Computational Intelligence, Volume 59, Number 000, 2010

Inner Entanglements: Narrowing the Search in Classical Planning by Problem Reformulation

LUKÁŠ CHRPA[1,2], MAURO VALLATI[3] AND THOMAS LEO MCCLUSKEY[3]

[1]*Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic*

[2]*Faculty of Mathematics and Physics, Charles University in Prague, Czech Republic*

[3]*School of Computing and Engineering, University of Huddersfield, UK*

In the field of Automated Planning, a central research focus is on domain-independent planning engines which accept planning tasks (domain models and problem descriptions) in a description language such as PDDL, and return solution plans. Performance of planning engines can be improved by gathering additional knowledge about specific planning domain models/tasks (such as control rules) which can narrow the search for a solution plan. Such knowledge is often learnt from training plans, solutions of simple tasks. Using techniques to reformulate the given planning task to incorporate additional knowledge, while keeping to the same input language, allows to exploit off-the-shelf planning engines. In this paper we present *inner entanglements* that are relations between pairs of operators and predicates which represent exclusivity of predicate achievement or requirement between the given operators. Inner entanglements can be encoded into a planner's input language by transforming the original planning task, so planning engines can exploit them. The contribution of this paper is to provide an in depth analysis and evaluation of inner entanglements, covering theoretical aspects such as complexity results, and an extensive empirical study using IPC benchmarks and state-of-the-art planning engines.

Key words: Knowledge Representation, Classical Planning, Machine Learning, Problem Reformulation

1. INTRODUCTION

Automated planning is an important research area of Artificial Intelligence (AI) where an autonomous entity (e.g. a robot) reasons about the way it can act in order to achieve its goals. AI planning has therefore a great potential for applications where a certain level of autonomy is required such as in the Deep Space 1 mission (Bernard et al., 2000). Classical planning is a subarea of AI planning that deals with a static and fully observable environment and actions have deterministic and instantaneous effects. Classical planning is, however, intractable (PSPACE-complete) (Bylander, 1994).

In the last few decades, there has been a great deal of activity in the research community designing planning techniques and planning engines. Since 1998, the International Planning Competition (IPC)[1] is being organised and is increasingly attracting the attention of the AI planning community. Thanks to the IPC we have PDDL (Ghallab et al., 1998), that is a widely used language for describing planning tasks, and a wide range of benchmarks that can be used for measuring planners' performance. Currently, PDDL is supported by a large number of advanced planning engines. Along with those planning engines, many novel planning techniques have been proposed, such as heuristic search (Bonet and Geffner, 1999), translating planning tasks into SAT (Kautz and Selman, 1992) just to mention a few.

Performance of planning engines can be improved by restricting the search space, i.e., by introducing pruning techniques that "cut off" branches that are unnecessary

[1] http://ipc.icaps-conference.org

or redundant. *Commutativity pruning* eliminates all but one permutation of commutative actions (can be applied in any order) (Godefroid and Kabanza, 1991). *Symmetry breaking* reuses information about one object to its symmetric "twin" in such a way that "bad" states of one object can be avoided for its symmetric "twin" (Fox and Long, 1999). *Reachability analysis* can determine whether the goal is unreachable from a current state (Bonet and Geffner, 1999).

Another way how performance of planning engines can be improved is by gathering Domain Knowledge (DCK), i.e., additional knowledge about planning tasks indicating how solution plans would look like. DCK can be expressed, for instance, in the form of Control Rules (Minton and Carbonell, 1987), Temporal Logic formulas (Bacchus and Kabanza, 2000), or Decision Trees (de la Rosa et al., 2011). With growing interest in extracting DCK automatically, the emphasis was given to exploiting machine learning techniques that can acquire useful DCK, usually, by analysing "training plans", which are solutions of simple planning tasks. This motivated the foundation of the Learning Track in the IPC, which has been organised since 2008. It should be noted that an approach that learns DCK from relaxed plans (obtained by solving planning tasks while omitting negative effects of actions) (Yoon et al., 2008) won the best learner award at IPC 2008. However, such types of knowledge often require specific planning engines such as TALPlanner (Kvarnström and Doherty, 2000) in case of control rules. Alternatively, domain knowledge can be directly encoded into the domain and problem descriptions (usually in PDDL). Such an approach is planner-independent, so a standard planning engine can straightforwardly exploit it.

The best known planning task reformulation technique, macro-operators ("macros"), which encapsulate sequences of PDDL operators, can be encoded as normal planning

operators, so they can be easily added into domain models (Korf, 1985; Newton et al., 2007; Chrpa, 2010; McCluskey and Porteous, 1997). Abstracting planning tasks by their reformulation in order to reveal their hierarchical structures can mitigate "accidental complexity" of their domain models[2] (Haslum, 2007; Tozicka et al., 2016).

Beside macros, another type of domain-independent domain knowledge are *Entanglements* (Chrpa and Barták, 2009; Chrpa and McCluskey, 2012), which represent relations between planning operators and predicates, aiming at eliminating unpromising alternatives in a planning engine's search space. Technically speaking, entanglements are task-specific, i.e., relations described by entanglements hold in at least one solution plan of a given task. Entanglements usually generalize well, that is, a set of entanglements hold for a class of planning tasks with the same domain model.

Outer Entanglements (Chrpa and Barták, 2009) are relations between planning operators and predicates whose instances are present in the initial state or the goal. *Inner Entanglements* (Chrpa and McCluskey, 2012; Chrpa and Vallati, 2013), we focus on in this paper, are relations of exclusivity of predicate achievement or requirement between between pairs of operators. Inner entanglements can be encoded in planning tasks, effectively re-formulating them, and thus they are planner-independent. Deciding whether a given inner entanglement holds in a given planning task is generally intractable (PSPACE-complete) and thus as hard as solving a planning task. Such a theoretical result indicates practical infeasibility of enumerating entan-

[2] "accidental complexity of domain models" means that their inefficient encodings decrease performance of planning engines

glements for a given task prior solving it. Since inner entanglement generalize well as reported in literature (Chrpa and McCluskey, 2012; Chrpa and Vallati, 2013), i.e., they are rather domain-specific than task-specific, for extracting them, we have exploited the "Learning for Planning" paradigm, which identifies domain knowledge from a set of "training" planning tasks. Therefore, inner entanglements can be learnt on simple training tasks, which are easy to solve, and then used for more complex tasks (in the same class) for speeding-up plan generation process. Our initial work on inner entanglements has been reported in a couple of shorter papers detailing their discovery, use and effectiveness (Chrpa and McCluskey, 2012; Chrpa and Vallati, 2013). In this paper, we integrate and extend our previous work, with:

- A detailed description of the encodings of inner entanglements, including formal proofs of their correctness;
- A collected summary of the known complexity results, and trivial cases where inner entanglements hold;
- Case studies in which we investigate knowledge engineering aspects of (re)using inner entanglements;
- An analysis of the potential impact of inner entanglements on the planning process;
- An approximation method for extracting entanglements enriched by filtering unpromising inner entanglements;
- An extensive empirical study of the impact of inner entanglements in the planning process using all the domains from the 7th IPC's learning track[3], and 7 state-of-the-art planning engines based on very different principles.

[3]Learning track benchmarks are more natural, since inner entanglements extraction phase can be understood as a learning process

Although our approximation method for learning inner entanglements does not theoretically guarantee that the reformulated tasks remain solvable, the main empirical findings from this paper are that the use of inner entanglements improve the planning process generally through the considered planner and domain model combinations. Also, in the experimental scenarios we used, the potential for identification of incorrect inner entanglements stemming from our approximation method for their extraction, did not explicitly manifest itself in the results. Using the "learning for planning" paradigm, i.e., learning domain specific knowledge on a small set of training tasks, has demonstrated its usefulness in the inner entanglements case. Noticeably, the issue of making some reformulated tasks unsolvable can be alleviated i) by running the planner on the original task if the reformulated task was unsolved, ii) by domain engineers who can verify the correctness of learnt inner entanglements, or iii) by incorporating reformulated tasks along with the original ones into portfolios such as PbP (Gerevini et al., 2014).

The paper is organised as follows. After discussing related work, basic terminology is provided. Then, inner entanglements are introduced. After that reformulation of planning tasks in order to enforce inner entanglements is presented. Then, theoretical analysis of inner entanglements is provided, and an approximation algorithm for extracting inner entanglements is presented (including the filtering technique for unpromising inner entanglements). After that, empirical analysis of impact of inner entanglements in the planning process is provided. Finally, we give conclusions and present future avenues of research.

2. RELATED WORK

Generating domain knowledge which can be exploited by planning engines dates back into 1970s, when systems such as REFLECT (Dawson and Siklóssy, 1977) were developed. Macros are one of the best known type of domain knowledge in classical planning, because they can be encoded as normal planning operators and thus easily added into planning domain models (Korf, 1985). Macro-FF CA-ED version (Botea et al., 2005), which learns macros through analysis of relations between static predicates, Wizard (Coles et al., 2007), which learns macros by genetic algorithms, and BLOMA (Chrpa and Siddiqui, 2015), which exploits a block decomposition technique (Siddiqui and Haslum, 2012) to learn "long" macros, are good examples of planner-independent macro learning systems. Although macros and inner entanglements are based on a similar idea, i.e., enforcing (primitive) operators to be applied in certain order, inner entanglements do not require the affected operators to be applied strictly consecutively and inner entanglements can be represented in such a way that the number of operators' instances (after grounding) is not higher than when the original models are considered. The relation between inner entanglements and macros and how inner entanglements can be exploited for macro learning has been studied by Chrpa et al. (2013).

A general technique, called *commutativity pruning*, is used to discard all but one permutation of commutative (or independent) actions, which do not influence each other and thus can be executed in any order (Godefroid and Kabanza, 1991). Graphplan (Blum and Furst, 1997), one of the best-known planning algorithms, allows the execution of commutative actions in parallel (in one step). *Symmetry breaking*

is a well known technique for pruning unneeded alternatives in the search space. In planning, some objects might be symmetric which can be exploited for avoiding alternatives concerning one object that have been already tried with the object's symmetric "twin" (Fox and Long, 1999). In the spirit of works of Emerson and Sistla (1996) and Rintanen (2003), Pochter et al. (2011) present a pruning technique which identifies symmetries by exploring automorphisms in state-transition systems. This approach has been recently extended for cost-optimal planning (Domshlak et al., 2012). Motivated by the idea of partial order based reduction used in model checking (Valmari, 1996), Chen and Yao (2009) introduce an *Expansion Core* method, focusing on cost-optimal SAS+ planning (Bäckström and Nebel, 1995), which in a node expansion phase (in the A* search) restricts on relevant Domain Transition Graphs rather than all of them. The idea of "expansion cores" is extended into *strong stubborn sets* that guarantees stronger pruning than "expansion cores" (Wehrle et al., 2013). In contrast, inner entanglements prune asymmetrical alternatives. Outer Entanglements (Chrpa and Barták, 2009) are relations between operators and initial or goal atoms which aim to prune unpromising instances of these operators. Outer and inner entanglements are complementary as has already been demonstrated in literature (Vallati et al., 2014).

Recent work which is to some extent similar to inner entanglements proposes a method to learn "bad" causal links in order to generate plans of better quality (Celorrio et al., 2013). In contrast to this work, inner entanglements aim to capture possibly "good" causal links that are enforced in the planning process. Also, "bad" causal links are learnt by exploring differences between (different) plans solving a single

planning task while entanglements are learnt by exploring similarities in structures of solution plans of several planning tasks.

3. PRELIMINARIES

This section is devoted to introducing the terminology that will be used throughout the paper.

3.1. Classical Planning

Classical planning is concerned with finding a (partially or totally ordered) sequence of actions transforming the static, deterministic and fully observable environment from the given initial state to a desired goal state (Ghallab et al., 2004; Fox and Long, 2003).

In the classical representation, a *planning task* consists of a *planning domain model* and a *planning problem*, where the planning domain model describes the environment and defines planning operators while the planning problem defines concrete objects, an initial state and a set of goals. The environment is described by *predicates* that are specified via a unique identifier and terms (variable symbols or constants). For example, a predicate at(?t ?p), where at is a unique identifier, and ?t and ?p are variable symbols, denotes that a truck ?t is in a location ?p. Predicates thus capture general relations between objects.

Definition 1: A **planning task** is a pair $\Pi = (Dom_{\Pi}, Prob_{\Pi})$ where a **planning domain model** $Dom_{\Pi} = (P_{\Pi}, Ops_{\Pi})$ is a pair consisting of a finite set of predicates

P_Π and planning operators Ops_Π, and a **planning problem** $Prob_\Pi = (Objs_\Pi, I_\Pi, G_\Pi)$ is a triple consisting of a finite set of objects $Objs_\Pi$, initial state I_Π and goal G_Π.

Let ats_Π be the set of all **atoms** that are formed from the predicates P_Π by substituting the objects $Objs_\Pi$ for the predicates' arguments. In other words, an atom is an **instance** of a predicate (in the rest of the paper when we use the term instance, we mean an instance that is fully *ground*). A **state** is a subset of ats_Π, and the **initial state** I_Π is a distinguished state. The **goal** G_Π is a non-empty subset of ats_Π, and a **goal state** is any state that contains the goal G_Π.

Notice that the semantics of state reflects the full observability of the environment. That is, that for a state s, atoms present in s are assumed to be true in s, while atoms not present in s are assumed to be false in s.

Planning operators are "modifiers" of the environment. They consist of *preconditions*, i.e., what must hold prior an operators' application, and *effects*, i.e., what is changed after operators' application. Specifically, we distinguish between *negative effects*, i.e., what becomes false, and *positive effects*, i.e., what becomes true after operator's application. *Actions* are instances of planning operators, i.e., operators' arguments as well as corresponding variable symbols in operators' preconditions and effects are substituted by objects (constants). Planning operators capture general types of activities that can be performed. Planning operators can be instantiated to actions in order to capture given activities between concrete objects.

