ECAI 2012
Luc De Raedt et al. (Eds.)

doi:10.3233/978-1-61499-098-7-240

On Exploiting Structures of Classical Planning Problems: Generalizing Entanglements

Lukáš Chrpa and **Thomas Leo McCluskey**[1]

Abstract.

Much progress has been made in the research and development of automated planning algorithms in recent years. Though incremental improvements in algorithm design are still desirable, complementary approaches such as problem reformulation are important in tackling the high computational complexity of planning. While machine learning and adaptive techniques have been usefully applied to automated planning, these advances are often tied to a particular planner or class of planners that are coded to exploit that learned knowledge. A promising research direction is in exploiting knowledge engineering techniques such as reformulating the planning domain and/or the planning problem to make the problem easier to solve for general, state-of-the-art planners. Learning (outer) entanglements is one such technique, where relations between planning operators and initial or goal atoms are learned, and used to reformulate a domain by removing unneeded operator instances. Here we generalize this approach significantly to cover relations between atoms and pairs of operators themselves, and develop a technique for producing *inner entanglements*. We present methods for detecting inner entanglements and for using them to do problem reformulation. We provide a theoretical treatment of the area, and an empirical evaluation of the methods using standard planning benchmarks and state-of-the-art planners.

1 Introduction

In the area of Automated Planning, many successful planning engines have been developed [2, 3, 18], however, little use is made of knowledge that might be characteristic for a given class of planning problems. Knowledge that might be used to help inform planning falls into a spectrum between planner dependent at one extreme, and planner independent at the other. At the planner independent extreme, knowledge must be learned or hand-encoded within planning domain models and tasks, utilizing the syntax and semantics of a domain independent planner input language such as PDDL [12]. On the other hand, planner dependent knowledge is created to fit with the particular design of a planner, and is normally encoded in structures additional to the planner input language.

As the use of general planning engines within AI technology becomes more widespread, planner independent approaches such as reformulation or pre-processing are very attractive, as they tend to decouple the choice and development of planning engines from the choice and development of the learning or encoding technique. A good example of a such a technique is learning macro-actions. This is a widely studied area [4, 7, 20] because macro-actions can be encoded as regular actions, although they share the behavior of a sequence of actions, and are useful for a class of planning engines.

While the main disadvantage of using macro-actions is the risk of a significant increase of the branching factor during searching, there are several techniques which are used for reducing the branching factor. A well-known technique, called commutativity pruning, is used to discard unnecessary symmetries caused by commutative actions. Commutative actions, informally said, do not influence each other and can be executed in any order. Graphplan [2], one of the best-known planning algorithms, allows the execution of commutative actions in parallel (in one step). In linear-memory algorithms (e.g. IDA$*$) commutativity pruning can be done by giving the commutative actions fixed ordering (see e.g. [14]). Existing planners usually do not deal with all instances of planning operators. The FF planner [18] instantiates only actions appearing at some level of relaxed Planning Graph. FastDownward [15] (a predecessor of LAMA [21]) uses the idea called reach-one-goal (i.e. achieve the goals of the planning task consecutively), where the solver focuses on such actions that may be relevant for a particular goal. Recent work [6] focusing on cost-optimal SAS+ planning [1] introduces an 'Expansion Core' method which in a node expansion phase (in A$*$ search) restricts on relevant Domain Transition Graphs rather than all of them. Other work focusing on cost-optimal SAS+ planning [10] prunes irrelevant actions (e.g. actions changing a value of a variable having no dependants from a goal value) or exploits 'tunnel macro-actions' (i.e. if a certain action is executed then there is no other choice than to execute specific actions forming the 'tunnel'). Finally, capturing relations between planning operators and initial or goal atoms [8], later called *outer entanglements*, is also used for pruning potentially unnecessary instances of operators.

The contributions of this paper are as follows. We generalize the idea of outer entanglements [8] to cover relations between pairs of planning operators and atoms, called *inner entanglements*. Informally, inner entanglements determine exclusivity of 'achievement' or 'consumption' of atoms between operators. Entanglements (both outer and inner) aim to capture the causal relationships characteristic for a given class of planning problems; despite making no guarantee of preserving optimality, in many cases they enable a reduction of the branching factor. Since deciding outer entanglements is PSPACE-complete [9] and a similar property is assumed (although not proved yet) for inner entanglements, an approximation method for detecting entanglements is provided. The method finds entanglements in a set of reference plans which are then transferred to a wider set of planning tasks. Entanglements can be encoded directly in planning problems (by problem reformulation) which ensures planner independence. Our approach is evaluated on International Planning Competition (IPC) benchmarks[2] using several state-of-the-art planners.

