On the Robustness of Domain-Independent Planning Engines: The Impact of Poorly-Engineered Knowledge[*]

Mauro Vallati
University of Huddersfield
m.vallati@hud.ac.uk

Lukáš Chrpa
Czech Technical University in Prague, and
Charles University in Prague
chrpaluk@fel.cvut.cz

Abstract

Recent advances in automated planning are leading towards the use of planning engines in a wide range of real-world applications. As the exploitation of planning techniques in applications increases, it becomes imperative to assess the robustness of planning engines with regards to poorly-engineered (or maliciously modified) knowledge models provided as input for the reasoning process.

In this work, to understand the impact of poorly-engineered knowledge on planning engines, we consider the perspective of a hypothetical attacker that is interested in subtly manipulating such knowledge to introduce unnecessary overheads that consequently slow down the planning process. This narrative ploy allows us to describe different types of knowledge engineering issues that cannot be detected via validation of the models, and to measure their impact on the performance of a range of planning engines exploiting very different approaches for steps like pre-processing and search.

Introduction

Automated planning is one of the most prominent AI challenges; it has been studied extensively for several decades, and it is now exploited in a wide range of applications. Examples include network security penetration testing (Hoffmann 2015), urban traffic management (McCluskey and Vallati 2017), battery load management (Fox, Long, and Magazzeni 2012), and control of robots (Kvarnström and Doherty 2010; Capitanelli et al. 2018). The availability of tools such as planning.domains,[1] or itSIMPLE (Vaquero et al. 2007), is also fostering the independent testing and evaluation of automated planning techniques by domain experts, that may have limited expertise in AI planning and knowledge engineering.

The vast majority of planning engines have been traditionally designed and developed for working on "good quality" and rather simple domain models. These models are developed by planning experts who leverage their experience to encode the available domain knowledge into suitable and efficient planning models. Despite the fact that there is no unified view on what good quality model means in the automated planning context (McCluskey, Vaquero, and Vallati 2017), existing pre-processing and pruning techniques address a limited number of possible issues of planning models (e.g., symmetries that may occur in the search space), rather than addressing potential engineering issues of the provided models. However, the more automated planning is exploited in real-world applications, the higher is the probability that planning engines are provided with low-quality (or poorly-engineered) models, due to models being encoded by practitioners or non-planning experts, or even by planning experts with limited understanding of the application domain and its dynamics. Furthermore, it is well-known that any computer system can be the target of an attack, and attacks can be crafted to target knowledge models. As we are more and more using planning techniques in real-world applications, it is incumbent on us to investigate how well planning techniques can cope with malicious attacks targeting the input provided to engines. This need is particularly pressing in cases where planning is used for security or safety-related purposes. In those cases, planning must become secure itself.

In order to investigate the robustness of domain-independent planning engines to issues in the input knowledge models, and to raise awareness of the impact of maliciously modified knowledge, in this work we take an attackers' perspective. Our hypothetical attacker can manipulate the input knowledge of planning engines, with the aim of reducing planning capabilities. The attacker carefully crafts her attacks so that they cannot be detected by a post-hoc analysis of the generated solution plans, and can only be spotted by an expensive and time-consuming inspection of the input models. In fact, this is exactly what happens in cases that are usually classified as accidental complexity issues (Brooks 1987), that can typically arise when encoding a planning model. It should be noted that the considered attacker's perspective is a narrative ploy used to highlight how engineering issues –hard to identify by human experts and automated validation– affect the performance of planning engines. It is therefore different from traditional adversarial planning (e.g., (Brafman et al. 2009; Speicher et al. 2018)). For the sake of our analysis, here we

[*]This paper has been published in the proceedings of the ACM Conference on Knowledge Capture (K-CAP) 2019.

[1]http://planning.domains

assume that the defender is *unaware* of the attacker and of her goals. In that, if we have to draw a parallel with other AI areas, our narrative settings can be seen as more similar to adversarial machine learning (Laskov and Lippmann 2010): the only way to address (potential) attacks is by improving the robustness of the exploited planning engines.

As additional contributions, we describe and empirically evaluate two techniques that –by manipulating either the problem or the domain model files by adding dummy objects or operators– can be used to degrade the performance of state-of-the-art domain-independent planning engines. Whereas the introduced attack techniques are easy to perform, and can be easily happen as the result of a poor engineering process, their impact on performance of planning engines can be considerable.

While taking an attacker's perspective and provide suggestions to reduce performance may seem counterproductive, we believe that learning the weaknesses of current planning engines is the only way to fix them in the future. In fact, the main aim of this work is to raise awareness of the sensibility of existing planning engines to seemingly unimportant issues of knowledge models, and advocate for the development of more robust techniques for performing planning and for better tools for supporting the knowledge engineering of future models.

