Towards Believable Non-Player Characters with Domain-Independent Planning

Nicklas Slorup Johansen[1], Lasse Brink Kær[1], Jonas Alexander Baungård Stolberg[1], Rasmus Grønkjær Tollund[1], Nikolaj Hyldig[2], Paw Oktober[2], Álvaro Torralba[1]

[1] Aalborg University, Denmark
[2] GenNarrate, Denmark
{nsjo18,rtollu18,jstolb14,lkar18}@student.aau.dk, nikolajh@gennarrate.com, pawoktober@gmail.com, alto@cs.aau.dk

Abstract

This paper explores how domain-independent classical planners can be used to control non-player characters in an interactive storytelling videogame engine. Our planning models take into account the emotional state of the characters to decide the course of action, resulting in believable agents that act in a goal-driven manner with their own motivations. Moreover, despite using single-agent models, we incorporate reasoning about the actions of other agents, so that characters interact with each other to achieve their goals. We analyze the performance of current classical planners on our NPC planning models and show that they are capable to find plans under short time limits.

1 Introduction

In Videogames, it may be desirable that the Non-Player Characters (NPCs) dynamically adapt their behaviour depending on the player actions. This can create a more living and thus believable game world, hereby increasing social immersion in the game world. One option is to specify the possible courses of action in advance, in so called behaviour trees (Lim, Baumgarten, and Colton 2010). While this provides a lot of control to the game designer, it requires a design process that takes into consideration all possible courses of action in advance. This may be hard to do, specially for games where NPCs have a broad range of actions, items, and/or variables that influence the decision-making process. Furthermore, behaviour trees may need to be updated whenever new abilities or items are introduced. On the other hand, domain-independent planning is a very flexible way of modelling the character's decision making process. By declaring the new abilities and/or items in the planning model, this will be automatically taken into account on the behaviour of all characters and situations. Furthermore, any advances on planning technology can be adopted without a large implementation effort. Planning systems have been used mainly for strategic decision making in videogames such as first person shooters. While the first approaches considered classical planning systems (also known as Goal-Oriented Action Planning, GOAP) such as the AI of the game F.E.A.R. (Orkin 2006), most approaches have focused on using Hierarchical Task Network (HTN) planning models (Nau et al. 2003; Menif, Jacopin, and Cazenave 2014).

In this paper, we apply domain-independent planning technology within a game engine to create believable NPCs by making them act in a goal-driven manner, where their behavior is influenced by a number of factors such as their personality and/or emotions. Thus, the storytelling narrative emerges out of the interaction of characters that plan individually according to their own goals, in contrast to other previous approaches using planning for interactive storytelling which follow a centralized approach (Riedl and Young 2010; Haslum 2012). Our focus is not only on strategical agents, but also on social and believable agents that act according to their emotions (Pérez-Pinillos, Fernández, and Borrajo 2011; Belle, Gittens, and Graham 2022). Contrary to those approaches, we focus on classical (GOAP) planning, as this is suitable for our game engine in which where agent's have goals established by the game events.

In this setting, NPCs should be able to interact with each other in order to achieve their goals. Furthermore, they should be believable, i.e., their actions should be consistent with the expectations of the player according to the established personality traits and/or emotions of the characters.

We design planning models that are able to (1) tailor the behaviour of the characters according to their personality traits and/or emotional state, and (2) achieve plans where characters collaborate and/or compete. We achieve (1) through changing the cost of the actions accordingly. For (2), we achieve implicit coordination through actions that allow the NPC to plan for other characters to help them.

A critical question is whether current state-of-the-art planners are capable to provide plans in real-time during the execution of the game. To focus on scalability, we keep our models simple using classical planning where the environment is deterministic, plans are sequential, and there is a single agent that has full control over the actions that will change the environment. Even though the models do not reflect the game dynamics in a precise manner, the plans provided can still result in meaningful plans that could arise from characters under limited forms of reasoning. While some of the actions in the plan might fail, this can be addressed through re-planning.

We conduct an empirical evaluation comparing the performance of several planners on different planning models of different degrees of complexity. Our results show that current planners (e.g., the Fast Downward planning system with

the LM-cut heuristic (Helmert 2006; Helmert and Domshlak 2009)) can provide optimal plans on scenarios with a realistic number of characters and that, by sacrificing optimality guarantees (e.g., using the FF heuristic (Hoffmann and Nebel 2001)), it is possible to obtain close to optimal solutions in less than a second.

2 Background

We consider STRIPS planning models (Fikes and Nilsson 1971), where a *planning task* is specified by a tuple $\langle \mathcal{F}, \mathcal{A}, s_I, s_G \rangle$. $\mathcal{F}$ is a set of facts, i.e., Boolean variables that describe the current state. A *state* $s \subseteq \mathcal{F}$ is a set of facts corresponding to those facts that are true in s. $s_I \subseteq \mathcal{F}$ is an *initial state* and $s_G \subseteq \mathcal{F}$ is a *goal* specification. $\mathcal{A}$ is the set of actions. An *action* $a \in \mathcal{A}$ is a tuple $a = \langle \mathrm{pre}(a), \mathrm{add}(a), \mathrm{del}(a), \mathrm{c}(a) \rangle$, where $\mathrm{pre}(a) \subseteq \mathcal{F}$ is a set of preconditions of the action o, and $\mathrm{add}(a) \subseteq \mathcal{F}$ and $\mathrm{del}(a) \subseteq \mathcal{F}$ are sets of add and delete effects, respectively, and $\mathrm{c}(a) \in \mathbb{R}_0^+$ is a cost of the action. All actions are well-formed, i.e., $\mathrm{add}(a) \cap \mathrm{del}(a) = \emptyset$ and $\mathrm{pre}(a) \cap \mathrm{add}(a) = \emptyset$. An action o is *applicable* in a state s if $\mathrm{pre}(a) \subseteq s$. The *resulting state* of applying an applicable action o in a state s is the state $a[\![s]\!] = (s \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$. A state s is a *goal state* iff $s_G \subseteq s$.

