Replaying Type Sequences in Forward Heuristic Planning

Tomás de la Rosa and **Daniel Borrajo** and **Angel García Olaya**
Departamento de Informática, Universidad Carlos III de Madrid
Avda. de la Universidad, 30. Leganés (Madrid). Spain
trosa@inf.uc3m.es, dborrajo@ia.uc3m.es, agolaya@inf.uc3m.es

Abstract

Heuristic Planning is nowadays one of the top approaches for AI Planning. Although current heuristic planners are quite efficient, the time spent in computing heuristics is still an issue, since this task must be repetitively done for each state explored in the search process. We propose that domain type sequences can be learned to support the heuristic search and to avoid continuously computing heuristics. We present in the paper a CBR approach for extracting, retrieving and replaying type sequences within a heuristic search. We also present the results of testing this idea within two of the classical IPC domains.

Introduction

Heuristic Planning has proven to be a successful approach for STRIPS Planning (Fikes & Nilsson 1971). Most of the state of the art planners such as HSP (Bonet & Geffner 2001), FF (Hoffmann & Nebel 2001) or YAHSP (Vidal 2004) do heuristic search to generate plans. The common idea of this kind of heuristic search is to compute heuristics solving a relaxed version of the original problem, ignoring the delete list of actions in the domain description. This was first introduced by (McDermott 1996) and nowadays is the most common relaxation for planning tasks. Although these planners are quite efficient, the time spent in computing heuristics is still an issue. The basic problem is that for each state explored during the search process, the relaxed task is solved to obtain the heuristic. We propose that Case Based Reasoning (CBR) can be used as a learning technique to guide the search process and help with the burden of computing heuristics.

In the past, CBR seemed an interesting approach for not planning from scratch. ANALOGY (Veloso & Carbonell 1993) fully integrated CBR with generative planning based on a derivational analogy process, in which lines of reasoning are transfered and adapted to a new problem. This idea was attractive to planning because it reasons over the planning trace rather than other CBR techniques that just search the solution with a transformational adaptation. In forward heuristic search it is not possible to directly obtain a causal justification as it was stored in ANALOGY's cases. However, features of the planning trace can be still recorded. We suggest that sequences of visited states, stored as sequences of domain types, can be learned and later transfered as control knowledge to guide the search. These sequences can be viewed as traces of successful paths from previous solved problems.

This paper presents the current state of this work, in which we are using a CBR approach to support a heuristic planning process. In the following sections we introduce the SAYPHI Architecture, composed of a heuristic planner, and a CBR system. Then, we describe the behaviour of this system with the phases of the classical CBR cycle (Aamodt & Plaza 1994). First, we explain the extraction of type sequences and how they are stored as cases. Then, we describe the retrieval of sequences needed in order to solve the new problem and the adaptation of these retrieved sequences. Afterwards, we explain the replay process, which integrates the CBR technique within the heuristic search in an algorithm that we call mixed hill-climbing. We also present results of testing this idea within two of the International Planning Competition (IPC) domains. Finally, we discuss related work and present some conclusions.

The Planner

We are currently building SAYPHI, a learning architecture in which several techniques for control knowledge acquisition will be integrated with a common heuristic planner. We have seen in the past as in PRODIGY (Veloso *et al.* 1995), that this kind of systems greatly promotes research in planning and learning. The SAYPHI planner is an heuristic planner that performs forward state-space search. The planner uses the heuristic function of the relaxed plan as FF (Hoffmann & Nebel 2001). This heuristic is computed extracting a solution from a relaxed GRAPHPLAN (Blum & Furst 1995). The planner also integrates the helpful actions technique for pruning state successors and only evaluate promising actions that add sub-goals needed later. SAYPHI has been developed in LISP, and at the current time solves STRIPS problems written in PDDL (Fox & Long 2002), the standard language for describing planning tasks within the AI planning community.

Without any control knowledge, the planner performs a standard hill-climbing algorithm as follows. Given an initial state S_0 and a set of goals G, a search tree is expanded with all applicable actions belonging to the helpful actions. For

each state successor a heuristic value is computed and the node with the best heuristic is chosen. The applied action is added to the plan and the process starts over again in the selected node, until G is contained in a reached state. The resulting plan is the forward chaining of all applied actions of the selected nodes. Considering only the helpful actions for expanding the search tree makes the search incomplete, since many applicable actions are pruned. If the search process fails, a complete best-first algorithm is performed as a second attempt to obtain a plan.

