Case-Based Search Control for Heuristic Planning

Tomás de la Rosa
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
trosa@inf.uc3m.es

Abstract

The great success of heuristic search as an approach to AI planning is due to the the right design of domain-independent heuristics. Although many heuristic planners perform reasonably well with the only guidance of the heuristic function, few planners incorporate additional domain-dependent heuristics generated through a domain-independent automatic procedure in order to improve their performance in terms of time or plan quality. In this work we present a case-based reasoning approach that learns abstracted sub-state transitions that serve as domain control knowledge for improving the planning process. Search nodes matching the retrieved episodes are recommended as good choices when decisions of pruning or ordering nodes are done in the search algorithm. We show that the concept of recommended nodes can be applied in different algorithms depending on whether the aim of the planning task is finding a solution rapidly or achieving a plan of good quality.

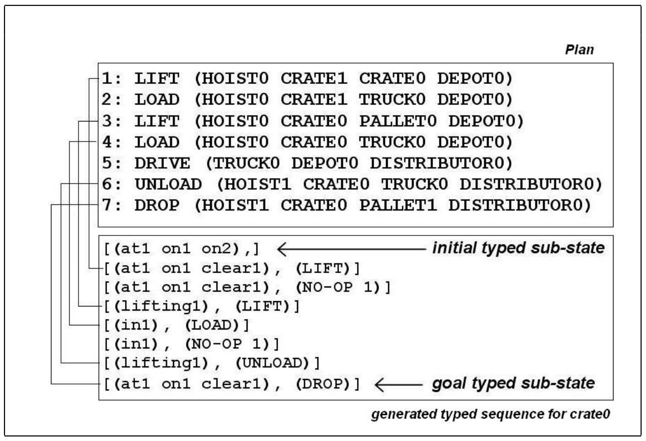
Introduction

Past IPCs have shown heuristic search as one of the top approaches in automated planning. This great success mostly relies on the domain-independent heuristic used for guiding the search. The heuristic of the relaxed planning graph introduced in FF (Hoffmann & Nebel 2001) is the most used heuristic and some state-of-the-art planners use variations of FF or its heuristic. Although heuristic planners perform reasonably well in many benchmark domains, additional domain-dependent control knowledge can be used to improve the search process. Given that manually defining this control knowledge is a difficult task, we advocate for learning it with machine learning techniques that take some planning examples and extract knowledge from them. Recently, macro-actions approaches in MACRO-FF (Botea *et al.* 2005) and MARVIN (Coles & Smith 2007), show that domain specific control knowledge can speed up the planning process. Another example is learning the heuristic function (Yoon, Fern, & Givan 2006) through observing in a particular domain the error between the estimation and the real cost of achieving the goals. Although we find in the literature many case-based planners (Cox, Muñoz-Avlia, & Bergmann 2005), no recent case-based reasoning (CBR) approach has been developed to support state-of-the-art-planners. We argue that heuristic planning offers some learning opportunities which can be learned within a CBR cycle of storing, retrieving and reusing planning episodes. We present in this paper a summary of our PhD work at its current stage. The aim of this work is to integrate CBR techniques in a heuristic planner to improve the planning process. We will show that a case-based search control can be useful depending on whether the target of the planning task is rapidly finding a solution or finding a plan of good quality.

Learning Typed Sequences

The basic piece of knowledge in a CBR system is a case. So, we describe our cases, called typed sequences, as abstracted sub-state transitions relative to an object type. A typed sequence of a given type is formed by an ordered list of pairs (typed sub-state, action to reach the state) which partially collects a planning episode from an object instance perspective. A typed sub-state is the set of all properties that an object has in a particular state. A property, first introduced by the domain analysis in TIM (Fox & Long 1998), is defined as a predicate subscripted with the object position of a literal (e.g., at_1 is a property of object *truck1* in the literal *(at truck1 depot0)*). In addition, an object sub-state is the set of the state literals in which the object is present. Then, the set of object properties that forms the typed sub-state is extracted from the object sub-state. For instance, suppose we have an initial state like [*(at truck1 depot1) (on crate0 crate1) (at crate0 depot0) (available hoist1) (clear crate0)*...]. Then, the object sub-state of *crate0* would be [*(on crate0 crate1) (at crate0 depot0) (clear crate0)*]. This is generalized to $(on_1\ at_1\ clear_1)$ which is a typed sub-state of type *crate*. If the action *lift(hoist1,crate0,crate1,depot0)* is applied in the initial state, we would generate first a pair with the initial sub-state and no action, [$(on_1\ at_1\ clear_1)$,$\emptyset$], and a second pair with the next sub-state and its corresponding action: [$(lifting_1)$, *lift*].

