Computational Intelligence, Volume 59, Number 000, 2010

To Plan or To Simply React?

An Experimental Study of Action Planning in a Game Environment

MARTIN ČERNÝ, ROMAN BARTÁK, CYRIL BROM, JAKUB GEMROT

Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic

Many contemporary computer games, notably action and role-playing games, represent an interesting class of navigation-intensive dynamic real-time simulations inhabited by autonomous intelligent virtual agents (IVAs). Although higher level reasoning of IVAs in these domains seems suited for action planning, planning is not widely adopted in existing games and similar applications. Moreover, statistically rigorous study measuring performance of planners in decision making in a game-like domain is missing. Here, five classical planners were connected to the virtual environment of Unreal Development Kit along with a planner for delete-free domains (only positive preconditions and positive effects). Performance of IVAs employing those planners and IVAs with reactive architecture was measured on a class of game-inspired test environments of various sizes and under different levels of external interference. The analysis has shown that planning agents outperform reactive agents if a) the size of the problem is small or if b) the environment changes are either hostile to the agent or infrequent. In delete-free domains, specialized approaches are inferior to classical planners because the lower expressivity of delete-free domains results in lower plan quality. These results can help to determine when planning is advantageous in games and for IVAs control in other dynamic real-time environments.

Key words: Action Planning, Comparison, Dynamic Environments, Intelligent Virtual Agents, Delete-free planning

1. INTRODUCTION

Dynamic, real-time and continuous environments pose a big challenge for the design of intelligent virtual agents (IVAs). 3D first person role-playing (RPG) and shooter (FPS) games are canonical examples of a subclass of such environments that are motion-intensive while offering the agent only limited options to interact with the environment and with other agents. Many serious games and some cultural heritage applications also fit this description (Connolly et al., 2012; Anderson et al., 2010).

One of the fundamental problems faced by an IVA in such an environment is the action selection problem — what to do next? In computer games, the prevalent approach involves reactive techniques, the most common being behaviour trees (Champandard, 2007) and finite state machines (FSMs) (Fu and Houlette-Stottler, 2004). Although the reactive techniques handle the dynamic aspects of the world well, they have some limitations: their strategies are fixed and cannot be altered during runtime, and they require a 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. Planning could theoretically allow IVAs to act smarter while easing the design burden. In this paper we focus on the longest studied approach — classical planning as solved by STRIPS (Fikes and Nilsson, 1972) — but mention also HTN planning (Ghallab et al., 2004) and systems based on Markov decision processes (MDP) as related work.

Unfortunately, the gap between the game AI community on the one hand and the planning and agent communities on the other hand is still huge. There are only a few applications of agent-based languages to game environments (detailed in (Gemrot et al., 2013)). Also only a few attempts were made to employ classical planning or its derivatives for controlling IVAs in dynamic environments. There are also numerous issues to be addressed for successful application of planning in complex domains (Pollack and Horty, 1999). While planning implementations in FPS-like domains, including released commercial games, do exist, we are not aware of any rigorous comparison of classical planning to reactive techniques in such environments.

In classical planning, the state of the world is modelled by a set of logical propositions. The agent can change the world by performing actions, each having a set of preconditions and effects. Preconditions are propositions that must hold for the action to be applicable, effects are either positive (propositions that became true after the action executes) or negative (propositions that no longer hold, once the action is executed). A planner then tries to find a sequence of actions that is applicable to the initial state of the world and that transforms the initial state into one of a set of goal states.

Since classical planning is computationally expensive, simplifications of the general planning problem were proposed in the past. One such simplification is that all effects of all actions are irreversible and the set of applicable actions never shrinks. Such domains are equivalent to *delete-free domains* (domains with only positive preconditions and effects). In delete-free domains, finding a plan becomes polynomial, although finding an optimal plan is NP-complete (Barták

et al., 2012). Because delete-free domains do actually correspond to problems present in some computer games, they are considered as a special case in this paper. One example of such problem is finding a plan to complete a set of quests that depend on each other — completing a quest is usually an irreversible action and it only adds new quests that are available.

The goal of this paper is to determine the conditions that allow AI planning to outperform reactive techniques in controlling IVAs in game-like environments. We focus on controlling individual agents and leave out of our present scope other possible planning applications in games (squad coordination, level generation, interactive storytelling, ...). We have designed a class of agent centric game-like motion-intensive test environments that allow a smooth adjustment of their dynamicity. Performance of several reactive agents and agents controlled by planners is then compared under different levels of external interference. Performance is measured by the solution time and the number of problems solved before timeout. Note that since this work is to our knowledge a first attempt at thorough evaluation of planning in a game-like environment, it is necessarily simplistic. Many more dimensions could be added to the problem, but we deliberately omitted most of them for the sake of clarity and interpretability of results.

This paper addresses two open questions. The first question is whether IVAs in real-time environments can actually perform better if they use a planner instead of a reactive architecture for their decision making. While planning has been adopted in some commercial games, the prominent reason to do so was not to improve performance, but to reduce complexity of AI code and increase manageability (Orkin, 2006). Intuition hints us that planners should be beneficial in

puzzle-like static environments, but they will be outperformed by reactive techniques in fast-paced dynamic worlds with multiple "good" actions at any time. Planners are also likely not to scale well to large problems, due to exponential complexity of planning. We want to make such claims precise by examining a whole spectrum of environment dynamicity and size. This comparison can hint the game industry, under which conditions action planning is a viable choice to improve IVAs intelligence, and academic planning community, along which lines to improve the current planning technology. The agent community may regard it as a preliminary test of real-time applicability of complex BDI-based languages, such as GOAL (Hindriks, 2009) or Jason (Bordini et al., 2007), to virtual agents in case planning is used as a part of the deliberation.

Our second question is whether limiting the planning problem to delete-free domains may mitigate the computing requirements of action planning to a significantly lower level while not sacrificing IVA performance in the environment. If this is the case, planning in delete-free domains might be another option for game developers to consider when choosing an appropriate AI technique.

To assess agent performance in practice, it is useful to present case studies at first. However, experimental results — even if performed *in silico* — are subject to substantial random noise. It is thus simple to misinterpret a random coincidence as a significant result. Therefore it is necessary to continue with a series of rigorous experiments after presenting simple case studies and to analyse the results statistically. Here, we conduct such series of experiments and analyse the results for significance, report respective p-values and effect sizes so that the readers have as much information as possible to judge the believability of our conclu-

sions by themselves. While some case studies of action planning in the context of 3D games were published (Champandard, 2013; Vassos and Papakonstantinou, 2011; Cartier, 2011), we consider lack of rigorous statistical analysis in game AI experiments an issue in the field.

This paper is an extension to our previous work on comparing action planning and reactive techniques (Cerny et al., 2013) and of the thesis (Cerny, 2012a). The extension in this paper consists of the study of planning in delete-free domains, detailed description of experimental design and much more detailed results and discussion.

The rest of the paper starts with discussion of related research (Sec. 2). Afterwards the actual experimental setup is introduced (Sec. 3 — 5). Then the experimental results are presented (Sec. 6) and the final part discusses the results and points out possible future research (Sec. 7, 8). Appendix A presents domain and problem descriptions for our test case, Appendix B shows a full diagram of a scenario we used in the experiments and Appendix C contains auxiliary tabular data on statistical significance of the results presented.

2. RELATED WORKS

To our knowledge, the only published papers on planning implementation in a commercial game describe the work of Orkin on F.E.A.R. and the GOAP system (Orkin, 2006, 2003) and were published in 2003 and 2006. GOAP is a planning system derived from STRIPS, but enhanced to better suite game needs. It has been applied to multiple games since: a good overview of planning applications

in games (including GOAP and Hierarchical Task Networks planning) is given in (Champandard, 2013). However, no research papers have been published yet.

Vassos and Papakonstantinou (2011) tested the BlackBox (Kautz and Selman, 1998) and Fast Forward (Hoffmann and Nebel, 2001) planners on a domain representing an FPS game. They show that the planners are able to plan in sub-second time for reasonably sized problems. However, the planning component is not connected to any real simulation. Thompson and Levine (2009) compared the performance of an agent employing a classical planner on several runs in static and dynamic versions of the same environment. The paper is however focused on the agent architecture and the performance comparison is very brief.

In (Bartheye and Jacopin, 2009), the Fast Forward and Qweak (Bartheye and Jacopin, 2005) planners are connected to a 2D arcade game and a 3D serious military game. However the paper is very unclear about the actual planners' performance and only states that most of the cases were solved in 4 seconds limit.

Long (2007) run a set of matches in Unreal Tournament between bots controlled by FSMs and bots controlled with GOAP. Bots controlled with GOAP win the matches more often, but no fine-grained statistical analysis has been done.

Cartier (2011) has run a series of matches between bots controlled by FSMs, GOAP and hierarchical task networks (HTN) planning in an abstracted FPS game. He concludes that unless the environment "waits" for agents to make decisions, the three architectures perform quite similarly, GOAP being significantly better than the other two by a small margin. However, only single environment size was investigated and the domain seems very sensitive to particular FSM design tested. The test domain also does not capture the continuous aspect of game environ-

ments as the agents move in large discrete steps. This work is also available only in French.

We know no other performance comparison of classical planning techniques in game-like domains. There are however other related papers where different planning approaches are included. Two of the alternative approaches to classical planning are the hierarchical task networks (HTN) formalism and Markov decision processes (MDP).

Hawes (2004) presents an anytime planning system based on HTN and evaluates it in the environment of Unreal Tournament. The planner is however compared only to a non-anytime planning system and not to a reactive architecture. In (Hoang et al., 2005), a set of "Capture the flag" matches between reactive bots and bots controlled with HTN was run. It is concluded that HTN planning brought relative advantage to the agents, but no statistical analysis is done and too few experiments were performed to give the results statistical significance. For instance one of the results is presented as "HTN clearly dominating", but analysing the published results with one-sided paired t-test (McKillup, 2011) gives p-value of 0.06 and thus the result is mildly unconvincing. Other relevant test variants report even higher p-values.

HTN planners have been also used in commercial games, most notable is the implementation in Killzone (Champandard et al., 2009), see the aforementioned planning overview (Champandard, 2013) for a list of other HTN applications.

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 (Balla and Fern, 2009). The planner is shown to outperform simple reactive behaviour and is better or

comparable to a human player, but no significance analysis was done and the number of experiment runs (5 per scenario) is not large enough to be convincing without such analysis.

CAPIR (Nguyen et al., 2011) is a system based on MDP for controlling an agent in a non-deterministic dynamic environment that is trying to infer human player goals and aid him in achieving them. CAPIR's performance was shown to be similar to a human in both absolute performance and perceived helpfulness, but it is not compared to any reactive architecture, and the statistical analysis performed is only basic.

Overall, the aforementioned papers show that planning in dynamic real-time environments is feasible and performs well against various baselines, but the papers either do not provide a rigorous comparison or do not compare planners directly to reactive techniques. This paper addresses this gap by a deep comparison of classical and reactive planners.

To our knowledge there is no implementation of a planner for delete-free domains actually connected to a game-like environment to control IVAs.

Planning techniques are also experimentally employed in games for other tasks than action selection. For example, the MIST interactive storytelling system (Paul et al., 2011) takes advantage of HTN planning in order to generate plots and dynamically repair them when changes in the world preclude their successful completion. There are many more planning implementations in the interactive storytelling domain, but as our focus is on action selection, we do not give a thorough review. An interested reader should consult the archives of the ICIDS conference[1].

[1] International Conference on Interactive Digital Storytelling, `http://www.icids.org/`

Planning has also been used for offline creation of game content. For instance Kelly et al. (2008) 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. Such uses are out of scope of this paper and our results are not directly transferable to those domains.

Finally, a parallel endeavour is being made to evaluate use of agent-based languages to develop game AI as in (Gemrot et al., 2013; Hindriks et al., 2012). While this is quite a different problem, there is a common ground since multiple agent-based programming paradigms assume that high-level deliberation, which is often supposed to use planning, constitutes a part of the reasoning cycle. To our knowledge, all of these works present user-studies or case-studies but they do not directly assess usage of planning within an agent-based language, let alone in a statistically rigorous manner.

3. EXPERIMENTAL DESIGN — ENVIRONMENT

Comparing reactive techniques to planning is a multi-faceted problem and there are many possible experimental design options. Since the area of action planning in dynamic game-like domains is not well studied, it is important to focus on a well-defined problem with a limited number of parameters first. The dynamicity of the environment was chosen as the most important factor for this paper, while all the other factors were either left out completely or kept as simple as possible. Still there are many ways how dynamicity may be achieved and it will be useful to investigate the nature of dynamicity present in games first.

In most game-like environments, the changes are continuous — or, more precisely, can be considered continuous for almost all practical purposes. Planning on the other hand, as other symbolic AI approaches, is discrete by nature. A natural way to discretize the dynamics is to consider only "important" changes, i.e., the changes that would affect a chosen discrete representation of the world. On a very abstract level, discrete dynamics may be considered as 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 size of the impact of a single change to the state of the environment;
- *attitude* — whether hostile or friendly changes are dominant. The hostile changes interfere with the agent's goals, while the 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 the reader should keep in mind that such summary necessarily involves a large amount of subjective interpretation and therefore is by no way definitive.

[Table 1 about here.]

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. 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 should mitigate the influence of implementation details of the agents on overall result trends. Moreover, unless a model of opponent behaviour is available or unless the problem is solved as a game, an opponent may be effectively perceived as a part of the environment. It is also important that the problem is not overly complex, so that there is not much room for improvement of reactive techniques by fine-tuning of the reactive plans by hand.

To keep the present work focused, we expect the world to be fully observable and the actions available to the agent to be deterministic. However, partial observability and non-deterministic actions can be to some extent modelled as changes in the environment (via interference).

3.1. Test Environment

[Figure 1 about here.]

