Using Cases Utility for Heuristic Planning Improvement

Tomás de la Rosa, Angel García Olaya, and Daniel Borrajo

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

Abstract. Current efficient planners employ an informed search guided by a heuristic function that is quite expensive to compute. Thus, ordering nodes in the search tree becomes a key issue, in order to select efficiently nodes to evaluate from the successors of the current search node. In a previous work, we successfully applied a CBR approach to order nodes for evaluation, thus reducing the number of calls to the heuristic function. However, once cases were learned, they were not modified according to their utility on solving planning problems. We present in this work a scheme for learning case quality based on its utility during a validation phase. The qualities obtained determine the way in which these cases are preferred in the retrieval and replay processes. Then, the paper shows some experimental results for several benchmarks taken from the International Planning Competition (IPC). These results show the planning performance improvement when case utilities are used.

1 Introduction

AI planning consists of the computational task of given a domain theory (problem space represented in a form of first order logic as a set of predicates, actions and types), and a problem to be solved (instances of types, initial state and goals), obtain a plan. The plan usually consists of an ordered set of instantiated actions that transform the initial state into a state where the goals are met. Some of the most useful current approaches to planning are based on heuristic planning. Heuristic planners (e.g., FF [1], YAHSP [2] or SGPlan [3]) are mainly composed of an efficient search algorithm guided by a heuristic function. The standard heuristic used consists of computing a solution to a relaxed planning problem, and then returning the cost of that solution. It was first introduced by FF and has proven to be accurate enough to guide efficiently the planners towards reasonable solutions in most of the benchmark domains[1]. One of the drawbacks of this heuristic is its computational cost, since it consumes most of the total planning time. To address this issue, among other solutions, researchers have incorporated additional heuristics to make the heuristic values more accurate, thus reducing, for instance, the number of ties of heuristic values through the search tree. Another option for improving planning time consists of ordering the way in which nodes are evaluated when a greedy algorithm is used. FF uses as the standard search algorithm a variation of hill-climbing called enforced hill-climbing (EHC). In order to select the next successor of the current node, EHC evaluates one successor after another, until it finds one that returns a heuristic value better than the current node. Therefore, if node evaluations are correctly ordered, it might imply a reduction on the number of evaluations: the sooner a good successor is evaluated, the more probable will be to continue the search further.

In our previous work [4,5], we showed that a CBR approach could improve the planning time deciding the node evaluation order in EHC. This domain-dependent knowledge is stored in a case base. The training phase consists of solving a set of planning problems, and then extracting cases from the solution path. Cases were structures called typed sequences which are abstracted state transitions incorporated with the set of actions performed to solve the problem for each object type in the domain, as we will describe in more detail in the next section. This CBR cycle worked very well in the tested domains. However, we did not assess how good the learned knowledge was, since the cases were extracted from non-optimal solutions. Furthermore, there was no maintenance of the case base, apart from merging new solutions to problems with previous ones. Also, cases were used regardless of their efficiency of replaying them previously. In this paper, we present an improved approach that dynamically learns the case qualities, in terms of how useful they were for supporting planning search.

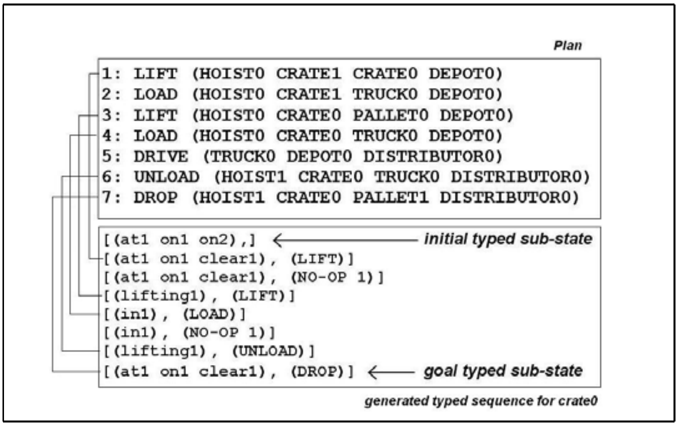
In the following sections we present a summary of how typed sequences are used to support EHC. Then, we introduce the scheme for assessing case quality based on two utility measures, one related to the sequence steps and the other one related to the global use of the sequences during the replay process. Afterwards, we show the experimental results comparing EHC, the previous approach, and the use of cases based on their quality. We also include a study of training the case base to recognize how much it may be populated depending on a particular domain. We also use this study to select a good case base for assessing case qualities. Finally we discuss some conclusions and future work.