Definition 2: A **planning operator** $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{eff}^-(o), \mathit{eff}^+(o))$ is specified such that $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$, where *op_name* is a unique identifier and $x_1, \ldots x_k$ are all the variable symbols (arguments) appearing in the operator,

pre(o) is a set of predicates representing an operator's precondition, *eff*$^-(o)$ and *eff*$^+(o)$ are sets of predicates representing an operator's negative and positive effects. **Actions** are instances of planning operators that are formed by substituting objects, which are defined in a planning problem, for operators' arguments as well as for corresponding variable symbols in operators' preconditions and effects. An action $a = (\textit{pre}(a), \textit{eff}^-(a), \textit{eff}^+(a))$ is **applicable** in a state s if and only if *pre*$(a) \subseteq s$. Application of a in s, if possible, results in a state $(s \setminus \textit{eff}^-(a)) \cup \textit{eff}^+(a)$.

A solution of a planning task is a sequence of actions transforming the environment from the given initial state to a goal state.

Definition 3: A **plan** is a sequence of actions. A plan is a **solution** of a planning task Π, a **solution plan** of Π in other words, if and only if a consecutive application of the actions from the plan starting in the initial state of Π results in the goal state of Π.

Determining equality of predicates (needed for set operations such as intersection) is done such that predicates are *equal* if they have the same name and their arguments (including their order) are identical. So, an expression $p \in X \cap Y$, where X and Y are sets of predicates, means that p has the same name and arguments (in the same order) in both X and Y. A predicate p is a *variant* of a predicate q[4] if by renaming p's variable symbols (arguments) we get a predicate equal to q.

[4]We can also say that p is unifiable with q

3.2. Relations between Actions and Operators

By analysing preconditions and effects of actions or operators we can identify how these influence each other. As discussed in Chapman's earlier work (Chapman, 1987), an action having some atom in its positive effects is a possible achiever of that atom for some other action having that atom in its precondition. The opposite for being a possible achiever is being a possible clobberer (below referred to simply as clobberer) which means that action a_i deletes atom(s) that a_j has in its precondition. Note that being a clobberer refers to the notion of "threat" in plan-space planning (Sacerdoti, 1975).

Definition 4: Let a_i and a_j be actions. We say that a_i **possibly achieves** an atom p for a_j if and only if $p \in \mathit{eff}^+(a_i) \cap \mathit{pre}(a_j)$.
We say that a_i is a **possible clobberer** for a_j if and only if $\mathit{eff}^-(a_i) \cap \mathit{pre}(a_j) \neq \emptyset$.

Notions of a possible achiever and clobberer can be easily extended for planning operators.

Definition 5: Let o_i and o_j be planning operators and p be a predicate. We say that o_i **possibly achieves** a predicate p for o_j if and only if there exist a_i, a_j and p_g, instances of o_i, o_j and p respectively, such that a_i possibly achieves p_g for a_j, i.e., $p_g \in \mathit{eff}^+(a_i) \cap \mathit{pre}(a_j)$. Similarly, we say that o_i is a **possible clobberer** for o_j if and only if there exist a_i, a_j, instances of o_i, o_j respectively, such that $\mathit{eff}^-(a_i) \cap \mathit{pre}(a_j) \neq \emptyset$.

In every solution plan, every atom in a precondition of an action a_j is (necessarily) achieved in the sense that there exists a possible achiever action a_i for the atom

before a_j, and there is no action in between a_i and a_j which deletes the atom (here the initial state can be viewed as the initial action which only adds atoms, and the goal can be viewed as the final action which only has precondition atoms). Notice that being an achiever relates to the notion of "causal link" in plan-space planning.

Definition 6: Let $\langle a_1, a_2, \ldots a_n \rangle$ be a solution plan of some planning task. We say that an action a_i **achieves** an atom p for an action a_j if and only if $i < j$, $p \in \mathit{eff}^+(a_i) \cap \mathit{pre}(a_j)$ and $p \notin \mathit{eff}^-(a_k)$ for every $k \in \{i+1, \ldots, j-1\}$.

Of course, an action can achieve an atom which then appears in preconditions of several following actions. Likewise, several actions can achieve an atom for one action. For the purpose of defining inner entanglements, we have to introduce special cases of the achiever relation. If a_i achieves an atom required by a_j and no action in between them also achieves the atom, then a_i is the primary achiever of the atom. In another case, where an action a_i achieves an atom for another action a_j and no other action in between has that atom in its precondition or its positive effects, we say that a_i firstly achieves the atom required by a_j. If a_i firstly achieves an atom, it follows that it is also the primary achiever of it.

Definition 7: Let $\langle a_1, a_2, \ldots a_n \rangle$ be a solution plan of some planning task. We say that an action a_i **is the primary achiever** of an atom p for an action a_j if and only if a_i achieves p for a_j, and $p \notin \mathit{eff}^+(a_k) \cup \mathit{eff}^-(a_k)$ for every $k \in \{i+1, \ldots, j-1\}$. We also say that an action a_i **firstly achieves** an atom p required by an action a_j if and only if a_i achieves p for a_j, and $p \notin \mathit{eff}^+(a_k) \cup \mathit{pre}(a_k) \cup \mathit{eff}^-(a_k)$ for every $k \in \{i+1, \ldots, j-1\}$.

3.3. BlocksWorld Domain

We briefly introduce the *BlocksWorld* domain (Gupta and Nau, 1992; Slaney and Thiébaux, 2001), which is one of the best known planning domains, that will be used as a running example in the paper.

The BlocksWorld domain describes an environment where we have a finite number of blocks, one table with unlimited space, and one robotic hand. A block can be either stacked on another block, placed on the table or held by the robotic hand. No block can be stacked on more than one block at the same time as well as no more than one block can be stacked on a block at the same time. The robotic hand can hold at most one block. The BlocksWorld domain consists of four operators: pickup(?x) refers to a situation when the robotic hand picks-up a block ?x from the table, putdown(?x) refers to a situation when the robotic hand puts down the block ?x it is holding to the table, unstack(?x ?y) refers to a situation when the robotic hand unstacks a "clear" block ?x from a block ?y, and stack(?x ?y) refers to a situation when the robotic hand stacks the block ?x it is holding to a "clear" block ?y. As mentioned before, planning operators are instantiated by substituting constants (objects) for variable symbols that appear in operators' definition. For example, putdown(?x) can be instantiated by substituting a, which refers to a concrete block "a", for ?x. We then obtain an action putdown(a) that requires the robotic hand to hold the block a, and the effect is that the block a is placed on the table, the block a is clear (no other block is stacked on it), and the hand no longer holds it.

4. INNER ENTANGLEMENTS

Inner Entanglements are relations between pairs of planning operators and predicates. Inner entanglements, informally speaking, represent exclusivity of "achieving" or "requiring" predicates between operators. That is that for a given planning task there exists at least one solution plan where a given inner entanglement holds. In other words, considering that inner entanglement while solving the task will not prune all possible solution plans. Typically, a predicate can be achieved by more than one operator as well as more than one operator might require the same predicate. However, it is often the case that some combinations "achiever-requirer" are not useful.

Specifically, we have two types of Inner Entanglements, *entanglements by succeeding* and *entanglements by preceding*. An entanglement by succeeding represents exclusivity of achievement of a predicate p by an operator o_i for an operator o_j. For a planning task, where such an entanglement holds, there exists a solution plan such that instances of o_i firstly achieve instances of p exclusively only for instances of o_j. An entanglement by preceding, on the other hand, represents exclusivity of requirement of a predicate p by an operator o_j from an operator o_i. For a planning task, where such an entanglement holds, there exists a solution plan such that only instances of o_i are exclusive primary achievers of instances of p for instances of o_j.

For example, in the BlocksWorld domain, it may be observed that operator pickup(?x) possibly achieves predicate holding(?x) for operators stack(?x ?y) and putdown(?x). Similarly, it may be observed that predicate holding(?x) is possibly achieved for operator putdown(?x) by operators unstack(?x ?y) and pickup(?x). We may re-

quire that every instance of pickup(?x) firstly achieves an instance of holding(?x) exclusively for a corresponding instance of stack(?x ?y) since putdown(?x) would just reverse the effects of pickup(?x) (see Figure 1 right). In other words, pickup(?x) is entangled by succeeding stack(?x ?y) with holding(?x). Analogously, we may require that for every instance of putdown(?x), a corresponding instance of unstack(?x ?y) is the exclusive primary achiever of an instance of holding(?x) because, again, putdown(?x) would just reverse the effects of pickup(?x) (see Figure 1 left). In other words, putdown(?x) is entangled by preceding unstack(?x ?y) with holding(?x).

Roughly speaking, inner entanglements provide restrictions to the plan generation process since they allow only some combinations of action sequences while not affecting solvability of considered planning tasks. Whereas the BlocksWorld example (see Figure 1) indicates one possible nature of inner entanglements, in general case, the reason why given inner entanglements hold in a given domain model might vary. Hence, our definition of inner entanglements does not explicitly capture their nature and "maintains" only solvability of considered planning tasks.

We distinguish two variants of them, *strict* and *non-strict*. The strict variant captures the exclusivity of predicate achievement strictly between involved operators, while the non-strict variant allows situations where some instances of the predicates are present in the initial state or can be present in the goal state. For example, if the initial state of some planning task contains an atom holding(a), then the strict version of the above entanglement by preceding prevents applying putdown(a) in the initial state, while the non-strict variant of the entanglement allows to apply putdown(a) in the initial state. Both strict and non-strict variants of

inner entanglements are defined as follows. Notice that we assume that operators o_1 and o_2 share arguments that are relevant to p. For example, pickup(?x) and stack(?x ?y) share the argument ?x, since it is relevant for holding(?x).

[Figure 1 about here.]

Definition 8: Let Π be a planning task. Let o_1 and o_2 be planning operators, p be a predicate (o_1, o_2 and p are defined in the domain model of Π) such that $p \in \mathit{eff}^+(o_1) \cap \mathit{pre}(o_2)$. We say that o_1 is **strictly entangled by succeeding** o_2 with p in Π if and only if there exists a solution plan π of Π and for each $a_1 \in \pi$ being an instance of o_1, there exists $a_2 \in \pi$ being an instance of o_2 such that a_1 firstly achieves an atom p_{gnd}, where p_{gnd} is an instance of p, required by a_2.

We also say that o_2 is **strictly entangled by preceding** o_1 with p in Π if and only if there exists a solution plan π of Π and for each $a_2 \in \pi$ being an instance of o_2, there exists $a_1 \in \pi$ being an instance of o_1 such that a_1 is the primary achiever of an atom p_{gnd}, where p_{gnd} is an instance of p, for a_2.

Henceforth, strict entanglements by preceding and succeeding are denoted as **strict inner entanglements**.

Definition 9: Let Π be a planning task. Let o_1 and o_2 be planning operators and p be a predicate (o_1, o_2 and p are defined in the planning domain model of Π) such that $p \in \mathit{eff}^+(o_1) \cap \mathit{pre}(o_2)$. We say that o_1 is **non-strictly entangled by succeeding** o_2 with p in Π if and only if there exists a solution plan π of Π and for every $a_1, a_2 \in \pi$ such that a_1 firstly achieves an atom p_{gnd}, where p_{gnd} is an instance of p, required by a_2, it holds that if a_1 is an instance of o_1 then a_2 is an instance of o_2.

We also say that o_2 is **non-strictly entangled by preceding** o_1 with p in Π if and

only if there exists a solution plan π of Π and for every $a_1, a_2 \in \pi$ such that a_1 is the primary achiever of an atom p_{gnd}, where p_{gnd} is an instance of p, for a_2, it holds that if a_2 is an instance of o_2 then a_1 is an instance of o_1.
Henceforth, non-strict entanglements by preceding and succeeding are denoted as **non-strict inner entanglements**.

Inner entanglements (both strict and non-strict) can be used for pruning some unpromising alternatives in the search space, in other words, reducing the branching factor. Notice that a predicate involved in some inner entanglement relation might be true for some time after it is achieved, in other words, the predicate does not have to be 'used' immediately after being achieved. Since the previous example of BlocksWorld might be confusing in this sense (the predicate holding(?x) is immediately 'used' after being achieved), we provide another example in a modification of the BlocksWorld domain that considers more than one robotic hand. Let pickup(?h ?x) be strictly entangled by succeeding stack(?h ?x ?y) with holding(?h ?x) in some planning task. If action pickup(h1 a) in applied at step i, then action stack(h1 a ?y) (any other block than a can be substituted for ?y) must be applied at step j such that $j > i$. The entanglement prohibits applying action putdown(h1 a) at step k such that $i < k < j$. On the other hand, other actions that utilises different robotic hands than h1 can be applied in between i-th and j-th step.

A single inner entanglement requires only existence of one solution plan of the given planning task where the entanglement conditions are met. However, different entanglements might hold in different solution plans. To consider multiple (different) inner entanglements rather than a single one, there must exist a solution plan in

which all considered entanglements hold. Also, in practice, inner entanglements are domain- or class of problems-specific rather than problem-specific. The above definition can be extended to reflect these aspects.

Definition 10: Let Π be a planning task. Let ENT_Π be a set of inner entanglements, where each element of ENT_Π is specified by a type of the inner entanglement relation and involved pair of planning operators and predicate. We say that a set of inner entanglements ENT_Π holds for Π if and only if there exists a solution plan of Π in which all the entanglements from ENT_Π hold.
Similarly, $ENT_\mathcal{P}$ holds for a set of planning tasks $\mathcal{P}$ sharing the same planning domain model if and only if $ENT_\mathcal{P} = \bigcap_{\Pi \in \mathcal{P}} ENT_\Pi$.