A plan is a sequence of actions $\pi = \langle a_1, \ldots, a_n \rangle$ that goes from the initial state to a state that satisfies the goal. That is, there are states $s_0, \ldots, s_n$ such that a_i is applicable in s_{i-1} and $s_i = a_i[\![s_{i-1}]\!]$ for $i \in \{1, \ldots, n\}$, $s_0 = s_I$, and $s_G \subseteq s_n$ The cost of the plan is the sum of the cost of its actions, $\mathrm{c}(\pi) = \sum_{i=1}^{n} \mathrm{c}(a_i)$. A plan is *optimal* if its cost is minimal among all plans for the task.

We specify planning tasks using the Planning Domain Definition Language (PDDL) (McDermott et al. 1998; McDermott 2000). In PDDL, STRIPS tasks are specified in terms of fact and action templates, with parameters that are to be replaced by a set of objects. For example, the action `Travel (?c - character ?l ?l' - location)` has three parameters, where the first parameter can be instantiated by objects of the type "character", whereas ?l and ?l' can be instantiated with objects of type location. In the rest of the paper we leave the object types implicit, as they are always clear from the context. Given a set of objects, one can instantiate these templates to obtain the STRIPS task. For example. given characters {Alice, Bob, Carol}, and locations $\{l_A, \ldots, l_E\}$, we would obtain actions `Travel(Alice,` l_A, l_B), `Travel(Bob,` l_A, l_B), and so on for every character and pair of connected locations. The process of enumerating all possible substitutions to translate a PDDL task into a STRIPS task is called *grounding* (Helmert 2009).

3 The Game Engine

We assume a game engine that focuses on dynamic and interactive storytelling, through the interaction with several goal-driven characters. The game state $\langle \mathcal{M}, \mathcal{C}, \mathcal{I} \rangle$ consists of an scenario $\mathcal{M}$, a set of characters $\mathcal{C}$, and a set of items $\mathcal{I}$.

The scenario represents the game map here as a directed graph where each node corresponds to a location, and edges represent the possible movements among locations. Furthermore, some locations may be locked. Each item in $\mathcal{I}$, may have diverse properties, such as power, or being the key to unlock certain locations. Each character in $\mathcal{C}$ has a set of associated resources, some of which are numeric (e.g., health, wealth), and some are binary (e.g., having certain items). Moreover, the character has a set of emotions (e.g., the level of happiness), which influence their behaviour. Social relations are modelled through a function $\mathrm{friend} : \mathcal{C} \times \mathcal{C} \mapsto [-100, 100]$, where $\mathrm{friend}(c, c') > 0$ if c considers c' to be their friend; and $\mathrm{friend}(c, c') < 0$ if c considers c' to be their enemy.

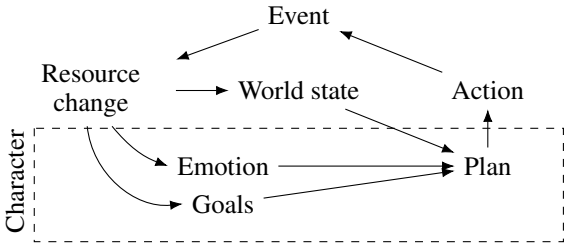

Figure 1: Game loop.

Characters are goal-driven, so that their actions can be explained in terms of what goals they intend to achieve. Figure 1 shows the game loop and how characters interact with the rest of the world. The game flow is guided by a series of events. Every time a character executes an action, this results in an event, which may change the resources of one or more characters. Changes in resources motivate a change in the emotional state of the characters (e.g., upon loss of an item, a character happiness will decrease), and this leads to the occurrence of new goals (e.g., recovering the lost item, or causing damage to the actor who caused the item loss).

We consider two types of goals that a character could have, social and quest goals. Social goals aim to affect the emotions/resources of other characters in a positive or negative way. This includes, for example, complimenting another character, giving them an item, or causing them damage. Quest goals correspond to obtaining specific items that are valuable for the character and often involve going into locked locations.

Furthermore, characters can interact with each other. For example, a character may ask another to perform an action in their behalf. If they agree, the event resolves by introducing a new goal for the second character, which now will plan independently how to best achieve such a goal.

After any event, re-planning may be triggered for a character if the character's plan is not executable anymore (e.g., an action has failed or another character does not agree to achieve the requested sub-goal) and/or their goals have changed. In that case, the character will make a new plan for the updated goal with the available resources.

Example 1. As an illustration, consider the following scenario, where there are 3 characters Alice, Bob, and Carol. Our main character, Alice, has a goal of getting an object, which requires to enter a blocked location. Alice does not

have the key, however, nor she has any lockpicking skills, so in order to achieve her goal, she will have to rely on the help of other characters. There are two main alternatives. On the one hand, Alice's friend, Bob, has some lockpicking skills. So plan π_B consists of asking Bob the favour of lockpicking the door. On the other hand, Carol has the door's key but she is not in very good relationship with Alice. So an alternative plan π_C could be to threaten Carol and demand her to give Alice the key. In order to be believable the plan of Alice should be consistent with her emotional state and/or personality. When in "good mood", π_B is preferable as it is less confrontational, even though Bob may fail at lockpicking the door. If Alice is angry, it may be more believable to choose π_C, as it is more confrontational and has higher changes of success.

After choosing a plan, Alice will start its execution (e.g. going to Bob/Carol and asking them the favour). However, as Bob and Carol are also following their own plans this may trigger a re-planning in a number of cases, such as if they become unavailable, or if they reject doing the favour. In that case, Alice will have to find an alternative course of action, provided that she cannot repeat the failed action anymore.

In this paper, we focus on the planning process of each character. We assume that the character's goals and emotions have already been decided by the game engine and focus on how to find a plan that is consistent with the character's goals and emotions.

4 Planning Models

In this section, we describe the planning models we use to represent the reasoning process of an NPC character to decide what action to perform depending on the current state of the world and its current goals. These planning models are only a small sample of what would be possible in a game engine featuring different sets of actions, emotions, and character resources.[1] The model is kept simple, but still representative of what could be the expected performance on other game engines with similar number of characters, items, and actions.