The basis of the CBR cycle (Aamodt & Plaza 1994) is that past experience is stored in a case base and when a new problem needs to be solved, the most similar case is retrieved. Then, the retrieved case is adapted and reused to solve the new problem. In this sense, we obtain the experience from solved problems as follows. For each object instance in the problem, a typed sequence is generated. This process is straightforward, since each step in the sequence is created with the corresponding object sub-state from the solution path. If we consider $U(o, S)$ the transformation function that gets the typed sub-state (properties of object o) from the state S, we say that given a plan $P = \{a_1, \ldots . a_n\}$, $Q = \{(U(o, S_0), \emptyset), \ldots, (U(o, S_n), a_n)\}$ is a typed sequence where the state S_i is the resulting state of applying action a_i to the state S_{i-1}. Sequences are grouped in the case base by domain types, so a new case is inserted in the type of object from which it was generated. Fig.1 shows an example of a typed sequence extracted from an object of type `crate`. If the object sub-state does not change when an action is applied, a `no-op` is saved to represent a void action from the object perspective. A merge process verifies that equivalent sequences only varying in the number of `no-op` do not repeat in the case base.



```
1: LIFT (HOIST0 CRATE1 CRATE0 DEPOT0)
2: LOAD (HOIST0 CRATE1 TRUCK0 DEPOT0)
3: LIFT (HOIST0 CRATE0 PALLET0 DEPOT0)
4: LOAD (HOIST0 CRATE0 TRUCK0 DEPOT0)
5: DRIVE (TRUCK0 DEPOT0 DISTRIBUTOR0)
6: UNLOAD (HOIST1 CRATE0 TRUCK0 DISTRIBUTOR0)
7: DROP (HOIST1 CRATE0 PALLET1 DISTRIBUTOR0)
```

```
[(at1 on1 on2),]
[(at1 on1 clear1), (LIFT)]
[(at1 on1 clear1), (NO-OP 1)]
[(lifting1), (LIFT)]
[(in1), (LOAD)]
[(in1), (NO-OP 1)]
[(lifting1), (UNLOAD)]
[(at1 on1 clear1), (DROP)]
```

Figure 1: An example of a typed sequence relevant to a crate.

The next step in the CBR cycle is the retrieval. For each object instance in the new problem a typed sequence is retrieved. For this purpose, we have a simple retrieval scheme that only considers the first step of the sequence referred to the initial state and the last step of the sequence referring to the goal state. The system performs two matches to retrieve a sequence. The first one matches the typed sub-state generated from the goals against the last step of all sequences of the corresponding type. For the second match the typed sub-state generated from the initial state is matched against the first step of the sequences resulting from the first match. Thus, if we consider an object o of type t and $Q = \{(q_0, \emptyset), \ldots, (q_n, a_n)\}$ an arbitrary typed sequence in the case base of type t, the first match holds when $U(o, G) \subseteq q_n$ where G is the set of goals, and the second match holds when $q_0 \subseteq U(o, S_0)$.

Before using retrieved sequences, they must be adapted to fit into the new problem. To achieve this, sequences are partially re-instantiated using the information from the relaxed planning graph of the initial state. The idea consists of transforming a typed sub-state back into an object sub-state, so the replay of sequences can be more accurate during the search. A relaxed planning graph is formed by a sequence of layers of actions and propositions. If we take only the facts achieved by applying the relaxed plan we get a subset M of the graph that represent the goals and sub-goals achieved by the relaxed plan. Then, for each layer j in M, and for each retrieved sequence Q, if $U(o, M_j) \subseteq q_i$ then we augment the step i information including the corresponding object sub-state from layer M_j. Since facts in the relaxed planning graph are not ordered as they appear in a real solution path, not all steps can be re-instantiated. Therefore we differentiate since this stage between a *typed step* (the pair (q_i, a_i)) and *instantiated step* (the set (q_i, a_i, w_i) where w_i represents the instantiated object sub-state). Since the retrieval and the adaptation are performed once at the beginning of the search, they do not produce considerable overhead in the overall performance of the search process. For additional explanation of these processes see (De la Rosa, García-Olaya, & Borrajo 2007).

Typed sequences are used during the search as control knowledge that supports exploring decisions together with the heuristic function. The search control is performed with recommendations given by the retrieved sequences, which are replayed while the search is advancing. We say that a successor node is recommended when it matches the next step of a sequence retrieved for any object involved in the applied action to reach the node state. Thus, we say that a child node S' achieved by applying action a' is a recommended node when $q_{k+1} \subseteq U(o_j, S)$ where o_j is the parameter (object instance) j of the action a' and k is the current step number for the sequence retrieved for o_j. The notion of *recommended node* is independent of the search algorithm, so it can be used for different types of search control like selecting, pruning or ordering nodes. According to this, we explain in the next sections how these CBR recommendations can be used in different search algorithms.