Taking the aforementioned requirements into account, we proposed a simple yet flexible domain loosely inspired by the 1984 computer game Spy vs. Spy (Anonymous author, 2012). The environment consists of rooms on a grid that are connected by corridors. There is a door in the middle of each corridor. On both ends of each corridor, there is a button. A button may open one or more doors and/or close one or more doors all over the map. Initially, all doors are closed. The agent starts at a predefined room and has a goal room to reach. The agent is aware of all effects of all buttons. See Figure 1 for an example scenario in such an environment. The shortest solution to go from A1 to C2 is to:

(1) Push the east button at A1.

(2) Go to B1.

(3) Push the west button at B1.

(4) Go to A2 (through A1).

(5) Push the north button at A2.

(6) Go to C1 (through A1 and B1).

(7) Push the west button at C1.

(8) Go to C2 (through B1 and B2), 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 to move away from the goal room twice. Also there is a dead end: if the agent performs Steps 1 — 4 and then pushes the east button at A2 to get to B2, he traps himself and is no longer able to reach the goal.

An easy and efficient way for introducing interference into the environment is to repeatedly choose a subset of doors at random and alter their state. The interference parameters are then implemented in a straightforward way. The *delay* is the mean interval between two successive changes. 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. Note that opening a single door might not be necessarily beneficial for the agent, for instance when it opens a path to a room where the agent might get trapped. Similarly a door that is closed might actually help the agent if it forces him to take

a safer path. Nevertheless, averaging over large amount of interference, opening door is definitely friendly, while closing a door is not.

3.2. PDDL Representation

The most widely accepted formalism for expressing classical planning domains is PDDL (Fox and Long, 2003) — the language used at the International Planning Competition (IPC). 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.

We will call the most straightforward PDDL representation *standard* as opposed to *delete-free* described in the next section. In the standard representation the agent position is expressed with a set of "at" atoms and the door status with a set of "adjacent" atoms. "move" actions are available for each pair of neighbouring rooms and there is a "push" action for each button, which may change the "adjacent" status of the rooms. An example domain and a problem in the standard representation are given in Appendix A.

3.3. Delete-free Domains

One of the experiment goals was to separately evaluate planner performance in delete-free domains (domains with only positive effects and preconditions). Note that a domain is equivalent to a delete-free domain, if and only if it contains no reversible actions and the set of applicable actions never shrinks by performing an action. Note that the standard representation of any map is not delete-free, because the move action itself is reversible by its very definition. However

simplified maps where buttons only open doors and do not close them may be represented by a delete-free domain, if we change the way we represent movement.

In our delete-free representation, the predicates for adjacency stay the same, so do the actions for pushing buttons (except they are not permitted to close doors), but instead of explicitly reasoning about its actual position, the agent reasons about the set of locations reachable from the agent's current position. In the initial state, only the agent's current location is marked as reachable. Then for every pair of neighbouring rooms A and B, there is a "reach" action that adds B to the set of reachable locations under the condition that:

- A is already reachable and
- the door from A to B is open, that is, if "adjacent" holds for A and B.

Thus we have no negative effects and no negative preconditions. Because buttons may not close doors, the set of locations reachable from the agent's current position may only grow. The problem at hand then has a solution if and only if the goal room is eventually added to the set of reachable locations, so the representation is both correct and complete. An example domain and a problem in the delete-free representation are given in Appendix A.

In further text, map variants that do not contain buttons that close doors and thus allow for a delete-free representation will be referred to as the *delete-free* or shortly *DF maps* as opposed to the *standard maps*.

Unlike the standard representation, the plans that solve problems in the delete-free representation cannot be directly interpreted as actions the agent may exe-

cute in the environment. Instead, the "reach" actions are ignored and the "push" actions are expanded to first move to the button location by a series of what-would-be "move" action in the standard domain and then push the button. After successfully executing such plan, the agent has not yet reached the goal room, but the path should be open. So the last step of the plan interpretation is to find a path from the current agent position to the goal and to execute the respective move actions. Technically, the paths are provided during plan execution from a separate pathfinding module. In our experiments, the time spent in pathfinding was negligible and was not included in the timing results.

Note that planning in delete-free domains precludes reasoning about the agent's position at the planner level. This means that the planner does not optimize agent movement in such representation and may well decide to push buttons at remote locations even though closer buttons with similar effect are available. Also, the order of an uninterrupted "push" actions sequence is arbitrary, while different orderings often correspond to fairly different routes. This is an example of a more general issue with delete-free 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 the delete-free representation in practice.

3.4. Map Generation

The maps for the experiments were created randomly, only ensuring that in the initial state of the map, there is always a solution path. See Algorithm 1 for the actual procedure. Note that although every room in our map is connected to all its

neighbours, there may be no button opening a particular door. Such a door may be opened only by interference. Similarly, there may be buttons that neither open nor close any door. Once again those buttons are ignored by the agents.

Algorithm 1 The map generation algorithm.

procedure GENERATEMAP($width, height, avgDoorClose, avgDoorOpen$)
  create square map with $width \times height$ rooms and no button interactions
  $guaranteedPath \leftarrow$ RANDOMPATH(point(0, 0), point(width, height))
  $numOpens \leftarrow 0$, $numCloses \leftarrow 0$
  for all door d on $guaranteedPath$ **do**
    randomly select a button b on $guaranteedPath$ prior to d
    add $opens(b,d)$ to map definition, $numOpens \leftarrow numOpens + 1$
  end for
  while $numOpens$ / number of buttons in map < $avgDoorOpen$ **do**
    randomly select a button b and door d
    add $opens(b,d)$ to map definition, $numOpens \leftarrow numOpens + 1$
  end while
  while $numCloses$ / number of buttons in map < $avgDoorCloses$ **do**
    randomly select a button b and door d
    if not PATHTHREAT($b, d, guaranteedPath$) **then**
      add $closes(b,d)$ to map definition, $numCloses \leftarrow numCloses + 1$
    end if
  end while
end procedure

▷ $avgDoorClose$ and $avgDoorOpen$ is the average number of doors open (closed) by a single button in the map.
▷ RANDOMPATH returns a path with possibly repeated rooms, the length of the path varies from $width + height$ (the minimum of steps needed) to $3 * (width + height)$.
▷ PATHTHREAT($b, d, guaranteedPath$) is a heuristic function that checks whether adding $closes(b,d)$ to the map may possibly render $guaranteedPath$ impossible to pass. The function may report false positives, but when it returns false, it is guaranteed that the path is not threatened.
▷ In our setting, the $guaranteedPath$ never required so many opens interactions so that no more randomized opens interactions would be added in the first while block — there was always a significant margin.

The standard maps for our experiments were created with the average number

of doors opened/closed by a single button varying from 0.2 to 2.5. For the DF maps the number of doors opened by a single button varied between 0.2 and 1.7 and the number of doors closed was set to 0. The lower parameter bound represents only slightly more interactions than those needed for a solution to exist, while the upper bound represents a situation where nearly all of the buttons in the map interact with multiple doors at once. The parameter range for the DF maps was lowered a little, because the difficulty of a DF map decreases with more door opening interactions. An example standard map generated by the algorithm is shown in Appendix B.

4. EXPERIMENTAL DESIGN — AGENTS

This section discusses the agent design. It starts with the overall description of the agent action selection mechanism, continues with the specifics of reactive and planning agents tested, and finally the actual planners that were tested are introduced.

4.1. Agent Action Selection

The agents have only two classes of actions to choose: move to a reachable room and push a button in the current room. The details of execution of the actions are delegated to an abstract interface to the virtual world, which is the same for all agents. This interface also includes a pathfinding module. Note that from the experiment point of view the total time spent in pathfinding was negligible (less than 1%).

Apart from the agent specific action selection mechanism there are two kinds of reactive rules available to the agents:

(R1) If there is a clear path to the goal location then follow that path.

(R2) If there is a button in the same room as the agent that will open an unopened door and will not close any open door, then push the button. If there are multiple such buttons, one is chosen randomly.

Note that both reactive rules are designed to never increase the number of actions required to reach a goal and are necessarily insufficient on their own to let an agent reach the goal. It is up to the agent's main logic to choose actions for the non-obvious cases.

Reactive rules have a higher priority than the agent's main logic — if the rule is active and the conditions are met, the rule's actions are always triggered. However, not all agents use both rules (see below).

In the context of computer games, planning is never used as the sole decision making mechanism, but is built on top of a reactive layer that handles high priority events where immediate reaction is vital. We thus considered using reactive rules in both planning and reactive agents.

4.2. Reactive Agents

The overall action selection mechanism for reactive agents is shown in Algorithms 2 and 3. Initially, three reactive agent types were examined with different

reactive rules. After a set of preliminary experiments one instance of each type was chosen for the final comparison. Two were chosen for their high performance and one was chosen as a baseline:

- *Inactive* — the agent performs only the actions triggered by rule R1, i.e. it can succeed only if a path to the goal is opened by chance (baseline agent).
- *Random* — the agent repeatedly chooses a reachable button at random, moves to its location and pushes it. The agent uses both rules R1 and R2.
- *Greedy* — if it is possible to move to a place closer to the goal, the agent moves there. The agent does not push any buttons, unless it is an action initiated by a reactive rule; both rules are active.

Algorithm 2 The main action selection function for reactive agents.

```
function SELECTACTIONREACTIVEAGENT
    ruleAction ← EVALUATEREACTIVERULES
    if ruleAction ≠ nil then
        return ruleAction
    else
        return SELECTACTIONREACTIVEAGENTSPECIFIC
    end if
end function
```

▷ SELECTACTIONREACTIVEAGENTSPECIFIC performs an agent type specific deliberation (the implementation is different for each reactive agent type — see text for further description).
▷ EVALUATEREACTIVERULES selects a reactive rule and returns the selected action (Algorithm 3).

Preliminary experiments have shown that rule R1 is beneficial for all reactive agents. All reactive agents performed better with rule R2, but except for the Greedy agent the difference was not statistically significant in the standard maps (at 0.05 level) and in the DF maps, it was significant only for a minority of cases.

Algorithm 3 The reactive rules used by agents.

```
function EVALUATEREACTIVERULES
    if r1Active and ∃ a path to goal then
        return "moveToAction(goal)"
    end if
    if r2Active then
        b ← any button in currentRoom, such that opens(b) ∩ closedDoors ≠
∅ and closes(b) ∩ openDoors = ∅
        if b ≠ nil then
            return "pushAction(b)"
        end if
    end if
    return nil
end function
```

▷ The values $r1Active$ and $r2Active$ are true in case the respective rule is active; otherwise it is false. The values are set specifically for each agent class.
▷ $opens(b)$ is the set of all doors opened by button b, $closedDoors$ is the set of all doors, that are closed in the current world state. $closes(b)$ and $openDoors$ are defined analogically.

Note that if there is interference and it is not extremely unfriendly, the Greedy agent is likely to eventually succeed in solving a map if the agent is given enough time. However it is also likely that this agent will produce "plans" far away from the theoretical optimum. This is a common situation in game AI: usually planning or some other more advanced technique is not required for the agent to act "somewhat OK", but an advanced technique may (or may not) help the agent to perform better.

4.3. Planning Agents

The action selection mechanism for a planning agent is shown in Algorithms 4 and 3. This algorithm is invoked for every tick of agent's logic (approximately every 250ms). The agent translates the actual state of the world into the PDDL language

and sends it to the planner. Until the planner responds, the agent initiates no action (except for rule actions). When the plan is received, it is executed sequentially and it is continuously checked for validity. If the check fails or if an action fails to execute, the current plan is discarded and the planner is called to yield a new one. The planner is never invoked, while the agent has a valid plan. This is a typical approach in computer games using planning, as replanning is expensive and is thus performed only when really necessary.

Preliminary experiments have shown that rule R1 is beneficial for all planning agents, improving the fraction of the runs when the agent reached goal by several percentage points in smaller maps and by over 30 percentage points in the largest maps. The R2 rule showed to be more problematic. Activating R2 for planning agents never improved the performance and it had statistically significant adverse effect in all but the largest maps. Based on those results all planning agents used R1 as their only reactive rule.

Note that in a dynamic hostile environment, even a freshly computed plan may be invalid because the environment may have changed while the agent was planning. Also, in a dynamic but friendly environment the R1 rule may be activated if a path to the goal becomes available while the agent is planning. In practice, both situations occur, especially for large maps where planning sometimes takes over a minute.

4.4. Chosen Planners

For our experiments we have chosen five classical planners based on their performance in recent International Planning Competition (IPC) and our subjec-

Algorithm 4 The action selection mechanism specific for the planning agents.

```
function SELECTACTIONPLANNINGAGENT
▷ The agent has two persistent variables: state that describes the current state
of the agent and plan, which is the current plan being executed.

    ruleAction ← EVALUATEREACTIVERULES
    if ruleAction ≠ nil then
        return ruleAction
    end if
    if state is waiting_for_plan then
        return "no_op_action"
    end if
    if state is just_received_plan then
        plan ← the plan received
        state ← executing_plan
                                        ▷ continues in executing_plan branch
    end if
    if state is executing_plan then
        if last action failed or PLANINVALID?(plan) then
            STARTPLANNINGTHREAD
            state ← waiting_for_plan
            return "no_op_action"
        else
            return EXTRACTFIRSTACTION(plan)
        end if
    end if
end function

▷ Action failures are evaluated by the environment.
▷ PLANINVALID? tests whether the given plan can be executed in the current
world state and if its execution reaches the goal state.
▷ STARTPLANNINGTHREAD translates the current world state into PDDL (the
representation is set separately for each agent) and starts the associated planner
in a background thread. Once the thread finishes, it sets the state variable to
just_received_plan and sends the plan to the agent.
▷ EXTRACTFIRSTACTION removes the first action from the plan and returns it.
```

tive assessment of their popularity in the game AI community. Out of the four fastest planners at the IPC 2011 three were based on the Fast Downward platform (Helmert, 2006), including the winner. The winning planner — LAMA 2011

(Richter et al., 2011) — was chosen to represent this platform. The second fastest planner at IPC 2011 was the Probe (Lipovetzky and Geffner, 2011) and so it was chosen too. Apart from the two very recent planners, three older planners, which have earned reasonable respect in the past years, were chosen. The first is SGPlan 6 (Hsu and Wah, 2008), which won IPC 2006. The Fast Forward (FF) planner (Hoffmann and Nebel, 2001), a top performer at IPC 2002, was also chosen. All four aforementioned planners are based on forward state space search. The last included planner is the BlackBox (BB) (Kautz and Selman, 1998) that tries to extract solution directly from a planning graph for the problem. The extraction problem is formulated as a SAT problem which is then solved by Walksat (Selman et al., 1994) and Satz (Li and Anbulagan, 1997) solvers. No partial-order planner was tested, since none has succeeded in recent IPCs neither there is (to our knowledge) one with wider academic use.