The heuristic of the relaxed GRAPHPLAN consists on building in any search state a planning graph from which a solution to the relaxed task could be extracted. This relaxed problem is the same problem, but ignoring the delete list of actions. The relaxed GRAPHPLAN is represented as a sequence $P_0, A_0, \ldots, A_{t-1}, P_t$ of propositions layers P_i and actions layers A_i. The planning graph starts with $P_0 = S_0$ as the initial proposition layer. All applicable actions are inserted in the next action layer and a new proposition layer is built with the add effects of applied actions. This process repeats until a proposition layer contains G. After the relaxed GRAPHPLAN expansion, a solution to the relaxed problem (a relaxed plan) is extracted, performing a backward chaining of actions through the graph. Then, the number of actions that appear in the solution to the relaxed problem is taken as an estimate of how far the goal is from the current state.

Learning Type Sequences

Most PDDL domains group objects in types. The idea of learning type sequences comes from the observation of typical state transitions that each type of object has. We propose that this planning feature can be learned with a CBR approach. In a CBR cycle, type sequences are extracted from previous solved problems and are stored in the case base. With a new problem to solve, a retrieve process selects most similar sequences to the new problem. The retrieve sequences are adapted and used in the replay process in order to solve the new problem. For the explanation of these CBR processes, we are going to use the Depots domain. This domain is part of the IPC collection, in which trucks transport crates around depots and distributors, and crates must be stacked onto pallets or on top of other crates at their destination.

Storing

To extract a case, we assume that the planner has solved a problem. A type sequence is formed by a set of pairs (object sub-state, applied action to reach the state). An object sub-state is the set of all facts in a state in which the object is present. Thus, a subset of the initial state like `[(on crate0 pallet0) (at crate0 depot0) (clear crate0)]`, can be translated to a sub-state of type "crate" as `[(on <x>__) (at <x>__) (clear <x>)]` where `<x>` represents any instance of a crate and the underscore sign represents another object in the literal which is not important for the sub-state of that type. The pair, object sub-state and applied action, forms a step in the type sequence and that is what we call a sub-state relation. Since many actions in the sequence are not relevant to the object, assuming that the object sub-state does not change either, a `no-op` is stored with the same sub-state. Then, from the solution path to a problem we can compute the complete sequence of each object, storing all sub-state relations in the order they appear.

Figure 1 shows a type sequence of a crate, extracted from a problem in which it was initially on a pallet, with another crate on top of it, and in the goal state it was at another place, on another pallet, and clear. Dots in the applied action represent the action parameters not relevant to the sub-state relation. The first step has no applied action, since it corresponds to the crate initial sub-state. The `no-op` in the sequence corresponds to a `Drive` action, for which the crate sub-state has not changed. A number is stored with the `no-op`, in the example 1. It represents the numbers of applied actions in which the current sub-state has not changed.

```
[(at <x>_)(on <x>_)(on _<x>),]
[(at <x>_)(on <x>_)(clear <x>), (LIFT..<x>.)]
[(lifting <x>_), (LIFT .<x>..)]
[(in <x>_), (LOAD .<x>..)]
[(in <x>_), (NO-OP 1)]
[(lifting <x>_), (UNLOAD .<x>..)]
[(at<x>_)(on<x>_)(clear <x>), (DROP.<x>..)]
```

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

After the construction of sequences, cases are grouped by type of objects and a merge process determines if the new case to store is part of the case base or it is just a different case. Though a case could have `no-op` in different places, the merge process saves the sequences with less number of `no-op` For example, actions in a typical transition of a crate sequence are `Lift`, `Load`. If a new case of type crate has the transitions `Lift`, `no-op`, `Load`, it is considered the same case by the merge process, and the first sequence is kept.

Retrieving

When a new problem to be solved arrives to the planner, cases for each object appearing in the goals must be retrieved. For this, we perform a retrieval step comparing the goal sub-state of an object with the last sub-state of all sequences in the case base of the corresponding type. Then, the initial sub-state of an object is compared with the first sub-state of the sequences that meet the first criteria. All retrieved sequences are kept in a replay table that will be used in the search process. The size of the replay table depends on how many sequences are retrieved, but at most there is one sequence for each different object instance appearing in the goals. Future work will use more sophisticated retrieval, but at this time this retrieval scheme works reasonably well.

