Exploring the Synergy between two Modular Learning Techniques for Automated Planning

Raquel Fuentetaja
Planning and Learning Group
Universidad Carlos III. Spain

Lukáš Chrpa and **Thomas L. McCluskey** and **Mauro Vallati**
PARK research group
University of Huddersfield

Abstract

In the last decade the emphasis on improving the operational performance of domain independent automated planners has been in developing complex techniques which merge a range of different strategies. This quest for operational advantage, driven by the regular international planning competitions, has not made it easy to study, understand and predict what combinations of techniques will have what effect on a planner's behaviour in a particular application domain. In this paper, we consider two machine learning techniques for planner performance improvement, and exploit a modular approach to their combination in order to facilitate the analysis of the impact of each individual component. We believe this can contribute to the development of more transparent planning engines, which are designed using modular, interchangeable, and well-founded components. Specifically, we combined two previously unrelated learning techniques, entanglements and relational decision trees, to guide a "vanilla" search algorithm. We report on a large experimental analysis which demonstrates the effectiveness of the approach in terms of performance improvements, resulting in a very competitive planning configuration despite the use of a more modular and transparent architecture. This gives insights on the strengths and weaknesses of the considered approaches, that will help their future exploitation.

Introduction

Automated planning is one of the most prominent AI challenges; it has been studied extensively for several decades and led to many real-world applications. The International Planning Competition (IPC), a nearly bi-annual major event of the planning community, has encouraged the development of planning engines to a level of optimisation that has led to the utilisation of some of these engines in applications such as narrative generation (Porteous, Cavazza, and Charles 2010), machine tool calibration (Parkinson, Longstaff, and Fletcher 2014), traffic management (Jimoh et al. 2013), to mention a few. These planning engines incorporate well known techniques, like approximations of the $h+$ heuristic (Betz and Helmert 2009), and generation and use of landmarks (Hoffmann, Porteous, and Sebastia 2004). Thus, these competition tuned engines are serving the purpose of promoting the use of planning in a wide variety of applications. For the sake of optimisation, the integration and implementation of these techniques has been performed in such a way that the effect of one particular component, within the overall performance of the system, is difficult to judge. Under these circumstances, machine learning, in the context of automated planning has a long history. A recent and extensive review can be found in (Jiménez et al. 2012). One important topic in planning regards learning control knowledge, i.e., domain-dependent knowledge related to the structure of related planning tasks that is useful to guide the search algorithm. Examples can be found in (de la Rosa et al. 2011), that describes a technique to learn control knowledge in the form of relational decision trees, and (Krajnansky et al. 2014), that describes a technique that learns control rejection rules. Other tendencies in this line are macro-action learning (Botea et al. 2005; Chrpa, Vallati, and McCluskey 2014) and learning to improve heuristic functions (Yoon, Fern, and Givan 2008). Machine learning has been applied to planning for other purposes also, such as learning domain theory (Yang, Wu, and Jiang 2007; Cresswell, McCluskey, and West 2009); learning to predict planner performance (Roberts and Howe 2009) or learning portfolio configurations (Cenamor, De La Rosa, and Fernández 2013).

In the context of automated planning, very few works have integrated different learning techniques in the same framework. Usually, a typical system involves a planning engine and a single learning approach. This is mainly due to the fact that it is much easier to develop planner-specific tools, rather than general ones. On the other hand, planner-specific tools are rarely re-implemented for a different engine, and their impact on a different engine is unpredictable. Examples of works in planning that combined learning approaches are the work of Durán et. al. (García, Fernández, and Borrajo 2006), PbP (Gerevini, Saetti, and Vallati 2014), ASAP (Vallati, Chrpa, and Kitchin 2013) and Cedalion (Seipp et al. 2015). The first, combines macro-actions with control knowledge rules for PRODIGY.[1] The learning task consists of finding a set of rules to decide when to use some previously acquired macro-actions. PbP and ASAP configure a domain-specific portfolio of planners (a single planner in the case of ASAP), which possibly exploits a reformulated version of the domain model. Finally, cedalion gener-

[1] A planner from the nineties that uses a lifted representation and performs a kind of bi-directional search.

ates a sequential portfolio of automatically configured planners.

This paper has its roots in the ideas of combining general learning planner-independent techniques for generating specific heuristics, which has a long history in several fields of AI (McCluskey 1987; García, Fernández, and Borrajo 2006). Part of the appeal of this line of research is that techniques can be studied in isolation, and in combination, in such a way that the performance change of a planner can be traced to the effect of a particular learning technique. Our work is also related to the idea of the modular construction of planners as espoused by Hertzberg (1995); the benefits of a modular approach are numerous. Modular techniques are easier to efficiently develop and to maintain; moreover, new techniques can quickly be added or exchanged.

This paper's contributions are: (i) a modular approach to the construction of planning engines, building on a "vanilla" search strategy; (ii) adding to this search strategy a combination of two unrelated learning techniques, namely decision trees (de la Rosa et al. 2011) and entanglements (Chrpa and McCluskey 2012), which both acquire domain specific heuristics from simple training sessions; (iii) a demonstration of the relative strengths of the techniques, and (iv) a demonstration of how the techniques can be combined in order to reach a level of performance comparable to competition planners.

Our aim is to re-invigorate research into planners which are made up of modular, interchangeable, and well-founded components, which learn domain specific heuristics, and which can be combined to achieve competition winning performance.

This paper is organised as follows. We first provide background information on classical planning, entanglements and decision trees. Next, we describe the different ways in which the techniques can be combined, and we present in detail our experimental analysis and results, followed by concluding remarks and future work.