[1] University of Huddersfield, UK, email: {l.chrpa, t.l.mccluskey}@hud.ac.uk

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

2 Preliminaries

Classical planning (in state space) deals with finding a sequence of actions transforming the static, deterministic and fully observable environment from some initial state to a desired goal state [13].

In the set-theoretic representation **atoms**, which describe the environment, are propositions. **States** are defined as sets of propositions. **Actions** are specified via sets of atoms specifying their preconditions, negative and positive effects (i.e., $a = (pre(a), eff^-(a), eff^+(a))$). An action a is **applicable** in a state s iff $pre(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus eff^-(a)) \cup eff^+(a)$.

In the classical representation atoms are predicates. A **Planning operator** $o = (name(o), pre(o), eff^-(o), eff^+(o))$ is a generalized action (i.e. action is a grounded instance of the operator), where $name(o) = op\_name(x_1, \ldots, x_k)$ (op_name is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator and $pre(o), eff^-(o)$ and $eff^+(o)$ are sets of (unground) predicates. The set-theoretic representation can be obtained from the classical representation by grounding.

A **planning domain** is specified via sets of predicates and planning operators (alternatively propositions and actions). A **planning problem** is specified via a planning domain, initial state and set of goal atoms. A **plan** is a sequence of actions. A plan is a **solution** of a planning problem if and only if a consecutive application of the actions in the plan (starting in the initial state) results in a state, where all the goal atoms are satisfied.

3 Theoretical Background

An insight into how actions in plans can be ordered is given by the fact that some action creates an atom (a grounded predicate) which is required by another action. This relation is called general dependence. General dependence can be generalized in terms of defining it between planning operators.

Definition 1. *Let a_1 and a_2 be actions. We say that a_2 **generally depends** on a_1 by a (grounded) predicate p_{gnd} if and only if $p_{gnd} \in eff^+(a_1) \cap pre(a_2)$.*
*Let o_1 and o_2 be planning operators. We say that o_2 **generally depends** on o_1 by a predicate p if and only if there exists a substitution Θ such that $p \in eff^+(o_1) \cap pre(o_2\Theta)$.* ■

A more specific notion than general dependence, which is well known in the planning community (e.g. as a causal link in plan-space planning), is that of an action being an 'achiever' of a predicate for some other action occurring after it in a plan (in a similar way as discussed in Chapman's earlier work [5]). Formally:

Definition 2. *Let $\langle a_1, a_2, \ldots a_n \rangle$ be a sequence of actions. We say that an action a_i **achieves** a (grounded) predicate p_{gnd} for an action a_j if and only if $i < j$, $p_{gnd} \in eff^+(a_i) \cap pre(a_j)$ and $\forall k \in \{i+1, \ldots, j-1\} : p_{gnd} \notin eff^+(a_k)$.* ■

3.1 Entanglements

'Outer Entanglements' are relations between planning operators and initial or goal atoms, and have been introduced as a tool for eliminating potentially unnecessary actions [8]. In the BlocksWorld [22] we may observe, for example, that *unstacking* blocks only occurs from their initial positions. In this case an 'entanglement by init' will capture that if atom `onblock(a,b)` is to be achieved for a corresponding instance of operator `unstack(?x,?y)` (`unstack(a,b)`), then the atom is an initial atom. Similarly, it may be observed that *stacking* blocks only occurs to their goal position. Then, an 'entanglement by goal' will capture that atom `onblock(b,a)` achieved by a corresponding instance of operator `stack(?x,?y)` (`stack(b,a)`) is a goal atom.

Definition 3. *Let P be a planning problem, where I is an initial situation and G is a goal situation. Let o be a planning operator and p be a predicate (o and p are defined in a planning domain related to P). We say that operator o is **entangled by init (resp. goal)** with predicate p in planning problem P if and only if $p \in pre(o)$ (resp. $p \in eff^+(o)$) and there exists a plan π that solves P and for every action $a \in \pi$ which is an instance of o and for every grounded instance p_{gnd} of the predicate p it holds: $p_{gnd} \in pre(a) \Rightarrow p_{gnd} \in I$ (resp. $p_{gnd} \in eff^+(a) \Rightarrow p_{gnd} \in G$). Hereinafter, it is denoted as $ent_P = (init, o, p)$ (resp. $ent_P = (goal, o, p)$).*
*Henceforth, entanglements by init and goal are denoted as **outer entanglements**.* ■

In this paper we extend the idea of outer entanglements to cover also relations between pairs of operators and predicates, called inner entanglements. Inner entanglements stand for operator exclusivity of 'providing' or 'requiring' predicates. In the BlocksWorld it may be observed, for instance, that operator `pickup(?x)` achieves predicate `holding(?x)` exclusively for operator `stack(?x,?y)` (and not for operator `putdown(?x)`). This relation is denoted as an 'entanglement by succeeding'. Similarly, it may be observed that predicate `holding(?x)` for operator `putdown(?x)` is exclusively achieved by operator `unstack(?x,?y)` (and not by operator `pickup(?x)`). This relation is denoted as an 'entanglement by preceding'.

