Generating Macro-operators by Exploiting Inner Entanglements

Lukáš Chrpa and **Mauro Vallati** and **Thomas Leo McCluskey** and **Diane Kitchin**
PARK Research Group
School of Computing and Engineering
University of Huddersfield
{l.chrpa, m.vallati, t.l.mccluskey, d.kitchin}@hud.ac.uk

Abstract

In Automated Planning, learning and exploiting additional knowledge within a domain model, in order to improve plan generation speed-up and increase the scope of problems solved, has attracted much research. Reformulation techniques such as those based on macro-operators or entanglements are very promising because they are to some extent domain model and planning engine independent. This paper aims to exploit recent work on *inner entanglements*, relations between pairs of planning operators and predicates encapsulating exclusivity of predicate 'achievements' or 'requirements', for generating macro-operators. We discuss conditions which are necessary for generating such macro-operators and conditions that allow removing primitive operators without compromising solvability of a given (class of) problem(s). The effectiveness of our approach will be experimentally shown on a set of well-known benchmark domains using several high-performing planning engines.

Introduction

Because even classical planning is intractable (up to PSPACE-complete (Bylander 1994)), exploiting additional knowledge which is somehow characteristic for a given class of planning problems is a promising way towards making the planning process more efficient. Since the 1970's, and lately with the help of the Learning Track of the International Planning Competition (IPC)[1], many such learning techniques have been developed. One of the most studied is the generation and use of macro-operators (macros), encoded in the same format as the operators forming the planning domain model, but encapsulating a sequence of such (primitive) operators (Dawson and Siklóssy 1977; Botea et al. 2005; Newton et al. 2007; Chrpa 2010b). Macros, whose power lies in providing "shortcuts" in the search space, have always been hampered by the problem of utility: used naively, their addition to a domain model can cause an explosion of operator instances.

The use of macros can be considered a technique for *planning problem reformulation*, a domain and planner independent way of preprocessing a planning problem (in PDDL (Mcdermott et al. 1998) defined by domain model and problem files) so that a planning engine may be able to solve the problem more efficiently. Another such technique is *outer entanglements* (Chrpa and Barták 2009), which can be used to reformulate the domain model by effectively removing unpromising operator instances. *Inner entanglements* (Chrpa and McCluskey 2012) are relations between pairs of planning operators and predicates, denoting exclusivity of 'achievement' or 'requirement' of a predicate. That is, one operator achieves a predicate exclusively for another operator, or an operator requires a predicate exclusively from another operator.

This paper proposes a preliminary investigation about how inner entanglements might be exploited to generate useful macros within a reformulation phase of a planning problem. After some theoretical background, we describe an automatic technique for (i) generating a macro from two primitive operators in an inner entanglement relationship, and (ii) safely removing one or both of the primitive operators from the domain model of a given problem. Being able to remove domain operators, while adding macros, ameliorates the main problem of macro utility. We present the results of an empirical evaluation of our inner entanglement-based technique on a set of well-known IPC benchmark domains using state-of-the-art planning engines, showing that in domains where such macros can replace both primitive operators, the reformulation is very effective.

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 (Ghallab, Nau, and Traverso 2004).

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 defining their preconditions, negative and positive effects (i.e., $a = (\mathit{pre}(a), \mathit{eff}^-(a), \mathit{eff}^+(a))$). An action a is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

In the classical representation atoms are predicates. A *planning operator* $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{eff}^-(o), \mathit{eff}^+(o))$ is a generalised action (i.e. an action is a grounded instance of the operator), where $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$

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

(*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 (ungrounded) predicates.

A *planning domain* is specified by a set of predicates and a set of planning operators. 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.

Basic Relations between Actions and Operators

By analysing action or operator preconditions and effects 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. A notion of being a possible achiever can be easily extended for planning operators. If an action achieves an atom for some other action in some plan, then the first action is a *necessary achiever* of the atom for the other one. Hereinafter, we will use only *achiever* unless it is not clear from the context whether an action is a possible or necessary achiever. Note that being 'achiever' refers to a notion "causal link" in plan-space planning.

The opposite for being a (possible) achiever is being a (possible) 'clobberer' which means that action a_i deletes atom(s) a_j has in its precondition. Clearly, a notion of necessary clobberer is meaningless unless negative preconditions are used. Note that being 'clobberer' refers to a notion "threat" in plan-space planning. A notion of (possible) clobberer can be also easily extended for planning operators.

Achiever and clobberer relations between planning operators require predicate comparison. Predicates are equal if they have the same name and their arguments (including their order) are identical. Arguments of ungrounded predicates are variable symbols. These variable symbols are also arguments of operators having the corresponding predicates in their preconditions or effects. Achiever and clobberer relations, therefore, indicate which arguments must be the same for the given planning operators.