We emphasize that the purpose of these planning models is not to have an accurate description of the game mechanics, reflect the precise knowledge of the character, nor consider all the possible dynamics with other characters and/or the non-deterministic environment. Instead, it is just an approximation that will result in characters choosing actions that are deemed believable when considering the character's goals, personality, and emotions. We use classical planning models, which offer a good trade-off between modelling capabilities and performance, as reasoning under temporal, adversarial, and/or probabilistic planning models is often much harder (Littman, Goldsmith, and Mundhenk 1998; Rintanen 2007). This means that our planning models will ignore action non-determinism, and assume that the current character has full control about all actions that will affect the world. While this may seem utterly unrealistic, we will consider alternatives to approximate these aspects within classical planning models. This means that, even when selecting optimal plans according to our models, characters will not necessarily always choose the best course of action under more realistic models. This is acceptable, as realistic characters are not necessarily fully rational, so such a "perfect" model may not be the best to obtain believable character behaviour.

[1]While the game engine is not open source, our approximated planning models are publicly available (Johansen et al. 2022).

Basic planning model

Table 1 shows the basic actions that are available to each character. Characters navigate through the world via the Travel action. Our model abstracts away any low-level motion details, except for the fact that certain locations are locked and cannot be directly accessed. In order to unlock them, a character must be in an adjacent location and either have a key that opens it or use lockpicking skills.

All actions are parameterized by the character that performs them, ?c. In principle, ?c corresponds to the main character, i.e., the one who is planning its own actions. However, as we will see in the next section, in certain situations ?c can also be another character.

As explained in Section 3, goals are set to the character due to the occurrence of some event, and then the characters will plan in order to achieve them. Following that, we can specify different types of goals in our planning models. We consider quest goals, which consist of getting an item typically located in a locked position (e.g., Alice's goal in Example 1), as well as the following social goals:

- Giving a present to a friend and/or complimenting them,
- Insulting an enemy and/or damaging their health.

Reasoning about the actions of other characters

During the planning process of an NPC, there are two types of considerations regarding other characters: adversarial and collaborative. Here, we focus on collaborative reasoning, and do not consider adversarial reasoning explicitly. Collaborative reasoning is important, and even necessary, as certain goals may only be achievable with collaboration from other agents, as illustrated in Example 1. We consider three ways in which the character can influence other character's actions:

- RequestFavour (?c, ?c', ?l): Character ?c asks character ?c' to perform some actions as a favour.
- DemandFavour (?c, ?c', ?l): Character ?c threatens character ?c' making them to perform some actions.
- BargainFavour (?c, ?c', ?i, ?l): Character ?c gives item ?i to character ?c' in exchange for them to perform some actions.

We will refer to these actions as *control* actions, and they have the effect that the target character ?c' will be controllable (by adding a fact (Controllable ?c')), so that they can be the actors of subsequent actions in the plan (instantiating the parameter ?c). This allows Alice to come up with a plan such as: Travel (Alice, l_A, l_B), RequestFavour (Alice, Bob, l_B), Travel (Bob, l_B, l_D), Lockpick (Bob, l_D, l_E), Travel (Alice, l_B, l_D), Travel (Alice, l_D, l_E), ...

Action	Description
Travel (?c ?l$_{from}$?l$_{to}$)	character ?c moves between two connected locations ?from and ?to if ?to is not locked.
Take (?c ?i ?l)	character ?c takes item ?i at location ?l.
Unlock (?c ?l ?i ?l$_{from}$)	character ?c unlocks location ?l from an adjacent location ?l$_{from}$ using item (key) ?i.
Lockpick (?c ?l ?l$_{from}$)	character ?c unlocks location ?l from an adjacent location ?l$_{from}$ by lockpicking it.
Give (?c ?i ?c' ?l)	character ?c gives item ?i to ?c' at location ?l.
Strike (?c ?i ?c' ?l)	character ?c attacks character c' using item (e.g. sword) ?i at location ?l.
Compliment (?c ?c' ?l)	character ?c compliments character ?c' at location ?l.
Insult (?c ?c' ?l)	character ?c insults character ?c' at location ?l.

Table 1: Actions available to the characters.

Note that this is only Alice's planning process, so actions involving Bob represent that Alice is planning for Bob to perform those actions before there has been any communication between them. Furthermore, even though the control actions do not specify explicitly what actions will ?c' be asked to perform, they are implicit in the rest of the plan. During the plan execution, when Alice executes the action RequestFavour(Alice, Bob, l_B), she already knows what actions Bob is supposed to perform in the rest of the plan, so this can be used to request the desired actions. In fact, by a simple analysis of the plan, one can identify which facts are achieved by Bob's actions that are relevant for the rest of Alice's plan, e.g., extract that Unlocked (l_E) is what Alice needs Bob to accomplish. Therefore, in our example the RequestFavour action would be resolved by Alice asking Bob to unlock l_E. If Bob agrees, Unlocked (l_E) will be a goal of Bob in the next planning process, and he may decide to accomplish it in the same way as Alice anticipated or in a different way (e.g. depending on his mood Bob could decide to ask Carol the key).

The model described above is slightly too unrealistic in the fact that, after asking a favour we gain full control of the other character and can use it to perform all kinds of actions. More fine-grained models are possible. We experiment with a partial control model, where the main character can gain different levels of control for another character. When using multiple control levels, actions RequestFavour, DemandFavour, and BargainFavour are parameterized by the level of control to achieve. Actions demanding a higher level of control have a higher cost as discussed in the next section, but are a pre-requisite for the controlled character to perform certain actions. This represents the fact that asking to perform actions like Strike is harder than simple actions like Insult. We introduce 4 levels of control, allowing for the following actions:

1. Travel and Take.
2. Insult and Compliment.
3. Strike, Give, and Lockpick.
4. Other control actions (asking another character to ask another character).

That is, it will be easier to control a character at level 1, but then that character can only be asked to travel to another location and/or take some objects. Each level provides a higher degree of control, until level 4, which provides full control so so the character can be asked to perform any action.