Background

In this section we provide the relevant background on classical planning, and we introduce the two modular learning techniques that will be investigated, namely entanglements (Chrpa and McCluskey 2012) and Roller (de la Rosa et al. 2011).

Classical Planning

Classical planning deals with finding a partially or totally ordered sequence of actions transforming the static, deterministic and fully observable environment from some initial state to a desired goal state.

In the classical representation *atoms* are predicates. *States* are defined as sets of ground predicates. A *planning operator* $o = (\mathit{name}(o), \mathit{pre}(o), \mathit{eff}^-(o), \mathit{eff}^+(o))$ is specified such that $\mathit{name}(o) = \mathit{op\_name}(x_1, \ldots, x_k)$ ($\mathit{op\_name}$ is an unique operator name and $x_1, \ldots x_k$ are variable symbols (arguments) appearing in the operator), $\mathit{pre}(o)$ is a set of predicates representing operator's precondition, $\mathit{eff}^-(o)$ and $\mathit{eff}^+(o)$ are sets of predicates representing operator's negative and positive effects. *Actions* are ground instances of planning operators. An action $a = (\mathit{pre}(a), \mathit{eff}^-(a), \mathit{eff}^+(a))$ is *applicable* in a state s if and only if $\mathit{pre}(a) \subseteq s$. Application of a in s (if possible) results in a state $(s \setminus \mathit{eff}^-(a)) \cup \mathit{eff}^+(a)$.

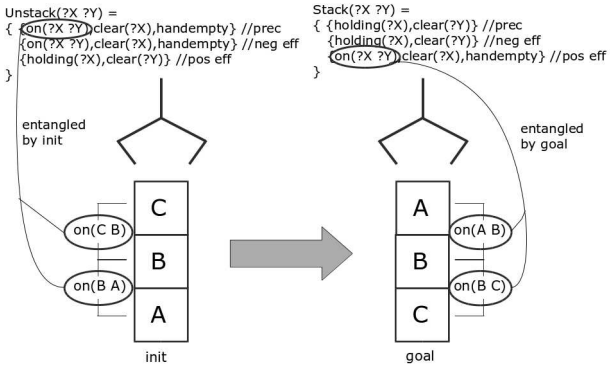

Figure 1: An illustrative example of outer entanglements.

A *planning domain* is specified via sets of predicates and planning operators. A *planning problem* is specified via a planning domain, initial state and set of goal atoms. A *solution plan* is a sequence of actions such that a consecutive application of the actions in the plan (starting in the initial state) results in a state that satisfies the goal.

Entanglements

Entanglements (Chrpa and McCluskey 2012) are relations between planning operators and predicates which aim to capture the *causal* relationships characteristic for a given class of planning problems. Once captured, they are able to prune unpromising alternatives in the search space. There are two kinds of entanglements: outer and inner.

Outer entanglements are relations between planning operators and initial or goal predicates. In other words, outer entanglements say that to solve a given planning problem some operators can be restricted to be applicable only when some preconditions are in the initial state or some positive effects are target goals. In the well-known BlocksWorld domain, it can be observed that unstacking blocks only occurs from their initial positions. In this case an *entanglement by init* will capture that if an atom `on(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 positions. Then, an *entanglement by goal* will capture the constraint that if atom `on(b a)` is achieved by a corresponding instance of operator `stack(?x ?y)` then (`stack(b a)`) must be a goal atom. Such an observation is illustrated in Figure 1.

Encoding outer entanglements is done by introducing supplementary static predicates having the same arguments as predicates involved in the outer entanglement relations. Instances of these static predicates, corresponding to instances of original predicates in the initial or goal state, are added to the initial state. For example, if `on(a b)` is in the initial state, then `static-on(a b)` is added to the initial state.

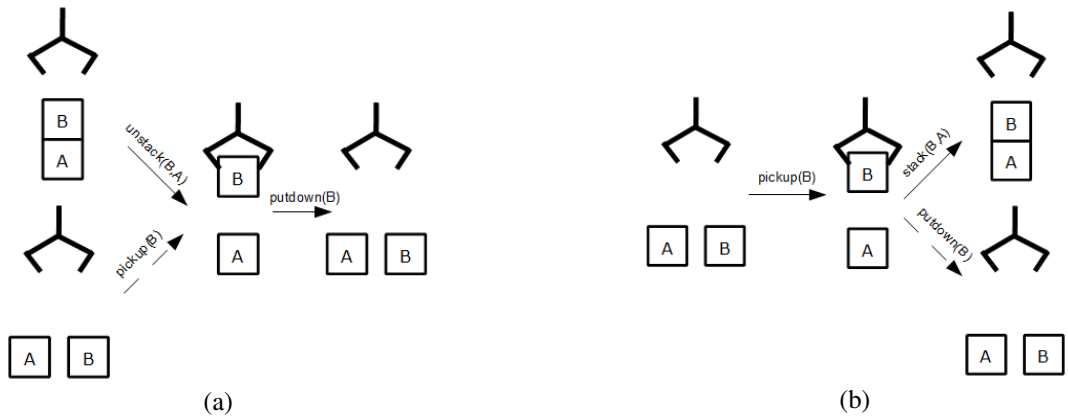

Figure 2: A motivating example for inner entanglements, concretely entanglements by preceding (a) and by succeeding (b).

These supplementary static predicates are added into preconditions of operators involved in the outer entanglement relations, so they allow only such operators' instances that follow conditions of outer entanglements.