Inner Entanglements

Inner Entanglements have been recently introduced as relations between pairs of planning operators and atoms (predicates) (Chrpa and McCluskey 2012). Inner entanglements stand for operator exclusivity of 'achieving' or 'requiring' predicates. The BlocksWorld domain (Slaney and Thiébaux 2001) which we will use as a running example consists of four operators: pickup(?x) refers to a situation when a robotic hand picks-up a block ?x from the table, putdown(?x) refers to a situation when a robotic hand putsdown the block ?x it is holding to the table, unstack(?x,?y) refers to a situation when a robotic hand unstacks a block ?x from ?y, and stack(?x,?y) refers to a situation when a robotic hand stacks a block ?x to ?y. For a typical problem in the BlocksWorld domain, 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) because putdown(?x) would just reverse the effects of pickup(?x)) (see Figure 1). 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) because again putdown(?x) would just reverse the effects of pickup(?x)) (see Figure 1). This relation is denoted as an *entanglement by preceding*. Entanglements by preceding and succeeding are denoted as *inner entanglements*.

Informally speaking, inner entanglements provide constraints affecting ordering of operators' instances in solution plans. If an operator o_1 is entangled by a succeeding operator o_2 with a predicate p in a given planning problem, then in some solution plan, instances of o_1 are at some point followed by corresponding instances of o_2 and no corresponding instance of other operator having p in its precondition can be placed in between them. Similarly, if an operator o_2 is entangled by a preceding operator o_1 with a predicate p in a given planning problem, then in some solution plan, instances of o_2 are at some point preceded by corresponding instances of o_1 and no corresponding instance of other operator having p in its positive effects can be placed in between them. Note that this refers to *strict* inner entanglements (Chrpa and McCluskey 2012).

A single (inner) entanglement requires only the existence of one plan solving the given planning problem where the entanglement conditions are met and, therefore, different entanglements might be met in different solution plans. A *set of compatible entanglements* ensures existence of at least one solution plan following all the entanglements in the set (Chrpa and McCluskey 2012). For example, the two BlocksWorld related entanglements mentioned throughout this section forms a set of compatible entanglements. Hereinafter, we will assume that multiple entanglements are a set of compatible entanglements unless stated otherwise.

Indirect Inner Entanglements

The Zeno domain addresses the problem of transporting passengers by planes between locations. Here, we may observe that the operator refuel(?aircraft, ?city, ?fuel1, ?fuel2) is entangled by the succeeding operator fly(?aircraft, ?city1, ?city2, ?fuel2, ?fuel1) with the predicate fuel-level(?aircraft, ?fuel2). The operator refuel increases the fuel level of an aircraft in some location. The operator fly moves the aircraft from one location to another reducing the fuel level. The entanglement says that the fuel level of the aircraft is increased for the next flight. However, no entanglement can capture that the refuelling is done at the same location the flight will start from. The reason is that refuel does not achieve the predicate at(?aircraft,?city1) for fly because at(?aircraft,?city1) is not in refuel's positive effects. However, adding at(?aircraft,?city1) to refuel's positive effects will satisfy the entanglement conditions. If adding a predicate into the positive effects of an operator given the fact that the predicate is present in its precondition but not in its negative effects enables the in-

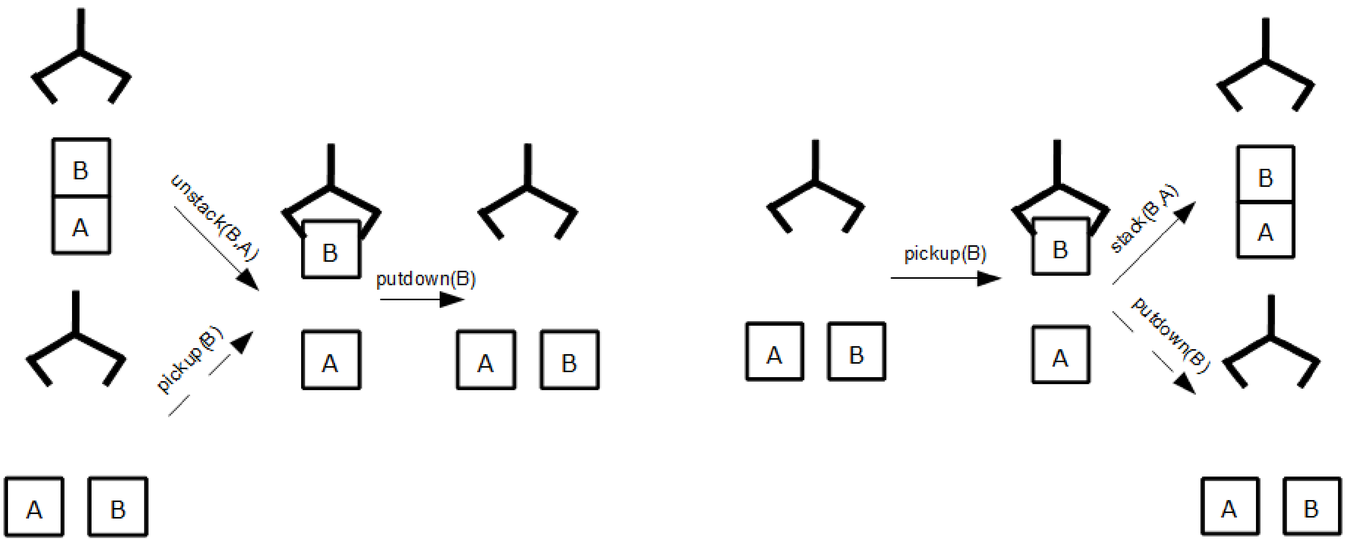

Figure 1: Motivating example for entanglements by preceding (left hand side) and by succeeding (right hand side)