As an example of the retrieval, suppose a set of goals G = `((on crate1 pallet1) (on crate2 crate1))`. We search in the case base a sequence of

type pallet for `pallet1`, and sequences of type crate for `crate1` and `crate2`. In order to retrieve a sequence for `crate1`, selected cases must have a sequence that ends with the sub-state `(on <x>__)(on __<x>)` with the corresponding initial sub-state. If no sequence matches, the object is not considered in the construction of the replay table.

Adapting

If cases were used simply as type sequences, they could represent many instantiated sequences in the new problem. Since sub-state relations only take care of a particular referenced object, the other objects in the literals could receive as many bindings as objects of the ignored type. To address this issue, we do an early extraction of a relaxed plan, and obtain the goals membership of the relaxed GRAPHPLAN. This goals membership is built in the relaxed plan extraction process, and is represented with the sequence $G_1,\ldots,$ G_t, where G_i is the set of facts achieved by applied actions of the relaxed plan, which belong to the A_{i-1} action layer. With this goal membership we can transform type sequences to real objects sequences, searching the right bindings through the goal layers. Suppose that the sequence in Figure 1 is retrieved for `crate1` appearing in the goal `(on crate1 pallet1)`. In the sixth step, the literal `(lifting crate1 __)` could be with any `hoist` of the problem description, but the G_{t-1} layer has the right binding, in example `(lifting crate1 hoist1)` since it is a precondition to achieve the goal present in G_t layer. Since the relaxed GRAPHPLAN does not always represent the real order of achieving goals and subgoals, not all sequences are fully instantiated and additional computing for the instantiation is done during the replay.

Replaying

For the search process we use a mixed hill-climbing algorithm, which combines the heuristic function and the advise of the sequences in the replay table. The search goes as follows: at any state, all helpful actions are considered as state successors. Helpful actions are all applicable actions that add any fact to G_1 layer in goals membership. Then, the process asks for a CBR advise of which node to select. If an advise is received, it goes to the recommended state; if not, it computes the heuristic for all successors and chooses the state corresponding to the best heuristic value.

To advise, the replay table holds the current pointer to all of the retrieved sequences, and if a state successor matches the next sub-state relation in the sequences, it is returned as the recommended successor. This match is performed converting the state successor to a sub-state relation relevant to the sequence with which it is going to be compared. In this way, the search behaves following sequences of learned experiences any time they are identified in the current search. Since not all sequences are fully instantiated at the adaptation phase, a stack of visited sub-states is kept in memory to avoid undoing the path reached in the sequence. Each sequence in the replay table keeps the visited sub-state when a step of the sequence is followed in order to hold the instantiated sub-state of the sequence used in the search. Since more than one state could be returned in the advise process, all cases in the replay table are sorted by the remaining length of sequences. Thus, when two or more states are good for advise, the one with more steps ahead is preferred. This kind of selection helps to avoid reaching the end of a sequence too early, and it allows to advance all retrieved sequences without forgetting any of them. Moreover, since retrieved sequences may share facts of their sub-states, if a sequence is advanced to the next sub-state relation, all sequences that share this sub-state are also advanced. Otherwise, if a node in the search is selected by the heuristic, the replay table is traversed to see if any sequence should be also advanced. If a sequence in the reply table points to a `no-op`, the sequence is not used again until it is advanced as many times as the number with the `no-op`. This guarantees at least that a number of actions, not relevant to the object, are applied before the sequence advises a new sub-state.

```
Function mixed_hill_climbing(S_0, G, replay_table)
S = S_0
while G ⊄ S do
  C = expand_state(S)
  for each q in seq_sorted(replay_table)
    for each c in C
      if (recommend(c, q) = true
       and not guided then
        guided = true
        S' = c
        advance(q)
      endif
    endfor
  endfor
  if guided
    S = S'
  else
    for each c in C
      h(c) = h_relaxedplan(c)
    endfor
    S = best_h_state(C)
  endif
endwhile
```

Figure 2: Pseudo-code for the mixed hill-climbing algorithm.

