mGPT: A Probabilistic Planner based on Heuristic Search

Blai Bonet
Departamento de Computación
Universidad Simón Bolívar
Caracas, Venezuela
bonet@ldc.usb.ve

Héctor Geffner
Departament de Tecnologia
Universitat Pompeu Fabra
Barcelona 08003, España
hector.geffner@tecn.upf.es

Abstract

We describe the version of the GPT planner to be used in the planning competition. This version, called mGPT, solves MDPs specified in the PPDDL language by extracting and using different classes of lower bounds, along with various heuristic-search algorithms. The lower bounds are extracted from deterministic relaxations of the MDP where alternative probabilistic effects of an action are mapped into different, independent, deterministic actions. The heuristic-search algorithms, on the other hand, use these lower bounds for focusing the updates and delivering a consistent value function over all states reachable from the initial state with the greedy policy.

Introduction

mGPT is a planner based on heuristic search for solving MDP models specified in the high-level planning language PPDDL. mGPT captures a fragment of the functionality of the GPT system that features non-determinism and incomplete information, in both qualitative and probabilistic forms, like POMDPs and Conformant planning (Bonet & Geffner 2001a; Bonet & Thiébaux 2003).

mGPT supports several algorithms and heuristic functions (lower bounds) that when combined generate a wide range of different solvers. The two main algorithms are `lrtdp` and `hdp`. Both are heuristic-search algorithms that make use of a given initial state s_0 and lower bound information. More precisely, they compute a value function V with a residual bounded by a user-provided threshold over all states reachable from s_0 when using the greedy policy π_V (Bonet & Geffner 2003b; 2003a).

The lower bounds are derived by solving relaxations of the input problem with an algorithms provided by mGPT. Since these algorithms are also based on heuristic search, we have implemented "stackable" components that are created in sequence for computing complex heuristic functions from simpler ones.

In this short document, we describe the features of the mGPT planner. This document is organized as follows. In the following two sections, we give a brief description of the most important algorithms and heuristics functions implemented in mGPT. Then, we describe how these algorithm and heuristics can combined in order to generate a wide range of different solvers. We conclude with a short discussion.

Algorithms

We divide the algorithms in two groups of optimal and suboptimal algorithms.

An optimal algorithm is one that computes an ϵ-consistent value function V over all states reachable from the initial state s_0 with the greedy policy with respect to V, denoted as π_V. A value function V is ϵ-consistent at state s if it residual at s is less than or equal to ϵ. It is known that if V is 0-consistent over all states reachable from s_0 with π_V, then π_V is optimal, as well as if V is ϵ-consistent for a sufficiently small ϵ. Here ϵ is a user-provided parameter.

The suboptimal algorithms, on the other hand, are provided in order to interleave planning and execution. In this group, we include algorithms that start selecting actions with respect to an initial lower bound (heuristic) that is improved over time.

(Although our main interest is towards optimal algorithms, we have included the suboptimal ones in order to cope with the format of the competition.)

The main optimal algorithms are `vi`, `lrtdp` and `hdp`, whilst the suboptimal ones are `asp` and `hdp-i`. In the following, we give a brief description and references for these algorithms.

The Value Iteration algorithm (`vi`) solves the problem in two steps. First, it generates the reachable state space from the initial state and the applicable operators, and second, uses the Value Iteration algorithm to obtain an optimal solution for the problem. `vi` is included in mGPT as a bottom-line reference.

Labeled Real-Time Dynamic Programming (`lrtdp`) is a heuristic-search algorithm that implements a labeling scheme on top of the `rtdp` algorithm (Barto, Bradtke, & Singh 1995). `lrtdp` works by performing simulated trials that start at the initial state and end at "solved" states by selecting actions with respect to π_V. Initially, V is the input heuristic function, and

the only solved states are the goal states. Then, each time an action is picked at state s, the value of s is updated by making its value consistent with the value of its successor states. At the end of each trial, a labeling procedure is called that checks whether new states can be labeled as solved: a state is solved if its value and the value of all its descendents are ϵ-consistent. The algorithm ends when the initial state is labeled solved since, at that time, all states reachable from s_0 with π_V are consistent. As shown in (Bonet & Geffner 2003b), this labeling mechanism adds a crisp termination condition to `rtdp` that features faster convergence time while retaining its good anytime behavior.