Definition 4. *Let P be a planning problem. Let o_1 and o_2 be planning operators and p be a predicate (o_1, o_2 and p are defined in a planning domain related to P) such that $p \in eff^+(o_1)$ and $p \in pre(o_2)$. We say that o_1 is **entangled by succeeding** o_2 with p if and only if there exists a plan π solving P and $\forall a_1, a_2 \in \pi$ such that a_1 achieves p_{gnd} (p_{gnd} is a grounded instance of p) for a_2 it holds that if a_1 is an instance of o_1 then a_2 is an instance of o_2. Hereinafter, it is denoted as $ent_P = (succ, o_1, o_2, p)$.*
*We also say that o_2 is **entangled by preceding** o_1 with p if and only if there exists a plan π solving P and $\forall a_1, a_2 \in \pi$ such that a_1 achieves p_{gnd} (p_{gnd} is a grounded 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. Hereinafter, it is denoted as $ent_P = (prec, o_2, o_1, p)$.*
*Henceforth, entanglements by preceding and succeeding are denoted as **inner entanglements**.* ■

The above definition allows situations where an instance of the predicate (e.g. `holding(a)`) is not exclusively achieved by an instance of a certain operator (e.g. `unstack(a,b)`) but it is present in the initial state. Similarly, it is allowed that an instance of the predicate (e.g. `holding(b)`) is not exclusively achieved for an instance of a certain operator (e.g. `stack(b,a)`) but it is present in the goal state. To keep 'providing' and 'requiring' exclusivity only between given operators, Definition 4 must be strengthened as follows.

Remark 1. *Let P be a planning problem, I be an initial state in P and $\pi = \langle a_1, \ldots, a_n \rangle$ be a plan solving P. Let $a_I = \{\emptyset, \emptyset, I\}$ and $a_G = \{s_G, \emptyset, \emptyset\}$ be actions where s_G is a state obtained by executing π in I. Universal quantifier ($\forall a_1, a_2 \in \pi$) used for defining both entanglement by succeeding and preceding can be modified to $\forall a_1, a_2 \in \langle a_I, a_1, \ldots, a_n, a_G \rangle$ (in both cases). Then we say that entanglement by succeeding (or preceding) is **strict**.*

Entanglements as defined before are related to a single planning problem. This assumption can be easily extended to cover a class of planning problems sharing the same predicates and operators.

A single entanglement requires only the existence of one plan solving the given planning problem where the entanglement conditions are met. Hence some of the solution plans do not have to meet the entanglement conditions. Obviously, different entanglements are met in different solution plans. Therefore, we define a set of compatible entanglements which ensures existence of at least one solution plan following all the entanglements in the set. For example, all the BlocksWorld related entanglements mentioned throughout this section forms a set of compatible entanglements.

Definition 5. *Let ent_P be an entanglement related to a planning problem P. Let Π_P be the set of plans solving P. Let $\Pi_{ent_P} \subseteq \Pi_P$ be the set of plans satisfying conditions of the entanglement ent_P. We say that a set $E_P = \{ent_P^1, \ldots, ent_P^k\}$ is a set of **compatible entanglements** for P if and only if $\bigcap_{i=1}^{k} \Pi_{ent_P^i} \neq \emptyset$.* ■

3.2 Theoretical Properties of Entanglements

Outer entanglements in fact say that we need only instances of operators which are directly related to initial or goal situations (e.g. unstacking a block from its initial position or stacking it to its goal position). Hence if outer entanglements are taken into account then the number of actions considered by planners are lower or at worst equal than in the original problem [8].

On the other hand, inner entanglements stand for exclusivity of operators in 'providing' or 'requiring' predicates. For example, if an action, an instance of some operator (e.g. `pickup(a)`), is executed, then only an instance of a certain operator (e.g. `stack(a,b)`) can 'require' an instance of a certain predicate (e.g. `holding(a)`). In this case we do not restrict a number of actions but we reduce the branching factor. In other words, we prune some unpromising alternatives during the search.

There are some trivial situations when we can decide entanglements. Considering static predicates, i.e., predicates whose instances can be defined only in an initial state and no instance can be added or removed during the planning process.