ner entanglement relation with some other operator, then we say that the operator is in the *indirect inner entanglement* relation with the other operator (and the predicate).

Determining Macro-operators from Inner Entanglements

Inner entanglements, as mentioned before, refer to exclusivity of 'achievement' and 'requirement' of predicates between planning operators. As a running example we will use the well-known Depots domain which is a combination of the BlocksWorld and Logistic domains. Operator lift unstacks a crate from its surface (another crate or pallet) by a hoist in a given place. Operator drop reverses lift by stacking a crate to a surface by a hoist in a given place. Operator load loads a crate lifted by the hoist into a truck in the given place. Operator unload unloads a crate from the truck by a hoist in a given place. Operator drive moves the truck between places.

Entanglement by succeeding says that a predicate achieved by an instance of a given operator can be required only by instances of a specific operator. For a typical problem in the Depots domain, we may observe that the operator lift(?hoist, ?crate, ?surface, ?place) is entangled by the succeeding operator load(?hoist, ?crate, ?truck, ?place) with the predicate lifting(?hoist,?crate). Hence, if an instance of lift (e.g. lift(h1,c1,c2,p1)) is executed at the i-th step of some solution plan, then a corresponding instance of load (e.g. load(h1,c1,t1,p1)) is executed at the j-th step of the plan, where $j > i$. Also, no instance of another operator requiring (having in its precondition) or consuming (having in its negative effects) lifting(h1,c1) (e.g. drop(?hoist, ?crate, ?surface, ?place)) can be executed in between. Similarly, we may observe that the operator load(?hoist, ?crate, ?truck, ?place) is entangled by the preceding operator lift(?hoist, ?crate, ?surface, ?place) with the predicate lifting(?hoist,?crate). Hence, if an instance of load (e.g. load(h1,c1,t1,p1)) is executed at the i-th step of some solution plan, then a corresponding instance of lift (e.g. lift(h1,c1,c2,p1)) is executed at the j-th step of the plan, where $j < i$. Also, no instance of other operator achieving lifting(h1,c1) (e.g. unload(?hoist, ?crate, ?truck, ?place)) can be executed in between. Analogously, we may observe entanglement by preceding and succeeding between the operators load(?hoist, ?crate, ?truck, ?place) and unload(?hoist, ?crate, ?truck, ?place) and the predicate in(?crate,?truck). Indirect inner entanglements can be considered as well. As discussed before, in the Zeno domain the operator refuel is indirectly entangled by the succeeding operator fly with a predicate at.

Macros, on the other hand, encapsulate situations where corresponding instances of given planning operators are executed consecutively. For a typical problem in the Depots domain, we may observe that the operators lift and load can be assembled (in this order) into a macro in such a way that the arguments ?hoist,?crate are shared because lift achieves a predicate lifting(?hoist,?crate) for load. However, in the case of the operators load and unload, where load achieves a predicate in(?truck,?crate) for unload and thus arguments (?truck,?crate) are shared by the operators, we may also observe that when the operators are applied consecutively the crate is unloaded at the same place where it was loaded. Therefore, it is necessary to execute the drive operator in between which moves the truck (and the crate which is loaded in it) from one place to another. This situation is illustrated in Figure 2. Hence, it is not reasonable to assemble load and unload into a macro.

Straightforwardly, operators cannot be assembled into a macro if the first one clobbers for the second one. Inner entanglements indicate pairs of planning operators which are candidates for becoming macros and also determine arguments the operators share since it is known which predicate or predicates are achieved by one operator exclusively for another operator or vice versa. However, we have to ensure that the corresponding instances of the operators can be applied consecutively without any other constraints. In other words, we must be sure that no action (instance of some other operator) must be applied in between them. There are three situations which might cause necessity of executing an action (an instance of some operator o) between corresponding instances of operators o_1, o_2 considered for a macro.

(a) o is a possible achiever of a predicate for o_2 but also a clobberer for o_1.

(b) o_1 is a possible achiever for o but o_2 is a clobberer for o.

(c) o_1 is a possible achiever for multiple instances of o_2.