Cost function

An important consideration is action cost. In principle, characters should prefer plans that have minimum effort, and maximum probability of success. This results in a multi-objective problem, where a trade-off must be chosen. To keep the model simple, we simply combine two factors into a single cost function, where the cost of each action correlates with both. This also allows to incorporate the personality of the characters' as well as their emotions into play, so that characters choose different plans depending on their current mood.

Table 2 shows the cost function used in our model. We assign simple actions such as Travel, Take, or Unlock have a minimum cost of 1 to reflect that they have full chances of success. Other actions have costs that depend on different factors. The formulas shown in Table 2 are arbitrarily chosen so that the range of action costs is reasonable while making plans that are consistent with the character's emotions and personality. While the formulas themselves are not too important, they illustrate how one can make believable NPCs by adjusting the cost function.

We model the emotions of the planning character as a single variable, $\mathcal{E}$, representing whether the character is in good mood or not. Admittedly, this is a simplified model of the multi-dimensional characterization of emotions that could be possible in the game engine. However, this simplifies our evaluation on how a change in emotions can affect the character's plan. Note that extending the cost functions to incorporate more nuanced emotions and or personality traits (e.g. how aggressive or risk-averse each character is) can be easily done by adjusting the cost function.

Going back to Example 1, we see that changing the value of $\mathcal{E}$ will immediately change the cost of multiple actions, possibly affecting the choice of plan. If $\mathcal{E} \approx 1$, then the cost of DemandFavour is very high, so Alice will prefer to use RequestFavour instead. As the cost of RequestFavour depends on whether the actor is friend of the actee, Alice will prefer to ask Bob, as expected. On the other hand, if $\mathcal{E} \approx 0$ is preferred, using DemandFavour with Carol will have lower cost.

While the functions may look complex, they are not state-dependent (Geißer, Keller, and Mattmüller 2015; Speck et al. 2021). All numerical variables, and in particular those de-

Travel	1
Take	1
Unlock	1
Lockpick	$10(1 - \text{lockpick-skill}(?c))$
Strike	$(1 + 10 \cdot \mathcal{E} \cdot \text{friend}^+(?c, ?c')) \cdot \text{str-diff}(?c, ?c', ?i)$
Compliment	$1 + 2 \cdot (1 - \mathcal{E}) \cdot \text{friend}^-(?c, ?c')$
Insult	$1 + 2 \cdot \mathcal{E} \cdot \text{friend}^+(?c, ?c')$
Give	$3\,\text{item-val}(?i, ?c)$
RequestFavour	$1 + 5 \cdot 2^{?l} \cdot (1 - \mathcal{E}) \cdot \text{friend}^-(?c, ?c')$
DemandFavour	$(1 + 5 \cdot 2^{?l} \cdot \mathcal{E} \cdot \text{friend}^+(?c, ?c')) \cdot \text{str-diff}(?c, ?c')$
BargainFavour	$4 \cdot 2^{?l} \cdot \frac{\text{item-val}(?i, ?c)}{1+\text{item-val}(i?, ?c)} + 2\,\text{friend}^+(?c, ?c')$

where ?c is the character performing the action, ?c' is the character receiving the action, ?i is an item, $?l$ is the level of control acquired ($?l = 1$ if the full control model is used), and the following expressions are used:

$$\text{friend}^+(?c, ?c') := (2 + \text{friend}(?c, ?c'))^2$$
$$\text{friend}^-(?c, ?c') := (2 - \text{friend}(?c, ?c'))^2$$
$$\text{str-diff}(?c, ?c', ?i) := 1 + \frac{\text{strength}(?c')}{1+\text{strength}(?c)+\text{power}(?i)}$$
$$\text{str-diff}(?c, ?c') := 1 + \frac{\text{strength}(?c')}{1+\text{strength}(?c)}$$

Table 2: Cost function. All numeric variables are considered here to be normalized in the $[0, 1]$ interval ($[-1, 1]$ for $\text{friend}(?c, ?c')$).

scribing emotions ($\mathcal{E}$), do not change their value within the planning process. At first glance, that may seem unrealistic, as certain actions will definitively influence the character's emotions. Our models have the assumption, however, that characters are not capable of reasoning in advance about this (e.g., a scared character will not specifically plan to do actions that calm him down). Within the game engine, if a change in emotions occurs during the execution of the plan, this event will trigger a re-planning process so that the character can find a plan according to the new emotion.

As costs are not state-dependent, each grounded action has a single constant cost. This allows us to use standard planning systems to solve our planning tasks. In order to do that, we perform a minor modification to the PDDL parser of Fast Downward (Helmert 2006), so that the costs are correctly computed for each grounded action.

Filtering irrelevant actions

Most classical planners are based on grounding, the process of enumerating all combinations of objects that could instantiate the parameters of the actions. An issue of these planning models is that the amount of ground actions may be huge, especially for the actions Give, RequestFavour, DemandFavour, and BargainFavour. For example, the amount of Give actions is $O(|\mathcal{C}|^2||\mathcal{I}||\mathcal{M}|)$ as any character can give any item to any other character at any location. Even though most of these actions will never be used, they can greatly diminish the efficiency of planners based on grounding, as with only 10 characters, 20 items and a map with 50 locations, we have 100000 combinations.

A possible solution is to filter some of the actions, if they can be deemed unnecessary for achieving the goal (Gnad et al. 2019). We do this by introducing additional preconditions that can be used to restrict some of the arguments. These preconditions are based on some newly added predicates that are static, i.e., their truth value is specified in the initial state and they are not in the effect of any action. For the give action, for example we introduce a precondition `(may_give_item ?actor ?actee)`, which is only set to true in the initial state if either `?actor` or `?actee` are the main character. While excluding these actions may forbid the actual optimal plan, or even make the main character's goals unreachable, this is unlikely to happen in practice as most plans are centered around actions that involve the main character.

