Bootstrap Planner: an Iterative Approach to Learn Heuristic Functions for Planning Problems

Shahab Jabbari Arfaee, Robert C. Holte
Computing Science Department
University of Alberta
Edmonton, AB Canada T6G 2E8
{jabbaria,holte}@cs.ualberta.ca

Sandra Zilles
Computer Science Department
University of Regina
Regina, SK Canada S4S 0A2
zilles@cs.uregina.ca

Abstract

Bootstrap planner is an iterative planner that improves an initial heuristic function by solving a sequence of successively larger training problems. It solves a set of very small problems using the initial heuristic function and learns a new heuristic from the plans found for the training problems. The planner then iteratively uses the newly learned heuristic to solve larger problems and learn better heuristics.

Introduction

Planners that competed in the first learning track of the international planning competition in 2008 (see Fern *et al.* (Fern, Khardon, and Tadepalli 2011) for more details about the competition) can be categorized into two major groups. The first group is consisted of the planners that focused on learning domain-specific macro-actions. For example, the Wizard planner (Newton et al. 2008) that performed well in terms of both quality and time metrics, uses the plan obtained from solving small training problems and generalizes the plan replacing the objects in the plan by variables. Then it builds macros from the generalized plan and only keeps the macros that improve the search performance comparing to a search that does not include the macros.

The PbP planner (Gerevini, Saetti, and Vallati 2009), which was the overall winner of the competition, used dovetailing (Valenzano et al. 2010) between a set of state-of-the-art satisficing planners in addition to using macros. The learning phase of the planner analyzes the performance of different planners on the training problems of each domain. Then, it allocates different amount of time to each planner in each domain depending on the performance of the planner on the training problems of the domain.

The success of such macro-learning planners depends on the amount of generalization that can be obtained from learning macros as they are learned from solving smaller training problems and are used to improve the search performance on larger test problems.

The second group of planners focused on learning a policy to guide the search. For example, the Obtuse Wedge planner (Yoon, Fern, and Givan 2008), that won the best learner award in the competition, first learns a reactive policy and then uses it to guide a best-first search. The planner uses a set of features extracted from the FF's relaxed plan (Hoffmann and Nebel 2001) and learns a reactive policy from solving the training problems. A reactive policy is a policy that maps each state in the problem to an action. During the search, the planner only adds the nodes that are created by the policy and their neighbors to the open list of a greedy best first search. Intuitively, this method uses the choices made by the policy to speed up the search, while overcoming the flaws of the policy by adding the neighbors of states selected by the policy to the open list.

Here we take a different approach. We learn a specific heuristic for each problem domain. Specifically, we learn a set of domain-specific weights to combine the features of the domain. Our approach is very close to the bootstrapping algorithms discussed in Jabbari Arfaee *et al.* (Jabbari Arfaee, Zilles, and Holte 2010) and Fern *et al.* (Fern, Yoon, and Givan 2004).

The algorithm discussed in Jabbari Arfaee *et al.* (2010) iteratively improves an initial heuristic function by solving a set of successively more difficult training problems in each iteration. This algorithm is ideal when a batch of problems with the same goal state should be solved quickly. It starts with a time limited search on the set of training instances using the initial (weak) heuristic function. Solving a sufficient number of the training instances at the end of the iteration, it learns a heuristic function from the training instances that are solved in this iteration. Failing to solve enough training instances in an iteration, the bootstrap algorithm increases the time limit to fill the gap between the difficulty of training instances and the weakness of the current heuristic function. This process is then repeated on the unsolved training instances using the newly learned heuristic. This algorithm is shown to create effective heuristics for a variety of classic heuristic search domains.

Different from the previous system, Fern *et al.* (Fern, Yoon, and Givan 2004) learn a control knowledge from solving smaller problems and use it to solve larger problems. For example, they learn a policy from solving blocksworld problems with n blocks and use it to solve blocksworld problems with $(n+1)$ blocks. This algorithms is shown to create very effective policies for a variety of planning domains. Here, we aim at learning effective heuristic functions to solve planning problems using the ideas discussed above.

The Bootstrap Planner

Our planner starts with an initial heuristic function and uses it to solve a set of training instances. There is a time limit assigned for solving each training instance. If enough training instances have been solved in an iteration, *i.e.,* a number of instances above a fixed threshold, the planner learns a better heuristic from the solved instances and repeats this process on a larger set of training instances with the newly created heuristic. If the planner fails to solve enough training instances, it increases the time limit and tries to solve the training instances using the new time limit.