Conditions (a) and (b) can be weakened by analysing other inner entanglement relations which might invalidate some achiever relations between operators. For example, the operator lift possibly achieves the predicate lifting to the operators load and drop. If lift is entangled by succeeding load with lifting in a given problem, then no instance of lift is an achiever for any instance of drop in some solution plan of the problem. Condition (a) is illustrated in Figure 2. Typically, we need to execute an instance of drive in between instances load and unload. Condition (b) is analogous (there is no example in the Depots domain). Condition (c) is illustrated in Figure 3. Typically, we need to unload more crates in one place from a given truck, i.e., one instance of drive achieves a predicate at for multiple instances of unload.

Operators o_1 and o_2 can be assembled into a macro if for no operator o any of conditions (a)-(c) is satisfied. Operators o_1 and o_2 are selected as a candidate for a macro only if there is at least one inner entanglement relation between them (and some predicate). An inner entanglement between operators also indicates that there exists a solution plan of a given planning problem where instances of one operator are always followed or preceded by instances of the other operator. This information is useful because a macro generated from these operators may replace one or both of these operators without affecting completeness for that planning problem. Concretely, an entanglement by succeeding means that instances of one operator (o_1) are always followed by corresponding instances of the other operator (o_2) and therefore, if a macro is generated o_1 becomes unnecessary. Similarly, an entanglement by preceding means that instances of one operator (o_2) are always preceded by corresponding instances of the other operator (o_1) and therefore, if a macro is generated o_2 becomes unnecessary. If both inner entanglements hold between o_1 and o_2, then if a macro is generated both o_1 and o_2 becomes unnecessary. Having a non-empty set of predicates which are in an inner entanglements relation with o_1 and o_2 we can specify the following conditions:

(i) o_1 is entangled by succeeding o_2 with all the predicates, or

(ii) o_2 is entangled by preceding o_1 with all the predicates, or

(iii) both (i) and (ii).

Then, after generating a macro $o_{1,2}$ from operators o_1, o_2 (in this order we can remove: o_1 if (i) is met, o_2 if (ii) is met, and both o_1 and o_2 if (iii) is met.

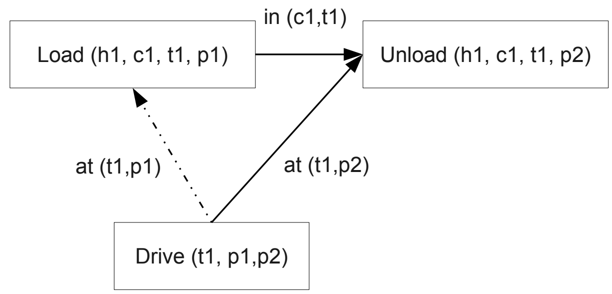

Figure 2: Example, in the Depots domain, where operator Load is entangled by succeeding operator Unload with a predicate in. Drive that is an achiever for Unload is also a clobberer for Load, so it has to be executed in between them.

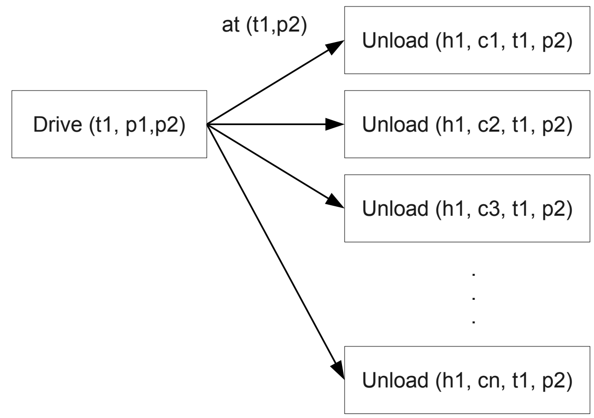

Figure 3: Example of an instance of operator Drive that achieves a predicate at for multiple instances of operator Unload, in the Depots domain

We are not restricted to generating macros from only two operators. When a new macro $o_{1,2}$ from operators o_1 and o_2 is generated, one of or both operators o_1 and o_2 are removed. It may be observed that an inner entanglement held between some other operator o and o_1 (or o_2) becomes true between o and the generated macro $o_{1,2}$ if o_1 (o_2) is removed. This is because the new macro encapsulated the primitive operators from which it was assembled, and when the primitive operator is removed the new macro becomes its only 'follower'. However, if the primitive operator is not removed the inner entanglement relation with it might be compromised since also the new macro consists of this primitive operator. Since macros are encoded in the same way as ordinary planning operators our approach can be applied recursively, which might result in generating 'longer' macros.