Similarly for the "Favour" actions, we filter them based on the relationship between the actor and actee characters (for RequestFavour, and BargainFavour) or their difference in strength (for RequestFavour). The rationale in this case is that all excluded actions have very high action costs and are therefore very unlikely to occur in any optimal plan. Admittedly, this is a heuristic criteria and more effective filters could be achieved by using machine learning techniques as done by Gnad et al. (2019).

Introducing these filters as preconditions over static predicates completely gets rid of the grounding issue, as grounding algorithms are able to skip the filtered actions without enumerating them in the first place (Helmert 2009).

5 Experiments

We evaluate the performance of several planners under the proposed planning models. In our evaluation, we consider four variants of the domain, depending on whether we apply a filter of feasible actions or not and on how we model the control of other characters. In the *full* version, control actions always allow to obtain full control of other characters, whereas the *partial* model uses different levels of control for the different actions.

For each of the versions, we generate instances randomly in a way that approximates the scenarios that could occur typically within the game engine. To analyze the scalability of planners, we generate instances of diverse sizes by varying the number of goals (1, 2, 3, 4, or 5), locations (10, 20, 50, 70), and characters (3, 5, 10, 15, 20). The number of items is always set to be the same as the number of characters, which is to be expected as often characters may carry one item. We consider all 100 combinations. For each combination, we generate two instances (with $\mathcal{E} = 0.25$ and 0.75, respectively), resulting in 200 instances in total for each variant of the domain. For all variants of the domain we always use exactly the same seed so that different in performances across versions are not affected by the random generation process.

The chosen ranges for our scalability analysis are representative, and go slightly beyond, of what is needed for a realistic use case. Even though in some cases there may be more objects (i.e., locations/characters/items) in the game, the planning process is only concerning the plan of a single agent, so one can typically include only those objects that are relevant.

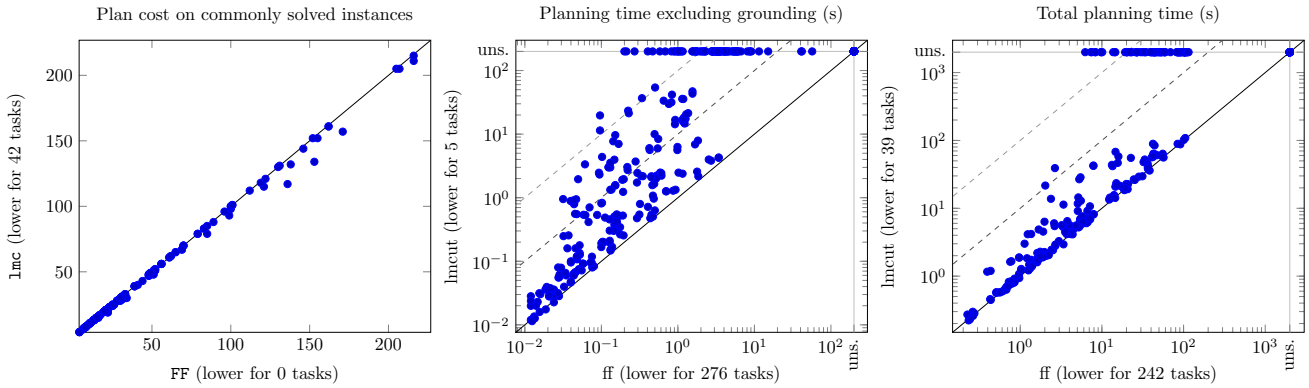

Figure 2: Results of optimal versus satisficing configuration in terms of cost, and planning time without/with grounding.

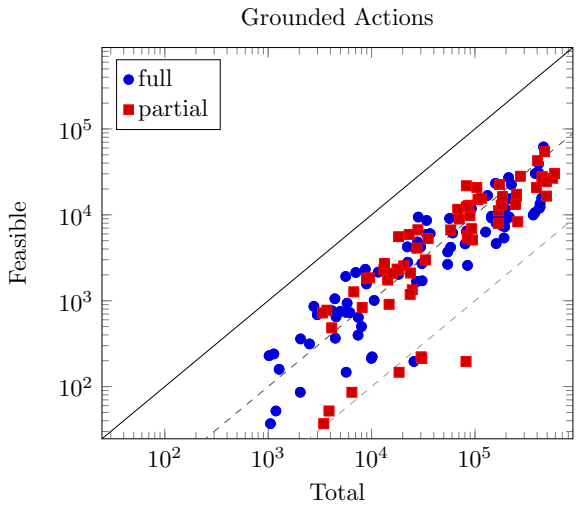

Figure 3: Number of grounded actions with and without the action filter.

We use a memory limit of 4GB, and a time limit of 2 minutes for the overall planning process (including the overhead of grounding), and 1 minute for the search. These time limits are very generous for the performance required in practice, where plans should be provided under a second. However, this allows us to obtain a more comprehensive view on how the difficulty of solving the problem scales with the size of the instance. All our experiments are run on a cluster using Downward Lab (Seipp et al. 2017). All benchmarks, scripts and experiment data are publicly available (Johansen et al. 2022).

We considered the best standalone planners from the sequential optimal track of IPC2018: Complementary2 (Franco et al. 2017; Franco, Lelis, and Barley 2018), Scorpion (Seipp 2018; Seipp, Keller, and Helmert 2020), and Symbolic Bidirectional Blind Search (Torralba et al. 2017). Complementary-2 and Scorpion, however, rely on a preprocessing phase that was optimized for maximizing the number of tasks solved within 30 minutes. The exact same

	Optimal						
	bl	sfw	sbd	sPDB	scrp	lmc	FF
full	76	55	52	73	**122**	109	163
full-filter	110	107	102	135	166	**171**	195
partial	66	50	47	65	87	**91**	118
partial-filter	97	96	86	117	137	**150**	176

Table 3: Coverage of blind explicit search (bl), symbolic forward (sfw) and bidirectional (sbd) search, and A* with symbolic PDBs (sPDB), Scorpion (scrp), LM-cut (lmc), and the FF heuristic. Best optimal planner in each kind of planning model highlighted in bold.

configurations perform very poorly under our time limit. For Scorpion, we adapt the configuration to consider only patterns up to size 3, with a shorter time limit. We replace complementary with a planner that interleaves search and symbolic PDBs (Kissmann and Edelkamp 2011; Franco and Torralba 2019), which also uses symbolic PDBs and is more optimized towards solving problems quickly. As baselines, we included blind search and A* search with the LM-cut heuristic (Helmert and Domshlak 2009), as implemented in version 21.12 of the Fast Downward Planning System (Helmert 2006). All those configurations are optimal, i.e., they are guaranteed to always provide a minimum-cost plan.