Automated Planning

Automated planning deals with finding a (partially or totally ordered) sequence of actions transforming the environment from some initial state to a desired goal state (Ghallab, Nau, and Traverso 2004). In the classical representation, the environment is represented by first-order logic *predicates*. *States* are defined as sets of grounded predicates (atoms). A *planning operator* $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{eff}^-(o), \mathit{eff}^+(o))$ is specified such that $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ (*op_name* is a unique operator name and $x_1, \ldots x_k$ are variable symbols (parameters) appearing in the operator), $\mathit{pre}(o)$ is a set of predicates representing the operator's preconditions, $\mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are sets of predicates representing the operator's negative and positive effects. *Actions* are grounded instances of planning operators. An action $a = (\mathit{pre}(a), \mathit{eff}^-(a), \mathit{eff}^+(a))$ is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

In classical planning, a *domain model* is specified via sets of predicates and planning operators. A *problem model* (or problem instance) is specified via objects (that are substituted for free variables in predicates and operators), an initial state and a set of goal atoms. A *planning task* consists of a domain model and a problem instance. A *solution plan* for a planning task is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that in which all the goal atoms are true.

When using the dominant family of planning knowledge representation languages – PDDL (Mcdermott et al. 1998), the planning domain model and problem instance are provided to planning engines as two different files, and the same domain model is used for a class of problem instances.

The classical planning model can be extended, in order to handle a wider range of constraints and increase expressiveness. For instance, this is the case in Temporal Planning, where actions have a duration (in this case PDDL 2.1 can be used for encoding). Uncertainty Planning studies cases in which the environment is not fully observable and effects are non-deterministic. A further extension is named PDDL+ (Fox and Long 2006), which contains constructs to define hybrid domains, including processes and events. On this matter, the interested reader is referred to Ghallab, Nau, and Traverso (2004) and Geffner and Bonet (2013).

Formalisation

Given our attacker's perspective, the idea lies in modifying domain models and/or problem instances in such a way that they still allow to produce correct solution plans, with regards to the original models, while reducing the performance of planning engines by introducing additional (and unnecessary) overheads. In other words, the aim of the attacker is to make the model harder for planning engines rather than damaging the plan, making tasks unsolvable, or terminating the engines. As it is apparent, the depicted attacker's behaviour can in fact be the result of a poor knowledge engineering process. Instead of attacks, they can be accidental complexity issues. Particularly, those that are the hardest to be spotted, due to the fact that generated plans are valid and do not present easily recognisable oddities.

Technically speaking, the modified planning task must not generate different solution plans than the original task. We define two relations between planning tasks, *plan equivalence* and *plan dominance*, that refer to whether two tasks generate the same set of solution plans or one task (the original, in this case) generates a superset of solution plans of the other task respectively.

Definition 1. *Let P and P' be planning tasks. We say that P and P' are* ***plan equivalent*** *if for each sequence of actions π it is the case that π is a solution plan of P if and only if π is a solution plan of P'. We also say that P is* ***plan dominant*** *over P' if for each sequence of actions π it is the case that π is a solution plan of P' if π is a solution plan of P.*

If the attacker provides a modified planning task such that the original task is plan dominant over it, then the solution plans generated from the modified tasks are correct with respect to the original task. On the other hand, such plans might have an odd structure, that could attract attention and thus reveal attacker's interference (or to notice knowledge engineering issues in the models). For example, in a simple logistic domain where trucks are delivering packages between different locations, a planning task can be encoded in such a way that only one truck can be used to deliver packages (leaving the other trucks unused). Assuming there are no restrictions for where a specific truck can go or what packages they can load, the original task (using all the trucks) is plan dominant over the modified task (using only one truck). However, "one-truck" plans because of their unusual structure could, very likely, attract attention and thus being quickly fixed by an expert of the domain. Hence, the modified task should be plan equivalent or "close" to it. Since the notion of being "close" to plan equivalence or, in other words, a "small" plan dominance cannot be reasonably quantified (as the set of solution plans is typically infinite), we will hereinafter stick to the notion of plan equivalence.

Given a planning task P, the attacker provides a planning task P' such that P and P' are plan equivalent while maximising the chance that a planning engine, or a class of engines: (i) needs more CPU time for solving P' than for solving P; (ii) by solving P' instead of P a lower quality solution is returned. On top of that, the modification should be rather small as larger modifications could reveal attacker's interference.

In the remainder of this work, in order to analyse and formalise two classes of knowledge engineering issues that can typically arise during the encoding of knowledge under the form of planning tasks, we focus on: changes in the objects that are named in the problem knowledge model, and the encoding of operators that are not useful for the considered application domain.

Problem Models: Increasing the Number of Objects

One typical example of hard-to-spot issues of a planning task that can make it less amenable for existing planning engines, is the presence of useless objects listed in the problem instance specification. The following condition ensures that the modified task (by adding objects, without modifying the initial and goal states) is plan equivalent with the original one.

Condition 1. *For each operator o defined in the domain model, it is the case that for each parameter x of o there exists a predicate $p \in pre(o)$ such that p contains x.*

The useless objects are not involved in the initial and in the goal states of the modified task, as we are considering objects that are not actually part of the "original" problem model. Condition 1 ensures that an atom (grounded predicate) involving an object can only be added (by some operator) if some other atom involving the object is true (otherwise the operator is not applicable).