Figure 2 shows a pseudo-code for the mixed hill-climbing algorithm we have described. The *recommend* function calls the CBR Advisor, which tries to match the child node c with a sequence q in the replay table. The *seq_sorted* function sorts the sequence by the remaining length of the sequences, and the *advance* function advances the used sequence to the next sub-state relation. The *h_relaxedplan* function computes the heuristic value for the child node c and the *best_h_state* function selects the node with the best heuristic, in this case the minimum value. As a result of this process, in many of the states that belong to the solution path, it is not necessary to compute the heuristic for the state siblings. This could intuitively suggest less effort for the planner, since it would call less times the heuristic function.

Results

For the experiments we have given the planner 10 random training problems from the Depots domain for extracting the type sequences. These problems had up to 14 objects instances and up to 3 goals. Then we generated the test set with 50 random problems up to 17 objects instances and up to 6 goals. All these problems were generated with the random problem generator supplied by the IPC Collection.

Then, we made our first test solving the problem set alone with the planner with a hill-climbing algorithm. The second test was supported by the CBR advise described in previous sections. The hill-climbing algorithm solved 96% of the problems and the mixed hill-climbing with CBR support solved 78%. The reason of these decrease is that we have not deal the goal ordering issue yet. This goal ordering could affect how the sequences may be selected in the replay and it yields the algorithm to get out of the way to the solution. Although the difference in the number of solved problems, the CBR-Supported search improves the planning time in problems solved by both techniques.

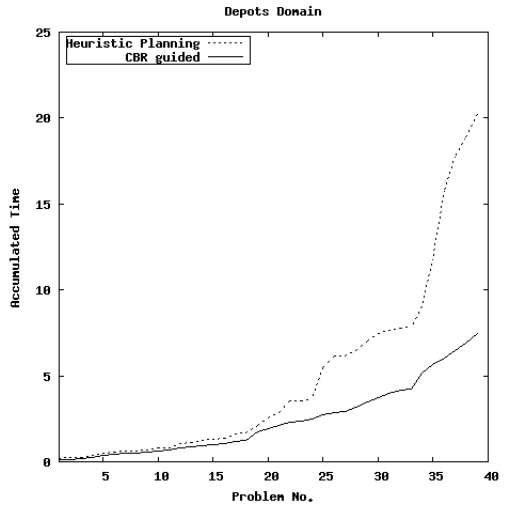

Figure 3: Accumulated time solving Depots random problems.

Figure 3 shows the result of the accumulated time used by the planner to solve the problems with the hill-climbing algorithm and with the search supported by the CBR advise. The accumulated time is from the problems solved by both techniques. The time improvement is due to the heuristic evaluations skipped when a state is advised by the CBR during the search.

We have also tested this idea within the Rovers domain. This domain is part of the IPC 2002 Collection, which was inspired by planetary rovers problems. This domain requires that a collection of rovers navigate a planet surface, finding samples, taking pictures and communicating them back to a lander. For the experiment, we have generated 10 training random problems, which were solved to populate the case base. Problems in the training set have 13 objects instances. Then, 30 test problems were generated to compare the performance of the search with and without the support of CBR. Problems in the test set have between 14 and 18 objects instances.

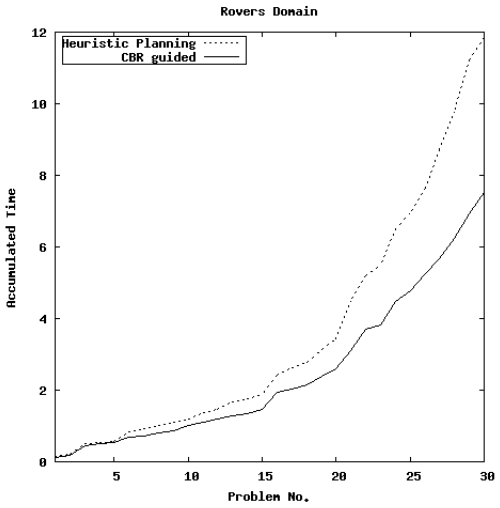

Figure 4: Accumulated time solving Rovers random problems.