Finally, to evaluate the gap with respect to satisficing planners, we also run A* search with FF. This is a non-typical choice, as satisficing planners typically use more greedy algorithms such as greedy best-first search and/or WA* (oftentimes in iterative anytime configurations that refine the quality of the plan found over time (Richter and Westphal 2010)). Here, we chose A* with FF due to its simplicity as well as its ability to find close-to-optimal plans in one go.

Coverage Analysis

Table 3 shows the coverage on the different types of models, that is, the number of instances solved within the time and memory limits by each planner. Even though the time limit is higher than what it would be required for it to be usable

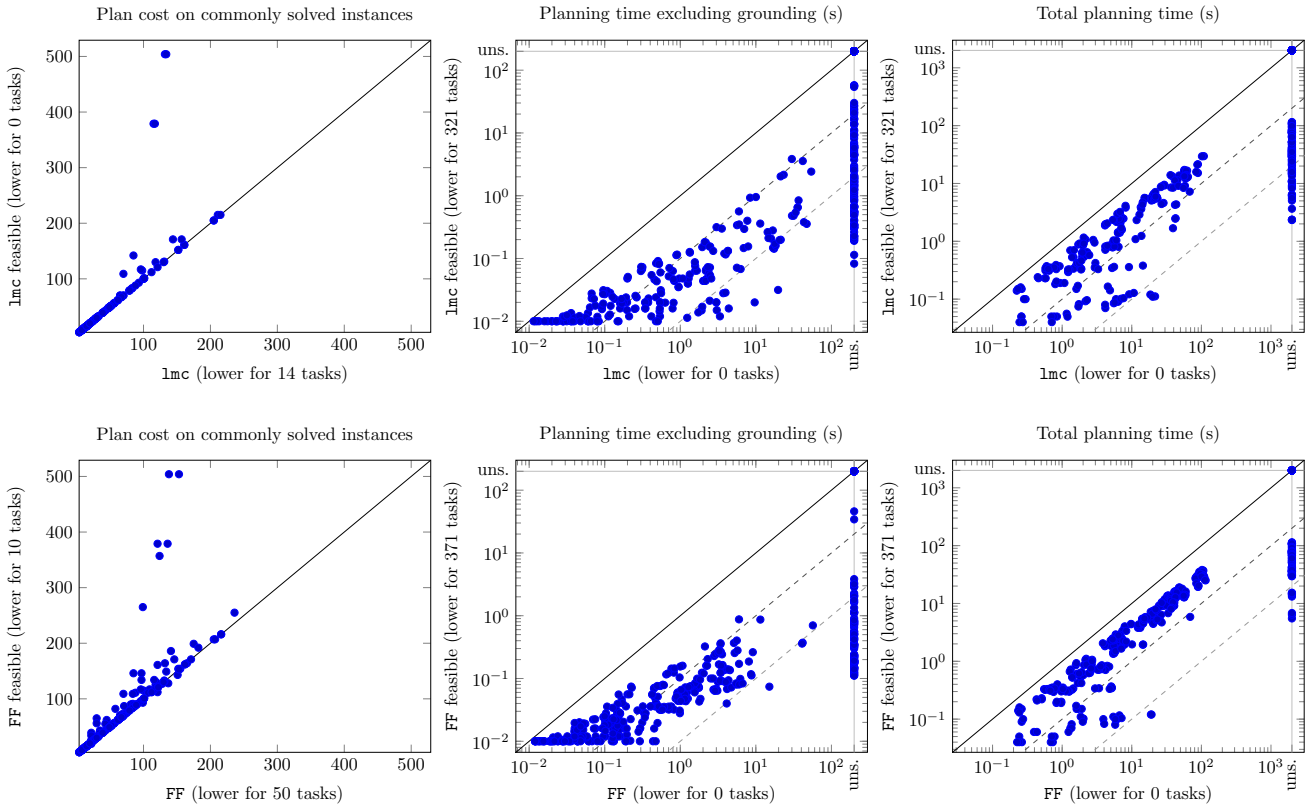

Figure 4: Impact of filter of feasible actions in cost and planning time without/with grounding on optimal (lmc, above) and satisficing (FF, below) configurations.

in the game engine, the results are representative of the general trends regarding which versions of the domain and/or planners lead to better results.

The results show a number of interesting observations. First of all, comparing the solved tasks within each of the four different planning models, it is always the case that full is easier than partial. This is to be expected, as the partial model is more fine-grained and has a larger state space due to the character having the choice of how much of a favour ask to other characters. Filtering out actions always improves performance regardless of how the interaction with other characters is modelled or the planner being used. This again is not surprising as the number of grounded actions without any filter can grow very large.

Comparing the performance of optimal planners, however, the trend is not exactly as expected. In particular, the baseline, lmc, is clearly superior to the best performing planners in the last IPC. Part of it could be due to the different time limits, specially given that these planners have not been fully optimized for short time limits. However, the domain seems to suit particularly well to heuristics based on the delete-relaxation such as lmc and FF, which compute accurate estimates. This could be related to the fact that the more costly actions, control actions, related to the control of other characters are "delete-relaxed" in the sense that they are about gaining resources that are never lost once acquired.

On the other hand, planners based on symbolic search tend to perform poorly, even under blind explicit search. This is likely related to the large number of grounded facts and actions, but also to the fact that reasoning backwards from the goal seems challenging, as suggested by the fact that bidirectional search (sbd) performs worse than only forward search (sfw) in several cases.