Both the BlocksWorld related entanglements hold for every BlocksWorld planning tasks. By adding two more inner entanglements, namely, unstack(?x ?y) to be (strictly) entangled by succeeding putdown(?X) and stack(?x ?y) to be (strictly) entangled by preceding pickup(?X), we restrict to solution plans where blocks are always put down to the table after being unstacked from other blocks and, eventually, picked up from the table and stacked on some other blocks. This might be useful since it introduces more restriction on decisions the planner has to take during the search. With unlimited table space, these inner entanglements hold for every task.

5. REFORMULATING PLANNING TASKS

To exploit inner entanglements during the planning process we have to develop a specific planner, modify an existing one, or we have to reformulate a planning task in such a way that the entanglements hold in every solution plan retrieved by a

planner. The last option is planner-independent: in fact, it involves the reformulation of domain and problem models using features of the PDDL (actually, STRIPS) language (see Section 3).

Hence, after inner entanglements are identified, we encode them directly into the planning task. The reformulated planning task is passed to a generic planning engine in order to generate a solution plan, which is also a solution plan of the original planning task. Encoding of inner entanglements as we show in this section prevents planning engines to explore branches of the search space that violate these entanglements. In other words, reformulated tasks "narrow" the search space for planning engines for improving their performance.

[Figure 2 about here.]

Encoding inner entanglements is done by introducing supplementary predicates, 'locks', that ensure that we cannot apply certain instances of operators in some stage of the planning process in order to enforce inner entanglements. Let Π be a planning task and Ops be the set of operators defined in the domain model of Π. Let an operator $o_1 \in Ops$ be (strictly or non-strictly) entangled by a succeeding operator $o_2 \in Ops$ with a predicate p (defined in the domain model of Π) in Π. Then Π is reformulated as follows:

(1) Create a predicate p' (not defined in the domain model of Π) having the same arguments as p and add p' to the domain model of Π.
(2) Modify the operator o_1 by adding p' into its negative effects. p' has the same arguments as p which is in the positive effects of o_1.

(3) Modify the operator o_2 by adding p' into its positive effects. p' has the same arguments as p which is in the precondition of o_2.

(4) Modify all operators $o \in Ops$ such that $o \neq o_2$ and having a variant of p in $\mathit{pre}(o)$ by adding p' into its precondition. p' has the same arguments as the variant of p.

(5) Modify all operators $o \in Ops$ such that $o \neq o_1$ and having a variant of p in $\mathit{eff}^{+}(o)$ by adding p' into its positive effects. p' has the same arguments as the variant of p.

(6) Add all possible instances of p' into the initial state of Π and if the entanglement is strict, then also to the goal of Π.

Figure 2 depicts the BlocksWorld operators encoding an entanglement by succeeding, concretely that pick-up(?x) is (strictly) entangled by succeeding stack(?x ?y) with holding(?x). In our terminology, pick-up(?x) refers to o_1, stack(?x ?y) to o_2, holding(?x) to p, and pick-up_stack_succ_holding(?x) to p'. Correctness of the reformulation is proved as follows.

Proposition 1: Let Π be a planning task, Ops be the set of planning operators defined in the domain model of Π. Let $o_1, o_2 \in Ops$ be planning operators and p be a predicate (p is defined in the domain model of Π) such that o_1 is strictly (resp. non-strictly) entangled by succeeding o_2 with p in Π. Let Π' be a planning task obtained by reformulating Π using the previous approach. π' is a solution plan of Π' if and only if π' is a solution plan of Π that satisfies the entanglement conditions (see Definitions 8 and 9).

Proof. Hereinafter, the modified operators o_1, o_2 will be denoted as o_1', o_2'. The strict entanglement by succeeding (see Definition 8) says that if an instance of o_1 that

achieves an atom p_{gnd} that is an instance of p is applied at step i and a corresponding instance of o_2 that requires p_{gnd} is applied at step j, or never in case of the non-strict entanglement, so $j = \infty$, then no corresponding instance of any operator other than o_2 having p_{gnd} in its precondition can be applied at step k unless p_{gnd} is re-achieved by any operator different than o_1 at step l. Formally speaking, $j > i$ and if $i < k < j$, then $i < l < k < j$.

Applying an instance of o'_1 results in removing an atom p'_{gnd} that is an instance of p' having the same arguments as p_{gnd} (notice that all the possible instances of p' are present in the initial state of Π'). From step 4 of the reformulation, p' is put into the precondition of any operator that has p in its precondition (both p and p' has the same arguments) except o_2. Hence, only instances of o'_2 having p_{gnd} in its precondition can be applied, since actions having p_{gnd} in their precondition that are not instances of o_2 have p'_{gnd} in their preconditions as well. If o'_2 is applied or p is re-achieved by any other (modified) operator than o'_1 (see step 5 of the reformulation), then a corresponding instance of p' is re-achieved as well.

For the strict version of the entanglement, all the instances of p' must be present in the goal state, so o'_2 must be applied at some point after o'_1. For the non-strict version of the entanglement, there is no need to re-achieve all the instances of p', so o'_2 does not have to be applied at some point after o'_1 in order to "use" the corresponding instance of p achieved by o'_1, however, no other operator can "use" it.

Straightforwardly, if π' is a solution plan of Π', then π' is a solution plan of Π that satisfies the entanglement conditions. The provided reformulation prevents only application of operators in $Ops \setminus \{o_2\}$ having p in their precondition after o_1 achieved

p. Therefore, if π' is a solution plan of Π that satisfies the entanglement conditions, then π' is a solution plan of Π'. □

[Figure 3 about here.]

Similarly, we use supplementary predicates, 'locks', to enforce entanglements by preceding. Let Π be a planning task and Ops be the set of operators defined in the domain model of Π. Let an operator $o_2 \in Ops$ is (strictly or non-strictly) entangled by a preceding operator $o_1 \in Ops$ with a predicate p (defined in the domain model of Π) in Π. Then Π is reformulated as follows:

(1) Create a predicate p' (not defined in the domain model of Π) having the same arguments as p and add p' to the domain model of Π.
(2) Modify the operator o_1 by adding p' into its positive effects. p' has the same arguments as p which is in the positive effects of o_1.
(3) Modify the operator o_2 by adding p' into its precondition. p' has the same arguments as p which is in the precondition of o_2.
(4) Modify operators $o \in Ops$ such that $o \neq o_1$ and having a variant of p in $\mathit{eff}^+(o)$ by adding p' into its negative effects (p' has the same arguments as the variant of p).
(5) If the entanglement is non-strict, then add all possible instances of p' to the initial state of Π.

Figure 3 depicts the BlocksWorld operators encoding an entanglement by preceding, concretely that put-down(?x) is (strictly) entangled by preceding unstack(?x ?y) with holding(?x). In our terminology, put-down(?x) refers to o_2, unstack(?x

?y) to o_1, holding(?x) to p, and put-down_unstack_prec_holding(?x) to p'. Correctness of the reformulation is proved as follows.

Proposition 2: Let Π be a planning task, Ops be the set of planning operators defined in the domain model of Π. Let $o_1, o_2 \in Ops$ be planning operators and p be a predicate (p is defined in the domain model of Π) such that o_2 is strictly (resp. non-strictly) entangled by preceding o_1 with p in Π. Let Π' be a planning task obtained by reformulating Π using the previous approach. π' is a solution plan of Π' if and only if π' is a solution plan of Π that satisfies the entanglement conditions (see Definitions 8 and 9).

Proof. Hereinafter, the modified operators o_1, o_2 will be denoted as o'_1, o'_2. The strict version entanglement by preceding (see Definition 8) says that if an instance of o_2 requiring an atom p_{gnd} that is an instance of p is applied at step j and a corresponding instance of o_1 is applied at step i achieving p_{gnd} ($i < j$), then no corresponding instance of any operator other than o_2 having p_{gnd} in its positive effects can be applied at step k such that $i < k < j$.

Adding p' into o_2's precondition results in the situation that any instance of o'_2 can be applied only after the corresponding instance of o'_1 since p' is in o'_1's positive effects. In particular, an instance of o'_1 that achieves an atom p_{gnd} (an instance of p) achieves also p'_{gnd} that is an instance of p' having the same arguments as p_{gnd}. The instance of o'_2 that requires p_{gnd} requires p'_{gnd} as well. If p_{gnd} is re-achieved by instance of other (modified) operator than o'_1, then p'_{gnd} is removed (step 4 of the reformulation). Then, o'_2 requiring p_{gnd} cannot be applied, since p'_{gnd} will not be true.

For the strict version of the entanglement, no instance of p' is present in the initial

state, so o'_1 must be applied at some point before o'_2. For the non-strict version of the entanglement all the instances of p' are present in the initial state, so o'_2 does not have be applied after o'_1, however, no other (modified) operator can re-achieve an instance p in between, since otherwise the corresponding instance of p' is removed.

Straightforwardly, if π' is a solution plan of Π', then π' is a solution plan of Π that satisfies the entanglement conditions. The provided reformulation prevents only application of o_2 having p in their precondition unless o_1 achieved p. Therefore, if π' is a solution plan of Π that satisfies the entanglement conditions, then π' is a solution plan of Π'. □

[Figure 4 about here.]

There are also situations where both the (strict) entanglements by preceding and succeeding hold for operators o_1, o_2 and a predicate p. Of course, we can reformulate the problem according to previous reformulation approaches. On the other hand, it requires two supplementary predicates and thus the process might not be very efficient. Given the fact that exclusivity of achievement and requirement of p is mutual between o_1 and o_2, we can replace p by its "twin" in the positive effects of o_1 and the precondition of o_2. Therefore, we introduce a more compact reformulation that exploits such a property.

Formally, let Π be a planning task and Ops be the set of operators defined in the domain model of Π. Let $o_1 \in Ops$ be non-strictly entangled by succeeding $o_2 \in Ops$ with p (p is defined in the domain model of Π) in Π, and o_2 be strictly entangled by preceding o_1 with p in Π[5]. Then Π is reformulated as follows:

[5]The non-strict entanglement by succeeding, a weaker form of the strict entanglement by succeeding, is required for correct

(1) Create a predicate p' (not defined in the domain model of Π) having the same arguments as p and add p' to the domain model of Π.

(2) Modify the operator o_1 by replacing p by p' in o_1's positive effects and adding p into o_1's negative effects.

(3) Modify the operator o_2 by replacing p by p' in o_2's precondition and (possibly) negative effects.

(4) Modify all operators $o \in Ops$ such that $o \neq o_1$ and having a variant of p in $\mathit{eff}^+(o)$ by adding p' into its negative effects (p' has the same arguments as the variant of p).

Figure 4 depicts the BlocksWorld operators encoding both entanglements by preceding and succeeding, concretely that pick-up(?x) is non-strictly entangled by succeeding stack(?x ?y) with holding(?x) and stack(?x ?y) is strictly entangled by preceding pick-up(?x) with holding(?x). In our terminology, pick-up(?x) refers to o_1, stack(?x ?y) to o_2, holding(?x) to p, and stack_pick-up_both_holding(?x) to p'. Correctness of the reformulation is proved as follows.

Proposition 3: Let Π be a planning task, Ops be the set of planning operators defined in the domain model of Π. Let $o_1, o_2 \in Ops$ be planning operators and p be a predicate (p is defined in the domain model of Π) such that o_2 is strictly entangled by preceding o_1 with p in Π and o_1 is non-strictly entangled by succeeding o_2 with p in Π such that both entanglements are compatible. Let Π' be a planning task obtained by reformulating Π using the previous approach. π' is a solution plan of Π' if and

capture of the entanglements by the encoding. Enforcing the strict entanglement by succeeding would require more complex encoding mitigating benefits of the introduced compact encoding.

only if π' is a solution plan of Π that satisfies the conditions of both entanglements (see Definitions 8 and 9).

Proof. Hereinafter, the modified operators o_1, o_2 will be denoted as o_1', o_2'. If both the entanglements are strict (see Definition 8), then it says that if an instance of o_1 achieving an atom p_{gnd} (an instance of p) is applied at step i and a corresponding instance of o_2 requiring p_{gnd} is applied at step j ($j > i$), then no corresponding instance of any operator other than o_1 or o_2 having p_{gnd} in its precondition or positive effects can be applied at step k such that $i < k < j$.

We can observe that o_1' is the only operator achieving p' but no longer achieving p. Similarly, o_2' is the only operator requiring p' (having it in the precondition) but no longer requiring p. The entanglement by preceding cannot be affected by applying any (modified) operator o achieving p, since it removes p' as well (see step 4 of the reformulation) and thus make o_2' inapplicable. Similarly, if p is true before applying o_1', it is removed after o_1' is applied and hence any operator requiring p becomes inapplicable. The strict entanglement by preceding is met since no instance of p' is in the initial state of P'. There is no restriction that prevents occurrences of p' in any of the goal states. Therefore, the entanglement by succeeding is non-strict.

Hence, if π' is a solution plan of Π', then π' is a solution plan of Π that satisfies the conditions of both entanglements. The provided reformulation prevents only application of o_2 having p in their precondition unless o_1 achieved p as well as application of any operator other than o_2 having p in its precondition after o_1 achieved p. Therefore, if π' is a solution plan of Π that satisfies the entanglement conditions, then π' is a solution plan of Π'. □

6. THEORETICAL FOUNDATIONS OF INNER ENTANGLEMENTS

This section is devoted to theoretical properties of inner entanglements such as complexity results as well as their expected impact on planners.

6.1. Landmark Theory

Landmark theory (Hoffmann et al., 2004) is a useful framework for studying structures of planning tasks. We will use a fragment of the landmark theory to prove intractability (PSPACE-completeness) of deciding whether a given inner entanglement holds. The notions we will use are briefly introduced in the following lines, for more details see (Hoffmann et al., 2004).

Landmarks are atoms which must be achieved at some point in every solution plan of a given planning task. Deciding whether atoms are landmarks is PSPACE-complete (Hoffmann et al., 2004).

Ordering landmarks is useful for computing heuristics (Richter and Westphal, 2010). Landmarks p and q are *greedily necessarily ordered* (we denote it as $p \rightarrow_g q$) if for every solution plan of a given planning task, p is achieved before q is achieved for the first time. Deciding greedy necessary ordering of landmarks is also PSPACE-complete (Hoffmann et al., 2004).