Implementation Details

Detection and Use of Inner Entanglements

Detecting inner entanglements is believed to be intractable in general, although some trivial cases can be easily identified (Chrpa and McCluskey 2012). Besides this, we can use an approximation method for detecting compatible sets of inner entanglements (including indirect inner entanglements) which has recently been published (Chrpa and McCluskey 2012). This method analyses a set of training plans, solutions of simpler planning problems, in order to identify a set of compatible (inner) entanglements which holds for every training problem and then it is assumed that this set of compatible (inner) entanglements holds for a whole class of planning problems using the same domain model. Of course, a set of compatible inner entanglements detected by this approach might not be correct for some (non-training) problems. Enforcing incorrect set of compatible entanglements while solving a problem results in losing its solvability (note that a correct set of compatible entanglements ensures the existence of a solution plan following all the entanglements in the set). Despite incompleteness of such an approach, we believe that selecting a 'good' set of training problems can almost eliminate this issue. In IPC benchmarks, selecting a few (around 5) training problems from the domain is usually sufficient to find a correct set of inner compatible entanglements for all the benchmark problems in a given domain. It was shown empirically (Chrpa and McCluskey 2012) that in a very few cases (in very complex domain) enforcing entanglements caused loss of solvability of a few problems.

Encoding Inner Entanglements into Domain and Problem Models

Work (Chrpa and McCluskey 2012) utilised a planner-independent approach to enable the reformulation of domains and problems in order to enforce (inner) entanglements during the planning process. In other words, alternatives which do not follow exclusivity of 'achieving' and 'requiring' predicates between operators must be avoided. The idea behind the reformulation is in introducing specific predicates, 'locks', which prevents executing certain instances of operators in some stage of the planning process. An instance of an operator having a 'lock' in its precondition cannot be executed after executing an instance of another operator ('locker') having a 'lock' in its negative effects until an instance of some other operator ('releaser') having a 'lock' in its positive effects has been executed. For example, a situation where pickup(?x) is 'entangled by succeeding' stack(?x,?y) with holding(?x) is modelled such that pickup(?x) is a 'locker' for putdown(?x) and stack(?x,?y) is a 'releaser' for putdown(?x). For details about encoding inner entanglements, see (Chrpa and McCluskey 2012).

Note that 'trivial' inner entanglements, i.e., whether there is only one achiever for a certain predicate or a certain predicate is required by only one operator, do not have to be encoded in the domain model since they do not provide any useful knowledge which can be used to prune some unpromising alternatives in the search.

Algorithm 1 A high-level description of our method for generating macros from inner entanglements

Require: Planning domain model with training planning problems and their solutions
Ensure: Reformulated domain model (added macros, removed some of primitive operators)

```
 1: Determine a set of compatible inner entanglements
 2: repeat
 3:   for all operators o1, o2 having an inner entanglement
      relation(s) between them do
 4:     if for each operator o in the domain model none of
        conditions (a)-(c) is satisfied then
 5:       generate a new macro o1,2 and remove o1, o2 or
          both (depends if (i), (ii) or (iii) is satisfied)
 6:       update inner entanglement relations
 7:       break
 8:     end if
 9:   end for
10: until No new macro has been generated
11: generate a reformulated domain model
```

Macro Generation

Our method is described in Algorithm 1. It utilises the original method for detection of inner entanglements (line 1), and then uses them to perform a macro generation phase (lines 2-10). The inner loop (lines 3-9) iteratively checks whether the macro candidates (pairs of operators in an inner entanglement relation) meet the conditions for becoming macros discussed in the previous section (line 4). If a candidate does meet the conditions, then a new macro is generated, one of or both (primitive) operators are removed according to conditions (i)-(iii), and inner entanglement relations are updated (as already discussed before). Then, we continue with the main loop (line 2). If no candidate meets the given conditions the macro generation phase finishes and a reformulated domain model is generated (line 18).

Condition (a) is verified as follows. Predicates which have to be achieved for o_2 by some different operator than o_1 are considered. In particular, we know that such predicates are not in o_1's positive effects or prevailing precondition (i.e. a predicate is in o_1's precondition but not in o_1's negative effects). We have to consider any operator o achieving some of these predicates, however, we can exclude situations where such a predicate achievement violates some of the inner entanglement relation. If o is also a clobberer for o_1, then condition (a) is satisfied and thus the macro $o_{1,2}$ cannot be generated. Condition (b) is verified analogously by finding an operator o which requires predicate(s) from o_1 which are deleted by o_2. Also, in this case we can discard situations which violates some inner entanglement relations.

Condition (c) is verified by checking whether the operator o_2 is a clobberer for itself or whether multiple application of different instances of o_2 will not bring any new information. The first can be easily identified by checking whether some predicate involved in the entanglement relation with o_1 is in negative effects of o_2. The latter can be identified by checking whether instances of o_2 with fixed arguments referring to all predicates involved in the inner entanglements relation with o_1 do not have different positive effects. For example, in the Gold-miner domain, the operator pick-gold(?location) achieves only a predicate holds-gold and thus it is not necessary to execute it more than once. As a counterexample, we might use the one depicted in Figure 3