In addition to those five planners, a specialized planner with a specific algorithm tailored for the delete-free domains was added — the ANA* planner (Barták et al., 2012). In contrast to the other planners, which are written in C/C++, ANA* is written in Java.

For the DF maps, the five classical planners were tested both with the standard PDDL representation and the delete-free PDDL representation. In the context of DF maps, those will be referred to as *DF* and *standard planner variants*.

5. EXPERIMENTAL DESIGN — PARAMETERS AND ANALYSIS

This section discusses the hardware and software setup for the experiments, the number and the types of maps used and the actual values of interference parameters, and statistical methods we used to analyse the results.

5.1. Hardware and Software Setup

The experiments were carried out in the virtual environment of Unreal Development Kit (UDK) (Epic Games Inc., 2012). The agents were written in Java using the Pogamut platform (Gemrot et al., 2009). Moving from one room to an adjacent one takes approximately one second, while approaching and pushing a button takes about 200ms. 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). 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 and left a big margin of free RAM so that the experiments did not compete for resources.

5.2. Experiment Scale

Since the simulations run in real-time, the experiments are very time consuming, especially for large maps (up to 15 minutes per run). Therefore the number of maps was limited. Table 2 summarizes the four map types used along with their time limits. The time limits were set (separately for each map size) to 5 times the time needed by all planning agents on average to reach the goal without interference. The reason is that the time planning agents take to complete the map

without interference is close to the shortest time the maps may be completed in and multiplying by 5 should give a sufficient margin so that agents that do not reach the goal in this limit can be considered incapable of reaching the goal in a reasonable time. Our preliminary experiments indicated that our limits are sufficient to let the agents reach goal in a substantial number of cases while keeping the total time needed for the experiments bearable.

The number of actions refers to the number of grounded "push button" and "move to adjacent room" actions. The standard maps for our experiments were created with the average number of doors opened/closed by a single button varying from 0.2 to 2.5. For the DF maps the number of doors opened by a single button varied between 0.2 and 1.7 and the number of doors closed was set to 0. More details on map generation can be found in Section 3.4.

[Table 2 about here.]

The interference parameters were set based on the estimates from Table 1 and observations from the preliminary experiments. The mean delay values were chosen as 0.5, 1.5 and 3 seconds. The mean impact (fraction of the doors changed at once) values were 0.05, 0.1 and 0.2 and the friendliness (probability a door opens) values were 0, 0.15, 0.3, 0.5 and 0.7. More focus was kept on hostile environments since the reactive agents clearly dominated with friendliness 0.3 and higher, as shown by the preliminary experiments.

The actual delay and impact values during the experiment run are sampled from a uniform distribution with zero minimum and respective mean. The randomness is realized through a built in random number generator of the Java lan-

guage. 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 interferences are the same for all agents.

For each combination of map, agent, and interference parameters 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.

5.3. Analysis

The primary metric for evaluation of the experiments is the *success rate*. It measures whether the agent managed to reach the goal before the timeout elapsed. Note that since the environment is stochastic, all agents would eventually reach the goal thanks to the R1 heuristic, if no time limit was enforced. The only exception is when friendliness is 0, where the agent may find itself in a dead end and no interference will help him.

Success rate was chosen as the primary metric, because it is easy to interpret and does not need to be rescaled to compare runs in different map sizes. Statistical results for the success rates are assessed using multiple comparisons of means with Tukey contrasts (Hothorn et al., 2008) over an ANOVA fit with a first order generalized linear model. In other words, we assume a binomial distribution for the success of the agent, and that the parameters of the distribution can be described by a linear combination of the agent properties transformed by a logit function. For each pair of the agents, we test a null hypothesis that the means of the binomial distributions are equal. Tukey's contrasts correct for the increased probability of a type I error resulting from the large amount of comparisons be-

ing made. We compute the effect size simply as the difference between the two success rates.

An important metric is also the time the agent spent from start of the simulation to the moment it reaches the goal — the *solution time*. The solution time is considered only for the runs where the agent actually reached the goal. To analyse the solution time, a linear model is fitted to the data with solution time log transformed to be closer to the normal distribution, and the Tukey's HSD test (McKillup, 2011) is performed to reveal significant differences between agent pairs. Tukey's HSD test performs all pairwise comparisons of means of the agents' metrics, i.e. for each pair of the agents, it tests a null hypothesis that their means are equal. Tukey's HSD corrects for the increased probability of a type I error resulting from the large amount of comparisons being made.

Effect size is asserted using Hedge's g, a less biased metric derived from Cohen's d (Hedges, 1981). For both Hedge's g and Cohen's d, effect size of 0.2 to 0.3 is usually considered a "small" effect, around 0.5 a "medium" effect and 0.8 to infinity, a "large" effect.

However, the results for solution time should be interpreted with caution, since the number of successful runs is likely to be different among the agents. Thus the longest times — the ones where the agent failed to reach the goal — are effectively not included. To partially remedy this, right censored accelerated failure-time survival model (Diez, 2013) was fitted to the data using R package survival (Therneau and Lumley, 2011). The error distribution was lognormal and multiple comparison was performed with Tukey contrasts (Hothorn et al., 2008). 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 given infinite time to solve the problem. Here, the multiplicative coefficient by which such estimated mean solution times differ between a pair of agents (as given by the model) is treated by us as the effect size. The results of the survival model have the advantage of being robust with respect to the choice of the time limit, as long as a large amount of runs is completed within the limit.

To measure the computing power needed by the agents we measure the time the agents spent deliberating. 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 and includes all processing related to deliberation (e.g. translating world state to PDDL). This related processing however took less than 5% of the deliberation time in all but few runs and less than 1.5% on average (with the exception of LAMA 2011 planner, see below). The deliberation time does not include the processing performed by the interface between agent’s action selection and the virtual world. Almost 100% of the time spent in the interface corresponds to finding low-level paths in the virtual environment when executing “move” actions. The time spent in these pathfinding calls was however negligible when compared to the time spent by planning and did not in any way slow the execution of the plan down. The purpose of deliberation time is to compare different planners in terms of computing power needed.

6. RESULTS

In the preliminary runs, the LAMA 2011 planner performed very poorly (worse than Random agent and only slightly better than Inactive agent). The main reason is that the Fast Downward platform carries out a quite costly translation of the PDDL input to different formalism before starting the actual planning. The pre-processing of our domains took from several hundred milliseconds to several seconds, which is a big performance hit, considering the interference delays. To save computing time, LAMA 2011 was removed from further experiments.

This section continues with the analysis of overall agent performance in the standard maps, then similar analysis is performed for DF maps and finally the effect of various interference parameters is discussed.

6.1. Overall Performance — Standard Maps

Averaging success rate over all standard maps (Table 3, row "Total") shows that all planners perform very similarly to each other and to Greedy. SG, FF, BB and Greedy are not significantly different and thus share the first place. All the other differences in row "Total" are significant (all $p < 10^{-3}$).

[Table 3 about here.]

Inspecting the results for individual map sizes shows that the planners dominate by large amount in small and medium maps, but yield lead to Greedy in large and 13×13 maps. The Inactive baseline bot showed that in many cases no smart acting is required to complete a map. The success rates of planning agents are not generally significantly different from each other, but nearly all of the other

differences are significant with $p < 10^{-3}$. For detailed significance analysis see Tables 15 and 16 in Appendix C. The performance of each agent was similar across individual maps of the same size.

While the success rate of planning agents decreases with the growing map size, the success rates of Greedy and Inactive behave differently — their success rate is higher for 13×13 maps than for the large ones. Examining the results more closely (see Figure 2) reveals that the raise of the time limit with growing map size was larger than the raise of the mean solution time for Greedy and Inactive: In small, medium and large maps, Inactive and Greedy agents reached the goal shortly before the respective timeout in many runs, indicating that the success rate is likely to grow if they were given more time. For 13×13 maps, most of the runs finished long before the timeout.

Although the success rate of the reactive agents in smaller maps would likely grow if the time limit was increased, the analysis of the survival model (see below) indicates that planning agents would still outperform reactive agents in small and medium maps. Similar observation can be made intuitively by examining Figure 2 — as the histograms of reactive agents are very flat (in log scale), we can reasonably expect that even a two fold increase/decrease in the time limit would change their success rate only by several percentage points, while the success rate of planning agents would be even less affected, as the bulk of their runs finished long before the time limit.

[Figure 2 about here.]

The solution times are presented in Table 4. Most of the solution time differences are significant at 0.05 level, for detailed significance results and effect sizes see Table 17 in Appendix C. The results show that when the planning agents did manage to reach the goal, they were faster than reactive agents in all map sizes. The effect size is small or negligible among the planning agents and medium to large between all planning and all reactive agents, except for 13 × 13 where the effect sizes diminish, probably in connection with lower number of successful agent runs. As mentioned above, it is tricky to interpret this result due to the uneven number of successful runs and thus we leave further analysis to the survival model.

[Table 4 about here.]

Table 5 shows the ordering of agents based on significant differences of the survival model, for detailed significance results and effect sizes see Table 18 in Appendix C. The survival model produces results similar to the success rate for small, medium and 13 × 13 maps, but for the large maps, there is an important difference: BB is considered to share the first place with Greedy, and Greedy is not significantly better than FF and SG. Thus it is possible to conclude that while BB, FF and SG have lower success rate than Greedy in large maps, they compensate this with a shorter solution time. The overall similarity to the results of our success rate analysis also indicates that the findings of the success rate analysis would be similar even if the time limits were chosen differently.

[Table 5 about here.]

The deliberation time results are shown in Table 6. For reactive agents, the time spent deliberating is almost negligible — less than 0.4% of the solution time in all cases, the planning agents however spent at least 21% of the solution time deliberating and in some cases climbed above 50%. SG showed the least growth of time for single planning execution with the growing map size.

An interesting thing to note is that in some instances of the 13 × 13 maps, the planning agents did not complete any planner execution, but they were still able to succeed, thanks to the R1 rule. Out of 540 runs of each agent on 13 × 13 maps, this was the case in 39 runs of BB and 32 runs of Probe. These correspond to relatively quick runs, and thus they represent cases where the agent terminated its deliberation early on, because a path to the goal became available.

[Table 6 about here.]

[Figure 3 about here.]

6.2. Overall Performance — Delete-free Maps

In general, the results for the DF maps show similar patterns as the standard maps, but the relative performance of reactive agents to planning agents has slightly improved. Averaging the success rate across all map sizes (Table 7, row "Total") shows that Greedy agent performs best by a reasonable margin and ANA* is worse than all other agents except Inactive, which is the worst. All those differences are significant (all $p < 0.01$), but most of the other differences are not. The performance of each agent was similar across individual maps of the same size.

Similarly to the standard maps, the planning agents outperform all reactive

agents in small maps, but their performance degrades with growing map size, yielding the lead position to the Greedy agent for large maps (Table 7). Nearly all differences among the classical planning agents (both DF and standard variants) are not significant. ANA* on the other hand is almost always significantly different than the rest and so are Greedy and Inactive ($p < 0.05$).

In many cases DF variants of planning agents perform worse than the standard ones (although the difference is not significant). There are two possible reasons for this: the difficulties in interpreting a DF plan (see Section 3.3, page 15) and small to negligible increase in planner performance on DF problems. As we show in further analysis, both seem to play a role.

Quite notably the performance of ANA* degrades very quickly. This is most likely because ANA* does not return, until it finds an optimal plan. Interestingly, the DF variants of planning agents almost never beat their standard counterparts.

Similarly to the standard maps and for the same reason, the actual success rate of reactive agents is larger in 13×13 maps than in the Large maps (Table 7, Figure 3). Once again, the analysis of survival model (see below) and the histograms indicates that the results would be similar even if the time limit differed. Surprisingly, the random agent has performance very similar to that of the worst planning agents. This is probably due to the fact that since the buttons cannot close the doors, just pushing random buttons is not a bad strategy.

Comparing the actual numbers to those for the standard maps (cf. Tables 3, 4 and Tables 7, 8), shows that while in small maps agents in DF maps performed better, for medium and larger maps, the agents have only slightly better or even worse performance in the DF maps. This hints that the DF maps are not much

simpler for planners to handle and simultaneously the R2 rule does not bring significantly better advantage in DF maps. The theoretical shift to lower problem class[2] therefore does not seem significant for practical use.

[Table 7 about here.]

[Table 8 about here.]

[Table 9 about here.]

[Table 10 about here.]

[Table 11 about here.]

Considering the solution times in the DF maps does not bring many interesting results (Table 8). There are nearly no significant differences among the planners, although the DF variants have slightly longer solution times in general, probably due to suboptimal interpretation of DF plans. Planners are able to have lower solution times than reactive agents in all map sizes, although in 13×13 maps some of the planners are worse than Greedy.

For small and medium maps survival analysis (Table 9) reports similar results to those of the success rate. In large and 13×13 maps planners are able to rank better in survival model than by the success rate only. Note that for 13×13 maps some of the agents were left out of the comparison, this is due the fact that their solution time distributions were grossly non-normal (Figure 3) and their analysis was thus impossible. However none of those was among candidates for top performers. For small and medium maps the planners are clear leaders in the survival model, but

[2] Classical planning is PSPACE-complete (Bylander, 1994), while planning in DF domains is in PTIME (Barták et al., 2012).

for larger maps they yield the first position to Greedy. There are also only a few significant differences among the planners. The similarity to the results of our success rate analysis also indicates that the findings of the success rate analysis would be similar even if the time limits were chosen differently.