6.2. Intractability of Entanglements

The intractability (PSPACE-completeness) of deciding whether a given inner entanglement holds in a given task is proved by the following theorem.

Theorem 1: Let Π' be a planning task, $o_{p'}$ and $o_{q'}$ be planning operators and p'' a

predicate defined in the domain model of Π'. The problem of deciding whether $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p'' in Π' is PSPACE-complete. The problem of deciding whether $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p'' in Π' is also PSPACE-complete.

Proof. First, we show that the problem of deciding whether $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p in Π' as well as the problem of deciding whether $o_{q'}$ is strictly entangled by preceding $o_{q'}$ with p in Π' belongs to the PSPACE class. To do this, we reformulate Π' by encoding the given inner entanglement as described in Section 5. Hence, the decision problem of whether the given inner entanglement holds can be encoded as a planning task, i.e., the entanglement holds if and only if the reformulated task is solvable. We know we can solve planning tasks in polynomial space, hence this decision problem belongs to PSPACE.

We reduce, in polynomial time, the problem of deciding whether landmarks p and q are greedily necessarily ordered, i.e., $p \rightarrow_g q$, in some planning task Π, which is PSPACE-complete, to the problem of deciding strict entanglements by succeeding or preceding between $o_{p'}$, $o_{q'}$ and p in Π'. Without loss of generality, we assume that p and q are nullary predicates (atoms) defined in the domain model of Π.

We create a planning task Π' by modifying Π as follows. Let Ops be the set of planning operators defined in the domain model of Π. Let $Ops_p = \{o \mid o \in Ops, p \in \mathit{eff}^+(o)\}$ be the set of operators achieving p and $Ops_q = \{o \mid o \in Ops, q \in \mathit{eff}^+(o)\}$ be the set of operators achieving q. We extend the domain model of Π by adding atoms (nullary predicates) r, p', p'', q' and q'' (without loss of generality we assume that none of these is defined in the domain model of Π). Then, we add r

into preconditions of every operator from Ops. Then, we modify operators in Ops_p and Ops_q as follows. For every $o \in Ops_p$: replace p by p' in $\mathit{eff}^+(o)$ and add r into $\mathit{eff}^-(o)$. For every $o \in Ops_q$: add q' into $\mathit{eff}^+(o)$ and add q'' into $\mathit{eff}^-(o)$. The initial state I of Π is modified as follows. If $p \in I$, then replace p by p'. If $p \notin I$, then add r. If $q \in I$, then add q', otherwise add q'' (if $q \notin I$). Notice that q' becomes and remains true when q has been achieved and q'' is true only before q is achieved (if q is true in the initial state, q'' is never true).

For the strict entanglements by succeeding case, we introduce the following operators (without loss of generality we assume that none of the operators is defined in the domain model of Π), i.e, $o_{p'} = (name(o_{p'}), \{p'\}, \{p'\}, \{p''\})$, $o_{q'} = (name(o_{q'}), \{p'', q'\}, \{p''\}, \{p, r\})$ and $o_{q''} = (name(o_{q''}), \{p'', q''\}, \{p''\}, \{p, r\})$, and add them into the domain model of Π. Notice that $name(o_{p'}), name(o_{q'})$ and $name(o_{q''})$ contain only unique operator identifiers (and no variable symbols). We can observe that if $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p'' in Π' (the modification of Π), then q must be true before or at the same time when p is achieved. This is, because q' becomes true after q is achieved (as mentioned before), and according to the entanglement there is a solution plan π' of Π' such that $o_{p'}$ always achieves p'' for $o_{q'}$. Removing instances of $o_{p'}$ and $o_{q'}$ from π' gives us a plan π which is a solution plan of Π. Given the modification of all operators from Ops_p, p becomes true in π in the same time as p' becomes true and r becomes false in π'. Then, only $o_{p'}$ and $o_{q'}$ can be applied (in this order) in π', because other operators have r in their preconditions, and r can be re-achieved by $o_{q'}$. From this, we can get that q' must be achieved before $o_{p'}$ is applied in π'. Therefore, q is achieved before or in the same time as p in π, which is a solution plan of Π, and thus $p \rightarrow_g q$ does

not hold in Π. Hence, $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p'' in Π' (the modification of Π) if and only if $p \rightarrow_g q$ does not hold in Π.

For the strict entanglements by preceding case, we introduce the following operators (without loss of generality we assume that none of the operators is defined in the domain model of Π), i.e, $o_{p'} = (name(o_{p'}), \{p', q'\}, \{p'\}, \{p''\})$, $o_{p''} = (name(o_{p''}), \{p', q''\}, \{p'\}, \{p''\})$, and $o_{q'} = (name(o_{q'}), \{p''\}, \{p''\}, \{p, r\})$ and add them into the domain model of Π. Notice that $name(o_{p'}), name(o_{p''})$ and $name(o_{q'})$ contain only unique operator identifiers (and no variable symbols). Analogously to the previous case, we can observe that if $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p'' in Π' (the modification of Π), then q must be true before or at the same time as p is achieved. Therefore, there exists π', a solution plan of Π' where the entanglement holds. Again, removing instances of $o_{p'}$ and $o_{q'}$ from π' gives us a plan π which is a solution plan of Π. Analogously to the previous case, after a modified operator from Ops_p is applied in π', then only $o_{p'}$ and $o_{q'}$ (in this order) can be applied before any other operator can be applied. Therefore, q' must be achieved before $o_{p'}$ is applied in π' and thus q is achieved before or in the same time as p in π, so $p \rightarrow_g q$ does not hold in Π. Hence, $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p'' in Π' (the modification of Π) if and only if $p \rightarrow_g q$ does not hold in Π.

Clearly, modification of Π in both cases is done in polynomial time. Hence, since the problem of deciding whether landmarks p and q are greedily necessarily ordered in Π is PSPACE-complete, the problem deciding whether $o_{p'}$ is strictly entangled by succeeding $o_{q'}$ with p in Π' as well as the problem deciding whether $o_{q'}$ is strictly entangled by preceding $o_{p'}$ with p in Π', where both problems belong to PSPACE, is PSPACE-complete as well.

□

Corollary 1: Let Π' be a planning task, $o_{p'}$ and $o_{q'}$ be planning operators and p'' a predicate defined in the domain model of Π'. The problem of deciding whether $o_{p'}$ is non-strictly entangled by succeeding $o_{q'}$ with p'' in Π' is PSPACE-complete. The problem of deciding whether $o_{q'}$ is non-strictly entangled by preceding $o_{p'}$ with p'' in Π' is also PSPACE-complete.

Proof. The problem of deciding of either of the non-strict inner entanglements can be encoded as a planning task (Π' is reformulated as described in Section 5), so it belongs to PSPACE. Since the strict version of inner entanglements are special cases of the non-strict version, the problem is PSPACE-complete. □

Intractability of deciding whether a single entanglement holds for a given planning task implies intractability of deciding whether a set of inner entanglements holds for that task.

Corollary 2: Let e_1 and e_2 be inner entanglements that hold in a planning task Π. The problem of deciding whether a set $\{e_1, e_2\}$ holds in Π is PSPACE-complete.

Proof. Without loss of generality, let Π_{e_1} be a planning task obtained by reformulating Π considering e_1. Then, the problem of deciding whether $\{e_1, e_2\}$ holds in Π is equivalent to the problem of deciding whether e_2 holds in Π_{e_1} which is PSPACE-complete. □

The presented theoretical results say that deciding whether a set of inner entanglements holds in a planning task is (theoretically) as hard as solving the task. Hence, in order to benefit from inner entanglements we have to spend (much) less time on

their generation than how much time we can save by their use. Learning them from simple planning tasks is a viable option, since such tasks can usually be solved and analyzed very quickly.

6.3. Trivial Entanglements

Despite the complexity results, there are some cases where we can trivially identify inner entanglements (hereinafter referred as trivial inner entanglements). The following situations refer to special cases where there is no way to violate inner entanglements in the planning process. However, trivial inner entanglements do not provide any new domain-specific information and hence we do not have to consider them in the reformulation.

We can observe that having only one achiever or ‘requirer‘ of some predicate trivially satisfy the conditions of exclusivity. In other words, if only one operator achieves a certain predicate, then it is its exclusive achiever for all the operators that require this predicate. Similarly, if only one operator requires a certain predicate, then it is its exclusive ‘requirer‘ from all the operators that achieve this predicate.

Lemma 1: Let Π be a planning task, Ops be the set of planning operators and p be a predicate defined in the domain model of Π. If there exists exactly one $o_i \in Ops$ such that $p \in \mathit{eff}^{+}(o_i)$, then for every $o_k \in Ops$ such that $p \in \mathit{pre}(o_k)$ it holds that o_k is non-strictly entangled by preceding o_i with p in Π.

Lemma 2: Let Π be a planning task, Ops be the set of planning operators and p be a predicate defined in the domain model of Π. If there exists exactly one $o_i \in Ops$

such that $p \in \mathit{pre}(o_i)$, then for every $o_k \in Ops$ such that $p \in \mathit{eff}^+(o_k)$ it holds that o_k is non-strictly entangled by succeeding o_i with p in Π.

6.4. Identifying Inner Entanglements: Case Studies

This section is devoted to investigate identifying and (re)using inner entanglements from knowledge engineering perspective. Whereas it is usually feasible to consider inner entanglements as domain-specific rather than task-specific, even small modifications in domain models can invalidate some of the entanglements and, possibly, introduce some other entanglements.

An illustrative example we used earlier in the text identified two inner entanglements in the BlocksWorld domain, i.e., the operator putdown is entangled by the preceding operator unstack with the predicate holding and the operator pickup is entangled by the succeeding operator stack with the predicate holding. Whether the entanglements are strict or non-strict depends on whether a block is initially held by the robotic hand or whether the same is required in the goal state. The entanglements in fact prevent applying the operators pickup and putdown consecutively since they just reverse each other's effects, thus doing so is clearly meaningless. Extending the BlocksWorld domain by introducing an operator paint which paints the block while it is held by the robotic hand might invalidate the entanglements in some cases. pickup can then achieve holding for both stack and paint, so the exclusivity required by the entanglement is not met. putdown can meaningfully use the predicate holding achieved by pickup since we can paint the block (apply the paint operator) in between, so the entanglement by preceding might not be met.

The Depots domain is a combination of the BlocksWorld domain and the Logis-

tics domain such that crates are arranged in stacks and operated by hoists in the same way as blocks in BlocksWorld but can be also transported by trucks between different locations. The lift and drop operators correspond with the BlocksWorld's unstack and stack operators respectively. The load and unload operators are variants of the BlocksWorld's putdown and pickup operators such that instead of putting to and picking up crates from the table they load crates to or unload crates from trucks respectively. In Depots, we may observe, for instance, that the operator lift is entangled by the succeeding operator load with the predicate lifting, load is entangled by preceding lift with lifting, the operator drop is entangled by the preceding operator unload with lifting and unload is entangled by succeeding drop with lifting. If no instance of lifting is present in the initial state or the goal, then the entanglements are strict. If there is no truck defined in the problem, then we cannot apply load or unload, hence the entanglements do not hold (otherwise it will not be possible to apply lift and drop consecutively). Modifying the domain model in such a way that particular trucks can move only between some locations might introduce the necessity of reloading crates from one truck to another. This will certainly affect two of the entanglements, in particular, load will no longer entangled by preceding lift with lifting and unload will no longer entangled by succeeding drop with lifting. However, tasks in which some crate(s) have to be reloaded can be easily identified.

It should be noted that the aforementioned examples indicate that the nature of inner entanglements varies per domain model. Therefore, it seems, in our opinion, that refining not very restrictive general rules for identifying inner entanglements might not be a feasible option. Domain model engineers can either identify inner

entanglements by hand, or exploit our method based on the "learning in planning" paradigm that is presented in Section 7.

6.5. Expected Impact of Inner Entanglements on the Planning Process

Inner entanglements eliminate unpromising alternatives in the search space which reduces the branching factor in search. Introducing supplementary predicates required for encoding inner entanglements, however, introduces additional facts (atoms) planners have to deal with during search and, moreover, memory requirements might therefore be higher. Hence, the impact of inner entanglements is determined by considering whether the potential benefits of reducing the branching factor outweigh overheads caused by handling supplementary predicates. An analogy can be seen in determining whether a macro-operator is useful, in literature also referred as a *utility problem* (Minton, 1988).

Taking a closer look on how inner entanglements are encoded provides insights into how they may influence delete-relaxed heuristics, which is a common technique used in planning engines. Having an operator o_2 strictly entangled by a preceding operator o_1 with a predicate p captures a situation where an instance of o_2 can be applied only if a corresponding instance of p is achieved by an instance of o_1. This is enforced by putting a supplementary predicate p' into o_1's positive effects and into o_2's precondition. In delete-relaxed plans, o_1 must be also applied at some point before o_2. However, an operator $o \neq o_2$ achieving p (and thus removing p') can be placed in between o_1 and o_2 in delete-relaxed plans which does not correspond with the entanglement conditions. Entanglements by preceding are therefore only partially taken into account while computing delete-relaxed heuristics. Having an

operator o_1 strictly entangled by a succeeding operator o_2 with a predicate p captures a situation where an instance of o_1 achieves a corresponding instance of p for an instance of o_2. This is enforced by putting a supplementary predicate p' into o_1's negative effects and into preconditions of operators other than o_2 that have p in their preconditions. However, in delete-relaxed plans, applying o_1 does not prevent applying any other operator having p in its precondition. Therefore, entanglements by succeeding are not taken into account while computing delete-relaxed heuristics. Intuitively, only entanglements by preceding might be beneficial on planners based on delete-relaxed heuristics (e.g. FF).