Inner entanglements are relations between pairs of planning operators and predicates. Inner entanglements capture situations where some operator is an exclusive *achiever* or *requirer* of a predicate to or from another operator. In the Blocksworld domain, it may be observed that operator `pickup(?x)` achieves predicate `holding(?x)` exclusively for operator `stack(?x ?y)` (and not for operator `putdown(?x)`), i.e., `pickup(?x)` is *entangled by succeeding* `stack(?x ?y)` with `holding(?x)` (for illustration, see Figure 2 right). 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)`), i.e., `putdown(?x)` is *entangled by preceding* `unstack(?x ?y)` with `holding(?x)` (for illustration, see Figure 2 left).

Encoding inner entanglements into planning domains and problems must ensure *achiever* and *requirer* exclusivity given by these inner entanglements. It is done by using supplementary predicates, *locks*, which prevent applying certain instances of operators in some stage of the planning process. An instance of an operator having a *lock* in its precondition cannot be applied after applying an instance of another operator (*locker*) having the *lock* in its negative effects until an instance of some other operator (*releaser*) having the *lock* in its positive effects has been applied. For example, a situation where `pickup(?x)` is *entangled by succeeding* `stack(?x ?y)` with `holding(?x)` is modelled such that `pickup(?x)` is the *locker* for `putdown(?x)` and `stack(?x ?y)` is the *releaser* for `putdown(?x)`.

Entanglements are extracted from training plans, solutions of simpler problems, in such a way that we check for each operator/pair of operators and related predicates whether the entanglement conditions are satisfied in all the training plans (some error rate might be allowed since training plans may consist of 'flaws' such as redundant actions). Although applying extracted entanglements might cause loss of solvability of some non-training problems, it has been shown empirically that it does not happen (or happens very rarely) if the structure of the training problems is similar to the structure of the testing problems. For deeper insights, see (Chrpa and McCluskey 2012).

Decision Tree Learning with Roller

Roller (de la Rosa et al. 2011) is a learning/planning system that learns relational decision trees for planning by induction. These decision trees contain control knowledge that sort the applicable actions in a given node for state space search planners. The Roller learning module receives a planning domain model and a set of training problems as inputs. Then, it extracts training instances from the search trees generated to solve training problems. The training instances are used to train TILDE (Blockeel and De Raedt 1998), an off-the-shelf relational classification tool that generates decision trees.

Decision trees are binary trees in which each node represents a query about a feature expressed as a positive predicate logic literal. The main features considered by Roller are extracted from the *context* of a search state. They are: (1) *helpful actions*, i.e., whether a particular action is helpful or not in the current state. Given a state, the set of helpful actions is computed heuristically and determines the most useful applicable actions. Specifically, helpful actions are those applicable actions that add a subgoal of the relaxed plan used to compute the heuristic of the FF planner (Hoffmann and Nebel 2001); (2) *target pending and achieved goals*, i.e., whether a target goal is pending or achieved in the current state; and (3) *static facts*, i.e., whether a fact is defined in the initial state and no action modifies it.

Roller generates two types of decision trees: operator trees and binding trees. The leaves of operator trees provide ordering of applicable operators in a given state, depending on the features indicated by the path from the root node to the leaf. Only one operator tree is generated per domain. Binding trees allow instantiations of each operator (actions) to be sorted into order. For a domain, Roller generates one binding tree per operator.

Roller operates in two phases: the learning phase and the planning phase. In the learning phase, the set of deci-

sion trees for a domain is generated using a set of training problems. In the planning phase, a state space search algorithm is applied to solve a new problem in that domain. The base search algorithm is Depth-First Search (DFS) with chronological backtracking endowed with the helpful actions heuristic and a strategy to explore first the space generated by helpful actions. When this base algorithm considers Roller decision trees, they are used to compute a priority for each applicable action in every expanded state. Then, applicable actions are sorted in decreasing priorities and reclassified as helpful/non-helpful.[2] The only actions considered as non-helpful are those with a priority of zero. The Roller planning algorithm is the union of both the base algorithm and the mechanisms to sort and re-classify actions using decision trees (originally this algorithm was called Depth-first Helpful Context Policy (DHCP) (de la Rosa et al. 2011)). This is a complete algorithm, since it considers all applicable actions in every node.

As an example, Figure 3 shows a part of the operator tree learned in the Depots domain. The branch in bold states that if there is a helpful action of type `load` and a helpful action `lift` and there is a non-helpful action `unload` for the current state, the operators have to be applied in the order given by the reached leaf. The number near each operator is the number of examples covered by the leaf during the training phase in which the operator was selected (i.e. it belongs to an optimal solution). This number is used as operator priority ($op\_priority$). For instance, $op\_priority(\texttt{lift}) = 3$, while $op\_priority(\texttt{drive}) = 1$.

Figure 4 shows part of the bindings tree for the operator `drive` in Depots. If an action of type `drive` matching with the branch in bold is applicable in the current state, the tree would recommend to reject it, since its $selection\_ratio$ is zero. The selection ratio of each action is computed as:

$$selection\_ratio(a) = \frac{selected(a)}{selected(a) + rejected(a)}$$

Again, $selected(a)$ ($rejected(a)$) is the number of examples covered by the leaf for which the action was selected (rejected) in training instances.

The global priority of an action a is computed as:

$$priority(a) = op\_priority(op(a)) + selection\_ratio(a)$$

where $op(a)$ stands for the operator name of a.

Combining Entanglements and Roller

For inductive learning tasks, one of the reasons that justify the use of a combination of several learning techniques, instead of a single one, is representational (Dietterich 2000). It could be the case that the target function cannot be represented. This is one of the ways in which a learning algorithm fails. There are also statistical reasons, related with the number of training problems; and computational reasons, related with the use of greedy algorithms to generate the model. However, we focus on the representational issue, since combining entanglements and relational decision trees learnt by Roller improves this aspect specifically.