It is of interest, that while in most cases the deliberation times of the DF and the standard variants of planners were very similar (Table 10), the DF variants of FF, Probe and BB have significantly lower average deliberation time in 13×13 maps than their standard counterparts ($p < 10^{-4}$, see Table 10) Those planners thus seem to benefit from the delete-free representation for very large maps. The same is however not true for SG ($p > 0.9$). This might be due to the fact that the problem decomposition method used in SGPlan allowed it to exploit the delete-free nature of the problem even in the standard representation. Also the lower deliberation time has not resulted in better performance which hints us that while BB, Probe and FF planners solve the delete-free problems faster in larger maps, they are slowed down by the difficulties of interpreting a DF plan (see Section 3.3).

While ANA* degrades quickly in medium and larger maps, it spent only 5% of solution time deliberating in small maps (other planners spent 18 – 21%). Thus for smaller domains ANA* got close to reactive agents in terms of speed of response. However, this did not result in a comparable increase in success rate, which once again shows that the lower quality of plans found in the DF representation is a major obstacle to practical applicability of DF planning. Moreover, if very quick optimal planning in delete-free representation for small maps did not result in superior performance, it is unlikely that switching to suboptimal planning would

result in superior performance in larger maps — even if the planning time shrunk considerably.

Interestingly, the deliberation time of ANA* is lower in large maps than in medium maps. This is likely because the planning time for ANA* grew faster from small to medium maps than the timeout, but in large maps the timeout grew faster than the ANA* planning time.

In some instances of the 13 × 13 DF maps, the planning agents did not complete any planning runs, but were able to succeed thanks to the R1 rule only. Out of 810 runs of each agent on 13 × 13 DF maps, this was the case in 209 runs of ANA*, 180 runs of BB, 1 run of BB-DF, 1 run of FF, 2 runs of FF-DF, 123 runs of Probe, 69 runs of Probe-DF, 2 runs of SGPlan and 1 run of SGPlan-DF. Similarly to standard domains, these correspond to relatively quick runs, where a path to the goal became available before the agent completed its initial planning.

Unlike in the standard domains, there were also cases when the agents did not complete any planning run on the 13 × 13 map, but did not succeed: 338 for ANA*, 255 for BB, 1 for BB-DF and 1 for Probe. These are the cases where the agent interrupted its planning, because a path became available, but the path was later blocked by interference and the agent took too long to find a solution in the new situation and either was once again interrupted by R1 or the time ran out. This once again hints that the DF maps were in some sense harder for the agents than the standard maps, probably due to smaller number of buttons opening doors. In addition, ANA* did not complete any planning computation in 190 unsuccessful and 80 successful runs on large maps and in 192 unsuccessful and 100 successful runs on medium maps, but that is probably due only to its low performance.

6.3. Performance and Dynamicity

In this section we focus mostly on analysing the success rate. Since the survival model is robust to a different choice of time limit and the results for the success rate are close to the results of the survival model, it indicates that our choice of the time limit lets us generalize the results drawn from analysing the success rate.

All the interference parameters have statistically significant impact on the agent performance (all $p < 0.01$ for both metrics); see Tables 11 and 12 for the actual effect sizes for SG and Greedy. Since the results were very similar for all planning agents, SG was taken as a representative. In general, friendliness is unsurprisingly the most important factor. For SG, impact and delay also play an important role, but for Greedy their effect is only marginal. The effect size of the combination of delay and impact is small, because the effect of combination is strongly dependent on friendliness. Both impact and delay show large effect sizes in combination with friendliness, but for SG and friendliness levels below 0.5, the larger impact mitigates the effect of small changes in friendliness. For Greedy the larger impact improves performance for all friendliness levels. Note that larger friendliness might not improve the solution time, because larger friendliness alters the number of runs where the agent reached the goal, but some of the "newly added" successful runs might have longer solution times. This is just another issue with the solution time metric.

It has been already noted that concerning the success rate, the Greedy bot performed the best on average. However, in hostile environments (friendliness = 0) and in less dynamic environments (delay = 3s), the planners prevailed. This effect

was more dramatic in the standard maps than in the DF maps. The differences between different planners were very small and followed the same overall trends.

Figure 4 shows a plane fitted through the average success rates of SG and Greedy bots in the standard maps, 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 the shorter interference delays, the success rate of SG 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 Rand is similar to planning agents.

[Figure 4 about here.]

In the DF maps the trends are very similar, although the performance of Greedy agent degrades much slower with the growing delay (Figure 5). There are no notable differences between the DF and the standard variants of planners and ANA* behaves similarly to other planners, although with a lower overall success rate.

[Figure 5 about here.]

The impact and delay values were chosen so that there are more combinations that yield the same number of mean door changes per second, so it is possible to determine whether delay or impact has a larger effect on agent performance. The results for the success rate are presented in Table 13. They show that SG (as a representative planning agent), Rand and Greedy all perform the same if the mean number of door changes per second is the same and thus the effect of delay and

impact may be considered identical. This is in accord with the effect sizes listed in Tables 11 and 12 and holds also for examining only specific friendliness settings or map sizes.

[Table 12 about here.]

As interference forces the agent to replan, it is worth examining how often did the agents actually need to replan. Table 14 shows the number of planner executions for the SGPlan agent, which was chosen as the representative. We report numbers with respect to friendliness, map size and delay. Friendliness and map size had the highest influence on the number of planner executions, the influence of delay was smaller and the influence of interference impact was similar to the influence of delay. The number of executions with respect to interference impact was not reported as they provide little additional insight, while requiring a large amount of space.

Overall, the number of planner executions is very high — in some cases it is more than one in every two seconds. Quite unsurprisingly, more hostile environments and shorter delay times result in more planner executions. The same is true for larger maps as plans in larger maps are longer and thus there is more time for the interference to invalidate the agent's plan. In fact, the number of replannings reacts to changes in interference parameters inversely to success rate, but it is not sufficient to explain the worse performance of planning agents in dynamic environments on its own. For example for friendliness 0.5 and delay 3s, the number of replannings is higher than for friendliness 0.5 and delay 1.5s. This is probably just due to chance, but the success rate is still higher in the former case and thus the

increase in number of planning executions cannot be linked directly to decrease in success rate. The correlation between the number of planner executions and success rate, measured by Pearson's R, is -0.42 in standard maps and -0.44 in DF maps. The differences between DF and standard maps are once again very small in general, signalling that in our particular case, the DF maps were not easier for the agents.

The numbers are however inflated by the cases where the planner could not find a solution. If no solution is found, the agent starts replanning immediately (see Algorithm 4). In situations where the non-existence of the plan can be proved quickly and the environment is not friendly, the agent may execute the planner a relatively large number of times before the environment changes enough to allow for a solution. In the case of both standard and DF 13×13 maps and friendliness 0.15 the effect of continual replanning is so strong that there is actually a higher number of planner executions for longer delay values.

The only case when the agent does not replan immediately is when friendliness is 0. In this case, the agent stops completely after the first time the planner has found no solution, because it means that the agent will never be able to reach the goal. This manifests in low number of planner executions for completely hostile environments.

A more detailed discussion of the experiment design and complete results are described in the thesis (Cerny, 2012a).

[Table 13 about here.]

7. DISCUSSION

7.1. Main Results

The most important result is that in small (< 200 atoms and/or actions) or hostile (friendliness = 0) or less dynamic (delay $\geqslant 3s$) domains, the contemporary planning algorithms are fast enough to provide advantage over the reactive approaches concerning action planning (Table 3 and Figure 4). Examining the sizes of our planning domains and logged planning times, we perceive the limits of real-time applicability (planning faster than 1s) of contemporary planners somewhere above one hundred atoms and two hundred grounded 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.

On the other hand, the results also explain why planning is not the first choice for action selection 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 (Figure 4 and 5). 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 is affected similarly by interference delay and interference impact (Table 13) — at least if the interference parameters lie within the game spectrum as we observed in Table 1.

Another result is that unless the problem cannot be modelled naturally as delete-free, involving delete-free representation may pay off only in domains that are significantly larger than the largest we have tested, if ever (Section 6.2). This can be because the speedup in planning is mitigated by the lower quality of plans. For some planners there is no significant planning speedup with delete-free maps at all. Except for the smallest maps, ANA*, a specialized planner for delete-free domains performed very poorly, but this might be improved if it did not search for an optimal solution.

7.2. Generalizations, Limitations

Initially we have set a spectrum to classify environment dynamicity (Table 1). Generalizing our results with respect to this classification indicates that if the planning domain is smaller than about a hundred atoms and/or grounded actions, contemporary planning techniques may improve IVA performance under the whole spectrum of dynamic conditions and thus in many different game genres (comparing to reactive agents similar to reactive agents used in our study).

If the domain grows larger, situation is more complicated. In less dynamic situations, such as fulfilling a quest in a RPG game or everyday activities in environments similar to The Sims, action planning may be beneficial to IVA performance. However if the environment is changing more frequently and does not introduce significant hostility, the reactive approaches might prove equal or even better. In even more dynamic situations in larger domains, action planning might be beneficial only if the environment is extremely hostile, which is probably not

the case for most games — even in FPS games a large number of changes in the environment is friendly to the agent (e.g., item respawn).

There are nevertheless some limitations to the applicability of results of this paper to a general case. Despite our efforts, 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, although it 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) affected the results in some way. Future studies investigating action planning by means of contemporary academic planners in more complex conditions are needed to pinpoint other situations when planning techniques can be fruitful.

To investigate the effect of dynamicity, we have used the success rate as a primary metric, because it is easy to interpret. Although the success rate seems to reflect the general trends well, it is a crude measure of performance and thus might have hidden some important details. Future studies could benefit from involving a real/integer valued reward signal instead of a simple success/failure.

8. CONCLUSION

We have compared action planning and simple reactive approaches as action selection mechanisms for IVAs in dynamic, game-like environments. Performance of four off-the-shelf classical planners and one special-purpose planner were analysed in detail. The agents were run in the whole spectrum of dynamic

conditions and in environments of various sizes. As a specialized case, we have tested environments that allow for delete-free representation.

We have shown that action planning performs better than the reactive approaches if the environment is either changing slowly, is very hostile to the agent or is small in terms of the planning domain size. Using the delete-free representation has not proven to bring any benefit to the agents. Tested classical planners exhibited surprisingly similar results to each other.

From the methodological perspective, the important contribution of this paper is the usage of a statistical analysis allowing us to quantify the importance of various patterns seen in the data and put our conclusions on a solid ground. At the same time, our test environment presents a new benchmark that is arguably closer to the real game environments than standard planning benchmarks (e.g. International Planning Competition).

Multiple possibilities for future research are open. It would be interesting to see if the given results hold for more extreme parameter values, larger maps and more complex domains. For the gaming AI community, it would be advantageous to invent other benchmarks mimicking more specific game-like scenarios and demonstrate usefulness of planning in these scenarios using analysis similar to ours. For the planning, agent and gaming AI communities, it would be interesting to see how HTN planners and/or systems based on MDP perform in these scenarios.

Exploring the effect of different plan validation strategies (e.g. validating only the first action of a plan) and modifications to the reactive rules would also yield more insights, as well as using a more diverse set of planners: As Blackbox per-

formed very well on small domains, modern SAT-based planners such as Madagascar (Rintanen, 2014) could be competitive even on the larger ones. Including optimal planners for the standard domains could also be interesting, although we suspect them to perform even worse than the satisficing ones in non-static environments.

Another research direction is a development one. Tightly integrating a 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. For instance, ANA* planner already has internal anytime capabilities, but those are not exposed to the user. Exploiting those could bring advantage to the ANA* planner.

To decide whether a reactive or a planning approach is better for IVA design for a particular scenario, it is also important to consider what are the costs associated with implementing either one. Is it easier to develop a reactive agent or to construct a PDDL model? To answer this question would require a qualitatively different analysis than we conducted in this paper.

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 an open source library Planning4J (Cerny, 2012b). We hope that this tool will help other researchers cross the gap between planning and IVAs.

ACKNOWLEDGMENT

This work was partially supported by the student research grant GA UK 559813/2013/A-INF/MFF, by the SVV project number 260 224 and by the grant P103/10/1287 from GAČR.

REFERENCES

ANDERSON, EIKE FALK, LEIGH MCLOUGHLIN, FOTIS LIAROKAPIS, CHRISTOPHER PETERS, PANAGIOTIS PETRIDIS, and SARA DE FREITAS. 2010. Developing serious games for cultural heritage: A state-of-the-art review. Virtual Reality, **14**(4):255–275.

ANONYMOUS AUTHOR. 2012. Spy a vs. Spy (1984 video game). `http://en.wikipedia.org/wiki/Spy_vs._Spy_%281984_video_game%29`. Accessed 2014-05-10.

BALLA, RADHA-KRISHNA, and ALAN FERN. 2009. UCT for tactical assault planning in real-time strategy games. *In* Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence, pp. 40–45.

BARTÁK, ROMAN, FILIP DVOŘÁK, JAKUB GEMROT, CYRIL BROM, and DANIEL TOROPILA. 2012. When planning should be easy: On solving cumulative planning problems. *In* Twenty-Fifth International FLAIRS Conference, pp. 405–410.

BARTHEYE, OLIVIER, and ERIC JACOPIN. 2005. New results for arithmetic constraints partial order planning. *In* Proceedings of the 24th Workshop of the UK Planning and Scheduling Special Interest Group.

BARTHEYE, OLIVIER, and ERIC JACOPIN. 2009. A real-time PDDL-based planning component for video games. *In* Proceedings of the Fifth Artificial Intelligence for Interactive Digital Entertainment Conference, pp. 130–135.