2 Typed Sequences Overview

Current planners use a common standard language for describing their inputs, the Planning Domain Definition Language (PDDL). This language is used in the planning competitions (IPC) held every two years for describing the domains and problems to be solved. Once of the features of the domain definitions is the possibility of assigning types to predicate arguments and action variables. This permits to recognize typical state transitions that each object type has [6]. In our work, we define a case as a sequence of these transitions, called typed sequence, which describes a particular episode of the object type. As an example, if we have a domain in which crates have to be moved among some depots, using trucks and hoists to load them into the trucks, we will have cases that refer to crates, cases that refer to trucks, and so on for each type in the domain.

A typed sequence of a given type is formed by an ordered list of pairs (typed sub-state, action to reach the state). A typed sub-state is a collection of all properties that an object has in a particular state. The notion of object properties was first introduced by TIM [6] with its domain analysis techniques. A property 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 action *lift(hoist1,crate0,crate1,depot0)* were applied in the initial state, we would generate first a pair with the initial state and no action, and a second pair with the state resulting from applying that action, and the action: [($lifting_1$ at_1), *lift*].

The typed sequences grouped by domain types form the case base. In a training phase, the case base is populated, solving the training problems and extracting a sequence for each object of the problem instance. Figure 1 shows an example of a typed sequence for *crate0* and the plan from which it was generated. The two *no-op* in the sequence are steps in which the typed sub-state does not change (no action related to *crate0* was executed), so there is no relevant action to store in the sequence.



```
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)]
```

Fig. 1. An example of a typed sequence relevant to a crate

Typed sequences are used as follows. In a new problem, the planner retrieves from the case base the most similar sequence for each instance (object) in the problem. First, for each object in the problem instance, we generate two typed sub-state, one from the initial state and the other from the problem goals (goals are also described as state literals). Then, we match the typed sub-state from the goals against the last step of all sequences in the case base. Then, we do the same with the initial typed sub-state and the first step of the sequences that resulting from the first match. The retrieved sequences are generalizations of sub-states, so in order to use them properly, they are partially instantiated in the adaptation phase by using objects found in applicable actions from the initial state.[2]

Then, a modified version of EHC generates at any state S its successors, and checks if any successor is recommended by the retrieved cases. If there are, the successors are evaluated in order for obtaining their heuristic value $h(S')$. If $h(S')$ is strictly less than $h(S)$, the successor is selected, and the search continues from it, until a state achieving the problem goals is reached. If $h(S')$ is equal or greater than $h(S)$, a second attempt with the next successor is done, and so on, until a node with a better heuristic is found. If the CBR module could not recommend any node, all skipped successors are evaluated in order, and the standard EHC is used. A successor is recommended when its object sub-state (typed sub-state if it is not fully instantiated) matches the current sequence step in one of the retrieved sequences.

This approach has been implemented in SAYPHI, a learning architecture in which several techniques for control knowledge acquisition can be integrated with a heuristic planner. The SAYPHI planner is an FF-like heuristic planner. The planner includes several search algorithms and the same heuristic function as FF. Figure 2 shows its architecture. We are currently using EHC as the search algorithm, but we could use any other included in the planner, like the standard hill-climbing technique used in [4]. Also, there is one case base for each domain.