[2]Ties are broken arbitrarily.

```
selected(-A,-B,-C)
helpful_load(A,B,-D,-E,-F,-G) ?
+--yes: helpful_lift(A,B,-H,-I,-J,G) ?
| +--yes: nothelpful_unload(A,B,-K,-L,-M,-N) ?
| | +--yes: [lift] 6.0
| |         [[drive:1.0,lift:3.0,drop:1.0,load:1.0,unload:0.0]]
| | +--no: [lift] 12.0
...
```

Figure 3: Partial view of the operators tree in Depots.

```
selected_drive(-A,-B,-C,-D,-E,-F)
helpful_drive(A,B,C,D,E) ?
+--yes: nothelpful_load(A,B,-G,-H,C,D) ?
| +--yes: helpful_unload(A,B,-I,-J,C,D) ?
| | +--yes: [rejected] 12.0 [[selected:0.0,rejected:12.0]]
| | +--no: nothelpful_lift(A,B,-K,-L,-M,E) ?
| |    +--yes: [selected] 20.0 [[selected:18.0,rejected:2.0]]
| |    +--no: helpful_unload(A,B,-N,-O,-P,-Q) ?
...
```

Figure 4: Partial view of the bindings tree for the operator drive in Depots.

The representational ability of Roller's decision trees depends strongly on the defined features that are included as queries there. In Roller, these features are related to the context of the current state, defined by (i) the applicable operators, divided into helpful and not helpful; and (ii) own characteristics of the state: facts in this state that are either static or achieved/pending target goals

Entanglements represent knowledge of a different nature. Outer entanglements restrict some operators to be applicable only when some preconditions are in the initial state or some positive effects are target goals. Inner entanglements impose constraints by relating pairs of operators and predicates, so that some operators are exclusive *achievers*/*requirers* of a predicate to/from other operators. This knowledge does not depend exclusively on the context of the current state as it is defined in Roller. Thus, Roller can not represent it.

Given the different nature of the knowledge represented by entanglements and Roller, it seems that considering both types of knowledge will improve the representational power of the learnt model. This idea is also supported by the fact that previous results for both techniques considered separately showed they are valuable to guide search algorithms for solving planning tasks (Chrpa and McCluskey 2012; de la Rosa et al. 2011).

In this paper, we combine entanglements and decision trees in a modular way, applying them independently in sequence. This is possible because entanglements are expressed by reformulating the planning domain and problem models. For combining the learning techniques, there are two possibilities:

1. **E+R**: Extract entanglements; apply Roller learning over

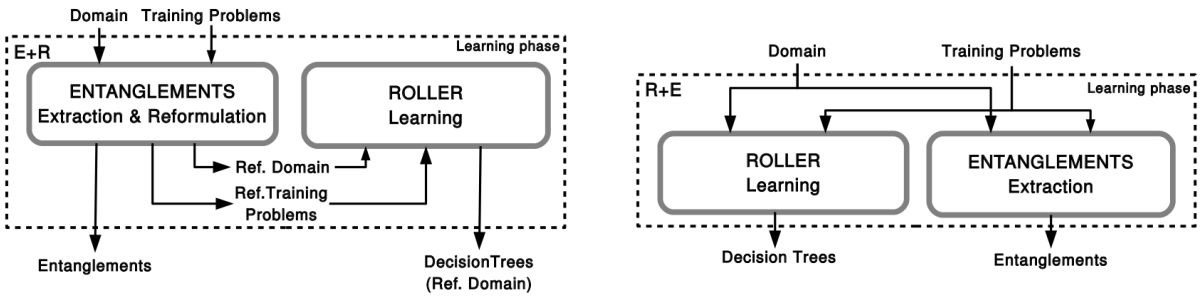

Figure 5: Schemes for the learning phase: E+R (left), R+E (right).

the reformulated domain model and reformulated training problems; and then Roller planning with the reformulated domain using the decision trees learnt for it;

2. **R+E**: Apply Roller learning over the original domain; extract entanglements; and then apply Roller planning with the reformulated domain using decision trees learnt from the original domain.

Figure 5 shows the learning phase for both integration schemes. Inputs are a domain model and a set of training problems.

In the **R+E** sequence, decision trees are not aware of entanglements-related knowledge, therefore this combination is straightforward. In the **E+R** sequence, the structure of decision trees is modified because they are learnt from a reduced search space. Regarding the generated knowledge, entanglements-related knowledge is incorporated naturally in decision trees since this knowledge is expressed by means of domain predicates. These new predicates are considered by the learning algorithm to be included as queries in decision trees. In this sense, the representational power of decision trees increases. The effect of introducing entanglements enables Roller to create five new entanglements-related features. Three of them are related to inner entanglements by succeeding and two to outer entanglements. Predicates introduced for inner entanglements by preceding do not generate new features as they are neither target goals nor static facts.

Predicates introduced in the reformulated domain for inner entanglements by succeeding ($o1$ generates p exclusively for $o2$) are represented by a positive literal $L = o1\_o2\_succ\_p_{pred}\_(p_{args})$, where p_{pred} is the predicate symbol of p, and p_{args} its lifted arguments. L is included in the initial state, target goals, and as precondition of all other operators requiring p but o2 (see (Chrpa and McCluskey 2012) for a detailed explanation). This new predicate enables Roller to formulate new queries related to information about previously applied operators with the following semantics:

- **`pending_goal(`L`)?`**: *true* when the first operator involved in the entanglement has been applied and then the plan should contain the second operator of that entanglement.
- **`achieved_goal(`L`)?`**: *true* when either any or both operators involved in the entanglement have been applied.
- **`static_fact(`L`)?`**: *true* when there are no corresponding instances of operators for a particular instantiation of the entanglement (it often happens in a combination with outer entanglements).