However, recent empirical results do not confirm this intuition by showing that in some cases entanglements by succeeding can be very beneficial even for planners based on delete-relaxed heuristics (Chrpa and Vallati, 2013). To understand potential benefits of entanglements by succeeding we have to take a different view. A heuristic may suggest to apply an operator $o \neq o_2$ requiring p from o_1. However, after actual application of o_1 it will become impossible to apply o (due to the entanglement conditions), since o_2 will be enforced. Although it might cause planners to be 'trapped' in a local minimum of the heuristics, it might also prevent planners to get into 'deeper' local minima which might eventually happen if o is applied instead of o_2.

If both (strict) entanglements by preceding and succeeding hold between o_1, o_2 and p, the compact encoding involves replacing p with p' in o_1's positive effects and o_2's precondition. In delete-relaxed plans, o_2 cannot be applied unless o_1 is, which is similar to the entanglements by preceding case, and, moreover, o_1 cannot achieve p for any other operator than o_2 (because p is replaced by p'). Although as in the entanglements by preceding case an operator achieving p can be placed in between

o_1 and o_2 in delete-relaxed plans, which does not correspond to the entanglements conditions, both the entanglements are taken into account to reasonable extent while computing delete-relaxed heuristics.

The compact encoding (when both entanglements by preceding and succeeding hold between a pair of operators and a predicate) is intuitively beneficial for planners. The potential impact of inner entanglements seems to be correlated with the shape of search space, in other words, whether inner entanglements can prevent planners to end up in undesirable states (e.g dead-ends, "deep" local minima). We believe that maximising sets of compatible inner entanglements does not imply maximising planners' performance, because some of the entanglements might in fact have a negative impact, for instance, by introducing supplementary predicates planners might deal with or introducing local minima in the heuristics landscape. Possible examples of "bad" inner entanglements are those that consist of operators whose instances appear sporadically in plans, because such inner entanglements bring only a little information for possibly high overheads. Also, if an inner entanglement prunes only a few alternatives, then overheads introduced with it might be higher than its possible benefit. For example, after picking up a block, we might either put it down or stack it on some other clear block. Clearly, the number of clear blocks might be up to $n - 1$, where n is the number of all blocks. If pickup(?x) is (strictly) entangled by succeeding stack(?x ?y) with holding(?x), then we cannot apply putdown(?x) after pickup(?x). Hence, we prune one alternative, keeping $n - 1$ alternatives in the worst case. Similarly, after unstacking a block from another block we can either put it down or stack it on some clear block. If putdown(?x) is (strictly) entangled by preceding unstack(?x ?y), then we cannot apply stack(?x ?z) after unstack(?x

?y). Hence, we keep only one alternative, pruning $n-1$ alternatives in the best case. Given this observation, the latter entanglement is much more informative than the former one. Intuitively, the former entanglement is not helpful and very likely will worsen the planning process. The latter entanglement, on the other hand, seems to be helpful and should improve the planning process.

7. EXTRACTING INNER ENTANGLEMENTS

Algorithm 1 Checking how many times a given operator achieves (requires) a predicate to (from) another operator in the training plans.

1: initialize_ent_arrays(); ▷ create empty arrays $entP, entS$ of size [Ops,Ops,Preds]
2: initialize_op_counter(); ▷ create an empty array $counter$ of size [Ops]
3: **for each** training plan $\pi = \langle a_1, \ldots a_n \rangle$ **do**
4: **for** $i := 1$ to n **do**
5: **for each** $p \in pre(a_i)$ **do**
6: $a := last\_achiever(p, \langle a_1 \ldots a_{i-1} \rangle)$;
7: **if** $a \neq NULL$ **then**
8: $entP[is\_inst(a_i), is\_inst(a), is\_inst(p)] + +$;
9: $entS[is\_inst(a), is\_inst(a_i), is\_inst(p)] + +$;
10: **end if**
11: **end for**
12: $counter[is\_inst(a_i)] + +$;
13: **end for**
14: **end for**

Deciding whether a given set of inner entanglements holds in a given task is generally PSPACE-complete (as discussed in Section 6). Moreover, trivial entanglements (see Section 6.3) are not informative and thus not considered for task reformulation. Therefore, we have to devise an effective approximation technique for extracting sets of inner entanglements. We assume that tasks having the same domain model have a similar structure, so the same set of inner entanglements holds in all of them. Hence, we can select a representative set of simple tasks for each domain model as training tasks, so those can be solved easily by standard planning engines. Generated training plans, that is the solutions of these training tasks, are then explored in order to find what inner entanglements hold in them.

The above approach can be formalised as follows. Let $\mathcal{P}$ be a class of planning tasks that has the same domain model. Let $\mathcal{P}_T \subset \mathcal{P}$ be a set of training tasks. In our approximation method, we assume that $ENT_{\mathcal{P}_T} = ENT_{\mathcal{P}}$, in other words, a set of inner entanglements holding on training planning tasks also holds on the whole class of planning tasks. This assumption is, of course, a source of incompleteness, since enforcing incorrect entanglements may cause some tasks becoming unsolvable. On the other hand, planning tasks having the same domain model are of similar structure (e.g. they differ only by number of objects), which is the case of the most of IPC benchmarks. Hence, we believe that selecting a small set of these tasks such that selected tasks are easy but not trivial can alleviate the incompleteness issue and thus support the assumption. Our empirical study that also explores these issues is provided in Section 8.

The method for extracting inner entanglements from (training) plans works as follows. For every action we check which actions achieved atoms for it or vice versa.

This information is used to determine the cases where exclusivity of predicate's achievement or requirement between a pair of operators applies. This concept is elaborated in Algorithm 1. For this purpose, we define an array $counter$, which stores information about how many instances of given operators occur in the training plans, 3D arrays $entP, entS$, which count how many times a given operator achieves/requires a predicate to/from another operator. Function $is\_inst(arg)$ returns either an operator if arg (action) is an instance of it or a predicate if arg (atom) is an instance of it. Function $last\_achiever(p, \langle a_1, \ldots, a_k \rangle)$ returns the last action in the sequence ($\langle a_1, \ldots, a_k \rangle$) that has p in its positive effects, or NULL if no such action exists (i.e., p is an initial atom).

Algorithm 1 requires linear time with respect to the lengths of given training plans if the number of atoms in actions' preconditions and effects is much lower than lengths of training plans, so it can be bounded by a constant. Notice that information retrieved by the $last\_achiever$ function can be stored in a hash table, hence constant time is needed.

7.1. Flaw Ratio

From Algorithm 1, it is easy to determine whether a given set of inner entanglements holds in all the training plans. However, it is often not a very efficient way to determine a useful set of inner entanglements. There are two main reasons. Firstly, training plans might contain redundant actions or very sub-optimal sub-plans which can prevent detecting some useful entanglements. Secondly, there might be several strategies how a task can be solved, where only some of these lead into discovery of some useful entanglements. For example, in BlocksWorld, we might "put aside"

blocks in two different ways: put them on the table, or stack them on other blocks. Only the former way leads to the discovery of two useful inner entanglements, i.e., unstack is (strictly) entangled by succeeding putdown with holding, and stack is (strictly) entangled by preceding pickup with holding.

Introducing a *flaw ratio* $\eta \in [0; 1]$ which is a parameter referring to an allowed percentage of "flaws" in training plans, can identify inner entanglements that can be discovered in plans that are "close" to the training plans. In other words, the exclusivity of predicate achievement or requirement between a pair of operators might only be satisfied to some extent in the training plans, while in some other solution plans the exclusivity can be fully satisfied. For example, in Blocksworld, the blocks might occasionally be "put aside" to other blocks in the training plans and thus cause that the useful inner entanglements (as above) are not detected. By considering flaw ratio, these inner entanglements can be found.

Let η be the flaw ratio, then the following equations determine when a given inner entanglement can be considered ($sprec$ and $ssucc$ stand for the strict version of entanglements by preceding and succeeding respectively):

$$(prec, o_1, o_2, p) \Leftrightarrow entP[o_1, o_2, p] > 0 \wedge \forall o \neq o_2 : \frac{entP[o_1, o, p]}{counter[o_1]} \leqslant \eta \quad (1)$$

$$(succ, o_1, o_2, p) \Leftrightarrow entS[o_1, o_2, p] > 0 \wedge \forall o \neq o_2 : \frac{entS[o_1, o, p]}{counter[o_1]} \leqslant \eta \quad (2)$$

$$(sprec, o_1, o_2, p) \Leftrightarrow \frac{entP[o_1, o_2, p]}{counter[o_1]} \geqslant 1 - \eta \wedge \forall o \neq o_2 : \frac{entP[o_1, o, p]}{counter[o_1]} \leqslant \eta \quad (3)$$

$$(ssucc, o_1, o_2, p) \Leftrightarrow \frac{entS[o_1, o_2, p]}{counter[o_1]} \geqslant 1 - \eta \wedge \forall o \neq o_2 : \frac{entS[o_1, o, p]}{counter[o_1]} \leqslant \eta \quad (4)$$

7.2. Filtering Unpromising Inner Entanglements

Following the discussion from Section 6.5 we can derive that pruning power of inner entanglements is crucial for having positive impact on the planning process. In other words, inner entanglements are more likely to be beneficial if they can prune a relatively large number of search alternatives. Otherwise, inner entanglements might have detrimental effect on performance of planning engines because of the overhead caused by their representation.

We identified two main cases in which inner entanglements do not have a strong pruning power. Firstly, inner entanglements where operators that are rarely applied in plans are involved. Training plans can provide a good indication of "rare" operators. So we can assume that if an operator appears rarely in training plans, then it will be used rarely also for other planning problems in a given domain. Hence, we define a threshold ϵ and filter out such inner entanglements where any of the involved operators (o_1 and o_2) has less instances in the training plans, i.e:

$$counter[o_1] < \epsilon \vee counter[o_2] < \epsilon$$

Secondly, comparing the number of arguments that "entangled" and "prohibited" operators have. Recall the example from Section 6.5, where pickup(?x) is (strictly) entangled by succeeding stack(?x ?y) with holding(?x). The entanglements prohibits applying putdown(?x) after pickup(?x). In our words, stack(?x ?y) is the "entangled" operator and putdown(?x) is the "prohibited" operator. Clearly, only 1 alternative is pruned (only one instance of putdown(?x) can be applied after pickup(?x)) while up to $n - 1$ alternatives are allowed (up to $n - 1$ instances of stack(?x ?y) can be applied after pickup(?x)), so the pruning power of the

entanglement is poor. The number of operators' arguments is thus a good indicator for estimating the numbers of pruned search alternatives. Hence, if the number of arguments of the "entangled" operator is higher than all the "prohibited" operators, then the entanglement is unpromising. Formally, let $arg(o)$ denote the number of arguments of an operator o. Let an operator o_1 be (strictly) entangled by a succeeding operator o_2 with a predicate p, the entanglement is considered as unpromising if:

$$\forall o \neq o_2, p \in \textit{pre}(o) : arg(o) < arg(o_2)$$

Analogously, let an operator o_2 be (strictly) entangled by a preceding operator o_1 with a predicate p, the entanglement is considered as unpromising if:

$$\forall o \neq o_1, p \in \textit{eff}^{+}(o) : arg(o) < arg(o_1)$$

Unpromising inner entanglements are filtered out except cases where both types of inner entanglements hold for the operators o_1, o_2 and the predicate p, and only one of the entanglements is unpromising. Such an exception follows the observation discussed in Section 6.5 that the compact encoding of such entanglements does not introduce more overheads than the encoding of a single (inner) entanglement.

7.3. Inner Entanglement Extraction

Algorithm 2 wraps up the method for extracting inner entanglements. Given generated training plans we can fill the arrays $entP, entS$ and $counter$ by running Algorithm 1. An initial value $init\text{-}fr$ of the flaw ratio η is assigned. The main loop (Lines 4-12) iteratively validates whether using the given flaw ratio does not lead to extraction of entanglements that do not hold in the training tasks. The validation is done by extracting the non-trivial inner entanglements using the current flaw

Algorithm 2 Extraction of entanglements with the flaw ratio.

1: generate training plans
2: fill arrays (Alg. 1)
3: $\eta = init\text{-}fr$
4: **while** $\eta > 0$ **do**
5: extract inner entanglements considering η (see equations (1)-(4))
6: filter unpromising inner entanglements (see Section 7.2)
7: generate reformulated training problems
8: **if** reformulated training problems are solvable **then**
9: **break**
10: **end if**
11: $\eta = \max(0, \eta - step)$
12: **end while**
13: generate reformulated (testing) problems

ratio η (Line 5), filtering unpromising inner entanglements (Line 6), generating reformulated training problems considering the extracted entanglements (Line 7) and running a planner on these reformulated problems (Line 8). Introducing the flaw ratio may cause that the set of extracted inner entanglements does not even hold for the training problems. If such a situation occur, the flaw ratio is decreased by $step$ (Line 11) and the process (for Line 4) is repeated. Clearly, if $\eta = 0$, then the set of extracted inner entanglements holds for the training tasks.

8. EXPERIMENTAL EVALUATION

This section is devoted to the empirical evaluation of the impact of entanglements in the plan generation process. The aims of the experiments are to analyse the impact of inner entanglements on state-of-the-art planning engines and how quality of training plans influences detection and extraction of inner entanglements. For the empirical evaluation purposes we used all the domains from the learning track of IPC-7; since inner entanglements are automatically extracted domain-specific knowledge, the learning track benchmarks seem to be appropriate. This test set is thus independent, open, and gives a relatively wide coverage.

In each domain, the planning tasks have the same domain model and thus differ only by planning problem specifications. Henceforth, *training problems* denote tasks that are used for learning entanglements, and *testing problems* denote tasks that are used as benchmarks.