When considering object types (i.e., the "typed STRIPS" representation) it is possible to add objects of a given type if Condition 1 is met for all operators' parameters involving that type.

In large planning instances, which are common in real-world planning applications, useless objects can be easily confused with actual useful objects if they follow the naming convention used in the problem model.

As it should be apparent, most of the state-of-the-art pruning techniques exploited in planning engines are able to identify that the described added objects are not relevant for solving the considered planning task. In that, the impact of such object on the engines' performance is nullified during the search step. However, such analysis is usually done after the first grounding step, when all the operators and predicates are grounded in order to create the data structures needed by the search phase. Therefore, this potential issue of the problem model mainly targets the grounding step, and it has the potential of slowing down planning engines by increasing the size of the grounded problem.

```
(:action inapplicable-drop
:parameters
  (?r - robot ?obj - object ?room - room ?g - gripper)
:precondition  (and  (carry ?r ?obj ?g)
                (at ?obj ?room)
                (at-robby ?r ?room))
:effect (and (free ?r ?g)
         (at ?obj ?room)
         (not (carry ?r ?obj ?g)))))
```

Figure 1: An example of an inapplicable dummy operator from the Gripper domain model. The operator is obtained by adding an additional precondition (at ?obj ?room), that is in a mutex relationship with (carry ?r ?obj ?g).

Domain Models: Dummy Operators

With regards to domain models, a possibility for an engineering issue that is hard to spot is the presence of dummy planning operators in the original models. In order to comply with the relation of plan equivalence, the attacker can add only operators whose instances cannot be present in any solution plan. Instances of such dummy operators must either be inapplicable at any point, or their application must lead to dead-ends. Intuitively, the former type of dummy operators is easier to prune, and therefore can slow down only the grounding process of a planning engine. Dead-ends dummy operators are much harder to prune, and hence they can slow down the search process as well.

Inapplicable Dummy Operators For an action a (a grounded planning operator), it is the case that a is never applicable if its precondition cannot be satisfied in any reachable state, i.e., there does not exist a sequence of actions whose consecutive application in the initial state results in a state in which all atoms from $pre(a)$ are true. One of the cases in which a is never applicable is when atoms $p_g, q_g \in pre(a)$ are *mutex*, i.e., p_g and q_g cannot be both true in any reachable state.

Generalising the notion of mutexes for (ungrounded) predicates, we say that predicates p and q are mutex with respect to substitutions Θ, Ω if and only if for all grounded instances of $\Theta(p)$ and $\Omega(q)$ it is the case that they are mutex. Practically speaking, Θ and Ω are used to unify corresponding predicates' variables. For example, let us consider the Gripper domain, where a robot with two grippers is required to move balls between different rooms. A ball can be either placed at some location, or being carried by a robotic gripper. Hence, predicates (at ?ball ?loc) and (carry ?robot ?ball ?gripper) are mutex (the variable ?ball is unified). An example of an inapplicable dummy operator for the Gripper domain is shown in Figure 1.

While there exist techniques being able to identify mutexes (for instance, the Fast Downward framework (Helmert 2006) includes a very effective approach), a simplistic way how to identify (some) mutexes is to analyse planning operators defined in the domain model.

```
(:action opposing-drop
:parameters
  (?r - robot ?obj - object ?room - room ?g - gripper)
:precondition  (and  (carry ?r ?obj ?g)
                (at-robby ?r ?room))
:effect (and (free ?r ?g)
         (not (carry ?r ?obj ?g)))))

(:action opposing-move-drop
:parameters
  (?r - robot ?obj - object ?f ?to - room ?g - gripper)
:precondition  (and  (carry ?r ?obj ?g)
                (at-robby ?r ?f))
:effect (and (at-robby ?r ?to)
         (not (at-robby ?r ?f))
         (free ?r ?g)
         (not (carry ?r ?obj ?g))))
```

Figure 2: Examples of dead-end dummy operators from the Gripper domain model. The top operator represents a dead-end dummy operator obtained by removing the atom (at ?obj ?room). The bottom operator is obtained in a similar way, but is then combined with the move operator to increase the grounded size of the planning task, and to make the operator more likely to be selected by planning engines.

Given predicates p, q and unifying substitutions Θ, Ω we say that p, q are *mutex* with respect to Θ, Ω if: for all operators o there exists a renaming substitution Ψ such that Ψ renames all variables of o it is the case that o has at most one variant of p, or q respectively, in $\mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$, $\Omega(q) \in \mathit{eff}^+(\Psi(o))$ (or $\Theta(p) \in \mathit{eff}^+(\Psi(o))$) implies $\Theta(p) \in \mathit{pre}(\Psi(o)) \cap \mathit{eff}^-(\Psi(o)) \setminus \mathit{eff}^+(\Psi(o))$ (or $\Omega(q) \in \mathit{pre}(\Psi(o)) \cap \mathit{eff}^-(\Psi(o)) \setminus \mathit{eff}^+(\Psi(o))$), and there do not exist grounded instances of $\Theta(p)$ and $\Omega(q)$ being both present in the initial state.