Lemma 1. *Let P be a planning problem, p be a predicate and o be an operator related to P. Let $ent_P = (init, o, p)$ be an entanglement. ent_P is valid, i.e., o is entangled by init with p and $\Pi_{ent_P} = \Pi_P$ if p is static.*

Proof. See [8]. □

There is a straightforward relationship between entanglements by succeeding or preceding and general dependence which says that general dependence is weaker than inner entanglements.

Lemma 2. *If either an operator o_1 is entangled by a succeeding operator o_2 with a predicate p or an operator o_2 is entangled by a preceding operator o_1 with p, then o_2 generally depends on o_1 by p.*

Proof. Follows immediately from the definitions. □

General dependence between planning operators can also reveal situations where the operators must follow inner entanglements with a certain predicate. That is if there is only one operator that can achieve a certain predicate or if there is only one operator having a certain predicate in its precondition. In other words, exclusivity is already given in the domain definition. It is formalized in the following lemmas.

Lemma 3. *Let O be the set of planning operators defined in the domain of a given problem P and p be a predicate. If $\exists! o_i \in O$ such that $p \in eff^+(o_i)$, then $\forall o_k \in O$ such that $p \in pre(o_k)$ it holds that o_k is entangled by preceding o_i with p.*

Proof. It is straightforward because there cannot be an action providing a grounded instance of p to any instance of o_k which is not an instance of o_i. □

Lemma 4. *Let O be the set of planning operators defined in the domain of a given problem P and p be a predicate. If $\exists! o_i \in O$ such that $p \in pre(o_i)$, then $\forall o_k \in O$ such that $p \in eff^+(o_k)$ it holds that o_k is entangled by succeeding o_i with p.*

Proof. It is straightforward because there cannot be an action requiring a grounded instance of p (achieved by any instance of o_k) which is not an instance of o_i. □

Remark 2. *Considering strict inner entanglements we have to take into account additional conditions (see Remark 1). Regarding Lemma 3 we have to ensure that there exists a plan where instances of o_i always achieve p for instances of (all) o_k. It holds if there is no instance of p in an initial state. Regarding Lemma 4 we have also to ensure that there exist a plan where execution of instances of (all) o_k is always followed at some point by execution of an instance of o_i.*

4 Problem Reformulation

To exploit entanglements during the planning process we have to develop a specific planner or we have to reformulate problems in such a way that every valid solution follows entanglements. Reformulation is provided within classical planning (see Section 2), therefore it is possible to use any planner supporting classical (STRIPS) planning.

Outer entanglements can be encoded as follows (for deeper insight, see [8]). Let P be a planning problem, I be its initial state and G its goal situation. Let $ent_P = (init, o, p)$ (resp. $ent_P = (goal, o, p)$) be an entanglement. Then the problem P is reformulated as follows:

1. Create a predicate p' (not defined in the domain of P) having the same arguments as p and add p' to the domain of P.
2. Modify the operator o by adding p' into its precondition. p' has the same arguments as p which is in precondition (resp. positive effects) of o.
3. Create all possible instances of p' which correspond to instances of p listed in I (resp. G) and add the instances of p' to I.

Adding p', which is in fact a static predicate, into precondition of o causes that instances of o that do not follow the entanglement ent_P to become unreachable. For a proof of correctness, see [8].

Inner entanglements can be encoded as follows. Firstly, it is shown how entanglement by succeeding is encoded. Let P be a planning problem and $ent_P = (succ, o_1, o_2, p)$ be an entanglement. Then the problem P is reformulated as follows:

1. Create a predicate p' (not defined in the domain of P) having the same arguments as p and add p' to the domain of P.
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 such that $o \neq o_2$ and o generally depends on o_1 by p by adding p' into its precondition. p' has the same arguments as p which is in the precondition of o.

5. Add all possible instances of p' into the initial state of P and if ent_P is strict, then also to the goal situation of P.

Secondly, it is shown how entanglement by preceding is encoded. Let P be a planning problem and $ent_P = (prec, o_2, o_1, p)$ be an entanglement. Then the problem P is reformulated as follows:

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

Due to space reasons we provide only sketch proofs of correctness of above encodings of inner entanglements. Regarding entanglements by succeeding we can see that all possible instances of p' are valid in the initial state therefore applicability of instances of any operator is not affected unless an instance of o_1 is executed. After execution of an instance of o_1 a corresponding instance of p' is removed. It causes inapplicability of corresponding instances of o (see step 4) until a corresponding instance of o_2 is executed. Such a situation follows the entanglement by succeeding (ent_P) because all instances of o_1 achieves predicates only for instances of o_2. If we consider a strict version of the entanglement ent_P, then all the instances of p' must be valid in the goal state, i.e., every instance of o_1 must be at some point of the planning process followed by the corresponding instance of o_2.