8.1. Benchmark Planners

In order to perform our analysis, we selected a number of planners according to i) their performance in the IPCs, and ii) the variety of techniques they exploit. Selected planners are: *Metric-FF* (Hoffmann, 2003), *LPG-td* (Gerevini et al., 2003), *LAMA* (Richter and Westphal, 2010; Richter et al., 2011), *Probe* (Lipovetzky and Geffner, 2011; Lipovetzky et al., 2014), *MpC* (Rintanen, 2012, 2014), *Yahsp3* (Vidal, 2014), and *Mercury* (Domshlak et al., 2015).

Metric-FF (Hoffmann, 2003) is an extension of the well known FF planner (Hoffmann and Nebel, 2001) which won the 2nd IPC. The FF's search strategy is a variation of hill-climbing over the space of the world states, and in FF the goal distance

is estimated by solving a relaxed task for each successor world state. Compared to the first version of FF, Metric-FF is enhanced with goal orderings pruning technique and with the ordering knowledge provided by a goal agenda.

LPG-td won the 3rd IPC. It uses stochastic local search in a space of partial plans represented through *linear action graphs*, which are variants of the very well-known planning graph (Blum and Furst, 1997). The search steps are graph modifications, transforming an action graph into a different one.

LAMA (Richter and Westphal, 2010; Richter et al., 2011) won the 6th and 7th IPC (sequential satisficing track). LAMA translates the PDDL problem specification into a multi-valued state variable representation ("SAS+") and searches for a plan in the space of the world states using a heuristic derived from the causal graph, a particular graph representing the causal dependencies of SAS+ variables. Its core feature is the use of a pseudo-heuristic derived from landmarks.

Probe (Lipovetzky and Geffner, 2011; Lipovetzky et al., 2014) was successful in IPC-7 and IPC-8. It implements a dual search architecture for planning that is based on the idea of *probes*: single action sequences computed without search from a given state that can quickly go deep into the state space, terminating either in the goal or in failure.

MpC (Rintanen, 2012, 2014) was a runner-up in the agile track of IPC-8. MpC is a SAT-based planner that exploits an extremely compact SAT representation of planning tasks and an integrated SAT solver.

Yahsp3 (Vidal, 2014) won the agile track of IPC-8. Yahsp is a heuristic search based planner that exploits information obtained from computation of the heuris-

tics, which is similar to the heuristic used in FF. Such information is used to find "lookahead states" that are reachable but "far" from the current state.

Mercury (Domshlak et al., 2015) was a runner-up in the satisficing track of IPC-8. Similarly to LAMA, Mercury translates the PDDL representation into a SAS+ multi-valued state variable representation. It then exploits the Red-Black heuristics, that uses only partial delete-relaxation.

8.2. Experimental Setup

In Machine Learning, it is important to have a good quality training set in order to maximise the outcome of the learning process. From the planning perspective, training plans should well capture the important structural aspects that are generalizable to the whole class of planning tasks. If training plans are too short, their structure might be over-constrained and thus we might extract some inner entanglements that do not hold for many typical tasks of a given class. On the other hand, planning is computationally very expensive and thus obtaining long training plans might be too time consuming or even impossible. Hence, we have observed that a reasonable size for a training problem is when the length of its solution plan is between 20 and 100 actions, depending on the number of defined operators in the domain models (more operators yields longer solution plans). Moreover, the number of training problems does not have to be high. This follows the observation made by Chrpa et al. (2013) that the set of extracted entanglements often does not change, or changes are very small, with increasing number of training problems. Similar observations have been made when configuring portfolios of planners (Núñez et al., 2012). On the other hand, using very few training problems increases the risk of extracting

inner entanglements that do not hold (we might be "lucky" to have a very atypical problem as a training one). Following these observations, 5 training problems per domain were used. Notice that in the learning track of IPC-7 (Coles et al., 2012), a set of training problems is not explicitly provided and thus the training problems were generated by existing problem generators.

Strict versions of inner entanglements were learnt[6]. The benchmark planners were used to generate training plans. The flaw ratio (η) was initially set to 0.2, and, in case of any of the training problems became unsolvable after incorporating entanglements[7], the flaw ratio was iteratively reduced by 0.05 until all the training problems became solvable while entanglements were considered, or the flaw ratio dropped to 0.0 (for details, see Algorithm 2). Although in the previous work (Chrpa and McCluskey, 2012) the flaw ratio is set to 0.1, we observed on some preliminary experiments, performed on a small set of benchmarks (not included in the rest of this experimental analysis) that such a value is too conservative. On the other hand, setting the value above 0.2 led to extraction of inner entanglements that often did not hold in the training problems. The threshold ϵ (see Section 7.2) is set to 20, which means that the operator must be used at least four times in average in each training plan.

A CPU-time cutoff of 900 seconds (15 minutes, as in learning tracks of IPC) was used for both learning and testing runs. All the experiments were run on a quad core 2.8 Ghz CPU machine with 4GB of RAM. In this experimental analysis, IPC scores

[6] Although the compact encoding for situations where both types of inner entanglements are involved requires the non-strict version of entanglements by succeeding, correctness is not compromised, since the strict versions of inner entanglements are special cases of the non-strict versions.

[7] by "unsolvable" we mean those problems where the planner did not find a solution in the time limit of 600s

as defined in IPC-7 are used. For a planner $\mathcal{C}$ and a problem p, *Time*$(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$, where $T_p(\mathcal{C})$ is the CPU time needed by planner $\mathcal{C}$ to solve problem p and T_p^* is the CPU-time needed by the best considered planner, otherwise. Similarly, *Qual*$(\mathcal{C}, p)$ is 0 if p is unsolved, and $N_p^*/N_p(\mathcal{C})$, where $N_p(\mathcal{C})$ is the cost of the plan, solution of p, obtained by $\mathcal{C}$ and N_p^* is the minimal cost of the solution plan of p among all the considered planners, otherwise. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

8.3. Experimental Results: The Learning Phase

As discussed in literature (Chrpa et al., 2013) structure of solution plans might differ according to a planner that generated them and hence the set of inner entanglements extracted from such plans can differ as well. In order to improve sets of extracted inner entanglements (i.e., maximise the number of useful entanglements and minimise the number of "peculiar" entanglements), we selected, for each training problem, the best quality (shortest) plan from those produced by all the considered planners. These best quality training plans were then used in the entanglement extraction method (see Section 7). Hereinafter, a set of inner entanglements extracted by exploiting this approach will be denoted as the *"best-plan set"* of inner entanglements.

Intuitively, using good quality training plans leads to extracting good quality domain knowledge (inner entanglements in this case). To test this intuition, we also considered the worst quality plans from those produced by all the considered planners (hereinafter denoted as *"worst-plan set"* of inner entanglements).

The results of the learning phase are as follows:

- In Gripper, Rovers, Satellite and Spanner no inner entanglements have been extracted, i.e., both the best- and worst-plan sets are empty.
- In Depots, Parking, TPP, the best- and worst-plan sets are the same.
- In BlocksWorld (Bw), the worst-plan set is empty while the best-plan set is not empty.
- In Barman, both the best- and worst-plan sets are not empty but different.

In the first case, the structure of the domain models prevents to capture any non-trivial inner entanglements. In the second case, the quality of training plans does not make any difference. This is due to the fact that the "important" part of training plans structure does not change that much than the quality of these training plans and by using the flaw ratio small structural changes of training plans are "absorbed". The third case refers to the situation where good quality plans usually follow the strategy of putting blocks to the table while bad quality plans usually temporarily stack blocks on other blocks. In the last case, the worst-plan set is a superset of the best-plan set. Although such a result is counter-intuitive, we observed that in the Barman domain, drinks can be prepared by using clean shots or by reusing "dirty" shots if the same ingredient is put into them. Using always clean shots provide "narrower" structure of solution plans, however, there plans are of worse quality since shots have to be always cleaned. It should be noted that best- and worst-plan sets were different only in 2 out of 9 domains. Although the work of Chrpa et al. (2013) indicates that the differences should be larger, incorporating the filtering technique for unpromising

inner entanglements (see Section 7.2) into the learning method "absorbs" some of these differences.

8.4. Experimental Results: The Testing Phase

[Table 1 about here.]

The results shown in Table 1 demonstrate the positive impact of inner entanglements on the planning process. It can be seen that only Probe solved all original testing problems in Bw and Depots. While inner entanglements (best-plan sets) were considered, in Bw, Depots and TPP, some planners were able to solve all the testing problems. In Parking and Barman, the results are mixed. In Parking, the overall results are rather negative, in Barman, Probe (best-plan set) and Lama (worst-plan set) benefit from inner entanglements while Mercury, on the other hand, has much worse performance on inner entanglements enhanced problems. Assuming that we can run all planners with original and inner entanglements enhanced domain models in parallel, then by using inner entanglements we can solve 2 more problems in Parking, and 3 more problems in Barman. Additionally, 9 problems in Barman can be solved faster when inner entanglements are considered.

Whereas the results generally support the claim that inner entanglements can effectively prune search space by eliminating unpromising alternatives, some results, however, require more attention. Lama does not perform well for the best-plan set in Bw, while it performs considerably well in the worst-plan set in Barman. This might lead to an observation that Lama performs well in the worst-plan sets rather than best-plan sets. We, however, believe that this observation is of domain and planner specific nature and thus might not be generalized. The reason for Lama's good

performance in the worst-plan set in Barman is in the fact that enforcing the planner to use only clean shots makes the landmark-based heuristics more informative. On the other hand, the best-plan set in Bw enforces the planner to put blocks on the table before stacking them in goal positions. Lama, however, has already a good performance on the original setting – its heuristics is well informed. Inner entanglements in this case might introduce some sub-optimalities (as the quality results indicate) and thus slow down the planning process of Lama. In Mercury's case, we can observe that it already performs well on the original Barman problems. Inner entanglements, however, seem to introduce overheads and possibly make Mercury's heuristics less informative.

We have also observed that using good quality training plans is useful for the learning process since the structure of the plans has less noise (e.g. redundant actions). Despite some results of Lama that contradicts the observation, we believe that the "best-plan" strategy will be useful also in other learning based techniques (e.g. generating macros).

8.5. Impact of Flaw Ratio and Filtering

[Table 2 about here.]

Table 2 provides a comparison of impact of the heuristics (flaw ratio, filtering) on "quality" of learnt set of inner entanglements. Only the best-plan sets were considered for this comparison. Noticeably, in Parking and TPP the sets are the same regardless of which heuristics is used or not and thus these domains are not listed in Table 2. Also, planners that did not solve any task in any of the encodings in a given domain are not listed in Table 2. The results provide clear evidence, mainly

in Barman and Bw, that both heuristics – flaw ratio and filtering – are useful when applied together.

Technically speaking, when only flaw ratio is considered, the set of inner entanglements is the superset or equal than the set of inner entanglements without considering flaw ratio. Filtering, on the other hand, removes possibly unpromising inner entanglements from the learnt set. In other words, flaw ratio and filtering heuristics provide a useful synergy for maximizing the potential of inner entanglements.

8.6. Discussion of Results

This subsection is devoted to discuss interesting aspects of the experimental analysis results.

8.6.1. *Summary of Performance Improvement.* Inner entanglements eliminate unpromising alternatives in the search space. As already discussed in the paper, pruning power of inner entanglements is a key factor for their usefulness. Therefore, we proposed a method for filtering inner entanglements whose pruning power is small (see Section 7.2). Our experiments confirmed that the filtering method often manage to filter out unpromising inner entanglements while keeping the promising ones. Inner entanglements are efficient if exclusivity of both predicate achievement and requirement between a pair of operators holds. The reason is mainly in the compact and informative encoding (see Section 5). Such inner entanglements were extracted in Bw (i.e. putting the block on the table always after it is unstacked), in Depots (i.e. loading a crate always after it is lifted) and TPP (i.e. loading goods always after buying it). Our experiments showed a performance improvement among

the planners in these domains. Such results indicate that inner entanglements have a good potential for improvement. We have also identified a few cases where inner entanglements have detrimental effect on planners (e.g. Mercury in Barman). As discussed in Section 6.5, representation of inner entanglements has an impact on heuristics computation. Generally speaking, despite pruning the search space, representation of inner entanglements might introduce local minima of heuristic functions that, in consequence, might have detrimental effect on planning engines since they need to search more nodes to escape such minima.

8.6.2. *Completeness Issues.* As discussed earlier, our method for extracting inner entanglements follows an assumption that a set of inner entanglements that holds for a set of training planning tasks also holds for the whole class of planning tasks (i.e., the testing ones). If this assumption does not hold for some tasks in the class they become unsolvable if inner entanglements are enforced. We observed in our experiments that the majority of reformulated tasks (by encoding inner entanglements) was solved by at least one of the planners. In Barman and Parking, 5 and 16 reformulated tasks, respectively, have not been solved by any of the planners. However, no evidence was obtained whether this was caused by their unsolvability or whether these tasks were too hard for the planners. In other words, the planners on these tasks run out of time or memory.

To alleviate the incompleteness issue we can try to solve the original task after the reformulated one failed. Specifically, we run the planner on the reformulated task, and if the task is considered unsolvable before the time limit is reached, then we run the planner on the original task. Theoretically, unsolvability of a planning

task can be identified in finite time if a complete planning engine is considered. In practice, we can identify some unsolvable tasks in a little time if the reachability analysis reveals that the goal cannot be reached (Bonet and Geffner, 1999). Since we have not identified any unsolvable reformulated task in the given time limit, the same results as for best-plan or worst-plan sets of entanglements would have applied for the aforementioned approach. Alternatively, we can alleviate the incompleteness issue by manually verifying the correctness of extracted inner entanglements or by incorporating reformulated tasks along with original tasks into planning portfolios such as PbP (Gerevini et al., 2014).

