006. ISBN 978-0-521-54316-3.

49. DAVID, D. M. *Survival Analysis in R*. 2012. Available at: http:// www.stat.ucdavis.edu/~hiwang/teaching/11fall/R_tutorial%201.pdf

50. THERNEAU, T. and LUMLEY, T. *Survival analysis, including penalised likelihood.* [online]. 2011 [accessed 2012-7-6]. Available at: http://CRAN.R-project.org/ package=survival

51. BYLANDER, T. The Computational Complexity of Propositional STRIPS Planning. 1994, vol. 69. 165-204.

List of Tables

List of Figures

List of Abbreviations

IVA	Intelligent virtual agent
UDK	Unreal Development Kit a 3D game engine by Epic Inc.
HTN	Hierarchical task networks – a planning approach
IPC	International Planning Competition
PDDL	Planning domain definition language

Appendix A – Contents of the Attached DVD

The attached DVD contains following files and folders:

- `plots` – directory containing more plots of the experiment results
- `plots/!naming_conventions.txt` – notes how to interpret names of plot files
- `javadoc` – directory containing reference documentation for code of the experiments
- `results` – directory containing the raw results of the experiments in CSV format
- `rcode` – directory containing code in `R` that process experimental results, generates graphs and performs statistical analysis
- `src` – directory containing all source files of the experiments
- `final_workspace.RData` – saved `R` workspace with all results prepared for statistical analysis
- `developer_docs.pdf` – developer documentation of the experiments including notes on how to compile and run the experiments
- `thesis.pdf` – electronic version of this thesis
- `UDKInstall-2012-02-BETA.exe` – installer of UDK, the version used in the experiments
- `readme.txt` – short summary of the DVD contents