Optimal versus Satisficing Configurations

Figure 2 shows a comparison of the best configuration that guarantees always finding the optimal solution (lmc) and our satisficing configuration (FF). Overall, A*with the FF heuristic (FF) offers a very good trade off between planning time and plan quality. The comparison in terms of plan cost, shows that the plans found by FF are quite often very close to the optimal one, whereas the advantage in search time (i.e., excluding grounding) is huge, of more than one order of magnitude. FF is able to solve the problem below a second in many cases, once that the problem has been grounded. However, the grounding process is still a bottleneck, so the difference in terms of planner time, even though still favorable to FF, is not that huge.

When considering which instances can be solved under a time limit of 1 second, we observe that search time is not a problem. For example, under full filtering actions, and fixing the number of goals to 1, lmc can easily address tasks

with 70 locations, and 10 characters in 0.11 seconds and FF takes only 0.01 seconds. However, grounding is an issue on problems with larger number of objects. This is specially sensitive to the number of characters so to keep time under a second it is desirable that the number of characters is 5 or less. In future work, we will analyze how to filter which characters are relevant for the planning process in automatic ways, e.g., by extending our action filters or using additional object filters (Silver et al. 2021)

Filtering Actions

As shown in Table 3, filtering some actions can improve the performance across all planners. The main reason is the speed-up of the grounding process, thanks to the reduction in the number of grounded actions. This is clearly a bottleneck of the suggested models. As illustrated by Figure 3, without any filter, some problems cannot be even grounded within the time/memory limits, which only allow for up to half a million actions. The simple filters described in Section 4 are sufficient to reduce the number of actions by an order of magnitude, which explains the positive results in Coverage.

However, as these filters are based on manual heuristic rules, it is possible that they eliminate important actions that are required in the optimal plan. If that happens, plans will have larger cost. As the cost function is tailored to obtain believable characters, this could result on characters choosing poorly believable plans.

Figure 4 analyzes in detail the impact of the action filter on both the optimal and the satisficing configurations. The cost increases significantly in some cases, suggesting that there is room of improvement for our filter criteria. However, in most cases the impact on the cost of the plans obtained is negligible, where as search and total planning time are both heavily reduced thanks to the reduction in the number of actions.

6 Related Work

There are different approaches that aim to obtain social and emotional NPCs. A popular approach is to use behaviour trees (Johansson and Dell'Acqua 2012; Agis, Gottifredi, and García 2020; Belle, Gittens, and Graham 2022), or focus on pathfinding (Aversa and Vassos 2014). In contrast, we focus on goal-oriented planning, where the NPC's actions are determined by finding a (near-)optimal plan for achieving a pre-specified goal.

Among approaches applying domain-independent planning to control NPCs, most of them are focused only on strategic considerations and do not take into consideration the character's emotional state (Orkin 2006; Menif, Jacopin, and Cazenave 2014; Vlachopoulos, Vassos, and Koubarakis 2014). A notable exception is the work by Pérez-Pinillos, Fernández, and Borrajo (2011), whose planning models take into account the emotions and personality traits of the characters. They introduce a numeric planning approach where the goal is to satisfy basic needs, described in terms of numeric variables such as "hunger", "neuroticism", etc. In contrast, our models are focused on achieving specific goals. Our model of the character's state is simpler, aiming for a scalable solution that can provide believable plans in real time. Furthermore, our NPCs take into consideration other agents in order to collaborate with them for achieving their goals.

Finally, there has been a lot of research on multi-agent coordination and/or negotiation, e.g. (Kraus 1997). Again, our focus is not on obtaining optimal co-ordination for solving problems (Davis and Smith 1983), or on how agents negotiate. Instead, we aim for a simpler and scalable approach that is used during planning to achieve a realistic NPC behaviour. The actual negotiation between the agents that arises after an agent executes the RequestFavour could be based on any of those negotiation frameworks.

7 Conclusions

In this paper, we have shown how domain-independent classical planners can be used to obtain believable characters in a social storytelling environment, by taking into consideration collaboration with other agents. By designing a cost function that depends on the emotional state of the characters, their behaviour can be controlled so that the character actions are consistent with the player's expectations. Furthermore, we showed that current planners can find optimal or close-to-optimal solutions in the order of milliseconds on instances with a realistic number of characters.

We highlight that this work also opens several promising avenues for future research. First, we have manually designed a cost function that results in characters choosing believable plans. An interesting direction, however, is how to more systematically evaluate whether plans are believable, and whether the NPC's actions are consistent with players' expectations. This can lead to an automate process where cost functions are personalized for the player, i.e., so that the game can adapt if the player does not consider certain plans believable.

Secondly, in this paper we have focused only on hard goals, as currently we assume the game engine is in charge of defining goals based on previous events with a rule-based system. However, our set-up is very amenable for considering goal selection as part of the planning process, using soft goals whose utility also depends on the character's emotions.

Finally, as the main bottleneck is currently the grounding process, this is an interesting application for lifted planners which avoid it (e.g. (Corrêa et al. 2020, 2021; Lauer et al. 2021)). Future work could analyze how to handle complex cost functions such as the ones in our model in near-optimal lifted planning.

Acknowledgements

This paper has been supported by Vision Denmark through the project "Automated Planning for Emotion-Aware Decision Making in Videogames".

References

Agis, R. A.; Gottifredi, S.; and García, A. J. 2020. An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, 155.

Aversa, D.; and Vassos, S. 2014. Belief-Driven Pathfinding through Personalized Map Abstraction. In *Proc. AIIDE 2014*.

Belle, S.; Gittens, C.; and Graham, T. C. N. 2022. A Framework for Creating Non-Player Characters That Make Psychologically-Driven Decisions. In *IEEE International Conference on Consumer Electronics, ICCE 2022*, 1–7.

Corrêa, A. B.; Francès, G.; Pommerening, F.; and Helmert, M. 2021. Delete-Relaxation Heuristics for Lifted Classical Planning. In *Proc. ICAPS 2021*, 94–102.

Corrêa, A. B.; Pommerening, F.; Helmert, M.; and Francès, G. 2020. Lifted Successor Generation using Query Optimization Techniques. In *Proc. ICAPS 2020*, 80–89.