3 Computing Quality of Cases

One of the drawbacks of this approach is that heuristic planners generate non-optimal plans, and the typed sequences are extracted from those non-optimal solutions. Moreover, our retrieval scheme returns the first choice when it finds an exact match of the initial and goal typed sub-states, regardless of any other case with an exact match too. This suggests that we could improve the behavior of the CBR approach, by assessing cases quality, and using the quality to prefer useful cases in the retrieval. An additional issue is that CBR could recommend more than one node to evaluate in EHC, since the replay process uses one sequence per object. This suggests that we can also improve the CBR behavior by assessing how useful the steps in a sequence are, so we can break ties among cases when they provide different recommendations of nodes to be evaluated first. In this section we introduce two utility measures to address these issues.

[2] We refer for details to [5].

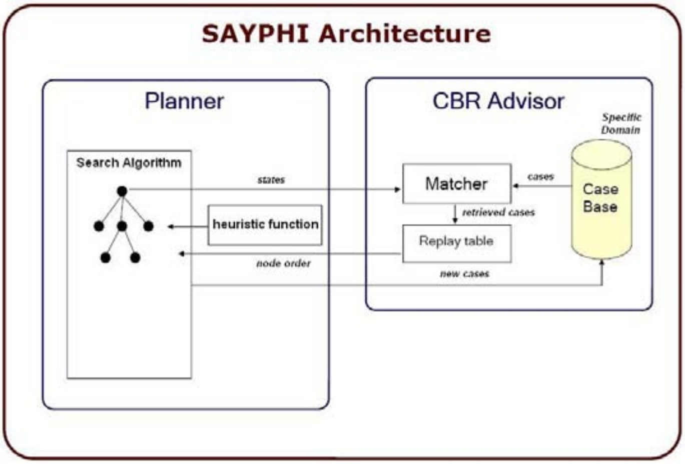


Fig. 2. The SAYPHI architecture

3.1 Step Utility Measure

During the search, the CBR approach continues following (advances to its next step) a retrieved case (typed sequence) in three different situations. The standard one is when a node (state) matches the current step of the case, and the evaluation of the recommended node improves the heuristic value of its parent. The second one is when the evaluation of the recommended node does not improve the heuristic value, but none of its siblings improve the heuristic value either. Even if it was not a good option, there was nothing better in the case base. The third one is when a node is recommended by two or more cases. All cases are then advanced to the next step if the node evaluation improves the heuristic value. Thus, we say that a "right choice" is a step sequence that recommended a node that belongs to the solution path (independently, of whether it improves the heuristic value of its parent). Likewise, we say that a "wrong choice" is a sequence step that recommended a node that does not belong to the solution path. If there is a wrong choice, there must be a sibling node that improves the heuristic value. The sum of right and wrong choices of a case is the number of recommendation attempts of the case. Thus, being g the number of right choices and A the number of recommendations, we define the step utility measure as:

$$\gamma = \frac{g}{A} \tag{1}$$

γ is the step frequency of recommending a good choice. This frequency can be easily computed after a problem is solved by just having a recommendation trace of evaluated nodes. We wanted to deal with the exploration vs. exploitation trade-off: when there is no good case for recommending a node, we can prefer the less used cases (exploration), or the most used ones (exploitation). Thus, we define a threshold μ_{step}, and a sorting function that orders the recommendations (steps of cases) by γ when $\gamma >= \mu_{\text{step}}$ (exploitation) and by increasing number of recommendation attempts when $\gamma < \mu_{\text{step}}$ (exploration: the less frequently used cases will be selected first).[3] Therefore, when none of the step options reaches the threshold we assume that no choice was good enough, and the less explored step is preferred. We can use high values of μ in a training phase if we want to explore the different options in order to learn the cases steps quality. In a test phase we can use lower values of μ to use the steps by their utility discarding only the known bad ones.