Regarding entanglements by preceding we can see that in a strict version we can execute an instance of o_2 only if a previously executed instance of o_1 created p and no other action removed it. It is because p' which is in the precondition of o_2 can be created only by o_1. If anther operator o removes p created by o_1 (see step 4), then p' is removed as well to prevent unwanted execution of o_2 if some other operator (than o_1) created p. However, if $o \neq o_1$ creates p, then p' must be removed to prevent execution of o_2 (otherwise the entanglement is violated). Such a situation follows a strict version of the entanglement by preceding (ent_P) because an instance of o_2 is applicable only if an instance of o_1 achieves p for it. In a non-strict version, instances of o_2 can be executed if corresponding instances of p are present in the initial state (all possible instances of p' are defined in the initial state).

5 Detecting Entanglements

Since deciding entanglements, except trivial cases discussed in Section 3.2, is a hard problem (as mentioned in the introduction), an approximation method is used for detecting entanglements. Having classes of planning problems where each class shares the same planning domain (i.e., predicate and operator sets), then we can select a set of simpler problems from each class, solve them by a common planner and then explore plans. Entanglements found in these plans are assumed to be valid also for the rest of problems from the class.

The above can be formalized as follows. Let $\mathcal{C}$ be a set of planning problems sharing the same planning domain. Let $\mathcal{C}_T \subset \mathcal{C}$ be a set of training problems. By generalizing the term 'set of compatible entanglements' for a class of problems we can say that $E_\mathcal{C}$ is a set of compatible entanglements for $\mathcal{C}$ if and only if $E_\mathcal{C} \subseteq \bigcap_{P \in \mathcal{C}} E_P$ (E_P is a set of compatible entanglements for a problem P). In our approximation method we assume that $E_{\mathcal{C}_T} \subseteq E_\mathcal{C}$. Obviously, this assumption might not be correct, however, we believe that if problems in each class differ only by number of objects, then the assumption remains valid.

The idea of detecting outer entanglements in (training) plans is presented in work [8]. For every action we check whether predicates in its precondition (resp. positive effects) correspond with predicates in initial (resp. goal) situations. Inner entanglements are detected by checking which action achieves a predicate for a current action or vice versa. By that we can reveal whether instances of a certain operator achieve instances of a certain predicate for instances of another operator. These ideas are elaborated in Algorithm 1. We count how many times the entanglement conditions between instances of operators are met. Results are stored in 2D arrays $entI$, $entG$ and 3D arrays $entP, entS$. Function $is\_inst(arg)$ returns either an operator if arg (action) is an instance of it or an unground predicate if arg (atom) is an instance of it. Function $apply(s, a)$ (Line 22) applies an action a in a state s and moreover ensures to keep information that predicates from $\mathit{eff}^+(a)$ are achieved by a. This information is restored by $achieved\_by$ function (Line 10) or *NULL* is returned if a predicate is achieved by the initial state.

When all the arrays are filled then we can determine where the particular entanglement holds (for training problems). Training plans obtained by using a non-optimal planner, however, might contain flaws which can prevent detection of some entanglements. On the other hand, using optimal planners might be computationally very expensive. Therefore we introduce a **flaw ratio** $\eta \in [0; 1]$ as a parameter referring to an allowed percentage of flaws in training plans. Let η be a flaw ratio, then the entanglements are detected as follows:

$$(init, o, p) \quad \Leftrightarrow \quad \frac{entI[o,p]}{counter[o]} \geq 1 - \eta \tag{1}$$

$$(goal, o, p) \quad \Leftrightarrow \quad \frac{entG[o,p]}{counter[o]} \geq 1 - \eta \tag{2}$$

$$(prec, o_1, o_2, p) \quad \Leftrightarrow \quad entP[o_1, o_2, p] > 0 \wedge \forall o \neq o_2 : \frac{entP[o_1, o, p]}{counter[o_1]} \leq \eta \tag{3}$$

$$(succ, o_1, o_2, p) \quad \Leftrightarrow \quad entS[o_1, o_2, p] > 0 \wedge \forall o \neq o_2 : \frac{entS[o_1, o, p]}{counter[o_1]} \leq \eta \tag{4}$$

For the strict versions of inner entanglements we have to replace $entP[o_1, o_2, p] > 0$ by $entP[o_1, o_2, p]/counter[o_1] \geq 1 - \eta$ in (3) and $entS[o_1, o_2, p] > 0$ by $entS[o_1, o_2, p]/counter[o_1] \geq 1 - \eta$ in (4).

However, introducing a flaw ratio might cause that some detected entanglements are not valid even for training problems, especially if the flaw ratio is too high. Therefore, it is reasonable to validate detected entanglements on the training problems, i.e., we reformulate the training problems according to detected entanglements and then we run the planner on them. If at least one of the reformulated problems become unsolvable then we have to decrease the flaw ratio and start again. We continue it unless the detected entanglements are valid for all the training problems (it obviously happens if the flaw ratio (η) is 0).