The heuristic is learned as follows. For each state on the plan of the solved instances, a set of features of the state and the distance of the state from the goal state are collected as the training data. The training data, that contains a wide range of distances to the goal, is then fed to a learning system to predict the distance to goal for new states. We used the following settings in our experiments.

The features used for our experiments were four heuristics, namely, the FF heuristic (Hoffmann and Nebel 2001), the landmark-cut heuristic (Helmert and Domshlak 2009), the h_{max} heuristic used by HSP (Bonet and Geffner 2001) and a heuristic that counts the number of goal predicates that are not satisfied for each state. Among these heuristics, only the landmark-cut heuristic is admissible and we used this heuristic as the initial heuristic of the planner. We chose these features to provide a diversity for different domains while maintaining a reasonable speed of heuristic computation for each state during the search.

The number of training instances for each iteration was set at 200 and the minimum number of instances that should be solved in each iteration was set at 50. The initial time limit was set at 8 seconds and it was increased by a factor of 2 whenever less than 50 training instances were solved in each iteration. The learning algorithm used was a neural network with one output neuron representing distance-to-goal and three hidden units trained using backpropagation (Rumelhart, Hinton, and Williams 1986) and mean squared error (*MSE*). Training was ended after 500 epochs or when *MSE*< 0.005.

The bootstrap planner is built on the Fast Downward (Helmert 2006) planner. The search algorithm used is greedy best first search that uses an alternation open list (Röger and Helmert 2008) with one queue for the learned heuristic and one queue for the preferred operator heuristic (Hoffmann and Nebel 2001). The search also uses deferred evaluation of heuristic function (Helmert 2006). All these choices were made based on the intuition that the planner should scale to solve large planning problems. The final heuristic that is learned by the planner is the output of the planner and will be used to solve the test problems.

The problem generator provided for each domain is used to create the training problems ranging from very small to very large problems. One possible addition to the planner would be to create the training instances for each iteration in an increasing order of difficulty using the random walk backwards from the goal state, similar to the approach used by Jabbari Arfaee *et al.* (Jabbari Arfaee, Zilles, and Holte 2010). As the goal states for planning problems are partially defined, techniques such as the one suggested by Kolobov *et al.* (Kolobov, Mausam, and Weld 2010) should be used to make a complete state from the goal state so that random walk backwards from the goal becomes feasible.

References

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

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning* 1–27.

Fern, A.; Yoon, S.; and Givan, R. 2004. Learning domain-specific control knowledge from random walks. In *International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 191–199.

Gerevini, A.; Saetti, A.; and Vallati, M. 2009. An automatically configurable portfolio-based planner with macro-actions: Pbp. In *International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 350–353.

Helmert, M., and Domshlak, C. 2009. Landmarks, critical paths and abstractions: What's the difference anyway? In *International Conference on Automated Planning and Scheduling (ICAPS 2009)*, 162–169.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research* 26:191–246.

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

Jabbari Arfaee, S.; Zilles, S.; and Holte, R. C. 2010. Bootstrap learning of heuristic functions. In *Symposium on Combinatorial Search (SoCS 2010)*, 52–60.

Kolobov, A.; Mausam; and Weld, D. S. 2010. Sixthsense: Fast and reliable recognition of dead ends in MDPs. In *AAAI Conference on Artificial Intelligence (AAAI 2010)*, 1108–1114.

Newton, M. A. H.; Levine, J.; Fox, M.; and Long, D. 2008. Learning macros that are not captured by given example plans. In *Poster Papers at the International Conference on Automated Planning and Scheduling (ICAPS 2008)*.

Röger, G., and Helmert, M. 2008. The more, the merrier: Combining heuristic estimators for satisficing planning. In *International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 246–249.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1986. Learning internal representations by error propagation. *Parallel distributed processing: explorations in the microstructure of cognition* 318–362.

Valenzano, R. A.; Sturtevant, N. R.; Schaeffer, J.; Buro, K.; and Kishimoto, A. 2010. Simultaneously searching with multiple settings: An alternative to parameter tuning for suboptimal single-agent search algorithms. In *International Conference on Automated Planning and Scheduling (ICAPS 2010)*, 177–184.

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