3.2 Sequence Utility Measure

Once a sequence is retrieved for an object, it stays selected regardless of whether it is used or not during the search. Though we could abandon it and select a different relevant sequence, this would lead to a higher computational load. Therefore, our replay process must deal with the problem of wrongly retrieved sequences, since either they produce wrong step choices or they are not followed at all. The problem of not using a retrieved sequence is different of the problem of having a sequence that produces wrong choices. For the sequence utility measure we have decided that is more important to recognize the "bad advisors" sequences, so the global utility measure for a sequence is a cumulative function of the step utilities. Thus, we define:

$$\lambda = \frac{\sum_{i=1}^{\mathrm{N}} g_i}{\sum_{i=1}^{\mathrm{N}} A_i} \tag{2}$$

where λ is the global frequency for a sequence of giving a good choice, N is the number of steps in the sequence, and i represents each step. As the step utility measure, we have defined a threshold μ_{case} to decide when it is bad to select a given sequence. We keep our retrieval scheme of selecting the most similar sequence, but the first one with an exact match. Then, we sort the cases of each type by λ when they have $\gamma >= \mu_{\text{case}}$ and the rest of cases are ordered in ascending order by the number of total attempts of recommendation when $\gamma <= \mu_{\text{case}}$. This utility measure adds all attempts independently from where the attempts came. Frequently, the same sequence is retrieved more than once, but assigned to different objects (two or more instantiations). Then, the λ value of a case is computed adding the right choices and the attempts of all such instantiations of the sequence. High values of μ_{case} ensure that unused cases are selected if there is more than one with an exact match. Lower values μ_{case} will guarantee that reasonably good cases are selected to avoid bad selections of unknown ones.

[3] We could have used any other way of implementing this trade-off as ϵ-greedy approaches in reinforcement learning.

4 Experimental Results

Before doing the tests with the utility measures we decided to perform a study of the learning curve for each domain. Since one training planning problem produces many cases (one per object) we intuitively know that the case base may perform reasonably well after training the system with few planning problems. Then, we selected the best training problem-set for each domain and used it for learning their qualities using the two defined utility measures. We have used four domains of the IPC in their version of classical planning (known as STRIPS). These benchmark domains are among the difficult ones: Satellite, Rovers, Depots and Zenotravel. The Satellite domain involves planning a set of observation tasks among multiple satellites. The Rovers domain requires that a collection of rovers navigate a planet surface, finding samples, taking pictures and communicating them back to a lander. In the Depots domain trucks transport crates around depots and distributors, and crates must be stacked onto pallets or on top of other crates at their destination. The Zenotravel domain involves transporting people around in planes, using different modes of movement.

4.1 Study on Training Problems

To set up the training cases tests we have generated for each domain a training set and a test set with the random problem generators supplied by the IPC. A training set consists of 20 problems subdivided in 10 groups by their difficulty. The test set consists of 100 problems subdivided in 20 sub-sets of incremental difficulty. The last training sub-set has the same difficulty than the 10th of the test sub-sets, so the test set has more complex problems. The aim of the experiment is to solve the test set after training the case base with the first sub-set, then after training with the first two sub-sets and so on. We expect that at some point an extra training will not produce any advantage of time or plan quality, and in some cases it could produce a disadvantage due to the overhead that produces a larger case base.

Table 1 shows the results of the study in the cited domains. These are the average length and evaluated nodes of problems solved with all training sub-sets. In none of the domains a considerable difference of solved problems was encountered, and the trace of these differences does not reflect any kind of convergence. In the Satellite domain, between 85 to 92 problems were solved depending on the training sub-set. In the Rovers domain between 92 and 93 problems were solved, while in the Depots domain between 61 and 64 were solved. The average of the number of evaluated nodes reflects more interesting results because we can observe the improvement as the case base grows. Though the average plan length could be a good measure for selecting a training sub-set, we have chosen the best training set in terms of the average of number of evaluated nodes. For those sub-sets the average plan length is either the best or quite good. In the Satellite domain the sixth sub-set was selected. It has 12 problems and generated 66 cases. In the Rovers domain the fifth sub-set was selected, that has 10 problems and generated 75 cases. In the Depots domain the fifth sub-set was selected that has 10 problems and generated 55 cases. In the Zenotravel domain the second sub-set was selected that has 4 problems and 24 cases.