Advising Local Heuristic Search

Local heuristic search is used in planning to find a solution as soon as possible. Therefore, the use of learning knowledge is focused on improving the planning time. Since computing the heuristic value of a node is expensive, skipping node evaluations is one the main learning opportunities in this kind of search. The first straightforward application of recommended nodes is directly selecting a recommended one for following the search. For instance, in the hill-climbing algorithm, after a node expansion, if a successor is recommended, it is selected and no evaluations are performed. If there is no advise, all nodes are evaluated and the one with the best heuristic value is selected. The main drawback of this hill-climbing variation is that the heuristic function is not been used to complement the selection. Although this approach can reduce node evaluations in many problems (De la Rosa, Borrajo, & García-Olaya 2006), in some others the plan quality decays, since a bad recommendation usually can not be recovered in a local search.

Another option of CBR search control is deciding the node evaluation ordering in a greedy algorithm like the enforced hill-climbing (EHC) used in FF. This algorithm performs a breadth-first search from a node S until it finds a node that has a better heuristic value. Thus, the idea consists of trying to evaluate first the most promising nodes in order to skip the evaluation of the rest of siblings. In our EHC variation, instead of evaluating nodes in the standard way, we only compute the heuristic function if the node is recommended. If none of the recommended nodes improves $h(S)$, the rest of nodes are evaluated in their standard order. The EHC supported by CBR improves EHC performance in some domains as shown in (De la Rosa, García-Olaya, & Borrajo 2007).

Advising Best First Search

A systematic global search like the best-first algorithm (BFS) is used in planning when the aim of the planning task is to find an optimal solution or at least a plan of good quality, (i.e., the metric optimization in numeric versions of the IPC benchmarks). One learning opportunity in this algorithm resides in which tree branches should be preferred to explore first, since the number of nodes that need to be expanded before finding a solution is very high. One basic difference with local search recommendations is that since different branches may receive different recommendations, each node must keep track of the pointers to current steps in the sequences.

A first variation over BFS is the case-based pruning BFS, which discards tree branches when there are sibling recommended nodes. After each node expansion, no-recommended nodes are pruned. Recommended nodes are evaluated and inserted into the *open list*. If there is no advise, all successors are inserted into the open list. The open list is resorted and the node with the best value of the evaluation function $f(S)$ is selected. This algorithm is not complete, but if the case base has good and enough planning episodes, the performance time can be improved without losing plan quality. The complete version for this algorithm is the case-based ordering BFS, in which no-recommended nodes are not pruned, but evaluated and postponed in a *delayed-list*. This algorithm follows a similar scheme, but no nodes are inserted in the open list if no advise is received. If the open list becomes empty at some point, the node with the best $f(S)$ in the delayed list is selected for expansion. In practice, this algorithm spends more time than the case-based pruning BFS, but at least it guarantees a complete search.

Future Work

The refinement of a case-based system usually involves improving some features particular to any CBR-cycle stage. For instance, a way of enhancing the retrieval phase is finding a more accurate similarity measure, permitting a more precise case retrieval. In this sense, the retrieval mechanism in our system only takes care of features from the initial state and the goals. We are currently developing a more complex retrieval scheme, that takes into account the intermediate steps of sequences. The idea consists of using the achieved facts layers of the relaxed planning graph of the initial state (explained before when sequences are adapted), as an extra key for retrieving typed sequences. Hence, if two sequences have the same first and last steps, they can be discriminated by their intermediate steps.

Another issue we have to address is the training of the system when we want to populate the case base with quality-oriented sequences in order to advise search algorithms that are optimizing a metric. The A* algorithm can only solve small problems in numeric domains, so we can not learn from medium or large problems. This drawback is also present when the case-based search control is applied to a complete algorithm due to the large size of the search tree and imprecise guidance of the heuristic. We are testing with alternative algorithms, so we can solve more problems without compromising too much the plan quality. One option is the real-time A* (RTA*), which is used in the search community to tackle this kind of problems. A different option is performing iterative search restarts with a branch and bound technique, so plans are refined gradually. In both cases we can use the case-based search control, since the storing and retrieving phase will remain the same, and the concept of recommended node is applicable to any search algorithm.

Acknowledgments

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project CCG06-UC3M/TIC-0831.

References

Aamodt, A., and Plaza, E. 1994. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications* 7, no.1:39–59.

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.

Coles, A., and Smith, A. 2007. Marvin: A heuristic search planner with on-line macro-actions learning. *Journal of Artificial Intelligence Research* 28:119–156.

Cox, M.; Muñoz-Avlia, H.; and Bergmann, R. 2005. Case-based planning. *Knowledge Engineering Review* 20(3):283–287.

De la Rosa, T.; Borrajo, D.; and García-Olaya, A. 2006. Replaying type sequences in forward heuristic planning. In Ruml, W., and Hutter, F., eds., *Technical Report of the AAAI'06 Workshop on Learning for Search*. Boston, MA (USA): AAAI Press.

De la Rosa, T.; García-Olaya, A.; and Borrajo, D. 2007. Case-based recommendation for node ordering in planning. In Dankel II, D., ed., *Proceedings of the 20th International FLAIRS Conference*. Key West, FL (USA): AAAI Press.

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *Journal of Artificial Intelligence Research* 9:317–371.

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

Yoon, S.; Fern, A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS-2006)*.