	Metric-FF				LAMA				SatPlan				LPG			
	Orig	OE	IE	IOE	Orig	OE	IE	IOE	Orig	OE	IE	IOE	Orig	OE	IE	IOE
Depots (5-22)	22.04	34.26	22.23	32.22	21.77	31.88	20.43	33.26	14.76	21.21	14.52	22.36	26.03	33.92	26.19	33.84
Zeno (10-20)	19.85	20.15	20.41	21.69	20.41	20.39	20.17	20.74	10.30	10.91	10.47	13.66	19.91	20.66	9.24	11.84
DriverLog (8-20)	18.29	19.54	18.17	17.36	21.59	22.84	21.56	22.84	14.71	21.58	15.98	21.61	22.73	17.56	20.17	18.78
Matching (1-20)	18.71	33.24	10.20	15.71	27.16	36.02	8.04	16.49	24.15	39.78	22.98	36.66	13.02	34.69	17.30	25.16
Parking (1-20)	36.25	N/A	38.14	N/A	30.53	N/A	28.01	N/A	0.00	N/A	0.00	N/A	0.00	N/A	2.00	N/A
FreeCell (8-20)	23.12	22.91	15.31	17.73	20.88	23.11	12.21	15.58	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 1. Cumulative results for typed strips IPC benchmarks (problem ranges are in brackets, target-typed for Matching and Parking Benchmarks). Values are computed according to scoring in IPC learning track (2011). OE - outer entanglements only, IE - inner entanglements only, IOE - both

Algorithm 1 Checking how many times the entanglement conditions are met.

```
initialize_ent_arrays(); {create empty arrays entI, entG of size [Ops, Preds] and entP, entS of size [Ops,Ops,Preds]}
initialize_op_counter(); {create an empty array counter of size [Ops]}
for all training plan π = ⟨a_1, ... a_n⟩ do
    s := I; {I is an initial state and G is a goal situation}
    for i := 1 to n do
        for all p ∈ pre(a_i) do
            if p ∈ I then
                entI[is_inst(a_i), is_inst(p)] + +;
            end if
            a := achieved_by(s, p);
            if a ≠ NULL then
                entP[is_inst(a_i), is_inst(a), is_inst(p)] + +;
                entS[is_inst(a), is_inst(a_i), is_inst(p)] + +;
            end if
        end for
        for all p ∈ eff+(a_i) do
            if p ∈ G then
                entG[is_inst(a_i), is_inst(p)] + +;
            end if
        end for
        counter[is_inst(a_i)] + +;
        s := apply(s, a);
    end for
end for
```

6 Experimental Evaluation

The aim of the experiments is to evaluate and compare how reformulating problems with inner and outer entanglements affects solving time and quality of plans. The evaluation is made according to rules used in the IPC learning track (see Section 6.3).

6.1 Implementation Details

We decided to use a strict version of entanglements by preceding and non-strict version of entanglements by succeeding. The reason why we use a non-strict version of entanglements by succeeding rests in a necessity of including all the instances of a 'special' predicate to the goal situation which is difficult to handle for some planners. Moreover we decided not to reformulate 'trivial' entanglements which are mentioned in lemmas 1, 3 and 4 because these entanglements do not bring any new information.

Methods for detecting outer and inner entanglements are implemented in C++. Both of the methods support typed STRIPS representation in PDDL.

6.2 Experimental Setup

For evaluation purposes we chose several IPC benchmarks (typed strips), namely Depots, Zeno, DriverLog, Matching-BlockWorld, Parking and Freecell. As benchmarking planners we chose Metric-FF [16], LAMA 2011 [21], SatPlan 2006 [19] and LPG-td [11]. All the planners successfully competed in the IPC. LPG was optimized for speed and ran with a random seed set to 12345. LAMA was set to use a lazy greedy best first search accommodated by Landmark and FF heuristics. Metric-FF and SatPlan ran in default settings. Timeout was set to 1000s. For each benchmark we selected 5-7 easy problems as training problems and produced training plans by Metric-FF. A flaw ratio was set to 0.1 but in Parking domain it had to be decreased to 0.0 (see Section 5). All the experiments were performed on Intel i5 2.8 GHz, 8GB RAM, where Ubuntu Linux was used for running planners and Windows 7 for running our method.