Table 1. Results of the study on the training problems

Cycle	Probs	Satellite			Rovers			Depots			Zenotravel		
		Cases	Len.	Eval.	Cases	Len.	Eval.	Cases	Len.	Eval.	Cases	Len.	Eval.
1	2	14	32.0	279.5	15	25.6	88.5	16	30.2	768.6	14	14.5	122.5
2	4	22	31.7	231.6	33	25.6	83.0	21	30.1	733.5	24	13.7	66.5
3	6	30	31.8	221.5	51	25.9	94.7	36	29.6	657.4	37	13.7	71.1
4	8	43	32.2	278.4	62	25.8	86.6	55	29.4	728.3	49	13.6	77.8
5	10	57	31.9	208.9	75	25.5	80.0	55	29.4	559.8	58	13.7	81.3
6	12	66	31.9	208.8	98	25.8	86.0	71	29.3	618.5	68	13.8	74.3
7	14	79	32.1	271.9	120	25.7	87.0	85	29.6	663.6	78	13.9	78.1
8	16	92	31.9	281.5	147	25.9	92.2	92	29.8	632.5	90	13.9	81.8
9	18	106	32.2	240.3	174	25.9	90.8	92	29.5	647.5	108	13.8	72.2
10	20	118	32.2	230.6	201	25.7	84.8	100	29.9	710.5	128	13.8	83.8

4.2 Test Using the Utility Measure

In the following experiments, we chose for each domain the best training problems-set in terms of the number of node evaluations. After the training phase, we performed a validation phase with a validation problems-set to determine the utility measures for each case. A validation set consists of 30 new problems with the same difficulty scheme as the training set. This validation is made with an on-line strategy, so the retrieval and the replay of sequences in one problem uses the utility measures of the previous problem in the validation set. The values of μ_{step} and μ_{case} for validation were set to 0.75 to prefer exploration.[4] Afterwards, with the test set of 100 problems, we compare the EHC performance by itself (no CBR advice), with the cases support (EHC-CBR), and with the cases support using both utility measures (EHC-CBR-Utility). The values of μ_{step} and μ_{case} for the test phase were set to 0.5 to use more exploitation, but not preferring cases that perform poorly more than half the time.

Table 2 shows the number of solved problems in each domain. In the Satellite and Rovers domains one problem more than EHC was solved. In the Depots domain the EHC-CBR does not perform quite well, but it was improved with the utilities. Table 3 shows the accumulated time, the average of plan length (solution quality) and the average of evaluated nodes for problems that were solved by all techniques.

Figure 3 shows the detail for the accumulated time using each technique. The results show that EHC-CBR-Utility greatly reduced the number of evaluations done by EHC-CBR in four domains. Compared with EHC, EHC-CBR-Utility

[4] We started with an "a priori" reasonable value for those thresholds. Since there are heuristic values, we will perform an analysis to understand the influence of those values in the results.

Table 2. Number of solved problems

Domains	EHC	EHC-CBR	EHC-CBR-Utility
Satellite	93	93	94
Rovers	94	95	95
Depots	74	69	74
Zenotravel	73	72	73

reduced the number of evaluations in three of four domains, specially in the Satellite domain, that had an improvement of 46%. In the Depots domain, both CBR approaches perform worse than EHC, even though EHC-CBR-Utility has improved EHC-CBR. The Depots domain is known as a hard benchmark in the IPC, but in spite of that, we can blame the poor performance to our current case representation. Typed sequences only store information about one object and do not take into account the relation that this object has in other problem goals. The Depots domain has strongly goal dependencies and the other domains are serializable (at least there is a sub-optimal plan that can solve each goal individually). However, the CBR approaches can partially deal with goal dependencies since the replay process holds more than one sequence at a time and selecting actions from different sequences interleaves actions within a plan to achieve the goals in the right order.