Experimental Evaluation

The goal of the experimental evaluation was to demonstrate the potential of reformulating problems by the replacement of original operators with inner entanglement-based macros, to compare this with inner entanglement reformulation, and to explore the range of domains and planners for which the techniques are successful. For evaluation purposes we chose several IPC benchmark domains (typed strips) from IPC-3, IPC-6 and 7 (learning track), where it was clear that this kind of reformulation would be applicable (for example, it would not be applicable to domains with one operator). The domains are BlocksWorld (BW), Depots, Zeno, DriverLog, Gold-Miner, Matching-BW, Satellite and TPP. As benchmarking planners we chose Metric-FF (Hoffmann 2003), LPG-td (Gerevini, Saetti, and Serina 2003), Probe (Lipovetzky and Geffner 2011), LAMA 2011 (Richter and Westphal 2010), SatPlan 2006 (Kautz, Selman, and Hoffmann 2006) and Mp (Rintanen 2012). All the planners successfully competed in the IPCs. Timeout was set to 900s. The experiment was performed on Intel Xeon™ 3 GHz, 2 GB RAM. For each benchmark we selected 5-7 easy problems as training problems and produced training plans by Metric-FF which were used to learn inner entanglements and generate macros from them. Metric-FF was selected due to the fact that it is usually fast, and provides good quality plans. Time spent on learning was usually in the order of tenths of seconds (rarely in the order of seconds) per one domain.

Cumulative results of the evaluation are presented in Table 1, with the macro technique compared to the existing reformulation technique of inner entanglements, and the original problem formulation. Values are computed according to rules used in IPC-7 learning track[2]. The score for every solved problem is computed according to the formula $(1/(1+\log_{10}(T/T^*)))$ for time or (N^*/N) for quality. T is a running time of a 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 the original problem or any of its reformulations. Similarly, N^* is the shortest solution. The score for unsolved problems is zero. Note that in the Satellite domain we identified only 'trivial' inner entanglements, so the reformulated domain (and problems) model was the same as the original one, hence the 'N/A' value for inner entanglements.

Discussion of Results

It is well known that using macros tends to reduce the depth of the search at the cost of increasing the branching factor. In the BW, Depots and TPP domains, generated macros always replaced *both* the primitive operators in the Depots and TPP domains, and in one case in the BW domain. Therefore, the branching factor did not increase much (note that macros often have more instances than the primitive operator it is assembled from), resulting in an overall improvement across those planning engines that could cope with the hard learning track problems. Indeed, Metric-FF and Mp were able to find solutions to almost all the problems in the BW domain which was not previously possible without the aid of macro reformulation. In other cases macro reformulation achieved mixed results, often worse than original or inner entanglement encodings. In these cases, generated macros replaced only one of the primitive operators causing increase of the branching factor which often had a negative impact on planners' performance. The technique of using inner entanglements to reformulate domains can reduce the search branching factor, but does so at the cost of introducing supplementary predicates, which causes an increase of the size of problem representation. Using inner entanglements brings some improvement against the original encodings in about half of the cases. However, using macros generated from inner entanglements outperforms the inner entanglement encodings in the majority of cases. Good results were achieved for this technique in the Zeno domain because the size of the representation increased only marginally. Interestingly, in some cases using the inner entanglements encodings resulted in getting much better plans (in terms of quality).

LPG uses greedy local search on the Planning Graph which might not work well in situations where the branching factor is large. Therefore, LPG seems to exploit more original or inner entanglement reformulations. On the other hand, LPG achieved very good results in BW, Matching-BW and TPP when using macro reformulation. Probe's performance improves with either inner entanglement or macro reformulations. Probe uses 'causal commitments' which are similar to 'causal links' in plan space planning and, therefore, it seems to better exploit inner entanglements which determine exclusivity of 'causal links' between operators. Probe also seems to be less vulnerable to larger branching factor, therefore, it can exploit macros as well. Metric-FF and LAMA, which are based on heuristic search, tend to perform better with macros than inner entanglements (except Gold-miner). It seems that in this case, having more actions, which eventually reduce the depth of the search, is better than having more atoms (facts). SatPlan and Mp, which are based on SAT, achieved mixed results. Mp seems to exploit macros better than SatPlan. In the SatPlan case, reformulations lead to more complex SAT formulae which might slow-down the planning process despite pruning some unpromising search alternatives. In the Mp case, it seems that more compact SAT formulae reduces the negative impact of having more operator or predicate instances.

Although neither the original inner entanglement technique, or the macro replacement technique, are generally effective, we found out some interesting outcomes. If generated macros replace both the primitive operators, then the results suggest that this reformulation will outperform both original and inner entanglement encodings. Planners based on heuristic search (e.g. Metric-FF, LAMA) incline to exploit macros more efficiently than inner entanglements.