Since π_V, the policy returned by `lrtdp`, is only guaranteed to be optimal over a subset of states, i.e. s_0 and those reachable from it, then π_V is said to be a partial optimal policy *closed with respect to* s_0.

Heuristic Dynamic Programming (`hdp`) is also a heuristic-search algorithm that computes a partial optimal policy closed with respect to s_0. The `hdp` algorithm works by performing depth-first searches in state space looking for ϵ-inconsistent states, and then updating their values to make them consistent. The searches are stopped when no inconsistent states are found (Bonet & Geffner 2003a).

Action Selection for Planning (`asp`) is a reactive algorithm that starts by selecting actions with respect to the input heuristic function. Each time an action is needed for state s, `asp` performs multiple depth-bounded `rtdp`-like trials starting at s before returning an action for s. These simulations implement a bounded-lookahead mechanism that improve the action selection task. This `asp` algorithm is a generalization of (Bonet, Loerincs, & Geffner 1997) for probabilistic planning.

Approximated Heuristic DP (`hdp-i`) is a heuristic-search algorithm that like `hdp` performs searches and updates. Unlike `hdp`, the `hdp-i` algorithm only enforces consistency over all states reachable from s_0 with plausibility no smaller than i. These plausibility levels form a *qualitative* scale based on *kappa* rankings (Spohn 1988; Pearl 1993) that quantify how improbable is to make a transition from the initial state to the given state. The `hdp-i` algorithm and some of its properties are described in (Bonet & Geffner 2003a).

Heuristics

The heuristics functions are also divided in two groups of admissible and non-admissible heuristics. An admissible heuristic is one that never overestimates the optimal cost, i.e. a lower bound. The main admissible heuristics are `zero`, `min-min`, `atom-min-forward` and `atom-min-backward`, whilst the main non-admissible heuristic is `ff`. All these heuristic are computed by solving deterministic relaxations of the input problem. In the case of admissible heuristics, these relaxations must be solved optimally (Pearl 1983).

The most important relaxations are the weak and strong relaxations. The weak relaxation is computed by transforming the input problem into a deterministic problem in which every operator of the form

$$\langle\, \texttt{prec}, [\, p_1 : \alpha_1, \ldots, p_n : \alpha_n \,] \,\rangle\,, \qquad (1)$$

where `prec` is the precondition and α_i is the i-th effect with probability p_i, is translated into the n deterministic and independent operators $\langle\, \texttt{prec}, \alpha_i \,\rangle$.

It is not hard to show that the optimal solution for the weak relaxation is a lower bound one the optimal solution for the original problem.

The strong relaxation is a STRIPS problem computed by firstly transforming the input into a problem in which every operator is of the form

$$\langle\, \texttt{prec}, [\, p_1 : (\texttt{add}_1, \texttt{del}_1), \ldots, p_n : (\texttt{add}_n, \texttt{del}_n) \,] \,\rangle \quad (2)$$

where $\texttt{prec}, \texttt{add}_1, \ldots, \texttt{del}_n$ are all *conjunctions* of literals and $\sum_i p_i = 1$. Observe that in order to take the input problem into the form given by (2), we must remove disjunctive preconditions, conditional effects, quantifier symbols, etc. The strong relaxation is then generated by translating each operator (2) into the n deterministic and independent STRIPS operators

$$\langle\, \texttt{prec}, \texttt{add}_i, \texttt{del}_i \,\rangle\,. \qquad (3)$$

As before, it is not hard to show that the optimal solution for the strong relaxation is a lower bound on the optimal solution for the original problem.

In the following, we give a brief description of the different heuristic and their relation to the relaxations.

The Min-Min (`min-min`) heuristic is the optimal solution to the deterministic problem given by the weak relaxation. Two flavors are provided: `min-min-lrtdp` that solves the relaxation with a deterministic version of `lrtdp` (a.k.a. `lrta` (Korf 1990)), and `min-min-ida*` that solves the relaxation with IDA*. Both versions are *lazy* in the sense that the values are computed on a need basis as the planner requires them. See (Bonet & Geffner 2003b; 2003a) for references. (Since the `min-min` heuristic is computed with a heuristic-search algorithm, another heuristic function is required for its computation. Below, we describe how to specify these multiple heuristics.)