If the following condition is met, then no instance of the dummy operator o is applicable at any point of the planning process.

Condition 2. *For some predicates p and q (defined in the domain model) that are mutex with respect to substitutions Θ, Ω it is the case that $\Theta(p), \Omega(q) \in \mathit{pre}(o)$.*

Additional preconditions and effects for the dummy operator can be random or completely meaningless, but involving predicates defined in the domain model. This is typically the case of models that are incrementally generated using simple tools such as a text editor, where changes to some part of the model are not reflected in the rest of it. Of course, for the sake of our attacker's perspective, instead of random generation, it is also possible to generate macro-operators, by incorporating in the dummy operator parameters, preconditions, and effects of suitable existing operators from the considered domain model. According to Condition 2, instances of the dummy operator are inapplicable in all reachable states (regardless of additional preconditions and effects) and thus cannot be part of any solution plan. Hence, a task modified by adding such a dummy operator (or more such dummy operators) is plan equivalent with the original task.

Dead-end Dummy Operators Dead-end states are those states from which a goal state is no longer reachable. Actions whose application always leads to dead-end states cannot be a part of any solution plan. For example, in the Gripper domain, if some (dummy) action "removes" a ball from the environment (i.e., the ball is not at any location as well as the ball is not carried by any gripper –the ball cannot be used anymore), then applying such an action leads to a dead-end state (assuming that we require the ball to be placed in some location in the goal).

With (ungrounded) operators we must ensure that all their instances when applied lead to dead-end states. The object-erasing strategy can be applied if a goal is defined for each object of a given class (e.g. all balls have to be placed somewhere).

If atoms p_g and q_g are mutex they can be both false. However, in some cases either p_g or q_g must be present in each reachable state. We say that predicates p and q *alternate* with respect to Θ, Ω if they are mutex (with respect to Θ, Ω) and for each operator o there exists a renaming substitution Ψ such that $\Omega(q) \in \mathit{eff}^-(\Psi(o))$ (or $\Theta(p) \in \mathit{eff}^-(\Psi(o))$) implies $\Theta(p) \in \mathit{eff}^+(\Psi(o))$ (or $\Omega(q) \in \mathit{eff}^+(\Psi(o))$).

If the following condition is met, then application of each instance of a dummy operator o leads to a dead-end state.

Condition 3. *Let p and q alternate with respect to Θ, Ω. Let x be an parameter (variable symbol) that both $\Theta(p)$ and $\Omega(q)$ share. For each object of type of x an instance of q is present in the goal, it is the case that $\Theta(p) \in \mathit{eff}^-(o) \cap \mathit{pre}(o)$ while $\Omega(q) \notin \mathit{eff}^+(o)$.*

Another object-erasing possibility is to delete atoms that cannot be re-achieved but are required to achieve goals. We say that a predicate q *strictly requires* a predicate p with respect to Θ, Ω if for each operator o there exists a renaming substitution Ψ such that $\Omega(q) \in \mathit{eff}^+(\Psi(o))$ implies $\Theta(p) \in \mathit{pre}(\Psi(o))$.

If the following condition is met, then application of each instance of a dummy operator o leads to a dead-end state.

Condition 4. *Let q strictly require p with respect to Θ, Ω. Let x be an parameter (variable symbol) that both $\Theta(p)$ and $\Omega(q)$ share. For each object of type of x an instance of q is present in the goal and there is no operator o' and a renaming substitution Ψ such that $\Theta(p) \in \mathit{eff}^+(\Psi(o'))$ it is the case that $\Theta(p) \in \mathit{eff}^-(o)$ while $\Omega(q) \notin \mathit{eff}^+(o)$.*

Analogously to the previous case, additional preconditions and effects for the dummy operator can be random (considering the predicates defined in the domain model), or by incorporating existing operators of the model, as long as Condition 3 or 4 is satisfied. Again, both cases can be the result of a poor knowledge engineering process, for instance due to incremental modifications done via a text editor to existing models. Both conditions ensure that if an instance of the dummy operator is applied, then a dead-end state is reached, since an object that has to be in a certain goal state has been erased from the environment. Figure 2 shows two examples of dead-end dummy operators for the Gripper domain, where the ball object is erased from the environment. The top operator is obtained by removing a single fact from the effects of the original drop operator; The bottom operator is obtained by generating a macro-operator that includes the original move operator and the modified drop operator. It should be noted that, for guaranteeing the solvability of planning instances, the original operators are left in the domain model.

Empirical Analysis

Our experimental analysis aims to evaluate the impact that the engineering issues described in previous sections have on the performance of state-of-the-art planning engines, in order to assess the robustness of the state of the art with regards to such common issues.