Davis, R.; and Smith, R. G. 1983. Negotiation as a Metaphor for Distributed Problem Solving. *Artificial Intelligence*, 20(1): 63–109.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2: 189–208.

Franco, S.; Lelis, L. H. S.; and Barley, M. 2018. The Complementary2 Planner in the IPC 2018. In *IPC-9 Planner Abstracts*, 32–36.

Franco, S.; and Torralba, Á. 2019. Interleaving Search and Heuristic Improvement. In *Proc. SoCS 2019*, 130–134.

Franco, S.; Torralba, Á.; Lelis, L. H. S.; and Barley, M. 2017. On Creating Complementary Pattern Databases. In *Proc. IJCAI 2017*, 4302–4309.

Geißer, F.; Keller, T.; and Mattmüller, R. 2015. Delete Relaxations for Planning with State-Dependent Action Costs. In *Proc. IJCAI 2015*, 1573–1579.

Gnad, D.; Torralba, Á.; Domínguez, M. A.; Areces, C.; and Bustos, F. 2019. Learning How to Ground a Plan – Partial Grounding in Classical Planning. In *Proc. AAAI 2019*, 7602–7609.

Haslum, P. 2012. Narrative Planning: Compilations to Classical Planning. *Journal of Artificial Intelligence Research*, 44: 383–395.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M. 2009. Concise Finite-Domain Representations for PDDL Planning Tasks. *Artificial Intelligence*, 173: 503–535.

Helmert, M.; and Domshlak, C. 2009. Landmarks, Critical Paths and Abstractions: What's the Difference Anyway? In *Proc. ICAPS 2009*, 162–169.

Hoffmann, J.; and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Johansen, N. S.; Kær, L. B.; Stolberg, J. A. B.; Tollund, R. G.; Hyldig, N.; Oktober, P.; and Torralba, Á. 2022. Code, Benchmarks, and Data of "Towards Believable Non-Player Characters with Domain-Independent Planning". https://doi.org/10.5281/zenodo.6594835.

Johansson, A.; and Dell'Acqua, P. 2012. Emotional behavior trees. In *2012 IEEE Conference on Computational Intelligence and Games, CIG*, 355–362.

Kissmann, P.; and Edelkamp, S. 2011. Improving Cost-Optimal Domain-Independent Symbolic Planning. In *Proc. AAAI 2011*, 992–997.

Kraus, S. 1997. Negotiation and Cooperation in Multi-Agent Environments. *Artificial Intelligence*, 94(1-2): 79–97.

Lauer, P.; Torralba, Á.; Fišer, D.; Höller, D.; Wichlacz, J.; and Hoffmann, J. 2021. Polynomial-Time in PDDL Input Size: Making the Delete Relaxation Feasible for Lifted Planning. In *Proc. IJCAI 2021*.

Lim, C.-U.; Baumgarten, R.; and Colton, S. 2010. Evolving behaviour trees for the commercial game DEFCON. In *European conference on the applications of evolutionary computation*, 100–110. Springer.

Littman, M. L.; Goldsmith, J.; and Mundhenk, M. 1998. The Computational Complexity of Probabilistic Planning. *Journal of Artificial Intelligence Research*, 9: 1–36.

McDermott, D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine*, 21(2): 35–55.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL – The Planning Domain Definition Language – Version 1.2. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Yale University.

Menif, A.; Jacopin, E.; and Cazenave, T. 2014. SHPE: HTN Planning for Video Games. In *Computer Games - Third Workshop on Computer Games, CGW 2014, Held in Conjunction with the 21st European Conference on Artificial Intelligence, ECAI 2014, Prague, Czech Republic, August 18, 2014, Revised Selected Papers*, 119–132.

Nau, D. S.; Au, T.-C.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20: 379–404.

Orkin, J. 2006. Three states and a plan: the AI of FEAR. In *Game developers conference*, volume 2006, 4. CMP Game Group San-Jose, California.

Pérez-Pinillos, D.; Fernández, S.; and Borrajo, D. 2011. Modeling Motivations, Personality Traits and Emotional States in Deliberative Agents Based on Automated Planning. In *Proc. ICAART 2011*, volume 271 of *Communications in Computer and Information Science*, 146–160. Springer. ISBN 978-3-642-29965-0.

Richter, S.; and Westphal, M. 2010. The LAMA Planner: Guiding Cost-Based Anytime Planning with Landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Riedl, M. O.; and Young, R. M. 2010. Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research*, 39: 217–268.

Rintanen, J. 2007. Complexity of Concurrent Temporal Planning. In *Proc. ICAPS 2007*, 280–287.

Seipp, J. 2018. Fast Downward Scorpion. In *IPC-9 Planner Abstracts*, 77–79.

Seipp, J.; Keller, T.; and Helmert, M. 2020. Saturated Cost Partitioning for Optimal Classical Planning. *Journal of Artificial Intelligence Research*, 67: 129–167.

Seipp, J.; Pommerening, F.; Sievers, S.; and Helmert, M. 2017. Downward Lab. https://doi.org/10.5281/zenodo.790461.

Silver, T.; Chitnis, R.; Curtis, A.; Tenenbaum, J. B.; Lozano-Pérez, T.; and Kaelbling, L. P. 2021. Planning with Learned Object Importance in Large Problem Instances using Graph Neural Networks. In *Proc. AAAI 2021*, 11962–11971.

Speck, D.; Borukhson, D.; Mattmüller, R.; and Nebel, B. 2021. On the Compilability and Expressive Power of State-Dependent Action Costs. In *Proc. ICAPS 2021*, 358–366.

Torralba, Á.; Alcázar, V.; Kissmann, P.; and Edelkamp, S. 2017. Efficient Symbolic Search for Cost-optimal Planning. *Artificial Intelligence*, 242: 52–79.

Vlachopoulos, I.; Vassos, S.; and Koubarakis, M. 2014. Flexible Behavior for Worker Units in Real-Time Strategy Games Using STRIPS Planning. In *Proc. SETN 2014*, 555–568.