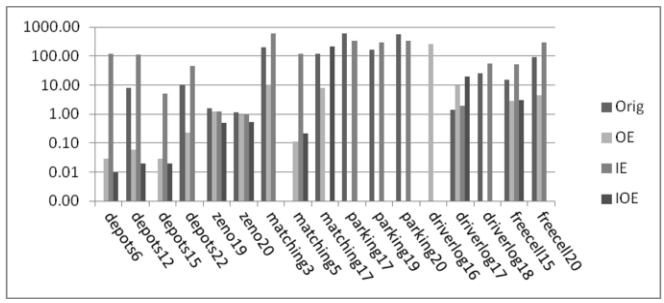

Figure 1. Selection of problems solved by Metric-FF. OE - outer entanglements only, IE - inner entanglements only, IOE - both

6.3 Experimental Results

Cumulative results of the evaluation are presented in Table 1. Values in Table 1 are computed according to rules used in IPC learning track[3]. Score for every solved (original or reformulated) problem is computed according to the formula $(1/(1+\log_{10} T/T^*)) + (N^*/N)$, where T is a running time of the certain planner for a certain (original or reformulated) problem, N is the length of the solution, T^* is the minimum running time achieved by a certain planner on either original problem or any of its reformulation. Similarly, N^* is the shortest solution. Score for unsolved (original or reformulated) problems is zero.

The results showed that reformulating planning problems by outer entanglements brought a significant improvement in most cases, except Zeno (Metric-FF and LAMA), Freecell (Metric-FF) and DriverLog (LPG). No outer entanglements have been detected in the Parking domain. The reason for this improvement rests in eliminating some potentially unnecessary but normally reachable instances of operators which pruned the search space and helped planners to navigate towards solutions more easily. However, sometimes it might happen that at some point of the planning process pruned actions might help to easily recover local maxima (e.g. if the goal is to build a tower of blocks A, B, C but at some point we have stacked A on B but not B on C, then unstacking A from B will help. However,

[3] http://www.plg.inf.uc3m.es/ipc2011-learning/Rules

if A is not on B in the initial state, then due to the entanglement we cannot unstack A from B and have to backtrack to the point before A was stacked on B.) This peculiarity of (outer) entanglements has been noticed previously [8].

In the inner entanglement case the results showed that the performance was improved in Zeno (Metric-FF), Parking (Metric-FF and LPG), DriverLog (SatPlan) and Matching-BW (LPG). However, the performance was much worse in Matching-BW and Freecell (Metric-FF and LAMA) and Zeno (LPG). In the case of Freecell at least two reformulated problems became unsolvable thus the assumption (see Section 5) does not hold in this case. Contrary to outer entanglements, inner entanglements do not restrict the number of actions considered by planners but prune some potentially unwanted alternatives coming across during the planning process. One shortcoming of planner independent approach is that planners have to take into account more atoms, which are introduced in encodings of inner entanglements (outer entanglements are encoded by static predicates that can be compiled away during preprocessing).

Combining outer and inner entanglements together brought the best results in Depots (LAMA, SatPlan), Zeno (Metric-FF, LAMA and SatPlan) and DriverLog (SatPlan). We found that the number of actions considered by these planners is lower than in when only outer entanglements are used. Even though inner entanglements did not restrict the number of actions in comparison to the original problem, in this case inner entanglements propagate knowledge given by outer entanglements. For instance, if we know that operator LIFT (Depots domain) is entangled by init with a predicate 'at' (referring to a location of a crate) and operator LOAD is entangled by preceding LIFT with a predicate 'lifting', then we can deduce that we can load a crate only at its initial location; thus some instances of LOAD can be pruned even though no outer entanglement is related to LOAD. The experiments show that in some cases the results were much better when inner entanglements were involved, while sometimes the results were much worse (for illustration, see Figure 1). In the case of LPG, it seems that the planner behavior is very dependent on the defined random seed. In the case of Metric-FF and LAMA, it appears that the efficiency of the planning process is tightly related to how the relaxed Planning Graph is affected by inner entanglements in different stages of the planning process and when there is a tendency for forming plateaux (the resolution of this conjecture is an interesting open problem which we aim to explore in future work). As mentioned before, outer entanglements restrict the number of actions therefore the action layers in Planning Graphs are smaller. In combination with inner entanglements the action layers can be even smaller. Inner entanglements are designed to ease the search (in Planning Graphs) by pruning possible 'dead-end' branches, but they may lead to fact layers which are larger than the original encoding.