Table 3. Accumulated time of solved problems using EHC, EHC-CBR and EHC-CBR-Utility

Domains	EHC			EHC-CBR			EHC-CBR-Utility		
	Time	Len	Eval	Time	Len	Eval	Time	Len	Eval
Satellite	1291.9	35.1	316.2	1040.2	34.6	254.6	695.9	34.1	146.8
Rovers	669.4	26.8	116.5	590.0	26.4	93.0	553.9	26.4	81.7
Depots	529.5	33.8	744.6	943.8	33.8	1432.8	667.5	33.4	816.0
Zenotravel	505.3	18.1	133.6	499.4	17.9	126.1	395.3	17.9	101.6

We can also analyze the plan quality measured in terms of the plan length. In all domains, the plan length was either equal or slightly improved by both CBR approaches. In EHC, the heuristic function can suggest irrelevant actions even if the heuristic value is improved with that action, producing non-optimal plans. EHC uses an inconsistent heuristic. Therefore, search can generate nodes with successors that improve the heuristic value, but with different estimations among them (the search will go through the first node evaluated of them). This fact is specially observed in bigger problems (in terms of size of initial state), and since typed sequences are learned from easy problems, the sequences do not store many of these irrelevant actions. Thus, the CBR recommendation is giving an additional heuristic to suggest the best successor even if there is another successor that also improves the heuristic value.