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

	+MA	-O		Metric-FF			LPG			Probe			LAMA			SatPlan			Mp		
				Orig	IE	Mcr	Orig	IE	Mcr	Orig	IE	Mcr	Orig	IE	Mcr	Orig	IE	Mcr	Orig	IE	Mcr
BW (60)	2	3	Time	0.0	0.0	**60.0**	21.1	35.8	**60.0**	29.3	33.1	**50.0**	21.1	15.8	**59.0**	0.0	0.0	0.0	0.0	0.0	**57.0**
			Quality	0.0	0.0	**60.0**	30.0	**59.8**	55.1	39.3	**49.2**	**49.2**	17.9	14.0	**58.9**	0.0	0.0	0.0	0.0	0.0	**57.0**
Depots (60)	2	4	Time	15.1	24.0	**42.8**	**39.5**	26.9	32.7	49.8	**55.9**	50.2	17.5	20.0	**51.9**	7.4	6.6	**13.3**	33.2	24.3	**35.4**
			Quality	22.1	23.9	**42.5**	40.2	**45.0**	37.2	54.0	**57.6**	52.5	18.6	20.9	**51.1**	6.3	9.3	**14.0**	30.6	36.8	**39.6**
Zeno (20)	1	1	Time	17.6	**19.3**	17.7	**19.8**	14.2	6.6	18.0	**19.6**	18.9	18.1	17.0	**19.8**	15.1	**15.3**	9.7	18.2	**19.2**	19.1
			Quality	**20.0**	19.9	16.9	16.6	**17.7**	9.0	19.0	**19.7**	16.8	**19.6**	18.8	18.1	13.9	**15.3**	11.8	18.3	**19.7**	17.5
DriverLog (20)	1	1	Time	14.6	13.7	**15.7**	16.8	**17.8**	17.6	**19.5**	18.3	19.1	**19.1**	16.6	18.7	14.6	**15.1**	14.4	**18.9**	18.6	18.2
			Quality	16.3	16.0	**16.5**	17.3	16.2	**18.4**	18.5	18.2	**19.3**	18.1	18.1	**18.5**	14.2	**15.7**	13.8	**19.4**	19.2	15.4
Gold-miner (60)	1	1	Time	37.6	**52.6**	33.0	**60.0**	33.0	48.3	55.1	34.4	**57.0**	44.2	53.2	**56.4**	**60.0**	54.9	53.4	**60.0**	32.6	49.9
			Quality	54.0	**58.5**	53.6	**60.0**	45.3	46.6	56.7	**58.6**	55.0	54.4	**57.7**	22.9	58.9	**58.9**	58.6	54.8	**56.8**	50.2
Matching-BW (60)	1	1	Time	**23.0**	8.1	17.9	28.8	21.7	**43.4**	15.5	25.1	**29.8**	**45.2**	11.6	20.6	**41.0**	27.3	36.7	1.0	1.7	**5.8**
			Quality	**24.3**	9.5	20.4	31.3	**34.4**	33.2	20.1	27.8	**28.4**	**42.8**	18.0	15.4	**41.0**	33.8	35.0	0.8	1.8	**6.0**
TPP (60)	1	2	Time	18.5	17.7	**31.9**	12.2	23.2	**60.0**	30.7	1.2	**51.0**	25.5	19.4	**56.8**	29.8	**34.5**	31.7	9.4	14.4	**27.0**
			Quality	27.2	28.7	**32.6**	15.4	**30.0**	24.5	37.3	3.0	**50.6**	37.5	33.8	**52.5**	26.7	36.8	**41.9**	13.0	17.4	**26.3**
Satellite (60)	1	1	Time	**28.0**	N/A	16.9	**60.0**	N/A	51.8	20.6	N/A	**30.0**	27.4	N/A	**29.7**	**7.0**	N/A	4.2	**25.0**	N/A	22.5
			Quality	**27.5**	N/A	24.5	**60.0**	N/A	54.6	25.4	N/A	**29.2**	**31.8**	N/A	26.9	**6.9**	N/A	4.5	**24.6**	N/A	23.9

Table 1: Cumulative results for typed strips IPC benchmarks (problem numbers are in brackets). +MA stands for the number of generated macros. -O stands for the number of removed primitive operators. Values are computed according to scoring in IPC learning track (2011). Orig - original PDDL domain encoding, IE - inner entanglements, Mcr - Macros

Probe tends to perform better using either inner entanglements (except Matching-BW) or macros than in the original domain encodings.

Related Work

Synthesising macros to aid AI plan generation has been a popular research area dating back to 1970s, in systems such as STRIPS (Fikes and Nilsson 1971) and REFLECT (Dawson and Siklóssy 1977). Some work has concentrated on "offline" problem independent macro generation using domain model analysis (McCluskey and Porteous 1997). A recent example of off-line macro generation is WIZARD (Newton et al. 2007; Newton and Levine 2010), which generates a useful set of macros using genetic algorithms, with generation and validation time of several hours for a given domain. Work (Alhossaini and Beck 2009) learns a set of domain-specific macros by WIZARD and then selects the most promising ones for a given problem in this domain by analysing problem features (e.g. numbers of objects). In fact, this approach provides problem-specific macros rather than domain specific which resulted in significant improvement of planning process in some domains. From this perspective, it seems to be reasonable to consider such an idea for improving our method since (inner) entanglements are also problem-specific, although we can identify the same entanglements for a whole class of 'typical' problems in a given domain.