7 Conclusions

In this paper we have generalized the idea of outer entanglements [8] to the idea of inner entanglements. We have presented a theoretical background to the work, and reviewed some relevant theoretical properties in this context. Methods for learning inner entanglements, and for reformulating problems using them, are detailed. The impact of inner (and outer) entanglements is experimentally evaluated on several IPC benchmarks using several state-of-the art planners. These planners already incorporates some pre-processing techniques for reducing the branching factor such as commutativity pruning (SatPlan) or pruning some operator instances (Metric-FF, LAMA), so the entanglement approach can be seen as complementary. The results indicate that the overall reformulation method is worthwhile, though the improvement is not universal. The experiments showed some interesting outcomes, indicating some fertile lines for future research. In particular, Metric-FF and LAMA performed on some reformulated problems significantly better while on some others significantly worse even in the same domain. This opens up an interesting problem of how relaxed planning graphs and heuristic values develop through the planning process and under what conditions reformulations help to avoid plateaux or, on the other hand, cause plateaux. Our future work will utilize recent research results on heuristic landscapes [17] in order to determine a theory of how inner entanglements affect search, and consider how related work designed for SAS+ planning [6, 10] might be incorporated into our approach.

REFERENCES

[1] C. Bäckström and B. Nebel, 'Complexity results for sas+ planning', *Computational Intelligence*, **11**, 625–656, (1995).

[2] A.L. Blum and M.L. Furst, 'Fast planning through planning graph analysis', *Artificial Intelligence*, **90**(1-2), 281–300, (1997).

[3] B. Bonet and H. Geffner, 'Planning as heuristic search: New results', in *Proceedings of ECP*, pp. 360–372, (1999).

[4] A. Botea, M. Enzenberger, M. Müller, and J. Schaeffer, 'Macro-ff: Improving ai planning with automatically learned macro-operators', *Journal of Artificial Intelligence Research (JAIR)*, **24**, 581–621, (2005).

[5] D. Chapman, 'Planning for conjunctive goals', *Artificial Intelligence*, **32**(3), 333–377, (1987).

[6] Y. Chen and G. Yao, 'Completeness and optimality preserving reduction for planning', in *Proceedings of IJCAI*, pp. 1659–1664, (2009).

[7] L. Chrpa, 'Generation of macro-operators via investigation of action dependencies in plans', *Knowledge Engineering Review*, **25**(3), 281–297, (2010).

[8] L. Chrpa and R. Barták, 'Reformulating planning problems by eliminating unpromising actions', in *Proceedings of SARA 2009*, pp. 50–57, (2009).

[9] L. Chrpa, T. L. McCluskey, and H. Osborne, 'Reformulating planning problems: A theoretical point of view', in *Proceedings of FLAIRS*, pp. 14–19, (2012).

[10] A. J. Coles and A. I. Coles, 'Completeness-preserving pruning for optimal planning', in *Proceedings of ECAI*, pp. 965–966, (2010).

[11] A. Gerevini, A. Saetti, and I. Serina, 'Planning in pddl2.2 domains with lpg-td', in *Proceedings of the fourth IPC*, (2004).

[12] M. Ghallab, C. Knoblock Isi, S. Penberthy, D. E Smith, Y. Sun, and D. Weld, 'Pddl - the planning domain definition language', Technical report, (1998).

[13] M. Ghallab, D. Nau, and P. Traverso, *Automated planning, theory and practice*, Morgan Kaufmann Publishers, 2004.

[14] P. Haslum and H. Geffner, 'Admissible heuristics for optimal planning', in *Proceedings of AIPS*, pp. 140–149, (2000).

[15] M. Helmert, 'The fast downward planning system', *Journal of Artificial Intelligence Research*, **26**, 191–246, (2006).

[16] J. Hoffmann, 'The metric-ff planning system: Translating "ignoring delete lists" to numeric state variables', *Journal Artificial Intelligence Research (JAIR)*, **20**, 291–341, (2003).

[17] J. Hoffmann, 'Analyzing Search Topology Without Running Any Search: On the Connection Between Causal Graphs and h+', *The Journal of Artificial Intelligence Research (JAIR)*, **41**, 155–229, (2011).

[18] J. Hoffmann and B. Nebel, 'The FF planning system: Fast plan generation through heuristic search', *Journal of Artificial Intelligence Research*, **14**, 253–302, (2001).

[19] H. Kautz, B. Selman, and J. Hoffmann, 'Satplan: Planning as satisfiability', in *Proceedings of the fifth IPC*, (2006).

[20] M. A. H. Newton, J. Levine, M. Fox, and D. Long, 'Learning macro-actions for arbitrary planners and domains', in *Proceedings of ICAPS 2007*, pp. 256–263, (2007).

[21] S. Richter and M. Westphal, 'The lama planner: guiding cost-based anytime planning with landmarks', *Journal Artificial Intelligence Research (JAIR)*, **39**, 127–177, (2010).

[22] J. Slaney and S. Thiébaux, 'Blocks world revisited', *Artificial Intelligence*, **125**(1-2), 119–153, (2001).