Outer entanglements introduce in the domain new static facts represented by literals $L_{init} = stai\_p_{pred}(p_{args})$ or $L_{goal} = stag\_p_{pred}(p_{args})$, meaning that facts with the form $p_{pred}(p_{args})$ should appear at the initial state (entanglement by init) or in the goals (entanglement by goal) respectively. These new static facts enable Roller to formulate the following queries related to the initial state and goals:

- **`static_fact(`L_{init}`)?`**: *true* when the positive literal L_{init} involved in an outer-by-init entanglement belongs to the initial state.
- **`static_fact(`L_{goal}`)?`**: *true* when the positive literal L_{goal} involved in an outer-by-goal entanglement is a target goal, which in fact is the disjunction of two existing features: pending target goal or achieved target goal.

These features can be interpreted as being an extension of Roller's contexts. Thus, they allow the learning algorithm to consider more information. When this information is useful to discriminate among different cases related to the order of actions/bindings, decision trees will include these features in queries.

The planning phase is the same for both the considered learning approaches. In the planning phase inputs are the original domain model and a problem. In a first step they are reformulated to include entanglements. Then, the Roller planning algorithm is applied using the decision trees generated in the learning phase.

Summarising, through this work we derive new heuristics for planning as to what learning techniques to combine together. They should be effective in their own right, learning effective domain-specific heuristics for each domain over a range of domains and learn distinct concepts. In both (**E+R** or **R+E**), we believe the planning algorithm will perform better than the individual Roller planning algorithm on the original domain because entanglements will prune some unpromising search alternatives for Roller. Hence, in the experimental section we compare both approaches which provides

an idea on how the knowledge in decision trees is affected by entanglements. Our hypothesis is that the **E+R** approach is the better order in which it can "accumulate" learning power. We explore this hypothesis empirically in the following section.

Experimental Analysis

The experiments in this section are designed to: (i) determine in which measure the combinations of entanglements and Roller are stronger than both techniques individually; and (ii) compare the behaviour of a planning system exploiting them with current state-of-the-art planning techniques.

We considered problem instances from the well-known benchmark domains used in the learning track of the last IPC (IPC-7) (Coles et al. 2012): Barman, Blocksworld, Depots, Gripper, Parking, Rovers, Satellite, Spanner and TPP. For each domain, we used the existing 30 benchmark problems as testing instances. For Roller, we generated an initial set of 50 training problems per domain, using available generators from IPC-7. As is usual in learning systems of this type (de la Rosa et al. 2011; Krajnansky et al. 2014), the training instances are extracted from solutions to training problems, so its number varies per domain. Specifically, learning instances consist of tuples $< c_i, a_{i+1} >$, where c_i is the context of the state s_i in the solution path, and a_{i+1} is the action applied in that state. We set a time-limit of 120 seconds to solve each training problem. Roller learning is inductive, so the best value for the training parameters for a domain is unknown. We selected them by experience and after some trials. A deeper study on the impact of selecting different values would be interesting, but is out of the scope of this paper. To extract entanglements we generated 10 additional instances. In practice, the whole learning (i.e., offline extraction of entanglements and decision trees for all the considered domains) process required at most few minutes per domain.

A runtime cutoff of 900 CPU seconds (15 minutes, as in the learning tracks of IPC) was used for testing runs. All the experiments were run on 3.0 Ghz machine CPU with 4GB of RAM. In this analysis, speed and quality score, as defined in IPC-7, are used. For a planner $\mathcal{C}$ and a problem p, $Score(\mathcal{C}, p)$ is 0 if p is unsolved, and $1/(1 + \log_{10}(T_p(\mathcal{C})/T_p^*))$ otherwise. T_p^* is the minimum time required by any considered planner to solve the problem. The quality score is defined as $Score(\mathcal{C}, p)$, which is 0 if p is unsolved, and $Q_p^*/Q(\mathcal{C}_p)$ otherwise ($Q_p^* \leq Q(\mathcal{C})_p$ for any $\mathcal{C}$). Q_p^* is the best plan quality found by any considered planner. Quality is measured in terms of number of actions. The IPC score on a set of problems is given by the sum of the scores achieved on each considered problem.

The base search algorithm can be used as-is, i.e., as a classical depth-first search with no heuristic, or it can be guided with a heuristic able to sort applicable actions instead of control knowledge (decision trees). Without any heuristic, the base search algorithm solves very few problems. Therefore, we have selected for comparison the FF approximation to h^+ (FF heuristic), since non-learning forward planners usually make choices with approximations to h^+. Such heuristic estimates the distance to the goal by solving a simplified planning task where all negative effects (i.e., effects that make false some predicate) are ignored. We will refer as DF-CK (Depth-First with Control Knowledge) to the base algorithm with control knowledge (decision trees) and as DF-FF to the same system using the FF heuristic instead of decision trees to sort applicable actions.

Firstly, it is important to understand the performance of DF-FF. For assessing its performance, we compared it with the state-of-the-art of domain-independent and learning-based planning; LAMA-11 (Richter, Westphal, and Helmert 2011; Richter and Westphal 2010), Yahsp3 (Vidal 2014) and PbP2 (Gerevini, Saetti, and Vallati 2009; 2014). LAMA-11 is the winner of the satisficing track of the IPC-7, Yahsp3 won the agile track of the IPC-8, while PbP2 is the winner of the IPC-7 learning track. The number of solved problems of DF-FF is very similar to LAMA-11, respectively 57 and 59. On the other hand, both of them are far away from PbP2, which is able to solve 238 testing problems. Yahsp3 is somehow in the middle, since it is able to solve 115 instances. It is worth to know that such testing problems are very large, in terms of number of objects involved, which make them hard to handle for most of the existing domain-independent planners.