Another line of work concentrates on specific planning engine techniques, and treats macro-generation as integrated with the planning process itself. For instance Macro-FF (SOL-EP) (Botea et al. 2005) and Marvin (Coles, Fox, and Smith 2007) exploit macros in order to help an FF-type planner escape plateaus. Our work fits into this area of "online" macro generation, but is aimed at providing a reformulation stage for input domain and problem specification, acting as a preprocessor (or "wrapper") for a range of planning engines and domains. The "CA-ED" technique of Macro-FF (Botea et al. 2005), involving learning macros via analysis of static predicates, is related but complementary to the inner entanglement approach. It is potentially useful for unifying some arguments of operators before generating a macro (e.g. lift and load are applied in the same location since the involved hoist can be only at one location), or by analysing successive actions in plans. Later work by Chrpa (2010b) extended the idea of macro-generation from adjacent actions in a solution, to non-adjacent actions which can be made adjacent in some valid plan permutation. His work utilised the idea of removing unnecessary primitive operators, though in an ad-hoc manner. In future, we should provide a rigorous comparison of this and our method in order to identify whether and how much are these methods complementary or competitive.

'Tunnel macros' (Coles and Coles 2010) refers to situations where given operators must be in a certain sequence. 'Tunnel macros' are related to 'trivial' inner entanglements, which can be easily determined without necessity for applying the approximation method, as these provide connections which must hold between 'adjacent' operators in such a specific sequence.

Given the potential for macros to degrade performance (McCluskey 1987), other work has emphasised the need for guiding heuristics and pruning techniques, and is largely complementary to our approach. Expansion Cores (Chen and Yao 2009), for example, restricts action use only to relevant domain transition graphs during the node expansion. Outer entanglements (Chrpa and Barták 2009), relations between operators and initial or goal atoms, are used for pruning unpromising operator instances. Combining macros and outer entanglements, even done in an ad-hoc way, provided very promising results (Chrpa 2010a).

Conclusions

In this paper we studied how inner entanglements, relations capturing exclusivity of predicate achievement or requirement between planning operators, can be exploited in order to generate macros. We provided an automatic technique for assembling macros from operators involved in an inner entanglement relation and, moreover, determining which of the primitive operators (or both) can be removed. For this purpose we defined indirect inner entanglements which can be understood as an extension of the original inner entanglements definition.

Our approach was evaluated empirically with eight IPC benchmark domains, which lead to 400 problem instances, using six state-of-the-art domain-independent planning engines. The results show that while the technique is not successful across all domains, it shows potential to be used as a reformulation technique for domains where a macro can replace two operator schema.

In the future we plan to provide a theoretical study of the complexity of the presented technique, and we are interested in extend our approach with ideas which have been applied in related works on macros (see the previous section). In particular we aim to investigate the possibility of combining our technique with action pruning (e.g Expansion Cores, Outer Entanglements) which should prevent generation of unpromising instances of macros.

Acknowledgements

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

References

Alhossaini, M., and Beck, J. C. 2009. Learning instance-specific macros. In *ICAPS Workshop on Planning and Learning*.

Botea, A.; Enzenberger, M.; Müller, M.; and Schaeffer, J. 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.

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

Chen, Y., and Yao, G. 2009. Completeness and optimality preserving reduction for planning. In *Proceedings of IJCAI*, 1659–1664.

Chrpa, L., and Barták, R. 2009. Reformulating planning problems by eliminating unpromising actions. In *Proceedings of SARA*, 50–57.

Chrpa, L., and McCluskey, T. L. 2012. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of ECAI*, 240–245.

Chrpa, L. 2010a. Combining learning techniques for classical planning: Macro-operators and entanglements. In *Proceedings of ICTAI*, volume 2, 79–86.

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

Coles, A. J., and Coles, A. I. 2010. Completeness-preserving pruning for optimal planning. In *Proceedings of ECAI*, 965–966.

Coles, A.; Fox, M.; and Smith, A. 2007. Online identification of useful macro-actions for planning. In *Proceedings of ICAPS*, 97–104.

Dawson, C., and Siklóssy, L. 1977. The role of preprocessing in problem solving systems. In *Proceedings of IJCAI*, 465–471.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3/4):189–208.

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

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

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

Kautz, H.; Selman, B.; and Hoffmann, J. 2006. Satplan: Planning as satisfiability. In *Proceedings of the fifth IPC*.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of ICAPS*.

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

McCluskey, T. L. 1987. Combining weak learning heuristics in general problem solvers. In *Proceedings of IJCAI*, 331–333.

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

Newton, M. A. H., and Levine, J. 2010. Implicit learning of compiled macro-actions for planning. In *Proceedings of ECAI*, 323–328.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of ICAPS*, 256–263.

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

Rintanen, J. 2012. Engineering efficient planners with SAT. In *Proceedings of ECAI*, 684–689.

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