We selected five planning engines, based on their performance in the International Planning Competition and/or the use of very different planning approaches: LAMA (Richter and Westphal 2010), Lpg (Gerevini, Saetti, and Serina 2003), Madagascar (Mp) (Rintanen 2014), Probe (Lipovetzky et al. 2014), and Dual-BFWS (Lipovetzky et al. 2018). In particular, the selected planners exploit very different pre-processing approaches, and can therefore provide some useful insights into the impact of the introduced techniques. Furthermore, by considering LAMA and Dual-BFWS, it is possible to evaluate how planners developed on top of, respectively, the Fast Downward (Helmert 2006) and the LAPKT (Ramirez, Lipovetzky, and Muise 2015) frameworks would react to the described attacks.[2]

We focused our study on domains that have been used in International Planning Competitions (IPC), that allows to meet the conditions specified in the previous section, and for which a randomised problem generator is available. The models we chose had at least four operators and, possibly, a large number of predicates. These can represent conditions under which knowledge engineering issues can more easily arise. Selected domains are: Barman, Childsnack, Floortile, Gripper, and Spanner. For each domain, we randomly generated 20 instances.[3]

Experiments were run on a dedicated machine equipped with Intel Xeon 2.50 Ghz processors and Linux operating system. Each run was limited to a single core, 4 GB of RAM, and 300 CPU-time seconds, as in the Agile track of IPCs. All the generated plans have been validated using the VAL tool (Howey, Long, and Fox 2004) on the original domain model and problem instance files. This has been done to check, at least for the generated solutions, that the manipulated models maintain the plan equivalence as defined in Definition 1. In other words, the original flawless model is used as ground truth. Validating against the original model simulates the fact that a formal specification of requirements for an application domain is available, and it can be used to validate the provided knowledge models and their output.

Table 1 shows, in terms of average runtime increase and average increase of grounded atoms, how *increasing the number of objects* of a planning instance affects the performance of the selected planning engines. The table shows the increase of grounded facts after the simplification step, therefore the pruning of non relevant facts has already been performed. In our experiments, the number of objects of each type have been increased by 10%. In cases where less than 10 objects are declared for a considered type, only 1 fictional object is added. In Gripper, objects of type `room` can be used even if there is no predicate referring to them in the initial state of the problem instance. For this reason, no additional room objects have been added in that domain as this type of objects violates Condition 1 (in all the other cases the object types satisfy Condition 1.).

Results in Table 1 indicate that planning engines can be, in some cases significantly, affected by an attack that artificially increases the number of objects of a planning instance, even if the objects are not relevant for the task to be solved. Among the selected planning engines, Probe is the most robust with regards to this sort of attack. Remarkably, Probe is the only engine that is completely unaffected by the addition of dummy objects in the Floortile domain. LAMA and Mpc are very sensitive to dummy objects added to instances of the Barman domain, while LPG performance is significantly affected in the ChildSnack domain. However, we observed that the increase in the grounded facts does not usually correspond to an increase in the grounded actions taken into account during the search phase. Out of the considered planning engines, only LPG shows a slight increase in grounded actions ($+5\%$ in ChildSnack and approximately $+1\%$ in both Gripper and Spanner). In that, the performance slow down can be attributed to the pre-processing phase and to the larger size of each search state. The behaviour of Dual-BFWS is extremely interesting: despite the fact that no additional facts are grounded, runtime performance can be significantly affected by an increased number of objects in the problem model. We observed that this is due to the fact that adding objects has an impact on the way in which the search space is explored: in many cases, this leads to a larger number of the visited nodes, which is reflected in an increased runtime. On the other hand, counter-intuitively, in some cases this can also boost the performance of the planner, as in Gripper. Our intuition is that the parsing of the additional objects affects the data structure of the planning engine and, despite the fact that the objects are pruned during pre-processing, changes of the data structures affect the subsequent search phase.

In summary, although planning engines actively prune unreachable actions and thus they should identify dummy objects, in about a quarter of cases, the planners are not able to effectively prune all the dummy objects. However, even if the dummy objects are pruned they can still affect the search phase, mostly in a negative way but not necessarily as shown by the Dual-BFWS planning engine in the Gripper domain.

In order to evaluate the impact of dummy operators, let us initially consider those that intuitively should maximise the impact on the planning engine: *dead-end dummy operators*. Results of this evaluation are presented in Table 2. In each domain, a single dummy operator has been generated by merging 2 operators: the one that allows to "erase"

[2]It should be noted that the LAPKT framework provides a range of techniques that can be used for parsing and pre-processing, so the observed impact may vary accordingly.

[3]Domain and problem models can be found at https://tinyurl.com/KCAP19.

Table 1: Results of how increasing the number of objects of a planning instance affects the performance of the selected planning engines. For each engine and domain, we show average percentage of runtime increase (considering only instances solved by both runs), and the average percentage of increase of the number of facts. "–" indicates that no instance has been solved by the planner in the corresponding domain.