In order to evaluate the impact of the considered types of knowledge, it should be noted that the knowledge extracted under the form of entanglements and decision trees can be used separately, i.e., exploiting only one of them for solving a testing problem, or combined, as stated in the previous section. When exploiting just entanglements, we use the DF-FF algorithm. For all the other cases involving decision trees DF-CK is used. Three different sets of entanglement can be extracted, namely, *inner*, *outer* and *both*, per domain. The last set, *both*, considers all *inner* and *outer* entanglements together.

Table 1 shows the results of all the possible combinations for each domain. The name of the combination indicates the order in which the different types of knowledge have been extracted, i.e., RO indicates a **R+E** combination that involves *outer* entanglements, while OR indicates the corresponding **E+R** one. Results are shown in terms of speed score and solved problems. The results shown in Table 1 (barman results are omitted since none of the considered techniques solved any testing instance of that domain) indicate that: (i) both the types of considered knowledge, when exploited singularly, are usually able to improve the performance of the base algorithm; (ii) there is a strong synergy between them in most of the benchmark domains; (iii) their **E+R** combination usually achieves better performance than the corresponding **R+E** one. This confirms our intuition that decision trees endowed with entanglements-related features learnt from the reduced search spaces contain more useful knowledge. For instance, the operators tree generated in Blocksworld for *both* and *inner* entanglements is able to capture information about applying the operators involved in entanglements by succeeding consecutively. Also, it identifies these cases as the only ones in which the second operator and any other has to be applied. Thus, the binding trees for these second operators (stack and putdown) contain only one leaf node recommending without doubt its selection with

	BW	Depots	Gripper	Parking	Rovers	Satellite	Spanner	Tpp	Total
RB	15.5 (30)	13.9 (16)	24.1 (29)	25.9 (30)	2.9 (5)	17.1 (18)	29.8 (30)	0.0 (0)	129.2 (158)
RO	3.8 (11)	26.4 (30)	24.1 (29)	**29.6** (30)	2.9 (5)	17.1 (18)	29.8 (30)	22.6 (24)	156.3 (177)
RI	0.0 (0)	5.8 (24)	16.0 (25)	25.9 (30)	2.8 (5)	14.6 (16)	29.8 (30)	0.0 (0)	94.8 (130)
BR	**29.0** (30)	**26.8** (30)	27.6 (30)	21.7 (28)	3.1 (5)	**20.0** (20)	**29.9** (30)	0.0 (0)	158.1 (173)
OR	26.5 (30)	26.8 (30)	**29.0** (30)	**29.6** (30)	**24.0** (24)	**20.0** (20)	29.8 (30)	**26.6** (30)	**212.1** (224)
IR	13.9 (29)	1.9 (8)	16.0 (25)	21.7 (28)	2.8 (5)	14.6 (16)	29.8 (30)	0.0 (0)	100.7 (141)
B	18.6 (30)	1.4 (6)	0.4 (1)	1.3 (4)	10.1 (20)	0.0 (0)	0.0 (0)	14.0 (20)	45.8 (81)
O	19.6 (30)	18.1 (30)	0.4 (1)	3.0 (9)	10.1 (20)	0.0 (0)	0.0 (0)	22.2 (30)	73.4 (120)
I	0.0 (0)	0.4 (2)	0.0 (0)	1.3 (4)	9.9 (20)	0.0 (0)	0.0 (0)	9.9 (24)	21.4 (50)
R	11.6 (29)	2.2 (9)	16.0 (25)	**29.6** (30)	2.8 (5)	14.6 (16)	29.8 (30)	11.4 (23)	117.9 (167)
base	0.4 (2)	0.6 (3)	0.0 (0)	3.0 (9)	9.9 (20)	0.0 (0)	0.0 (0)	9.4 (23)	23.2 (57)

Table 1: Speed score (number of problems solved) of all the combinations of Roller (R) and entanglements (B both, I inner, O outer); base is DF-FF. The name of the combination indicates the order in which the different types of knowledge have been extracted. Values in bold indicate the best results, also considering hidden decimals.

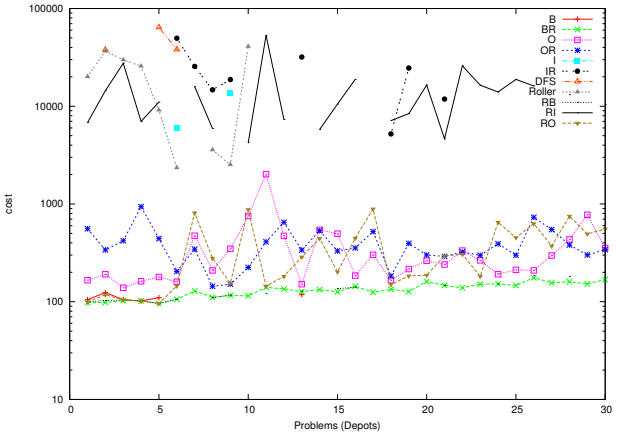

Figure 6: Quality of solutions found on Depots by all the combinations of Roller (R) and entanglements (B both, I inner, O outer).

any bindings. In fact, entanglement-related features appear in decision trees for all domains, which is a sign that they are relevant. Though in Satellite and Parking they appear in less measure, only in the bindings tree of one operator.

Moreover, we observed that even if entanglements and decision trees are mainly focused on improving the runtime, also the quality of resulting plans is usually improved, and the number of nodes expanded is decreased by several orders of magnitude. Figure 6 shows the quality of solutions found by all the considered combinations on the Depots testing problems.