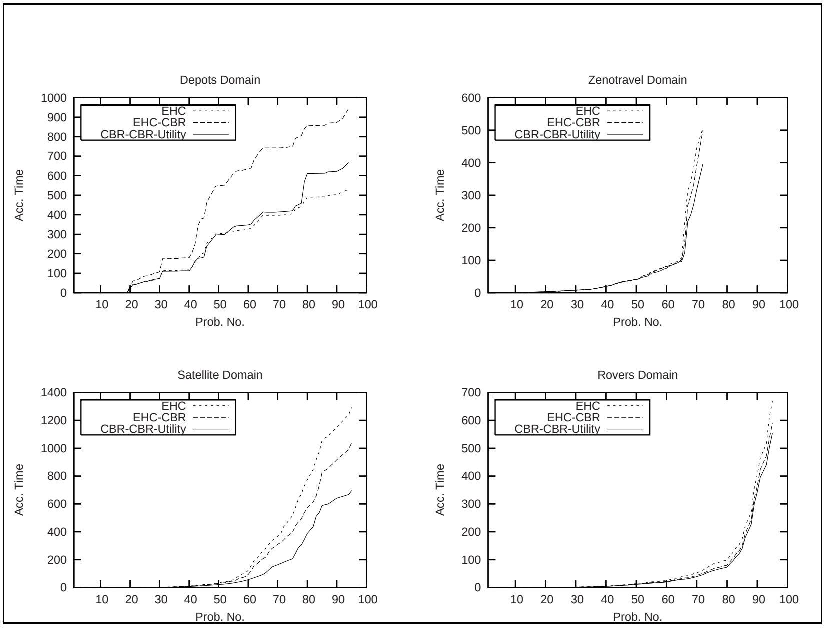


Fig. 3. Accumulated time of all techniques

5 Related Work

The relevance of object type transitions were shown by [6]. With a pre-processing tool, they obtain Finite State Machines that represent states in which a type of object can be and can move to (state invariants). A basic difference is that state invariants help planners to build efficient action schemas, needed to compute applicable actions, and in our approach typed sequences are used to guide the search ordering nodes during the search. Previous CBR approaches have supported different kinds of planning tasks. PARIS [7] stores cases in different abstraction levels of a solved problem, CAPlan-CBC [8] performs plan-space search and replay and Analogy [9] integrates CBR with generative planning based on a derivational analogy process, in which lines of reasoning are transfered and adapted to a new problem. More recently, in [10] CBR is applied to Hierarchical Task Networks Planning, another AI paradigm for planning. Our approach differs from these contributions, since ours is the first CBR approach applied to heuristic planning. Nevertheless, heuristic planning has been applied to support CBR retrieval. Tonidandel and Rillo [11] proposed a similarity metric based on the FF heuristic function. They use the heuristic estimation to measure the distance between the initial state of a problem and the initial state of a case, and between the problem goals and the goal state stored in the case. This idea was suitable for case-based planners that use whole plans as cases and needs significant effort to transform the solution to fit in the new problem, but this idea was not implemented to support heuristic planning.

6 Conclusions and Future Work

We have presented an approach that is based on previous work. The starting point is a CBR approach that advises a heuristic planner which is the best successor of each node to evaluate first during the search. In this paper, we describe a way of assessing the quality of the learned cases and results show that it can reduce the total planning time in some benchmark domains. This improvement is basically due to the reduction of the number of calls to the heuristic function, which is computationally expensive for planners.

In our future work we want to extend our system to numeric domains (incorporate cost functions different than plan length and handle numeric fluents). The IPC benchmark domains are very challenging in their numeric versions, and most planners can not find good solutions even for easy problems. We believe typed sequences, together with the quality of the solutions, could improve not only planning time, but also quality of solutions.

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

1. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
2. Vidal, V.: A lookahead strategy for heuristic search planning. In: Proceedings of the Fourteenth International Conference on Automated Planning and Scheduling, pp. 150–160 (2004)
3. Chen, Y., Hsu, C.W., Wah, B.: SGPlan: Subgoal partitioning and resolution in planning. In: ICAPS'04. Proceedings of the 4th International Planning Competition (IPC4) in Conference, pp. 30–33 (2004)
4. DelaRosa, T., Borrajo, D., Garcfa-Olaya, A.: Replaying type sequences in forward heuristic planning. In: Ruml, W., Hutter, F. (eds.) Technical Report of the AAAI'06 Workshop on Learning for Search, Boston, MA, AAAI Press, Stanford (2006)
5. de la Rosa, T., García-Olaya, A., Borrajo, D.: Case-based recommendation for node ordering in planning. In: Dankel II, D. (ed.) Proceedings of the 20th International FLAIRS Conference, Key West, FL, AAAI Press, Stanford (2007)
6. Fox, M., Long, D.: The automatic inference of state invariants in TIM. Journal of Artificial Intelligence Research 9, 317–371 (1998)
7. Bergmann, R., Wilke, W.: Paris: Flexible plan adaptation by abstraction and refinement. In: Voss, A. (ed.) ECAI (1996). Workshop on Adaptation in Case-Based Reasoning, John Wiley & Sons, Chichester (1996)

8. Muñoz-Avila, H., Paulokat, J., Wess, S.: Controlling nonlinear hierarchical planning by case replay. In: in working papers of the Second European Workshop on Case-based Reasoning, Chantilly, France, pp. 195–203 (1994)
9. Veloso, M.M., Carbonell, J.G.: Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. Machine Learning 10(3), 249–278 (1993)
10. Macedo, L., Cardoso, A.: Cased-based, decision-theoretic, HTN-Planning. In: Funk, P., González Calero, P.A. (eds.) ECCBR 2004. LNCS (LNAI), vol. 3155, Springer, Heidelberg (2004)
11. Tonidandel, F., Rillo, M.: An accurate adaptation-guided similarity metric for case-based planning. In: Aha, D.W., Watson, I. (eds.) ICCBR 2001. LNCS (LNAI), vol. 2080, pp. 531–545. Springer, Heidelberg (2001)