BORDINI, RAFAEL H, JOMI FRED HÜBNER, and MICHAEL WOOLDRIDGE. 2007. Programming multi-agent systems in AgentSpeak using Jason, Volume 8. Wiley-Interscience.

BYLANDER, TOM. 1994. The computational complexity of propositional strips planning. Artificial Intelligence, **69**:165–204.

CARTIER, JEAN-FRANÇOIS. 2011. Étude comparative des planificateurs appliqués au domaine des jeux-vidéos. Master's thesis, Université de Montréal, Québec, Canada.

CERNY, MARTIN. 2012a. Comparing reactive techniques to classical planning for intelligent virtual agents. Master's thesis, Charles University, Prague, Czech Republic.

CERNY, MARTIN. 2012b. Planning4J. http://code.google.com/p/planning4j/. Accessed 2015-05-10.

CERNY, MARTIN, ROMAN BARTAK, CYRIL BROM, and JAKUB GEMROT. 2013. Reactive and planning agents in dynamic game environments: An experimental study. *In* Proceedings of the 5th International Conference on Agents and Artificial Intelligence, pp. 234–240.

CHAMPANDARD, ALEX. 2007. Understanding behavior trees. *In* AIGameDev.com. http://aigamedev.com/open/article/bt-overview/. Accessed 2015-05-03.

CHAMPANDARD, ALEX. 2013. Planning in games: An overview and lessons learned. *In* AIGameDev.com. http://aigamedev.com/open/review/planning-in-games/. Accessed 2013-08-10.

CHAMPANDARD, ALEX, TIM VERWEIJ, and REMCO STRAATMAN. 2009. Killzone 2 multiplayer bots. *In* Game AI Conference 2009.

CONNOLLY, THOMAS M, ELIZABETH A BOYLE, EWAN MACARTHUR, THOMAS HAINEY, and JAMES M BOYLE. 2012. A systematic literature review of empirical evidence on computer games and serious games. Computers & Education, **59**(2):661–686.

DIEZ, DAVID. 2013. Survival analysis in R. http://anson.ucdavis.edu/~hiwang/teaching/10fall/R_tutorial%201.pdf. Accessed 2015-05-10.

EPIC GAMES INC. 2012. Unreal development kit. http://udk.com/.

FIKES, RICHARD E., and NILS .J. NILSSON. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial intelligence, **2**(3-4):189–208.

FOX, MARIA, and DEREK LONG. 2003. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, **20**:61–124.

FU, DANIEL, and RYAN HOULETTE-STOTTLER. 2004. The ultimate guide to FSMs in games. *In* AI Game Programming Wisdom II. Charles River Media, pp. 283–302.

GEMROT, JAKUB, ZDENĚK HLÁVKA, and CYRIL BROM. 2013. Does high-level behavior specification tool make production of virtual agent behaviors better? *In* Cognitive Agents for Virtual Environments, Volume LNCS 7764. Springer, pp. 167–183.

GEMROT, JAKUB, RUDOLF KADLEC, MICHAL BÍDA, ONDŘEJ BURKERT, RADEK PÍBIL, JAN HAVLÍČEK, LUKÁŠ ZEMČÁK, JURAJ ŠIMLOVIČ, RADIM VANSA, MICHAL ŠTOLBA, and OTHERS. 2009. Pogamut 3 can assist developers in building AI (not only) for their videogame agents. *In* Agents for Games and Simulations, Volume LNCS 5920. Springer, pp. 1–15.

GHALLAB, MALIK, DANA NAU, and PAOLO TRAVERSO. 2004. Automated Planning: Theory and Practice,

Chapter Hierarchical Task Network Planning, pp. 229–252. Morgan Kaufmann.

HAWES, NICHOLAS ANDREW. 2004. Anytime deliberation for computer game agents. Ph. D. thesis, University of Birmingham, Birmingham, UK.

HEDGES, LARRY V. 1981. Distribution theory for Glass's estimator of effect size and related estimators. Journal of Educational and Behavioral Statistics, **6**(2):107–128.

HELMERT, MALTE. 2006. The fast downward planning system. Journal of Artificial Intelligence Research, **26**(1):191–246.

HINDRIKS, KOEN V. 2009. Programming rational agents in GOAL. *In* Multi-Agent Programming:Languages, Tools and Applications. Springer, pp. 119–157.

HINDRIKS, KOEN V, BIRNA M RIEMSDIJK, and CATHOLIJN M JONKER. 2012. An empirical study of patterns in agent programs. *In* Principles and Practice of Multi-Agent Systems. Springer, pp. 196–211.

HOANG, HAI, STEPHEN LEE-URBAN, and HÉCTOR MUÑOZ-AVILA. 2005. Hierarchical plan representations for encoding strategic game AI. *In* Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 63–68.

HOFFMANN, JÖRG, and BERNHARD NEBEL. 2001. The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research, **14**:253–302.

HOTHORN, TORSTEN, FRANK BRETZ, and PETER WESTFALL. 2008. Simultaneous inference in general parametric models. Biometrical Journal, **50**(3):346–363.

HSU, CHIH-WEI, and BENJAMIN W WAH. 2008. The SGPlan planning system in IPC-6. *In* Proceedings of the Sixth International Planning Competition, pp. 5–7.

KAUTZ, HENRY, and BART SELMAN. 1998. BLACKBOX: A new approach to the application of theorem proving to problem solving. *In* AIPS98 Workshop on Planning as Combinatorial Search, pp. 58–60.

KELLY, JOHN-PAUL, ADI BOTEA, and SVEN KOENIG. 2008. Offline planning with hierarchical task networks in video games. *In* Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 60–65.

LI, CHU MIN, and ANBULAGAN ANBULAGAN. 1997. Heuristics based on unit propagation for satisfiability problems. *In* Proceedings of the 15th international joint conference on Artifical intelligence, Volume 1, pp. 366–371.

LIPOVETZKY, NIR, and HECTOR GEFFNER. 2011. Searching for plans with carefully designed probes. *In* Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS-2011), pp. 154–161.

LONG, EDMUND. 2007. Enhanced NPC behaviour using goal oriented action planning. Master's thesis, School of Computing and Advanced Technologies, University of Abertay Dundee, Dundee, UK.

MCKILLUP, STEVE. 2011. Statistics explained: an introductory guide for life scientists. Cambridge University Press, Cambridge.

NGUYEN, TRUONG-HUY DINH, DAVID HSU, WEE-SUN LEE, TZE-YUN LEONG, LESLIE PACK KAELBLING, TOMAS LOZANO-PEREZ, and ANDREW HAYDN GRANT. 2011. CAPIR: Collaborative action planning with intention recognition. *In* Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment International Conference, pp. 61–66.

ORKIN, JEFF. 2003. Applying goal-oriented action planning to games. AI Game Programming Wisdom, **2**(1):217–227.

ORKIN, JEFF. 2006. Three states and a plan: the AI of FEAR. *In* Game Developers Conference, Volume 2006, pp. 1–18.

PAUL, RICHARD, DARRYL CHARLES, MICHAEL MCNEILL, and DAVID MCSHERRY. 2011. Adaptive storytelling and story repair in a dynamic environment. *In* Proceedings of the 4th International Conference on Interactive Digital Storytelling, pp. 128–139.

POLLACK, MARTHA E., and JOHN F. HORTY. 1999. There's more to life than making plans. AI Magazine, **20**(4):71–83.

RICHTER, SILVIA, MATTHIAS WESTPHAL, and MALTE HELMERT. 2011. LAMA 2008 and 2011. *In* Seventh International Planning Competition (IPC 2011), Deterministic Part, pp. 50–54.

RINTANEN, JUSSI. 2014. Madagascar: Scalable planning with sat. *In* The 2014 International Planning Competition, pp. 66–69.

SELMAN, BART, HENRY A KAUTZ, and BRAM COHEN. 1994. Noise strategies for improving local search. *In* Proceedings of The Twelfth National Conference on Artificial Intelligence, pp. 337–343.

THERNEAU, TERRY, and THOMAS LUMLEY. 2011. Survival analysis, including penalised likelihood. `http://CRAN.R-project.org/package=survival`. Accessed 2015-05-10.

THOMPSON, TOMMY, and JOHN LEVINE. 2009. Realtime execution of automated plans using evolutionary robotics. *In* IEEE Symposium on Computational Intelligence and Games, pp. 333–340.

VASSOS, STAVROS, and MICHAIL PAPAKONSTANTINOU. 2011. The SimpleFPS planning domain: A PDDL benchmark for proactive NPCs. *In* Workshops at the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 92–97.

Appendix A

A standard PDDL domain representation of a simple environment with two rooms ($A0$, $A1$) and a button ($Button1$) that opens the door between them:

```
(define (domain SimpleStandardDomain)
  (:requirements :strips)
    (:predicates
      (at_A0)
      (at_A1)
      (adjacent_A1__A0)
    )

  (:action move_A0_to_A1
        :precondition
  (and (at_A0) (adjacent_A1__A0))
        :effect
  (and (at_A1) (not (at_A0)))
  )

  (:action move_A1_to_A0
      :precondition
  (and (at_A1) (adjacent_A1__A0))
      :effect
  (and (at_A0) (not (at_A1)))
  )

  (:action push_Button1
      :precondition (at_A0)
      :effect (adjacent_A1__A0)
  )
)
```

A PDDL problem formulation in the domain above:

```
(define (problem SimpleProblem)
  (:domain SimpleStandardDomain)
    (:init (at_A0))
    (:goal (at_A1))
)
```

A delete-free PDDL representation of the same environment:

```
(define (domain SimpleDeleteFreeDomain)
```

```
  (:requirements :strips)
  (:predicates
    (A0_reachable)
    (A1_reachable)
    (adjacent_A1__A0)
  )

  (:action reach_from_A0_to_A1
    :precondition (and (adjacent_A1__A0) (A0_reachable))
    :effect (A1_reachable)
  )

  (:action reach_from_A1_to_A0
    :precondition (and (adjacent_A1__A0) (A1_reachable))
    :effect (A0_reachable)
  )

  (:action push_Button1
    :precondition (A0_reachable)
    :effect (adjacent_A1__A0)
  )
)
```

A PDDL problem formulation for the delete-free problem:

```
(define (problem SimpleDeleteFreeProblem)
(:domain SimpleDeleteFreeDomain)
  (:init (A0_reachable))
  (:goal (A1_reachable))
)
```

Appendix B

[Figure 6 about here.]

Figure 6 shows one of the small maps used in the experiments. An example plan to reach goal, if there is no interference (as given by the Fast Forward planner) is:

(1) Push the east button at A0.

(2) Go to room B0.

(3) Push the west button at B0.

(4) Go to room C0.

(5) Push the south button at C0.

(6) Push the east button at C0.

(7) Go to room E0 (through D0).

(8) Push the west button at E0.

(9) Go to room B0 (through D0 and C0).

(10) Push the east button at B0.

(11) Go to room D0 (through C0).

(12) Push the south button at D0.

(13) Go to room B1 (through C0 and B0).

(14) Push the east button at B1.

(15) Go to room E4 (through B2, B3, C3, D3, E3 and E4).

In the same situation, Greedy agent would push the east button at A0 (rule R2) and then move to B0 (greedy behaviour), where it would push all three buttons (R2 - all the doors closed by the buttons are closed already). At this moment there are two equally good greedy actions: moving to B1 and moving to C0. The agent chooses one randomly.

If the agent moves to B1, it would press east button there (R2) and then wait for interference, as no reachable room would be closer to the goal.

If the agent moves to C0, it would press all three buttons (R2) and move to D0. There it would press south and east buttons (R2) and move greedily to E0 and

press the west button there (R2). At this moment, there is no room closer to goal than E0 — although path to B3 is clear, it has the same distance to goal as E0 and the Greedy agent would perform no action.

Appendix C

[Table 14 about here.]

[Table 15 about here.]

[Table 16 about here.]

[Table 17 about here.]

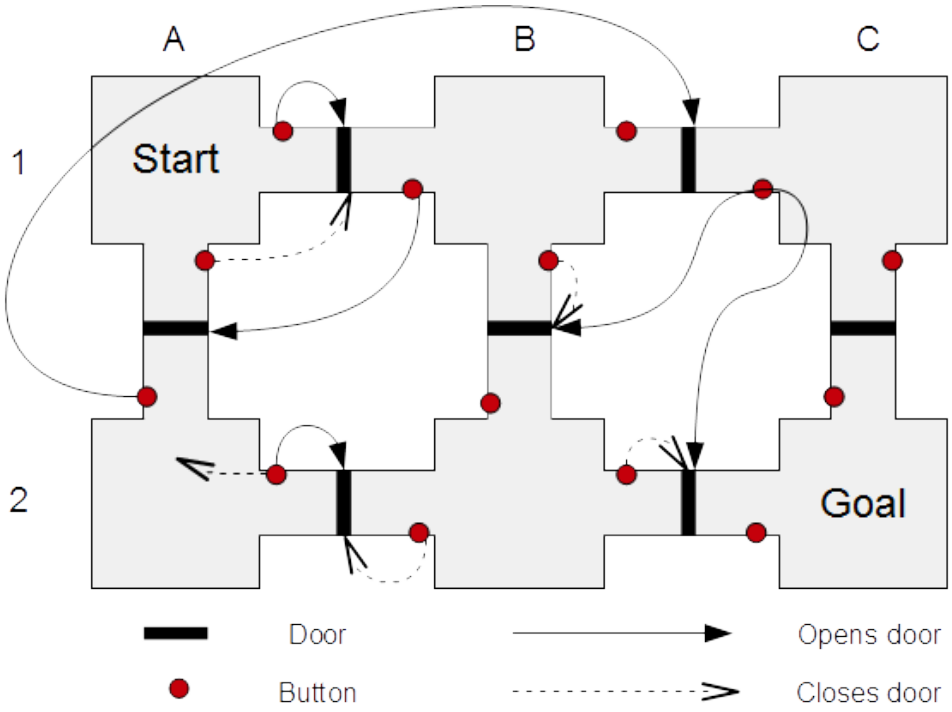

FIGURE 1. An example map representing the test environment class where the experiments were performed.