On the other hand, while in all the tested domains the use of both trees and entanglements offers better results than the use of entanglements alone, there are some domains in which it does not improve the results of Roller alone. This is the case of Parking and Spanner. In Spanner, each problem has a unique solution and the knowledge extracted by Roller leads the algorithm directly to it, without any room for improvement. Moreover, the extracted *outer* entanglements are not very informative, since they prune a part of the search space that is never explored by the planning engine. No *inner* entanglements were extracted there. In Parking, only one *inner* entanglement and no *outer* ones are extracted. The extracted entanglement, however, does not improve even against Roller alone, which can be explained by the fact that features related to this entanglement hardly appear in decision trees. Parking is the only domain in which learing Roller from the original models, rather than from reformulated ones, usually improves the performance of DF-CK. In Barman, none of the considered techniques solved any of the testing instances. This is probably due to the fact that there is not a common sequence of actions that is useful for reaching some of the goals and, moreover, considering all the goals at the same time make the solving process extremely hard. The best approach is to solve each goal separately, but neither Roller nor entanglements can capture this kind of knowledge.

Finally, we evaluate the effectiveness of the combined automatically learnt knowledge exploited by DF, with regards to the state of the art of automated planning. We called such a system [O]ER: it is the base algorithm that exploits outer entanglements (whenever available) and Roller, combined by the **E+R** approach. We selected outer entanglements because of their generally good impact on the performance of the base algorithm.We compared [O]ER with PbP2: it is learning-based and, as outlined before, its performance is significantly better than both LAMA-11 and Yahsp3. The results of this comparison are shown in Table 2. It should be noted that PbP2 configures different portfolios while optimising for speed or quality, for the number of solved problems we considered the largest one. We compared both systems with and without the extracted knowledge, in order to evaluate the overall performance, in terms of speed, quality and coverage, and also to understand the impact of the knowledge on the respective systems. [O]ER solves less problems than PbP2, but its Speed score is slightly better. From the quality point of view, PbP2 achieves a better score. This is mainly due to the fact that [O]ER is not able to solve any problem in Barman. It is worth mentioning that the only planner included in PbP2 that is able to solve testing instances of barman is SGPlan5 (Chen, Wah, and Hsu 2006), which is based on a partition-and-resolve strategy. Interest-

	Speed score				Quality score				# Solved			
Domain	DF	[O]ER	PbP2nk	PbP2	DF	[O]ER	PbP2nk	PbP2	DF	[O]ER	PbP2nk	PbP2
Barman	0.0	0.0	6.7	**30.0**	0.0	0.0	22.5	**30.0**	0	0	23	**30**
BW	0.4	**30.0**	4.3	11.4	0.0	**29.6**	11.9	29.3	2	30	17	30
Depots	0.6	**30.0**	2.0	8.0	0.0	17.8	7.2	**24.7**	3	**30**	8	27
Gripper	0.0	22.7	10.3	**30.0**	0.0	**30.0**	18.6	27.9	0	30	23	30
Parking	2.7	**30.0**	0.0	2.8	2.2	**29.6**	0.0	4.6	9	**30**	0	8
Rover	7.3	15.9	9.8	**27.4**	15.0	15.3	15.7	**29.0**	20	24	17	**28**
Satellite	0.0	7.2	4.7	**30.0**	0.0	15.3	9.6	**29.1**	0	20	11	**30**
Spanner	0.0	23.7	5.3	**30.0**	0.0	30.0	13.0	30.0	0	30	13	30
TPP	10.3	**29.5**	2.2	15.0	**22.0**	16.9	9.4	14.4	23	**30**	8	25
Total	21.4	**189.0**	45.3	184.6	39.2	184.5	108.0	**219.0**	57	224	120	**238**
Δ Score	**167.6**		139.3		**145.3**		111.0		**167**		118	

Table 2: Speed score, Quality score and number of problems solved by the proposed approach and PbP, with and without the automatically learned knowledge. PbP2nk indicates PbP2 the does not exploit the extracted knowledge. DF indicates DF-FF.

ingly, in TPP DF-FF achieves the best quality score. The FF heuristic works well in this model. On the other hand, the knowledge exploited by [O]ER is aimed at finding a solution quickly, by sorting the applicable actions (decision trees) and pruning the search space (entanglements). In fact, in the TPP domain, the speed score of [O]ER is significantly increased; however, this leads to lower quality plans.

Regarding the usefulness of the knowledge extracted, decision trees and entanglements have a substantial impact on all the considered metrics, as indicated by the delta between the scores of DF-FF and [O]ER. Moreover, such deltas are always bigger than PbP2 ones, thus we can argue that the quality of the combined knowledge exploited by [O]ER is very good.

Conclusion

Automated planning is an area with a rich spectrum of possibilities to apply machine learning. It provides a setting that allows the definition of many different learning tasks that can be exploited together with the general objective of optimizing the search to improve planners' performance.

The trend for the last decade for improving planners' performance has been in developing complex techniques merging different strategies, which makes it difficult to study and understand the planners' behaviour. In this paper we combined two learning techniques, and exploited a modular approach in order to facilitate the analysis of the impact of each individual component. We believe this can contribute to the research of planners designed by modular, interchangeable, and well-founded components. Specifically, we combined two previously unrelated learning techniques, entanglements and relational decision trees, to guide a "vanilla" search algorithm.

A large experimental analysis demonstrated the effectiveness of the approach. From the machine learning perspective, the combination produces more useful knowledge than the application of both techniques separately, which is reflected in the behaviour of the planner. From the planning perspective, we obtained competitive performance against PbP2, the winner of the last IPC learning track. This result is significant: PbP2 is a portfolio that includes several state of the art planners, while [O]ER is based on a single heuristic search algorithm.