	Barman		ChildSnack		Floortile		Gripper		Spanner	
	Time	**Facts**	**Time**	**Facts**	**Time**	**Facts**	**Time**	**Facts**	**Time**	**Facts**
Dual-BFWS	+4.3%	0.0%	+1.0%	0.0%	+33.4%	0.0%	−6.1%	0.0%	+48.7%	0.0%
LAMA	+16.4%	+2.6%	–	+0.3%	0.0%	+1.7%	+2.5%	+0.1%	+0.2%	+5.6%
LPG	+6.4%	0.0%	+56.6%	+1.3%	+2.0%	+1.3%	+3.7%	+1.0%	+5.8%	0.0%
Mpc	+15.3%	0.0%	+0.9%	0.0%	+3.4%	0.0%	+7.8%	0.0%	+25.2%	0.0%
Probe	+0.2%	0.0%	–	+1.8%	0.0%	0.0%	+1.5%	0.0%	+0.6%	0.0%

Table 2: Impact of dead-end dummy operators on the selected planning engines. For each engine and domain, we show between brackets the average runtime, followed by the average runtime slowdown (averages are calculated by considering only instances solved by both runs), the coverage delta (percentage), and the average increase of the number of actions considered during the search phase. $\times 1.0$ represents cases where no slowdown is observed: the higher the value, the higher the slowdown. "–" indicates that no instance has been solved by the planner in the corresponding domain.

	Time	**Δ Coverage**	**Actions**
	Barman		
Dual-BFWS	(27.9) ×2.3	−20.0%	×3.0
LAMA	(5.7) ×2.1	0.0%	×2.2
LPG	(7.1) ×6.4	0.0%	×3.0
Mpc	(5.5) ×1.2	−10.0%	×2.3
Probe	(2.1) ×3.7	−5.0%	×3.0
	ChildSnack		
Dual-BFWS	(43.1) ×1.4	0.0%	×1.7
LAMA	–	–	×2.0
LPG	(20.7) ×11.9	−25.0%	×2.0
Mpc	(0.2) ×5.8	−6.7%	×2.0
Probe	–	–	×1.7
	Floortile		
Dual-BFWS	(74.9) ×11.1	−5.0%	×1.5
LAMA	(7.9) ×1.1	0.0%	×1.5
LPG	(3.0) ×3.6	0.0%	×1.5
Mpc	(0.1) ×2.4	0.0%	×1.5
Probe	(67.2) ×2.1	0.0%	×1.5
	Gripper		
Dual-BFWS	(57.1) ×1.1	−25.0%	×5.0
LAMA	(29.5) ×6.5	−10.0%	×5.1
LPG	(3.1) ×34.4	−35.0%	×5.7
Mpc	(7.1) ×3.4	−50.0%	×4.5
Probe	(45.1) ×3.5	−40.0%	×4.9
	Spanner		
Dual-BFWS	(32.1) ×1.5	0.0%	×1.8
LAMA	(9.1) ×1.4	0.0%	×6.4
LPG	(1.9) ×1.1	0.0%	×1.8
Mpc	(0.5) ×4.2	−20.0%	×1.8
Probe	(5.6) ×1.4	0.0%	×1.8

objects from the environment, and another one selected following the criteria mentioned in the previous section. In this analysis, we erase objects listed in the goal state, by modifying the operators that allows to reach goal facts. Added operators are listed at the top of the domain model, following the empirical evidence provided by Vallati et al. (2015), suggesting that operators listed early are considered more often during the search process. We therefore assume that in this way, the detrimental impact on planning performance is maximised.

In terms of runtime, all the considered planning engines are very negatively affected by the additional dummy operator. LPG is the planning engine that generally gets the strongest slowdown: in the Gripper domain, it takes on average more than 34 times longer to solve a planning instance. In many cases, the injection of an additional dummy operator can reduce the coverage of most of the engines. It is worth remarking that performance reduction is not directly related to the increase in the number of grounded actions. In other words, the fact that a given engine grounds a larger number of actions than another planning engine, does not automatically result in the former planning engine being slower than the latter. According to our analysis, the larger number of actions leads to the generation of more search nodes that has quite a significant impact on runtime. As an example, in the Gripper domain Dual-BFWS generates on average 10% more nodes.

To better clarify the impact of dead-end dummy operators, we also performed experiments by considering "minimalistic" dummy operators. They satisfy the introduced criteria, but are generated as follows. From an operator that achieves goal facts, the "goal achieving" effect is removed. No additional preconditions or effects are considered. A sample "minimalistic" dummy operator is shown in the top part of Figure 2. The considered planning engines' performance are negatively affected also by this malicious input, but in a less substantial way. As an example, the results achieved by the considered planning engines in the Barman domain are shown in Table 3. The use of more complex dead-end dummy operators result in a stronger impact on planning performance, but the extension of domain models with minimally-modified operators still leads to noticeable performance degradation. In the other domains, similar figures can be derived.

Table 3: Barman: impact of a "minimalistic" dead-end dummy operator on the selected planning engines, in terms of runtime slowdown (average runtime), coverage, and average increment of the number of actions grounded and considered during the search phase.

	Time	**Δ Coverage**	**Actions**
	Barman		
Dual-BFWS	(27.9) ×1.9	-5.0%	×1.3
LAMA	(5.7) ×1.3	0.0%	×1.3
LPG	(7.1) ×2.3	0.0%	×1.3
Mpc	(5.5) ×1.1	-10.0%	×1.3
Probe	(2.1) ×1.4	0.0%	×1.3

Table 4: Impact of an inapplicable dummy operator on the selected planning engines, in terms of runtime slowdown (average runtime), coverage, and average increment of the number of actions grounded and considered during the search phase, in Gripper.