8.6.3. *Improvement to the Quality of Plans Generated.* In general, inner entanglements do not guarantee optimality of solution plans. Strengthening definitions of inner entanglements to guarantee plans optimality is, of course, theoretically possible. Given the complexity results of "normal" inner entanglements, we can expect the same for "optimal" inner entanglements. Using the approximation algorithm for extracting inner entanglements on optimal training plans with zero flaw ratio might extract some useful "optimal" inner entanglements. However, we believe that there is a high risk of extracting incorrect "optimal" inner entanglements. For example, the recently mentioned inner entanglements in the Depots domain are "optimal" for problems where each crate must be delivered to a different location. If in some problem a crate must be stacked on a different pallet but within the same location, such inner entanglements will force the planner (even the optimal one) to extract sub-optimal plans. Speaking about satisficing planning, these entanglements will prevent planners to find a plan only if no truck is available. Such a problem is very atypical.

Hence, there is a very low risk of extracting incorrect "normal" inner entanglements and similar observations can be made in other domains. Our experimental results have not clearly indicated any case in which extracted set of inner entanglements did not hold.

8.6.4. *Relationship to other Pruning or Problem Reformulation Techniques.* Although there are several techniques based on pruning or problem reformulation techniques (discussed in the Related Work section), inner entanglements are complementary to these techniques. Pruning techniques are often an inseparable part of advanced planning engines. We used several of such planning engines, which were successful in the past IPCs, for our experiments. We demonstrated that inner entanglements can often significantly improve their performance. Outer Entanglements (Chrpa and Barták, 2009; Chrpa and McCluskey, 2012) prune unpromising instances of planning operator according to their relations with initial or goal atoms. Inner entanglements are complementary to outer entanglements as has already been demonstrated in the previous work (Chrpa and McCluskey, 2012). Another well known technique for reformulating domain models is learning macros. A recent work introducing ASAP, a planner based on algorithm selection approach which selects the best couple (planner,encoding) for a given domain, has shown that inner entanglements and combination of outer and inner entanglements often outperformed macros (Vallati et al., 2014). Exploiting a natural property of inner entanglements, i.e., exclusivity of predicate achievement or requirement, has been also used for generating macros (Chrpa et al., 2013). Such macros can be in some cases beneficial, however, such an approach cannot be used in cases where operators in an

inner entanglement relation cannot be applied consecutively. Inner entanglements can support also other learning techniques that are used in planning. Roller (de la Rosa et al., 2011) is a system that learns decision trees that are then used to guide depth-first search. Combining Roller with entanglements (both inner and outer), such a system is called Rollent, brought promising results as well (Fuentetaja et al., 2015).

9. CONCLUSIONS AND FUTURE WORK

In this paper we presented Inner Entanglements, that are relations between pairs of planning operators and predicates such that an operator exclusively achieves a predicate for another operator, or an operator exclusively requires a predicate from another operator. To deal with the intractability of deciding a given inner entanglement holds for a given planning task (see Section 6), we used an approximation method for extracting "domain-specific" sets of inner entanglements from training plans, solution plans of simple tasks. Inner entanglements can be encoded into domain models without extending the input language of a planner (see Section 5) and, therefore, they can be understood and exploited as planner-independent knowledge.

Inner entanglements are able to considerably improve the planning process as our experiments demonstrated. In Bw, Depots and TPP, the considerable performance improvement was observed among almost all the planners. As discussed before, inner entanglements are especially powerful if exclusivity of both predicate achievement and requirement between a given pair of operators holds, which is the case of Bw, Depots and TPP. Generally, inner entanglements have a good potential

for a performance improvement if they are not "clashing" with a given planning technique, as demonstrated in Barman (Mercury) and Bw (Lama).

Pruning power of inner entanglements is a crucial aspect for their success. In particular, we need to avoid creating them with rarely used operators, and when the argument count of an entangled operator is higher than certain other operators in the domain model (as explained in section 7.2). Incorporating the aforementioned filtering technique into the inner entanglement learning method alleviated most of the performance concerns raised in the previous works (Chrpa and McCluskey, 2012; Chrpa and Vallati, 2013).

We identified several avenues for future research. Firstly, we believe that inner entanglements can be considered directly in heuristics –rather than being encoded in PDDL– by, for instance, penalising possibilities which violate these entanglements. Secondly, we believe that inner entanglements can be encoded, for instance, in decision trees or control rules. This might improve performance of related planners, i.e., Roller (de la Rosa et al., 2011) or TALPlanner (Kvarnström and Doherty, 2000). Thirdly, we will investigate in which cases deciding non-trivial inner entanglements is tractable. Given the insights in the paper (see Section 6.4) we believe that by analysing domain structure we can identify some useful inner entanglements in polynomial time. Finally, given the encouraging spread of results among sets of planners and domains, we intend to work towards including an inner entanglements generating facility as part of a knowledge engineering workbench.

Acknowledgments

The authors would like to acknowledge the use of the University of Huddersfield Queensgate Grid in carrying out this work.

The research was partly funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1).

REFERENCES

BACCHUS, FAHIEM, and FRODUALD KABANZA. 2000. Using temporal logics to express search control knowledge for planning. Artificial Intelligence, **116**(1-2):123–191. . http://dx.doi.org/10.1016/S0004-3702(99)00071-5.

BÄCKSTRÖM, C., and B. NEBEL. 1995. Complexity results for SAS+ planning. Computational Intelligence, **11**:625–656.

BERNARD, D. E., E. B. GAMBLE, N. F. ROUQUETTE, B. SMITH, Y. W. TUNG, N. MUSCETTOLA, G. A. DORIAS, B. KANEFSKY, J. KURIEN, W. MILLAR, P. NAYAL, K. RAJAN, and W. TAYLOR. 2000. Remote agent experiment DS1 technology validation report. Technical report, Ames Research Center and JPL.

BLUM, A.L., and M.L. FURST. 1997. Fast planning through planning graph analysis. Artificial Intelligence, **90**(1-2):281–300.

BONET, B., and H. GEFFNER. 1999. Planning as heuristic search: New results. *In* Proceedings of European Conference on Planning, ECP, pp. 360–372.

BOTEA, A., M. ENZENBERGER, M. MÜLLER, and J. SCHAEFFER. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. Journal of Artificial Intelligence Research (JAIR), **24**:581–621.

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

CELORRIO, SERGIO JIMÉNEZ, PATRIK HASLUM, and SYLVIE THIEBAUX. 2013. Pruning bad quality causal links in sequential satisfying planning. *In* ICAPS 2013 Workshop on Planning and Learning.

CHAPMAN, D. 1987. Planning for conjunctive goals. Artificial Intelligence, **32**(3):333–377.

CHEN, Y., and G. YAO. 2009. Completeness and optimality preserving reduction for planning. *In* Proceedings of International Joint Conference on Artificial Intelligence, IJCAI, pp. 1659–1664.

CHRPA, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. Knowledge Engineering Review, **25**(3):281–297.

CHRPA, L., and R. BARTÁK. 2009. Reformulating planning problems by eliminating unpromising actions. *In* Proceedings of Symposium on Abstraction, Reformulation, and Approximation, SARA, pp. 50–57.

CHRPA, LUKÁS, and THOMAS LEO MCCLUSKEY. 2012. On exploiting structures of classical planning problems: Generalizing entanglements. *In* Proceedings of European Conference on Artificial Intelligence, ECAI, pp. 240–245.

CHRPA, LUKÁS, and FAZLUL HASAN SIDDIQUI. 2015. Exploiting block deordering for improving planners efficiency. *In* Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, pp. 1537–1543. http://ijcai.org/Abstract/15/220.

CHRPA, LUKÁS, and MAURO VALLATI. 2013. Revisiting inner entanglements in classical planning. *In* Proceedings of Scandinavian AI conference, SCAI, pp. 75–84.

CHRPA, LUKÁS, MAURO VALLATI, THOMAS LEO MCCLUSKEY, and DIANE E. KITCHIN. 2013. Generating macro-operators by exploiting inner entanglements. *In* Proceedings of Symposium on Abstraction, Reformulation, and Approximation, SARA, pp. 42–49.

CHRPA, LUKÁS, MAURO VALLATI, and HUGH OSBORNE. 2013. Learnability of specific structural patterns of planning problems. *In* Proceedings of International Conference on Tools with Artificial Intelligence, ICTAI, pp. 18–23.

COLES, AMANDA, ANDREW COLES, ANGEL GARCÍA OLAYA, SERGIO JIMENEZ, CARLOS LINARES LOPEZ, SCOTT SANNER, SUNGWOOK YOON, and OTHERS. 2012. A survey of the seventh international planning competition (review). AI Magazine, **33**(1):83–88.

COLES, A., M. FOX, and A. SMITH. 2007. Online identification of useful macro-actions for planning. *In* Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS, pp. 97–104.

DAWSON, C., and L. SIKLÓSSY. 1977. The role of preprocessing in problem solving systems. *In* Proceedings of International Joint Conference on Artificial Intelligence, IJCAI, pp. 465–471.

DE LA ROSA, TOMÁS, SERGIO JIMÉNEZ CELORRIO, RAQUEL FUENTETAJA, and DANIEL BORRAJO. 2011. Scaling up heuristic planning with relational decision trees. Journal of Artificial Intelligence Research (JAIR), **40**:767–813.

DOMSHLAK, CARMEL, JÖRG HOFFMANN, and MICHAEL KATZ. 2015. Red-black planning: A new systematic approach to partial delete relaxation. Artificial Intelligence, **221**:73–114. .

http://dx.doi.org/10.1016/j.artint.2014.12.008.

DOMSHLAK, CARMEL, MICHAEL KATZ, and ALEXANDER SHLEYFMAN. 2012. Enhanced symmetry breaking in cost-optimal planning as forward search. *In* Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS, pp. 343–347.

EMERSON, E. ALLEN, and A. PRASAD SISTLA. 1996. Symmetry and model checking. Formal Methods in System Design, **9**(1/2):105–131. . http://dx.doi.org/10.1007/BF00625970.

FOX, MARIA, and DEREK LONG. 1999. The detection and exploitation of symmetry in planning problems. *In* Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages, pp. 956–961. http://ijcai.org/Proceedings/99-2/Papers/041.pdf.

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

FUENTETAJA, RAQUEL, LUKÁŠ CHRPA, THOMAS L MCCLUSKEY, and MAURO VALLATI. 2015. Exploring the synergy between two modular learning techniques for automated planning. *In* Eighth Annual Symposium on Combinatorial Search (SoCS), pp. 35–43.

GEREVINI, ALFONSO, ALESSANDRO SAETTI, and IVAN SERINA. 2003. Planning through stochastic local search and temporal action graphs. Journal of Artificial Intelligence Research (JAIR), **20**:239 – 290.

GEREVINI, ALFONSO, ALESSANDRO SAETTI, and MAURO VALLATI. 2014. Planning through automatic portfolio configuration: The PbP approach. Journal of Artificial Intelligence Research (JAIR), **50**:639–696. . http://dx.doi.org/10.1613/jair.4359.

GHALLAB, M., C. KNOBLOCK ISI, S. PENBERTHY, D. E SMITH, Y. SUN, and D. WELD. 1998. PDDL - the planning domain definition language. Technical report.

GHALLAB, MALIK, DANA S. NAU, and PAOLO TRAVERSO. 2004. Automated planning, theory and practice. Morgan Kaufmann Publishers.

GODEFROID, PATRICE, and FRODUALD KABANZA. 1991. An efficient reactive planner for synthesizing reactive plans. *In* Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 2., pp. 640–645. http://www.aaai.org/Library/AAAI/1991/aaai91-100.php.

GUPTA, NARESH, and DANA S. NAU. 1992. On the complexity of blocks-world planning. Artificial Intelligence, **56**(2-3):223–254. . http://dx.doi.org/10.1016/0004-3702(92)90028-V.

HASLUM, PATRIK. 2007. Reducing accidental complexity in planning problems. *In* Proceedings of International Joint Conference on Artificial Intelligence, IJCAI, pp. 1898–1903.

HOFFMANN, J. 2003. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. Journal Artificial Intelligence Research (JAIR), **20**:291–341.

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

HOFFMANN, J., J. PORTEOUS, and L. SEBASTIA. 2004. Ordered landmarks in planning. Journal of Artificial Intelligence Research (JAIR), **22**:215–278.

KAUTZ, H., and B. SELMAN. 1992. Planning as satisfiability. *In* Proceedings of European Conference on Artificial Intelligence, ECAI, pp. 359–363.

KORF, R.E. 1985. Macro-operators: A weak method for learning. Artificial Intelligence, **26**(1):35–77.

KVARNSTRÖM, J., and P. DOHERTY. 2000. TALplanner: A temporal logic based forward chaining planner. Annals of Mathematics and Artificial Intelligence, **30**(1-4):119–169.

LIPOVETZKY, NIR, and HECTOR GEFFNER. 2011. Searching for plans with carefully designed probes. *In* the 21st International Conference on Automated Planning and Scheduling (ICAPS-11), AAAI press, pp. 154–161.

LIPOVETZKY, NIR, MIQUEL RAMIREZ, CHRISTIAN MUISE, and HECTOR GEFFNER. 2014. Width and inference based planners: SIW, BFS(f), and PROBE. *In* The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track, pp. 6–7.

MCCLUSKEY, T. L., and J. M. PORTEOUS. 1997. Engineering and compiling planning domain models to promote validity and efficiency. Artificial Intelligence, **95**(1):1–65.

MINTON, STEVEN. 1988. Quantitative results concerning the utility of explanation-based learning. *In* Proceedings of AAAI Conference on Artificial Intelligence, pp. 564–569.

MINTON, S., and J. G. CARBONELL. 1987. Strategies for learning search control rules: An explanation-based approach. *In* Proceedings of International Joint Conference on Artificial Intelligence, IJCAI, pp. 228–235.

NEWTON, M. A. H., J. LEVINE, M. FOX, and D. LONG. 2007. Learning macro-actions for arbitrary planners and domains. *In* Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS, pp. 256–263.

NÚÑEZ, SERGIO, DANIEL BORRAJO, and CARLOS LINARES LÓPEZ. 2012. Performance analysis of planning portfolios. *In* Proceedings of the Fifth Annual Symposium on Combinatorial Search, SOCS, pp. 65–71.