For the future work, we plan to include more learning modules such as macro-operators learning and a wider experimental analysis involving different planning techniques. We are also interested in investigating techniques – based on empirical predictive models (Hutter et al. 2014) – for automatically selecting and combining learning modules, accordingly to the performance of the exploited planner.

Acknowledgements

This work has been partially supported by the Spanish project TIN2011-27652-C03-02. The research was funded by the UK EPSRC Autonomous and Intelligent Systems Programme (grant no. EP/J011991/1).

References

Betz, C., and Helmert, M. 2009. Planning with h^+ in theory and practice. In *KI 2009: Advances in Artificial Intelligence*, volume 5803 of *Lecture Notes in Computer Science*, 9–16.

Blockeel, H., and De Raedt, L. 1998. Top-down induction of first-order logical decision trees. *Artificial Intelligence* 101(1-2):285–297.

Botea, A.; Enzenberger, M.; Mller, M.; and Schaeffer, J. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research (JAIR)* 24:581–621.

Cenamor, I.; De La Rosa, T.; and Fernández, F. 2013. Learning predictive models to configure planning portfolios. In *ICAPS Workshop on Planning and Learning (PAL 2013)*, 14–22.

Chen, Y.; Wah, B. W.; and Hsu, C.-W. 2006. Temporal planning using subgoal partitioning and resolution in SGPlan. *Journal of Artificial Intelligence Research (JAIR)* 26:323–369.

Chrpa, L., and McCluskey, T. L. 2012. On exploiting structures of classical planning problems: Generalizing entanglements. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 240–245.

Chrpa, L.; Vallati, M.; and McCluskey, T. L. 2014. MUM: A technique for maximising the utility of macro-operators by constrained generation and use. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 65–73.

Coles, A.; Coles, A.; Olaya, A. G.; Jiménez, S.; Linares, C.; Sanner, S.; and Yoon, S. 2012. A survey of the seventh international planning competition. *AI Magazine* 33:83–88.

Cresswell, S.; McCluskey, T. L.; and West, M. M. 2009. Acquisition of object-centred domain models from planning examples. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 338–341.

de la Rosa, T.; Jiménez, S.; Fuentetaja, R.; and Borrajo, D. 2011. Scaling up heuristic planning with relational decision trees. *Journal of Artificial Intelligence Research (JAIR)* 40:767–813.

Dietterich, T. G. 2000. Ensemble methods in machine learning. In *Multiple classifier systems, LBCS-1857*, 1–15. Springer.

García, R.; Fernández, F.; and Borrajo, D. 2006. Combining macro-operators with control knowledge. In Otero, R., ed., *Proceedings of International Conference on Inductive Logic Programming (ILP'06)*, volume 4455 of *Lecture Notes on Artificial Intelligence*, 229–243. Santiago de Compostela (Spain): Springer Verlag.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: PbP. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 19–23. AAAI Press.

Gerevini, A.; Saetti, A.; and Vallati, M. 2014. Planning through automatic portfolio configuration: The pbp approach. *Journal of Artificial Intelligence Research (JAIR)* 50:639–696.

Hertzberg, J. 1995. On building a planning tool box. In *New Directions in AI Planning*, 3–18.

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

Hoffmann, J.; Porteous, J.; and Sebastia, L. 2004. Ordered landmarks in planning. *Journal of Artificial Intelligence Research (JAIR)* 22:215–278.

Hutter, F.; Xu, L.; Hoos, H. H.; and Leyton-Brown, K. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206:79 – 111.

Jiménez, S.; De la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(04):433–467.

Jimoh, F.; Chrpa, L.; McCluskey, T.; and Shah, S. 2013. Towards application of automated planning in urban traffic control. In *Intelligent Transportation Systems - (ITSC), 2013 16th International IEEE Conference on*, 985–990.

Krajnansky, M.; Buffet, O.; Hoffmann, J.; and Fern, A. 2014. Learning pruning rules for heuristic search planning. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*.

McCluskey, T. L. 1987. Combining weak learning heuristics in general problem solvers. In *Proceedings of the 10th International Joint Conference on Artificial Intelligence (IJCAI)*, 331–333.

Parkinson, S.; Longstaff, A.; and Fletcher, S. 2014. Automated planning to minimise uncertainty of machine tool calibration. *Engineering Applications of AI* 30:63–72.

Porteous, J.; Cavazza, M.; and Charles, F. 2010. Applying planning to interactive storytelling: Narrative control using state constraints. *ACM Transactions on Intelligent Systems and Technology (TIST)* 1(2):10.

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

Richter, S.; Westphal, M.; and Helmert, M. 2011. LAMA 2008 and 2011. In *Booklet of the 7th International Planning Competition*.

Roberts, M., and Howe, A. E. 2009. Learning from planner performance. *Artificial Intelligence* 173(5-6):536–561.

Seipp, J.; Sievers, S.; Helmert, M.; and Hutter, F. 2015. Automatic configuration of sequential planning portfolios. In *Proceedings of the Twenty-fourth National Conference on Artificial Intelligence (AAAI'15)*.

Vallati, M.; Chrpa, L.; and Kitchin, D. 2013. An automatic algorithm selection approach for planning. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, 1–8. IEEE.

Vidal, V. 2014. YAHSP3 and YAHSP3-MT in the 8th international planning competition. In *The 8th IPC. Description of Planners of the Deterministic Part*, 64–65.

Yang, Q.; Wu, K.; and Jiang, Y. 2007. Learning action models from plan examples using weighted MAX-SAT. *Artificial Intelligence* 171(2-3):107–143.

Yoon, S. W.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research* 9:683–718.