	Time	**Δ Coverage**	**Actions**
	Gripper		
Dual-BFWS	(57.1) ×1.0	-20.0%	×3.0
LAMA	(29.5) ×3.5	-10.0%	×3.7
LPG	(3.1) ×16.1	-20.0%	×5.2
Mpc	(7.1) ×1.3	-30.0%	×1.0
Probe	(45.1) ×2.5	-40.0%	×3.7

We also empirically tested the impact on planning engines' performance of the introduced *inapplicable dummy operators*. As expected, state-of-the-art planning engines can identify and prune such operators. In most of the benchmarks the considered engines proved to be robust with regards to this sort of attack: they were able to remove all the grounded inapplicable operators, with very limited overhead in terms of runtime. However, this is not the case in the Gripper domain, as shown by the results presented in Table 4. The mutex relationship between the (at ?ball ?loc) and (carry ?robot ?ball ?gripper) predicates is not correctly identified, and we exploited it in a "macro" composed by the move and drop operators sequence. The inability to recognise the mutex, and the fact that a large number of objects are involved in the tasks, lead to a significant surge in runtimes. Mpc is the only considered planning engine that is able to prune the added operator, but at the cost of a significant increase in terms of runtime and memory consumption of the pre-processing stage.

With regards to the impact of dead-end and inapplicable dummy operators on the quality of generated plans, we did not observe any significant variation. While it is true that the generated plans may differ, it is not the case that the use of domain models extended with the dummy operator leads to worse solutions.

Summarising, the presented results demonstrate that increasing the number of objects of problem instances can be a viable way for slowing down the planning process of domain-independent planning engines. Furthermore, from Tables 2 and 3, it can be seen that an attack based on modifying the domain model with an additional dummy operator, that is guaranteed to generate dead ends thus will never appear in a valid solution plan, can significantly reduce the performance of the selected domain-independent planning engines. Instead, attacks based on the injection of inapplicable operators in the domain model showed a limited impact performance, but can still be fruitfully exploited in some domains. Finally, it is worth emphasising that the two type of attacks have a different impact on performance: adding dummy objects has a more limited effect on planning engines than injecting of a dummy operator into the domain model.

Discussion

We reckon that the domain models used in our empirical analysis, and in general most of the benchmarks used in the International Planning Competition, are usually much less complex than those exploited in real-world applications. This is because such benchmarks are primarily chosen to be "challenging" for the participating planning engines, rather than to describe the actual issues that engines would have to face in real-world applications (Vallati, Chrpa, and McCluskey 2018). Worryingly, this implies that the impact of described knowledge engineering issues of models exploited in planning applications is expected to be amplified. In more complex models, difficulties for planning engines are usually exacerbated. In a nutshell, this can possibly imply that automated planning has been deemed not to be useful in some applications because poorly engineered models were used for trialling purposes.

Our experimental analysis does not include portfolio-based planners, even though they showed in recent competitions to be able to consistently deliver outstanding performance (see, e.g., (Cenamor, de la Rosa, and Fernández 2016; Seipp et al. 2015; Gerevini, Saetti, and Vallati 2014)). This is because the exploitation of different planning engines, combined in various ways, would make it hard to understand the actual impact of the attacks, and to isolate the step of the planning process that has been mostly affected. Out intuition with regards to portfolio approaches is that, since portfolios are composed by "basic" planning engines, if each basic engine is slowed down, also their combination will be negatively affected, probably to a similar degree. However, it may be the case that some configurations of planning engines could magnify the impact of the issues of domain models, but that has to be assessed in a very detailed case-by-case analysis.

Conclusion

Automated planning techniques are currently exploited in a wide range of real-world applications. Their widespread exploitation, and the fact that practitioners and non-planning experts are now in the position of testing planning techniques by themselves, raises questions about the robustness of planning engines with regards to issues in the provided input. To raise awareness on the weaknesses of planning frameworks, in this work we took an attacker's perspective. We provided three examples of attacks targeting the input models, that aims at reducing (or even disrupting) the performance of a planning engine without the possibility of being discovered by the analysis of the generated plans, and that are also representative of typical knowledge engineering issues that may arise. Our experimental analysis demonstrated that state-of-the-art planning engines can be strongly affected by such (to some extent trivial) issues.

Given the observed results, it is pivotal to investigate techniques and approaches that can improve the robustness of planning engines, in terms of their resilience to issues in the provided input models. This can be done, for instance, by performing more in-depth pre-processing steps before the actual grounding step. There is also an incumbent need for more supportive Knowledge Engineering techniques, that can perform both static and dynamic validation of planning models. Such tools would support the analysis of input models, and would make much harder for an attack to be undetected, or to affect the performance of a planning engine. On this matter, results from the recent International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS) highlighted the lack of tools and approaches for supporting a principled engineering of planning models (Chrpa et al. 2017). It is also worth noting that approaches such as DISCOPLAN (Gerevini and Schubert 2000) are able to detect invariants, and would help in improving the robustness of planning engines. For some reasons, such techniques are not exploited in most of the ready-to-use engines.