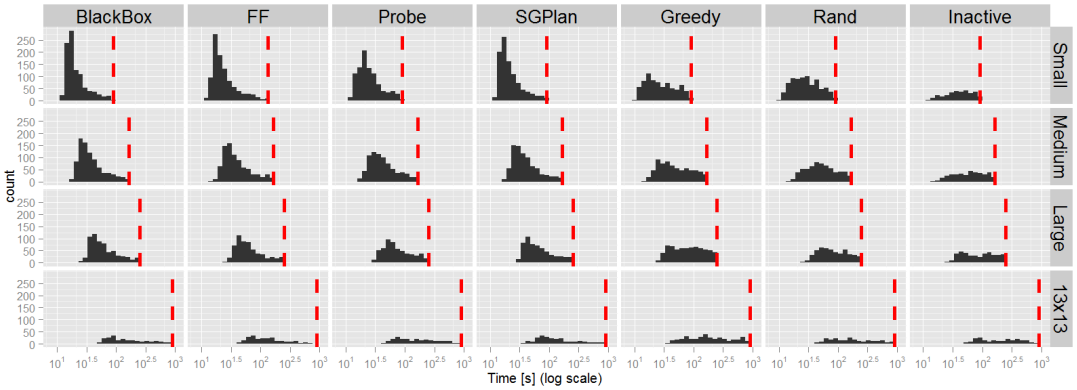

FIGURE 2. Histogram of solution times of agents in the standard maps of various sizes. The x-axis is in log-scale. The red dashed lines indicate the timeout for the respective map size.

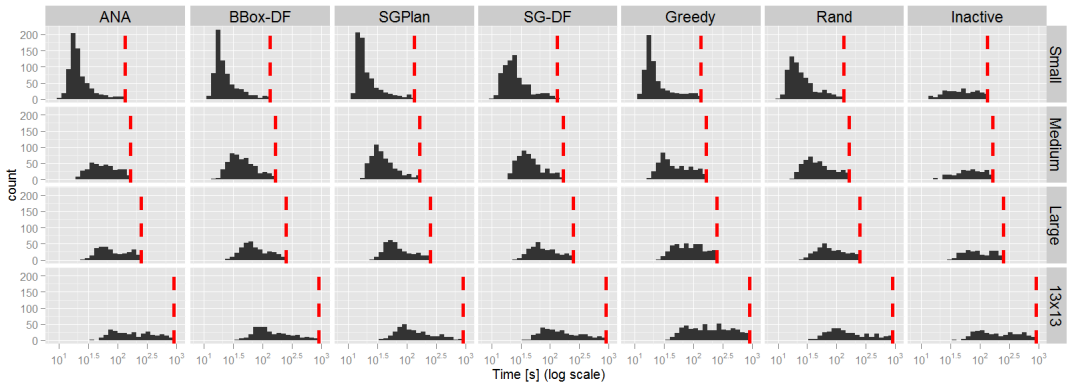

FIGURE 3. Histogram of solution times of agents in the DF maps of various sizes. The x-axis is in log-scale. For some agents in the 13×13 maps, the distribution is bimodal and could not be analysed with the standard parametric methods. The red dashed lines indicate the timeout for the respective map size.

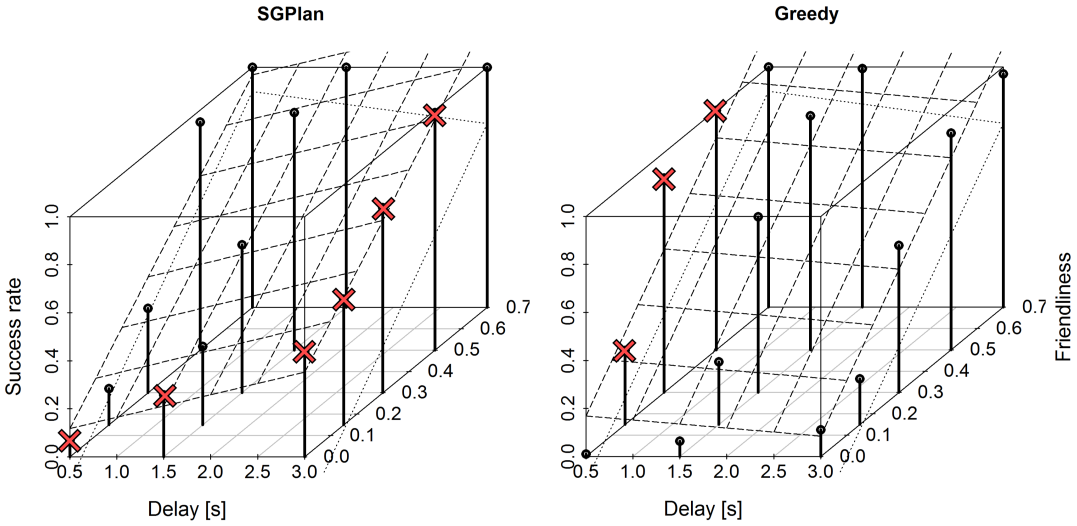

FIGURE 4. Success rate of SGPlan and Greedy bots under different dynamic conditions in the standard maps. The dotted lines show a plane fitted to the results of the Inactive bot. Planes are fitted to the averaged results and they are intended only as a visual cue. Crosses mark points where the respective agent is significantly better than the other (all $p < 0.01$).

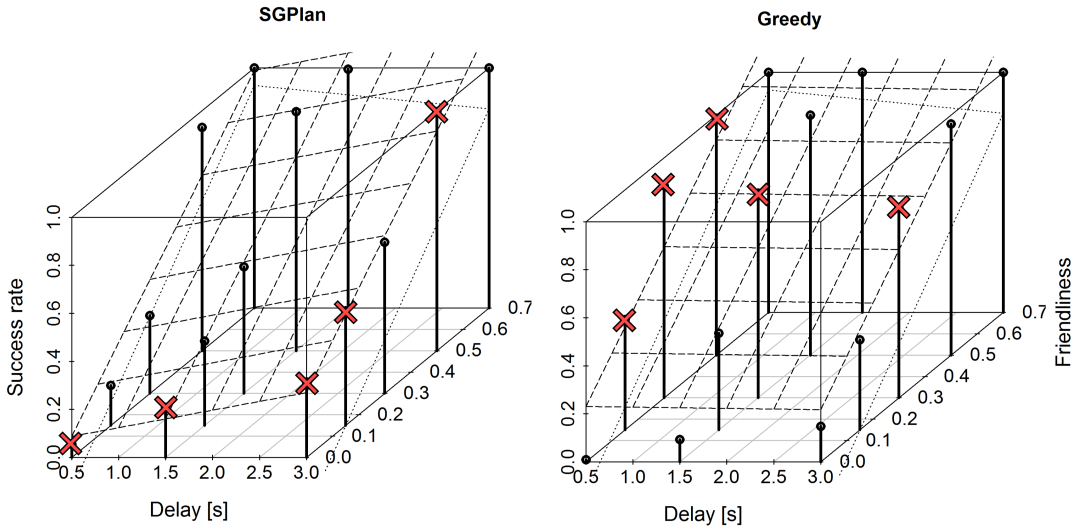

FIGURE 5. Success rate of SGPlan and Greedy bots under different dynamic conditions in the DF maps. The dotted lines show a plane fitted to the results of the Inactive bot. Planes are fitted to the averaged results and they are intended only as a visual cue. Crosses mark points where the respective agent is significantly better than the other (all $p < 0.01$).

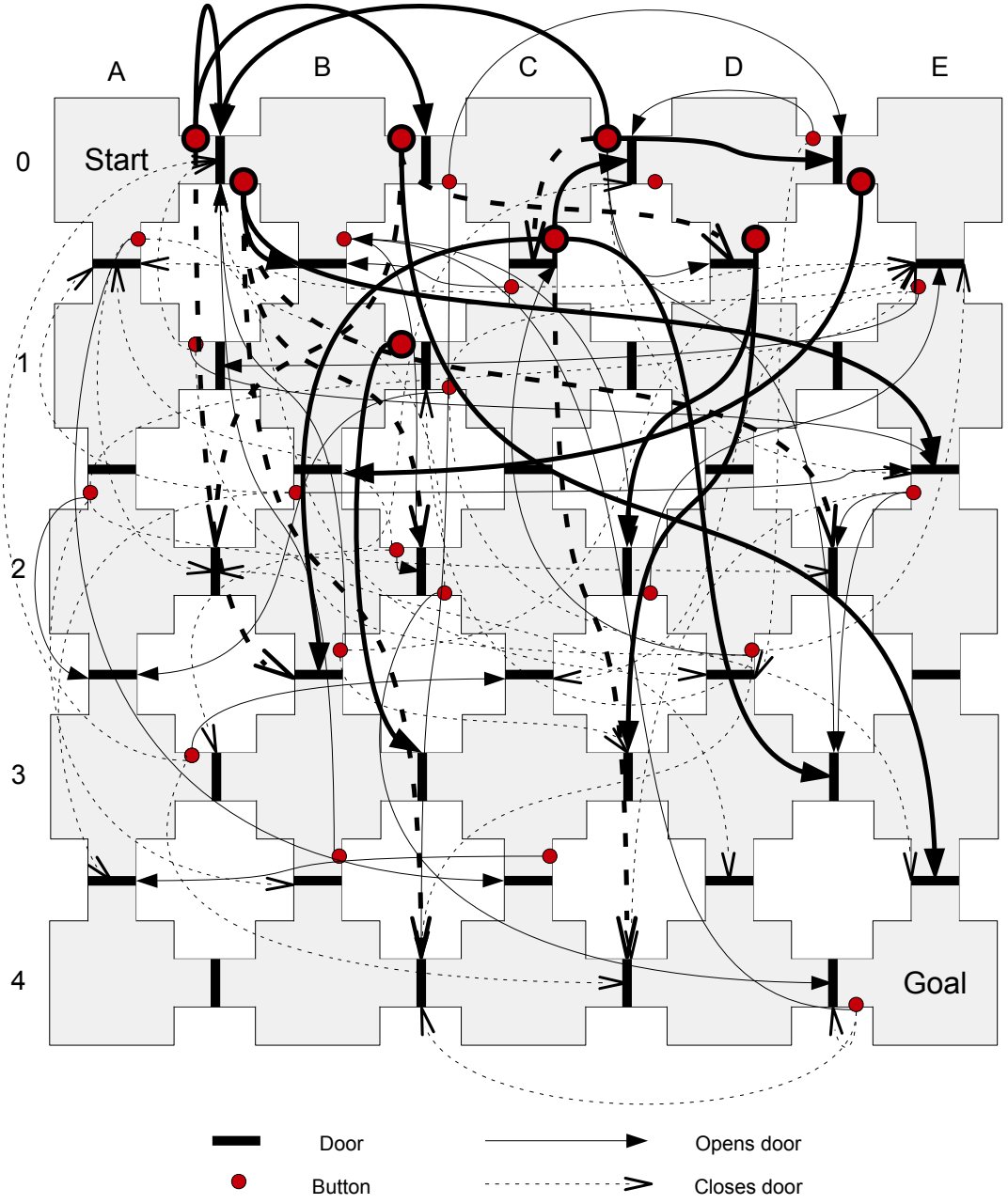

FIGURE 6. A scheme of one of the small maps. 51 buttons that do not open any door were removed for clarity. Buttons that are used on the shortest path found by a planner and their connections to doors are highlighted. The map was generated such that the average number of doors opened per button was 0.5 and average number of doors closed per button was 1.2.

TABLE 1. A subjective classification of several situations that arise in computer games by the three parameters we devised to describe interference in the world, i.e., the changes in the world state that are not controlled by the agent.

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	1 – 5s	Small	Friendly
Navigating through a spaceship falling apart	1 – 3s	Large	Hostile

TABLE 2. Map classes for the experiments and their respective sizes in the number of rooms along with the number of map instances tested during the experiments. The number of actions refers to the number of grounded actions and differs in different map instances.

Map class/size	**Number of maps**		**Domain size**	**Time limit**
	Standard	**Delete-free**	**(atoms/actions)**	**[s]**
Small (5 × 5)	9	6	65 / 90 - 160	135
Medium (7 × 7)	9	6	133 / 190 - 336	165
Large (10 × 10)	9	6	280 / 390 - 720	250
13 × 13	4	6	481 / 650 - 1248	900

TABLE 3. Average success rates over all experiment runs in standard maps. Best results in each row are highlighted.

Map	BB	FF	Probe	SG	Greedy	Rand	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
13 × 13	0.40	0.43	0.42	0.44	**0.68**	0.42	0.38
Total	**0.60**	**0.60**	**0.57**	**0.60**	**0.61**	**0.50**	**0.32**

TABLE 4. Average solution times [s] for standard maps with standard deviation (in brackets). Best results in each row are highlighted.

Map	BB	FF	Probe	SG	Greedy	Rand	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)	103.8 (61)
13 × 13	214 (188)	**167 (144)**	206 (167)	181 (172)	253 (218)	230 (199)	255 (187)

TABLE 5. Ordering of agents in standard maps according to survival model of solution time. If two agents have different ranks, their mean solution time is significantly different (all $p < 0.01$).
Note that in large maps ordering given by statistically significant differences is only partial: BB is significantly better than FF and SG, but not significantly better than Greedy. At the same time Greedy is not significantly better than either FF or SG. This is shown by overlapping ranks.

Map	BB	FF	Probe	SG	Greedy	Rand	Inactive
Small	**1–2**	3–4	3–4	**1–2**	5–6	5–6	7
Medium	**1**	2–3	4	2–3	5	6	7
Large	**1–2**	3–4	5	3–4	2–4	6	7
13 × 13	4–6	2–3	4–6	2–3	**1**	4–6	7

TABLE 6. Average deliberation times [%] for standard maps with standard deviation (in brackets). Deliberation time includes path finding. Best results among planning agents in each row are highlighted.