POCHTER, NIR, AVIV ZOHAR, and JEFFREY S. ROSENSCHEIN. 2011. Exploiting problem symmetries in state-based planners. *In* Proceedings of AAAI Conference on Artificial Intelligence, pp. 1004–1009.

RICHTER, S., and M. WESTPHAL. 2010. The LAMA planner: guiding cost-based anytime planning with

landmarks. Journal of Artificial Intelligence Research (JAIR), **39**:127–177. ISSN 1076-9757.

RICHTER, SILVIA, MATTHIAS WESTPHAL, and MALTE HELMERT. 2011. LAMA 2008 and 2011. *In* Booklet of the 7th International Planning Competition.

RINTANEN, JUSSI. 2003. Symmetry reduction for SAT representations of transition systems. *In* Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy, pp. 32–41. http://www.aaai.org/Library/ICAPS/2003/icaps03-004.php.

RINTANEN, JUSSI. 2012. Engineering efficient planners with SAT. *In* Proceedings of European Conference on Artificial Intelligence, ECAI, pp. 684–689.

RINTANEN, JUSSI. 2014. Madagascar: Scalable planning with SAT. *In* The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track, pp. 66–70.

SACERDOTI, EARL D. 1975. The nonlinear nature of plans. *In* Advance Papers of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, September 3-8, 1975, pp. 206–214. http://ijcai.org/Proceedings/75/Papers/028.pdf.

SIDDIQUI, FAZLUL HASAN, and PATRIK HASLUM. 2012. Block-structured plan deordering. *In* AI 2012: Advances in Artificial Intelligence - 25th Australasian Joint Conference, Sydney, Australia, December 4-7, 2012. Proceedings, pp. 803–814.

SLANEY, J., and S. THIÉBAUX. 2001. Blocks world revisited. Artificial Intelligence, **125**(1-2):119–153.

TOZICKA, JAN, JAN JAKUBUV, MARTIN SVATOS, and ANTONÍN KOMENDA. 2016. Recursive polynomial reductions for classical planning. *In* Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016., pp. 317–325. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13088.

VALLATI, MAURO, LUKÁS CHRPA, and DIANE E. KITCHIN. 2014. ASAP: an automatic algorithm selection approach for planning. International Journal on Artificial Intelligence Tools, **23**(6):1–25.

VALMARI, ANTTI. 1996. The state explosion problem. *In* Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996, pp. 429–528. .

VIDAL, VINCENT. 2014. YAHSP3 and YAHSP3-MT in the 8th international planning competition. *In* The Eighth International Planning Competition. Description of Participant Planners of the Deterministic Track, pp. 64–65.

WEHRLE, MARTIN, MALTE HELMERT, YUSRA ALKHAZRAJI, and ROBERT MATTMÜLLER. 2013. The relative pruning power of strong stubborn sets and expansion core. *In* Proceedings of the Seventeenth

International Conference on Automated Planning and Scheduling, ICAPS, pp. 251–259.

YOON, SUNG WOOK, ALAN FERN, and ROBERT GIVAN. 2008. Learning control knowledge for forward search planning. Journal of Machine Learning Research, **9**:683–718.

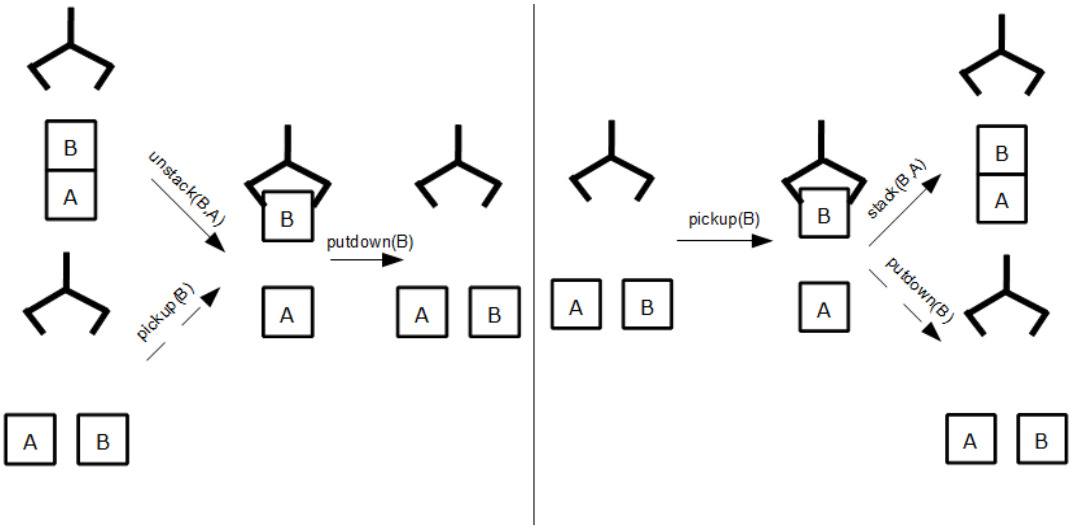

FIGURE 1. Motivating example for inner entanglements, concretely entanglements by preceding (left) and by succeeding (right). Whereas holding(B) can be achieved by either pickup(B) or unstack(B A), for putdown(B) requiring holding(B), only unstack(B A) is useful. Similarly, whereas holding(B) is required by either putdown(B) or stack(B A), only stack(B A) is useful if holding(B) is achieved by pickup(B).

```
(:action pick-up
:parameters (?x - block)
:precondition (and (clear ?x)(ontable ?x)(handempty))
:effect (and (not (ontable ?x))(not (clear ?x))(not (handempty))
             (holding ?x)(not (pick-up_stack_succ_holding ?x)))
)
(:action put-down
:parameters (?x - block)
:precondition (and (holding ?x)(pick-up_stack_succ_holding ?x))
:effect (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x))
)
(:action stack
:parameters (?x - block ?y - block)
:precondition (and (holding ?x)(clear ?y))
:effect (and (not (holding ?x))(not (clear ?y))(clear ?x)
             (handempty)(on ?x ?y)(pick-up_stack_succ_holding ?x))
)
(:action unstack
:parameters (?x - block ?y - block)
:precondition (and (on ?x ?y)(clear ?x)(handempty))
:effect (and (holding ?x)(clear ?y)
             (not (clear ?x))(not (handempty))
             (not (on ?x ?y))(pick-up_stack_succ_holding ?x))
)
```

FIGURE 2. An example of the BlocksWorld operators reformulated by an entanglement by succeeding (between pickup, stack and holding)

```
(:action pick-up
:parameters ( ?x - block)
:precondition (and (clear ?x)(ontable ?x)(handempty))
:effect (and (not (ontable ?x))(not (clear ?x))(not (handempty))
             (holding ?x)(not (put-down_unstack_prec_holding ?x)))
)
(:action put-down
:parameters ( ?x - block)
:precondition (and (holding ?x)(put-down_unstack_prec_holding ?x))
:effect (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x))
)
(:action stack
:parameters ( ?x - block ?y - block)
:precondition (and (holding ?x)(clear ?y))
:effect (and (not (holding ?x))(not (clear ?y))(clear ?x)
             (handempty)(on ?x ?y))
)
(:action unstack
:parameters ( ?x - block ?y - block)
:precondition (and (on ?x ?y)(clear ?x)(handempty))
:effect (and (holding ?x)(clear ?y)(not (clear ?x))(not (handempty))
             (not (on ?x ?y))(put-down_unstack_prec_holding ?x))
)
```

FIGURE 3. An example of the BlocksWorld operators reformulated by an entanglement by preceding (between unstack, put-down and holding)

```
(:action pick-up
:parameters ( ?x - block)
:precondition (and (clear ?x)(ontable ?x)(handempty))
:effect (and (not (ontable ?x))(not (clear ?x))(not (handempty))
             (stack_pick-up_both_holding ?x)(not (holding ?x)))
)
(:action put-down
:parameters ( ?x - block)
:precondition (and (holding ?x))
:effect (and (not (holding ?x))(clear ?x)(handempty)(ontable ?x))
)
(:action stack
:parameters ( ?x - block ?y - block)
:precondition (and (stack_pick-up_both_holding ?x)(clear ?y))
:effect (and (not (stack_pick-up_both_holding ?x))(not (clear ?y))
             (clear ?x)(handempty)(on ?x ?y))
)
(:action unstack
:parameters ( ?x - block ?y - block)
:precondition (and (on ?x ?y)(clear ?x)(handempty))
:effect (and (holding ?x)(clear ?y)(not (clear ?x))(not (handempty))
             (not (on ?x ?y))(not (stack_pick-up_both_holding ?x)))
)
```

FIGURE 4. An example of the BlocksWorld operators reformulated both entanglements by preceding and succeeding between pick-up, stack and holding

	Coverage			Δ IPC Score - speed		Δ IPC score - quality	
Planner	O	B	W	B	W	B	W
Barman							
FF	0	0	0	0.0	0.0	0.0	0.0
LPG	0	0	0	0.0	0.0	0.0	0.0
Lama	1	1	14	0.0	+13.4	0.0	+12.6
Probe	5	12	2	+7.5	-3.0	+7.1	-3.0
MpC	0	0	0	0.0	0.0	0.0	0.0
Yahsp	0	0	0	0.0	0.0	0.0	0.0
Mercury	25	21	2	-8.5	-24.1	-3.9	-23.5
Bw							
FF	0	0	-	0.0	-	0.0	-
LPG	25	30	-	+14.5	-	+7.9	-
Lama	28	28	-	-1.4	-	-2.3	-
Probe	30	30	-	+4.1	-	+2.8	-
MpC	0	14	-	+14.0	-	+14.0	-
Yahsp	29	30	-	+12.9	-	-7.1	-
Mercury	19	29	-	+11.0	-	+7.0	-
Depots							
FF	1	3	3	+2.3	+2.3	+1.8	+1.8
LPG	10	24	24	+16.7	+16.7	+14.1	+14.1
Lama	0	2	2	+2.0	+2.0	+2.0	+2.0
Probe	30	30	30	-3.2	-3.2	+2.0	+2.0
MpC	19	30	30	+17.6	+17.6	+11.3	+11.3
Yahsp	22	30	30	+18.7	+18.7	+20.8	+20.8
Mercury	0	0	0	0.0	0.0	0.0	0.0
Parking							
FF	11	9	9	-3.0	-3.0	-2.2	-2.2
LPG	0	0	0	0.0	0.0	0.0	0.0
Lama	8	7	7	-2.4	-2.4	-0.7	-0.7
Probe	7	6	6	-0.9	-0.9	-0.8	-0.8
MpC	5	5	5	+0.4	+0.4	-0.1	-0.1
Yahsp	0	0	0	0.0	0.0	0.0	0.0
Mercury	8	7	7	-1.7	-1.7	-0.9	-0.9
TPP							
FF	0	3	3	+3.0	+3.0	+3.0	+3.0
LPG	0	29	29	+29.0	+29.0	+29.0	+29.0
Lama	20	30	30	+20.3	+20.3	+10.3	+10.3
Probe	15	30	30	+23.6	+23.6	+14.7	+14.7
MpC	15	21	21	+14.1	+14.1	+6.0	+6.0
Yahsp	20	20	20	+1.7	+1.7	+0.0	+0.0
Mercury	26	30	30	+15.4	+15.4	+2.4	+2.4

TABLE 1. Comparing planners' performance on (O)riginal and (B)est- and (W)orst-plan sets of inner entanglements encodings. Δ IPC Score refers to a difference of the IPC score between the reformulated and the original encodings (positive values – higher score for the reformulated encoding). "-" means no inner entanglements were produced.

	Coverage					Δ IPC Score - speed				Δ IPC score - quality			
Planner	O	N	R	F	A	N	R	F	A	N	R	F	A
Barman													
Lama	1	0	0	-	1	-1.0	-1.0	-	0.0	-1.0	-1.0	-	0.0
Probe	5	0	0	-	12	-5.0	-5.0	-	+7.5	-5.0	-5.0	-	+7.1
Mercury	25	0	0	-	21	-25.0	-25.0	-	-8.5	-25.0	-25.0	-	-3.9
Bw													
LPG	25	-	1	-	30	-	-24.4	-	+14.5	-	-24.1	-	+7.9
Lama	28	-	0	-	28	-	-28.0	-	-1.4	-	-28.0	-	-2.3
Probe	30	-	0	-	30	-	-30.0	-	+4.1	-	-30.0	-	+2.8
MpC	0	-	0	-	14	-	0.0	-	+14.0	-	0.0	-	+14.0
Yahsp	29	-	21	-	30	-	-14.0	-	+12.9	-	-27.2	-	-7.1
Mercury	19	-	0	-	29	-	-19.0	-	+11.0	-	-19.0	-	+7.0
Depots													
FF	1	1	3	1	3	+0.4	+2.3	+0.4	+2.3	-0.1	+1.8	-0.1	+1.8
LPG	10	18	24	18	24	+8.0	+16.7	+8.0	+16.7	+6.5	+14.1	+6.5	+14.1
Lama	0	0	2	0	2	0.0	+2.0	0.0	+2.0	0.0	+2.0	0.0	+2.0
Probe	30	27	30	27	30	-9.3	-3.2	-9.3	-3.2	-6.7	+2.0	-6.7	+2.0
MpC	19	30	30	30	30	+16.2	+17.6	+16.2	+17.6	+13.2	+11.3	+13.2	+11.3
Yahsp	22	30	30	30	30	+18.7	+18.7	+18.7	+18.7	+23.0	+20.8	+23.0	+20.8

TABLE 2. Comparing planners' performance on (O)riginal, (N)o flaw ratio nor filtering, Flaw (R)atio only, (F)iltering only, (A)ll (flaw ratio and filtering) on the best-plan sets of inner entanglements encodings. Δ IPC Score refers to a difference of the IPC score between the reformulated and the original encodings (positive values – higher score for the reformulated encoding). "-" means no inner entanglements were produced.