We see several avenues for future work. First, we are interested in evaluating the impact of the proposed attacks on different types of planning, e.g. optimal, and with different versions of the PDDL language. Second, we plan to evaluate the impact of knowledge engineering issues can affect the quality of generated plans. Finally, we aim at investigating appropriate techniques for supporting knowledge engineers in the tedious task of encoding planning knowledge, in order to limit the arising of potentially disruptive issues in the generated models.

Acknowledgements

This research was funded by the Czech Science Foundation (project no. 18-07252S).

References

Brafman, R. I.; Domshlak, C.; Engel, Y.; and Tennenholtz, M. 2009. Planning games. In *IJCAI, Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 73–78.

Brooks, F. P. 1987. No silver bullet: Essence and accidents of software engineering. *IEEE Computer* 20:10–19.

Capitanelli, A.; Maratea, M.; Mastrogiovanni, F.; and Vallati, M. 2018. On the manipulation of articulated objects in human-robot cooperation scenarios. *Robotics and Autonomous Systems* 109:139–155.

Cenamor, I.; de la Rosa, T.; and Fernández, F. 2016. The ibacop planning system: Instance-based configured portfolios. *J. Artif. Intell. Res.* 56:657–691.

Chrpa, L.; McCluskey, T. L.; Vallati, M.; and Vaquero, T. 2017. The fifth international competition on knowledge engineering for planning and scheduling: Summary and trends. *AI Magazine* 38(1):104–106.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research* 27:235–297.

Fox, M.; Long, D.; and Magazzeni, D. 2012. Plan-based policies for efficient multiple battery load management. *J. Artif. Intell. Res.* 44:335–382.

Geffner, H., and Bonet, B. 2013. *A Concise Introduction to Models and Methods for Automated Planning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers.

Gerevini, A., and Schubert, L. K. 2000. Discovering state constraints in DISCOPLAN: some new results. In *Proceedings of AAAI*, 761–767.

Gerevini, A. E.; Saetti, A.; and Serina, I. 2003. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research (JAIR)* 20:239–290.

Gerevini, A.; Saetti, A.; and Vallati, M. 2014. Planning through automatic portfolio configuration: The pbp approach. *J. Artif. Intell. Res.* 50:639–696.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers.

Helmert, M. 2006. The fast downward planning system. *J. Artif. Intell. Res.* 26:191–246.

Hoffmann, J. 2015. Simulated penetration testing: From "dijkstra" to "turing test++". In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS*, 364–372.

Howey, R.; Long, D.; and Fox, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, 294–301.

Kvarnström, J., and Doherty, P. 2010. Automated planning for collaborative uav systems. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, 1078–1085.

Laskov, P., and Lippmann, R. 2010. Machine learning in adversarial environments. *Machine Learning* 81(2):115–119.

Lipovetzky, N.; Ramirez, M.; Muise, C.; and Geffner, H. 2014. Width and inference based planners: Siw, bfs(f), and probe. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Lipovetzky, N.; Ramirez, M.; Frances, G.; and Geffner, H. 2018. Best-first width search in the ipc2018: Complete, simulated, and polynomial variants. In *The Ninth International Planning Competition. Description of Participant Planners of the Deterministic Track.*

McCluskey, T. L., and Vallati, M. 2017. Embedding automated planning within urban traffic management operations. In *Proceedings of the Twenty-Seventh International Confer-*

ence on Automated Planning and Scheduling, ICAPS, 391–399.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering knowledge for automated planning: Towards a notion of quality. In *Proceedings of the Knowledge Capture Conference, K-CAP*, 14:1–14:8.

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

Ramirez, M.; Lipovetzky, N.; and Muise, C. 2015. Lightweight Automated Planning ToolKiT. http://lapkt.org/.

Richter, S., and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *J. Artif. Intell. Res.* 39:127–177.

Rintanen, J. 2014. Madagascar: Scalable planning with SAT. In *Proceedings of the 8th International Planning Competition (IPC-2014)*.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 3364–3370.

Speicher, P.; Steinmetz, M.; Backes, M.; Hoffmann, J.; and Künnemann, R. 2018. Stackelberg planning: Towards effective leader-follower state space search. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

Vallati, M.; Hutter, F.; Chrpa, L.; and McCluskey, T. L. 2015. On the effective configuration of planning domain models. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, 1704–1711.

Vallati, M.; Chrpa, L.; and McCluskey, T. L. 2018. What you always wanted to know about the deterministic part of the international planning competition (IPC) 2014 (but were too afraid to ask). *Knowledge Eng. Review* 33:e3.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE 2.0: An Integrated Tool for Designing Planning Domains. In *Proceedings of the International Conference on Planning and Scheduling (ICAPS)*, 336–343.