Figure 4 shows the result of the accumulated time used by the planner to solve the problems with the standard search and with the search supported by CBR. We can see in the curve a better performance in the CBR supported search. In a detailed review of solved problems, a 36.67% of them obtained the same solution length. A 13.33% of the problems had a better solution with the CBR supported search, but 50.0% had a longer solution length. Although here is a lost of plans quality, the difference is not really significant. The average plan length with the standard search is 14.53 vs. 15.13 of the CBR supported search. As we have mentioned earlier, the goal ordering may affect those problems in which the CBR supported search obtained a worse solution.

Related Work

The idea of extracting information from the domain types description is not new to the community. Our work is related to states invariants extracted from a domain analysis as in TIM (Fox & Long 1998) and (Long & Fox 2000). With a pre-processing tool, they obtain Finite State Machines (FSM) that represent states in which a type of object can be and can move to. In our case this knowledge is obtained dynamically while the case base is been populated in form of sequences. Another difference is that states invariants help planners to build efficient action schemas, needed to compute applicable actions, and type sequences are used to guide the search through the path of learned experiences.

Other works that integrate CBR with planning have been done in the past. ANALOGY (Veloso & Carbonell 1993) that we have mentioned earlier, PARIS (Bergmann & Wilke 1996) which stores cases in different abstraction levels of a solved problem and CAPLAN-CBC (H. Muñoz Avila 1994)

which performs plan-space search and replay. The novel contribution of our new approach is that the knowledge of the case base is abstracted in domain types and these cases represents sequences of state transitions. Moreover, cases do not represent directly plans of solved problems because a single plan can generate multiple type sequences. Our approach is also interesting because it gives an additional way to address heuristic planning.

Conclusions and Future Work

In this work we have shown a new approach to the forward state-space search heuristic planning, introducing a mixed hill-climbing that is advised by a CBR component. It uses type sequences learned from previous solved problem, for suggesting next states in the search process. The heuristic function is still present in states in which is not possible to get an advise. We have seen that the planner performance time can be improved with this technique, since the planner may do less heuristic computation. The key idea of this work opens a variety of possibilities for helping heuristic planning. Any kind of learning technique that could predict somehow a next state in the search, would be applicable within the mixed hill-climbing algorithm. Thinking on this, we continue developing SAYPHI, so we can research over extension to this approach.

The next target of this work is to extract additional information from the relaxed GRAPHPLAN or from the search tree of solved problems to address the issues of goals dependencies and goals ordering. This additional knowledge could be stored with type sequences within cases and could be used in the replay phase in order to perform a more informed search. We also want to know how this approach scales up when the case based is populated with more cases or when more challenging problems are given to the CBR planner.

Acknowledgments

This work has been partially supported by the Spanish MEC project TIN2005-08945-C06-05 and regional CAM-UC3M project UC3M-INF-05-016.

References

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

Bergmann, R., and Wilke, W. 1996. Paris: Flexible plan adaptation by abstraction and refinement. In Voss, A., ed., *ECAI (1996) Workshop on Adaptation in Case-Based Reasoning*. John Wiley & Sons.

Blum, A. L., and Furst, M. L. 1995. Fast planning through planning graph analysis. In Mellish, C. S., ed., *Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI-95*, volume 2, 1636–1642. Montréal, Canada: Morgan Kaufmann.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

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

Fox, M., and Long, D. 2002. *PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains*. University of Durham, Durham (UK).

H. Muñoz Avila, J. Paulokat, S. W. 1994. Controlling nonlinear hierarchical planning by case replay. In *in working papers of the Second European Workshop on Case-based Reasoning*, 195–203.

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

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *Proceedings of AIPS 2000*, 196–205.

McDermott, D. 1996. A heuristic estimator for means-ends analysis in planning. In *Proceedings of the Third International Conference on AI Planning Systems*.

Veloso, M. M., and Carbonell, J. G. 1993. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning* 10(3):249–278.

Veloso, M.; Carbonell, J.; Pérez, A.; Borrajo, D.; Fink, E.; and Blythe, J. 1995. Integrating planning and learning: The PRODIGY architecture. *Journal of Experimental and Theoretical AI* 7:81–120.

Vidal, V. 2004. A lookahead strategy for heuristic search planning. In *Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling*, 150–160.