Map	BB	FF	Probe	SG	Greedy	Rand	Inactive
Small	**21 (19)**	24 (21)	23 (21)	22 (20)	0.08 (0.05)	0.11 (0.07)	0.03 (0.07)
Medium	**25 (24)**	26 (26)	27 (25)	27 (25)	0.08 (0.07)	0.11 (0.06)	0.03 (0.06)
Large	**37 (31)**	38 (30)	41 (30)	**37 (31)**	0.14 (0.25)	0.18 (0.11)	0.05 (0.07)
13 × 13	78 (28)	50 (36)	62 (33)	**43 (36)**	0.21 (0.56)	0.32 (0.42)	0.06 (0.13)
Total	**34 (31)**	**32 (29)**	**34 (30)**	**30 (28)**	**0.11 (0.25)**	**0.16 (0.18)**	**0.04 (0.09)**

TABLE 7. Average success rates over all experiment runs in DF maps. Best results in each row are highlighted.

Map size	ANA*	BB	BB-DF	FF	FF-DF	Probe	Probe-DF	SG	SG-DF	Greedy	Rand	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
13 × 13	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 8. Average solution times [s] for DF maps with standard deviation (in brackets). Best results in each row are highlighted.

Map size	ANA*	BB	BB-DF	FF	FF-DF	Probe	Probe-DF	SG	SG-DF	Greedy	Rand	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)
13 × 13	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 9. Ordering of agents in DF maps according to survival model of solution time. Note that ordering given by statistically significant differences is only partial. For example, in large maps Greedy is significantly better SG, but not significantly better than BB. At the same time BB is not significantly better than SG. This is shown by overlapping ranks. If two agents have no intersection of rank intervals, their mean solution time is significantly different (all $p < 0.05$). Agents with no ranking for 13×13 maps had non-normal distributions of solution times and thus could not be analysed.

Map size	ANA*	BB	BB-DF	FF	FF-DF	Probe	Probe-DF	SG	SG-DF	Greedy	Rand	Inactive
Small	**1–4**	**1–4**	5–9	**1–4**	5–9	5–9	5–9	**1–4**	5–9	10–11	10–11	12
Medium	11	**1–3**	4–8	**1–3**	4–8	4–8	4–8	**1–3**	4–8	9–10	9–10	12
Large	9–11	2–5	4–6	4–6	6–10	6–10	9–11	3–5	6–10	**1–2**	9–11	12
13 × 13	N/A	N/A	N/A	3–5	3–5	N/A	N/A	2–3	4–5	**1**	N/A	N/A

TABLE 10. Average deliberation times [%] for DF maps with standard deviation (in brackets). Deliberation time includes path finding. Best results among planning agents in each row are highlighted.

Map size	ANA*	BB	BB-DF	FF	FF-DF	Probe	Probe-DF	SG	SG-DF	Greedy	Rand	Inactive
Small	**05 (06)**	19 (18)	20 (19)	19 (18)	20 (18)	21 (19)	18 (18)	20 (19)	20 (18)	0.05 (0.04)	0.07 (0.06)	< 0.01 ($< 10^{-4}$)
Medium	72 (29)	**27 (23)**	29 (24)	**27 (23)**	29 (24)	29 (25)	31 (24)	28 (24)	29 (24)	0.07 (0.05)	0.11 (0.08)	< 0.01 ($< 10^{-4}$)
Large	62 (41)	46 (31)	**41 (30)**	43 (30)	43 (30)	45 (30)	49 (28)	43 (30)	43 (30)	0.11 (0.20)	0.18 (0.11)	< 0.01 ($< 10^{-4}$)
13×13	89 (18)	87 (24)	50 (33)	51 (37)	46 (33)	80 (25)	74 (26)	48 (34)	**47 (33)**	0.18 (0.53)	0.30 (0.25)	< 0.01 ($< 10^{-4}$)
Total	**57 (41)**	**45 (36)**	**35 (29)**	**35 (30)**	**34 (29)**	**44 (33)**	**43 (32)**	**34 (30)**	**35 (29)**	**0.10 (0.29)**	**0.16 (0.17)**	**< 0.01** ($< 10^{-4}$)

TABLE 11. Effect sizes for the main metrics and variations of interference parameters for SGPlan and Greedy agents. Since all classical planners have very similar results, the effect sizes of SG apply very closely to other planning agents.
Recall that success rate effect size is computed simply as the difference in success rate, positive number reflects better performance with specified parameter values than with the baseline, difference > 0.10 is considered large. The effect for time is evaluated with Hedge's g, positive number reflects shorter time and thus better performance with specified parameter values than with the baseline values, absolute value > 0.8 is considered large. For survival, the effect size is computed as the multiplicative coefficient by which estimated mean solution times differ between specified values and baseline, coefficient larger than 1 reflects shorter time and thus better performance with specified parameter values than with the baseline values, coefficient lower than 0.75 or greater than 1.5 is considered large.
Solid fill represents large effect in favour of the specified parameters, dashed fill represents large effect in favour of the baseline.

Parameter		**SGPlan, standard maps**			**Greedy, standard maps**			**SGPlan, delete-free maps**			**Greedy, delete-free maps**		
Baseline	**Value**	**Success**	**Time**	**Surv.**	**Success**	**Time**	**Surv.**	**Success**	**Time**	**Surv.**	**Success**	**Time**	**Surv.**
Delay 0.5s	1.5s	0.12	−0.21	1.50	−0.03	−0.05	0.96	0.10	−0.28	1.43	0.00	−0.26	1.19
	3.0s	0.23	−0.31	2.03	−0.08	−0.00	0.84	0.16	−0.40	1.68	−0.01	−0.31	1.19
Impact 0.05	0.1	−0.08	0.08	0.78	0.03	−0.05	1.12	−0.06	0.13	0.84	0.01	0.06	0.99
	0.2	−0.18	0.19	0.58	0.06	−0.00	1.15	−0.13	0.20	0.70	0.02	0.20	0.93
Friendliness 0	0.15	0.09	0.47	1.20	0.19	1.31	1.55	0.13	0.58	0.66	0.32	1.33	2.6
	0.3	0.32	0.75	1.94	0.68	1.52	3.82	0.28	0.75	0.91	0.75	1.62	4.50
	0.5	0.74	0.84	4.69	0.89	1.27	7.40	0.77	1.3	2.39	0.89	1.41	8.49
	0.7	0.76	0.37	7.71	0.92	0.76	12.16	0.82	0.73	4.50	0.91	1.10	13.68

TABLE 12. Effect sizes for the main metrics and variations of interference parameters for SGPlan and Greedy agents. Since all classical planners have very similar results, the effect sizes of SG apply very closely to other planning agents.
Recall that success rate effect size is computed simply as the difference in success rate, positive number reflects better performance with specified parameter values than with the baseline, difference > 0.10 is considered large. The effect for time is evaluated with Hedge's g, positive number reflects shorter time and thus better performance with specified parameter values than with the baseline values, absolute value > 0.8 is considered large. For survival, the effect size is computed as the multiplicative coefficient by which estimated mean solution times differ between specified values and baseline, coefficient larger than 1 reflects shorter time and thus better performance with specified parameter values than with the baseline values, coefficient lower than 0.75 or greater than 1.5 is considered large.
Solid fill represents large effect in favour of the specified parameters, dashed fill represents large effect in favour of the baseline.
Assessing survival effect size for parameter combination would require a different model than the one used for all other data in this paper and thus was not included.

Parameter				**SGPlan, standard maps**		**Greedy, standard maps**		**SGPlan, delete-free maps**		**Greedy, delete-free maps**	
Baseline	**Value**			**Success**	**Time**	**Success**	**Time**	**Success**	**Time**	**Success**	**Time**
	Del 1.5s	*	Imp 0.1	0.04	−0.01	−0.04	−0.02	0.05	−0.01	−0.02	−0.13
Delay 0.5s *	Del 3.0s	*	Imp 0.1	0.16	−0.12	−0.06	0.04	0.10	−0.11	−0.05	−0.24
Impact 0.05	Del 1.5s	*	Imp 0.2	−0.06	0.02	0.03	0.04	−0.03	0.12	0.00	0.04
	Del 3.0s	*	Imp 0.2	0.05	0.08	−0.03	0.00	0.03	−0.20	0.01	−0.05
	Imp 0.1	*	Fr 0.15	−0.07	0.43	0.15	0.23	0.02	0.61	0.25	1.25
	Imp 0.2	*	Fr 0.15	−0.24	0.59	0.16	1.91	−0.08	0.70	0.35	1.92
	Imp 0.1	*	Fr 0.3	0.16	0.84	0.63	1.53	0.29	0.78	0.75	1.66
Impact 0.05 *	Imp 0.2	*	Fr 0.3	0.00	1.01	0.75	1.83	0.20	0.77	0.73	1.98
Friendliness 0	Imp 0.1	*	Fr 0.5	0.56	0.85	0.87	1.18	0.66	1.02	0.85	1.47
	Imp 0.2	*	Fr 0.5	0.54	1.00	0.88	1.20	0.64	1.13	0.86	1.44
	Imp 0.1	*	Fr 0.7	0.58	0.34	0.88	0.73	0.69	0.70	0.87	1.18
	Imp 0.2	*	Fr 0.7	0.58	0.35	0.89	0.47	0.68	0.70	0.87	1.05
	Del 1.5s	*	Fr 0.15	0.28	0.57	0.25	1.51	0.29	0.51	0.39	1.01
	Del 3s	*	Fr 0.15	0.45	0.51	0.17	0.94	0.42	0.27	0.36	0.62
	Del 1.5s	*	Fr 0.3	0.54	0.79	0.72	1.82	0.47	0.63	0.84	1.32
Delay 0.5s *	Del 3s	*	Fr 0.3	0.70	0.71	0.60	1.50	0.57	0.51	0.78	1.11
Friendliness 0	Del 1.5s	*	Fr 0.5	0.92	0.91	0.96	1.67	0.93	0.92	0.99	1.9
	Del 3s	*	Fr 0.5	0.92	0.81	0.89	1.76	0.93	0.76	0.95	1.8
	Del 1.5s	*	Fr 0.7	0.93	0.55	0.98	1.33	0.93	0.58	0.99	0.74
	Del 3s	*	Fr 0.7	0.93	0.57	0.96	1.53	0.94	0.64	0.99	0.85

TABLE 13. Success rate for three representative agents in different dynamic conditions with the same mean number of door changes per second.

Agent	0.067 door changes / s		0.033 door changes / s	
	Impact 0.1 Delay 1.5s	Impact 0.2 Delay 3s	Impact 0.05 Delay 1.5s	Impact 0.1 Delay 3s
SGPlan	0.64	0.64	0.75	0.75
Greedy	0.58	0.60	0.56	0.56
Rand	0.53	0.52	0.57	0.57

TABLE 14. Number of planner execution of SGPlan agent depending on friendliness, delay and map type. We report average number of planner executions per second (Exec./s) and average absolute number of planner executions (Exec. Total), with standard deviation in brackets.

Standard Maps

		Small Maps			Medium Maps			Large Maps			13 × 13 Maps		
Friendliness	**Delay**	**Exec./s**	**Exec. Total**		**Exec./s**	**Exec. Total**		**Exec./s**	**Exec. Total**		**Exec./s**	**Exec. Total**	
0	0.5	0.09	6.57	(5.52)	0.05	7.30	(4.96)	0.03	6.81	(5.53)	0.01	5.22	(2.51)
0	1.5	0.08	3.98	(3.13)	0.06	6.01	(4.54)	0.02	5.28	(3.68)	0.00	4.33	(1.77)
0	3	0.08	2.37	(1.76)	0.06	4.40	(3.83)	0.02	4.33	(2.43)	0.00	4.00	(1.77)
0.15	0.5	0.43	34.78	(20.21)	0.42	65.98	(24.33)	0.44	108.23	(26.53)	0.36	324.31	(172.86)
0.15	1.5	0.26	18.81	(24.55)	0.33	49.48	(31.34)	0.41	100.86	(32.96)	0.46	411.19	(111.45)
0.15	3	0.15	8.16	(17.96)	0.22	30.57	(34.15)	0.35	83.16	(43.83)	0.46	414.50	(127.47)
0.3	0.5	0.32	22.38	(17.46)	0.47	67.96	(37.22)	0.44	108.74	(35.77)	0.38	335.69	(138.97)
0.3	1.5	0.16	8.20	(12.77)	0.30	36.07	(37.85)	0.36	82.40	(47.88)	0.40	326.25	(109.08)
0.3	3	0.13	6.06	(12.62)	0.21	20.43	(30.01)	0.28	58.09	(50.98)	0.35	271.50	(173.80)
0.5	0.5	0.21	7.52	(7.71)	0.28	20.11	(19.56)	0.31	46.99	(34.03)	0.33	111.14	(74.17)
0.5	1.5	0.12	3.30	(3.67)	0.14	7.11	(6.66)	0.16	16.56	(16.62)	0.20	45.78	(39.27)
0.5	3	0.11	3.63	(7.34)	0.10	4.12	(5.43)	0.13	14.01	(19.37)	0.15	24.56	(25.04)
0.7	0.5	0.12	2.51	(1.86)	0.13	4.22	(3.30)	0.11	6.02	(4.09)	0.09	9.56	(7.70)
0.7	1.5	0.09	1.65	(1.35)	0.08	2.41	(1.68)	0.07	3.22	(2.30)	0.05	3.42	(2.56)
0.7	3	0.07	1.33	(0.82)	0.06	1.75	(1.22)	0.06	3.11	(5.18)	0.05	3.92	(3.26)

Delete-free Maps

		Small DF Maps			Medium DF Maps			Large DF Maps			13 × 13 DF Maps		
Friendliness	**Delay**	**Exec./s**	**Exec. Total**		**Exec./s**	**Exec. Total**		**Exec./s**	**Exec. Total**		**Exec./s**	**Exec. Total**	
0	0.5	0.25	9.54	(10.31)	0.26	6.98	(4.12)	0.27	5.26	(4.03)	0.26	6.07	(3.86)
0	1.5	0.14	3.85	(3.84)	0.20	6.09	(3.79)	0.19	4.20	(2.16)	0.18	4.93	(3.08)
0	3	0.11	2.06	(1.32)	0.14	5.56	(3.56)	0.15	4.43	(2.20)	0.14	4.13	(2.20)
0.15	0.5	0.40	45.41	(29.95)	0.47	74.63	(25.39)	0.54	136.22	(29.05)	0.55	491.17	(259.68)
0.15	1.5	0.21	15.94	(26.23)	0.36	51.19	(35.88)	0.51	126.91	(22.34)	0.59	533.98	(263.74)
0.15	3	0.12	4.67	(13.70)	0.23	29.30	(38.10)	0.43	105.06	(49.28)	0.50	448.74	(156.15)
0.3	0.5	0.31	23.67	(24.40)	0.40	58.06	(29.78)	0.52	131.19	(27.70)	0.44	389.43	(194.73)
0.3	1.5	0.16	7.02	(11.34)	0.22	22.87	(25.56)	0.45	106.76	(38.86)	0.43	371.76	(156.30)
0.3	3	0.11	3.80	(11.82)	0.15	10.89	(18.41)	0.34	80.98	(53.76)	0.42	331.63	(153.02)
0.5	0.5	0.20	7.72	(8.98)	0.25	17.19	(16.68)	0.35	61.24	(33.94)	0.36	129.13	(96.39)
0.5	1.5	0.10	2.13	(1.49)	0.13	6.46	(5.86)	0.18	23.17	(21.38)	0.23	52.31	(39.89)
0.5	3	0.10	2.41	(4.98)	0.08	2.83	(2.72)	0.13	15.76	(21.73)	0.15	28.61	(29.81)
0.7	0.5	0.12	2.30	(1.30)	0.12	3.76	(2.14)	0.11	6.20	(4.14)	0.10	10.33	(7.11)
0.7	1.5	0.09	1.41	(0.77)	0.07	2.19	(1.52)	0.06	3.35	(2.47)	0.06	4.56	(2.64)
0.7	3	0.07	1.22	(0.50)	0.07	2.20	(1.47)	0.06	3.69	(6.39)	0.05	4.43	(3.57)

TABLE 15. Comparison of success rate over all experiment runs in standard maps. The upper number in each cell is the difference between success rate of the row agent and the column agent, below is p-value for this difference. Solid fill represents significant difference in favour of the row agent (higher success rate), dashed fill represents significant difference in favour of the column agent.

	FF	**Probe**	**SG**	**Greedy**	**Rand**	**Inactive**
BB	0.00 1.00	0.03 $< 10^{-3}$	0.00 1.00	0.00 0.96	0.10 $< 10^{-3}$	0.28 $< 10^{-3}$
FF	—	0.03 $< 10^{-3}$	0.00 1.00	0.00 0.90	0.10 $< 10^{-3}$	0.28 $< 10^{-3}$
Probe		—	−0.03 $< 10^{-3}$	−0.04 $< 10^{-3}$	0.06 $< 10^{-3}$	0.25 $< 10^{-3}$
SG			—	−0.01 0.81	0.10 $< 10^{-3}$	0.28 $< 10^{-3}$
Greedy				—	0.10 $< 10^{-3}$	0.29 $< 10^{-3}$
Rand					—	0.18 $< 10^{-3}$

TABLE 16. Comparison of the success rate in the standard maps by map size. The upper number in each cell is the difference between the success rates of row and column agents, below is p-value for this difference. Solid background represents a significant difference in favour of the row agent (higher success rate), dashed background represents a significant difference in favour of the column agent.

Small maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	0.00 1.00	0.04 0.05	0.00 1.00	0.19 $< 10^{-3}$	0.16 $< 10^{-3}$	0.55 $< 10^{-3}$
FF	—	0.04 0.04	0.00 1.00	0.19 $< 10^{-3}$	0.16 $< 10^{-3}$	0.55 $< 10^{-3}$
Probe		—	−0.04 0.06	0.14 $< 10^{-3}$	0.12 $< 10^{-3}$	0.51 $< 10^{-3}$
SG			—	0.19 $< 10^{-3}$	0.16 $< 10^{-3}$	0.55 $< 10^{-3}$
Greedy				—	−0.03 0.36	0.36 $< 10^{-3}$
Rand					—	0.39 $< 10^{-3}$

Medium maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	0.02 0.33	0.06 $< 10^{-3}$	0.02 0.41	0.12 $< 10^{-3}$	0.16 $< 10^{-3}$	0.39 $< 10^{-3}$
FF	—	0.03 0.12	0.00 1.00	0.09 $< 10^{-3}$	0.14 $< 10^{-3}$	0.36 $< 10^{-3}$
Probe		—	−0.03 0.09	0.06 $< 10^{-3}$	0.10 $< 10^{-3}$	0.33 $< 10^{-3}$
SG			—	0.09 $< 10^{-3}$	0.14 $< 10^{-3}$	0.36 $< 10^{-3}$
Greedy				—	0.05 < 0.01	0.27 $< 10^{-3}$
Rand					—	0.22 $< 10^{-3}$

Large maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	0.02 0.06	0.04 $< 10^{-3}$	0.02 0.25	-0.05 $< 10^{-3}$	0.10 $< 10^{-3}$	0.18 $< 10^{-3}$
FF	—	0.01 0.76	0.00 0.99	-0.08 $< 10^{-3}$	0.07 $< 10^{-3}$	0.15 $< 10^{-3}$
Probe		—	−0.02 0.38	-0.10 $< 10^{-3}$	0.05 $< 10^{-3}$	0.13 $< 10^{-3}$
SG			—	-0.08 $< 10^{-3}$	0.08 $< 10^{-3}$	0.16 $< 10^{-3}$
Greedy				—	0.16 $< 10^{-3}$	0.24 $< 10^{-3}$
Rand					—	0.08 $< 10^{-3}$

13 × 13 maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	−0.02 0.44	−0.01 0.87	−0.04 0.03	−0.27 $< 10^{-3}$	−0.01 0.87	0.01 0.80
FF	—	0.00 0.99	−0.01 0.92	−0.25 $< 10^{-3}$	0.00 0.99	0.04 0.01
Probe		—	−0.02 0.53	−0.26 $< 10^{-3}$	0.00 1.00	0.03 0.11
SG			—	−0.23 $< 10^{-3}$	0.02 0.53	0.05 $< 10^{-3}$
Greedy				—	0.26 $< 10^{-3}$	0.29 $< 10^{-3}$
Rand					—	0.03 0.11

TABLE 17. Comparison of solution time in the standard maps by map size. The upper number in each cell is the effect size (Hedge's g) of the column agent compared to the row agent, below is p-value for the hypothesis that their solution times are drawn from the same distribution. The time has been log transformed to be closer to normal distribution. Solid background represents a significant difference in favour of the row agent (shorter solution time), dashed background represents a significant difference in favour of the column agent.

Small maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	0.34 $< 10^{-3}$	0.41 $< 10^{-3}$	0.12 0.12	0.54 $< 10^{-3}$	0.68 $< 10^{-3}$	1.44 $< 10^{-3}$
FF	—	0.05 0.85	−0.22 $< 10^{-3}$	0.22 $< 10^{-3}$	0.33 $< 10^{-3}$	1.00 $< 10^{-3}$
Probe		—	−0.29 $< 10^{-3}$	0.17 < 0.01	0.28 $< 10^{-3}$	0.98 $< 10^{-3}$
SG			—	0.43 $< 10^{-3}$	0.56 $< 10^{-3}$	1.29 $< 10^{-3}$
Greedy				—	0.09 0.2	0.67 $< 10^{-3}$
Rand					—	0.61 $< 10^{-3}$

Medium maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	0.11 0.48	0.29 $< 10^{-3}$	0.13 0.14	0.47 $< 10^{-3}$	0.62 $< 10^{-3}$	0.9 $< 10^{-3}$
FF	—	0.17 0.17	0.07 0.99	0.36 $< 10^{-3}$	0.50 $< 10^{-3}$	0.76 $< 10^{-3}$
Probe		—	−0.15 0.53	0.19 $< 10^{-3}$	0.33 $< 10^{-3}$	0.59 $< 10^{-3}$
SG			—	0.34 $< 10^{-3}$	0.49 $< 10^{-3}$	0.75 $< 10^{-3}$
Greedy				—	0.11 $< 10^{-3}$	0.34 $< 10^{-3}$
Rand					—	0.24 $< 10^{-3}$

Large maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	0.08 0.49	0.33 0.02	0.15 0.4	0.45 $< 10^{-3}$	0.51 $< 10^{-3}$	0.59 $< 10^{-3}$
FF	—	0.24 0.80	0.06 0.99	0.36 $< 10^{-3}$	0.42 $< 10^{-3}$	0.50 $< 10^{-3}$
Probe		—	−0.17 0.88	0.14 $< 10^{-3}$	0.18 $< 10^{-3}$	0.27 $< 10^{-3}$
SG			—	0.30 $< 10^{-3}$	0.35 $< 10^{-3}$	0.44 $< 10^{-3}$
Greedy				—	0.02 $< 10^{-3}$	0.11 $< 10^{-3}$
Rand					—	0.10 $< 10^{-3}$

13 × 13 maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	−0.26 $< 10^{-3}$	−0.01 $< 10^{-3}$	−0.22 $< 10^{-3}$	0.16 0.99	0.07 $< 10^{-3}$	0.26 $< 10^{-3}$
FF	—	0.25 $< 10^{-3}$	0.03 0.96	0.41 $< 10^{-3}$	0.33 $< 10^{-3}$	0.54 $< 10^{-3}$
Probe		—	−0.21 $< 10^{-3}$	0.17 $< 10^{-3}$	0.08 $< 10^{-3}$	0.28 $< 10^{-3}$
SG			-	0.37 $< 10^{-3}$	0.29 $< 10^{-3}$	0.49 $< 10^{-3}$
Greedy				—	−0.08 $< 10^{-3}$	0.08 $< 10^{-3}$
Rand					—	0.18 0.24

TABLE 18. Comparison of solution time in the standard maps by map size using the right-censored accelerated failure-time survival model. The upper number in each cell is the effect size: the quotient of the estimated mean solution time of the column agent to the estimated mean solution time of the row agent. Below is p value for the hypothesis that their solution times, including failure to reach the goal, are drawn from the same distribution. Solid background represents a significant difference in favour of the row agent (quotient larger than 1), dashed background represents a significant difference in favour of the column agent.

Small maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	1.20 $< 10^{-4}$	1.26 $< 10^{-4}$	1.04 0.76	1.77 $< 10^{-4}$	1.74 $< 10^{-4}$	4.99 $< 10^{-4}$
FF	—	1.04 0.79	0.86 $< 10^{-4}$	1.46 $< 10^{-4}$	1.44 $< 10^{-4}$	4.13 $< 10^{-4}$
Probe		—	0.82 $< 10^{-4}$	1.40 $< 10^{-4}$	1.38 $< 10^{-4}$	3.96 $< 10^{-4}$
SG			—	1.69 $< 10^{-4}$	1.66 $< 10^{-4}$	4.77 $< 10^{-4}$
Greedy				—	0.98 0.99	2.82 $< 10^{-4}$
Rand					—	2.86 $< 10^{-4}$

Medium maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	1.11 0.01	1.29 $< 10^{-3}$	1.11 < 0.01	1.58 $< 10^{-3}$	1.88 $< 10^{-3}$	3.76 $< 10^{-3}$
FF	—	1.16 $< 10^{-3}$	1.00 1.00	1.42 $< 10^{-3}$	1.69 $< 10^{-3}$	3.37 $< 10^{-3}$
Probe		—	0.86 $< 10^{-3}$	1.22 $< 10^{-3}$	1.45 $< 10^{-3}$	2.91 $< 10^{-3}$
SG			—	1.41 $< 10^{-3}$	1.68 $< 10^{-3}$	3.36 $< 10^{-3}$
Greedy				—	1.18 $< 10^{-3}$	2.37 $< 10^{-3}$
Rand					—	1.99 $< 10^{-3}$

Large maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	1.15 < 0.01	1.31 $< 10^{-3}$	1.14 $< 10^{-3}$	1.05 0.66	1.65 $< 10^{-3}$	2.22 $< 10^{-3}$
FF	—	1.16 $< 10^{-3}$	1.00 0.99	0.92 0.2	1.45 $< 10^{-3}$	1.95 $< 10^{-3}$
Probe		—	0.86 $< 10^{-3}$	0.79 $< 10^{-3}$	1.25 $< 10^{-3}$	1.68 $< 10^{-3}$
SG			—	0.91 0.11	1.44 $< 10^{-3}$	1.94 $< 10^{-3}$
Greedy				—	1.56 $< 10^{-3}$	2.11 $< 10^{-3}$
Rand					—	1.34 $< 10^{-3}$

13 × 13 maps

	FF	Probe	SG	Greedy	Rand	Inactive
BB	0.77 $< 10^{-3}$	0.92 0.64	0.73 $< 10^{-3}$	0.42 $< 10^{-3}$	0.96 0.99	1.28 $< 10^{-3}$
FF	—	1.19 < 0.01	0.95 0.96	0.54 $< 10^{-3}$	1.25 $< 10^{-3}$	1.66 $< 10^{-3}$
Probe		—	0.79 $< 10^{-3}$	0.45 $< 10^{-3}$	1.04 0.95	1.38 $< 10^{-3}$
SG			—	0.57 $< 10^{-3}$	1.31 $< 10^{-3}$	1.73 $< 10^{-3}$
Greedy				—	2.30 $< 10^{-3}$	3.04 $< 10^{-3}$
Rand					—	1.32 $< 10^{-3}$