Atom-Min Forward (`atom-min-forward`) is a heuristic function computed in *atom space* from the strong relaxation. `atom-min-forward` computes "costs" of reaching set of atoms of fixed cardinality from a given state. The name forward comes from the fact that the costs are computed by a forward-chaining procedure that begins with the given state and ends when the goal is generated. This heuristic is a generalization of the $h_{\min}$ heuristic in HSP (Bonet & Geffner 2001b). As in `min-min`, the heuristic values are computed on demand. `atom-min-k-forward` refers to the `atom-min-forward` heuristic for sets of cardinality k. The `atom-min-forward` heuristic is from (Haslum & Geffner 2000).

Atom-Min Backward (`atom-min-backward`) is a heuristic similar to `atom-min-forward` except that it

computes costs of reaching sets of atoms from the *goal* state in an inverted version of the strong relaxation. Thus, before the search starts, all costs for all sets of atoms of fixed cardinality are computed and stored in a table that are later used to compute the heuristic function. The inverted relaxation is described in (Bonet & Geffner 2001b).

The FF (`ff`) heuristic implements the heuristic function used in the FF planner with respect to the strong relaxation (Hoffmann & Nebel 2001). This heuristic is informative but non-admissible and can only be used for non-optimal planning.

Combining Algorithms and Heuristics

The main parameters for mGPT are "`-p <planner>`" that specify the algorithm to use for the planner, "`-h <heuristics>`" that specify the heuristic function, and "`-e <epsilon>`" that specify the threshold ϵ for the consistency check.

One typical call looks like

```
mGPT -p lrtdp \
     -h "atom-min-1-forward" \
     -e .001 <rest>
```

which instructs mGPT to use the `lrtdp` algorithm with the `atom-min-1-forward` heuristic and $\epsilon = 0.001$. Since the algorithm is optimal and the heuristic is admissible, this call produces an optimal policy. The `atom-min-1-forward` heuristic is admissible but very weak. The following example shows how to use the `min-min-lrtdp` heuristic using `atom-min-1-forward` as the base heuristic:

```
mGPT -p lrtdp \
     -h "atom-min-1-forward|min-min-lrtdp" \
     -e .001 <rest>
```

Note how the pipe symbol is used to stack the components of the heuristic function.

Another possibility is to use mGPT as a reactive planner in which decisions are taken on-line with respect to a heuristic function that is improved over time. For example,

```
mGPT -p asp -h "ff" <rest>
```

uses the `asp` algorithm with the `ff` heuristic, while

```
mGPT -p asp -h "zero|min-min-ida*" \
     -e .001 <rest>
```

uses the `asp` algorithm with the `min-min-ida*` heuristic computed from the constant-zero heuristic. In the first case, the heuristic being used is non-admissible, so the planner will deliver a suboptimal policy. In the latter case, the `asp` algorithm is seeded with an admissible heuristic so it is guaranteed to converge to a partial optimal policy as the number of trials increase.

Other combinations of algorithms and heuristics are possible. mGPT also implements other heuristic functions and parameters to control number of simulation trials and cutoff length for `asp`, initial hash size, heuristic weight, dead-end value, verbosity level, etc.

Discussion

At the moment of writing these pages, it is not clear for us which combination of algorithm and heuristic is going to be used during the competition. Moreover, we could enter the competition either with a fixed choice, or with a more complex planner that picks a choice upon an analysis of the input problem. In any case, we plan to evaluate (after the competition) the different choices separately in order to obtain meaningful data for future research.

The mGPT planner will be publicly available after the competition with the default settings corresponding to those actually used.

Acknowledgements We thank the chairs of IPC-4 for making this competition possible. mGPT was built upon a source code developed by John Asmuth from CMU and distributed by the organizers.

References

Barto, A.; Bradtke, S.; and Singh, S. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72:81–138.

Bonet, B., and Geffner, H. 2001a. GPT: a tool for planning with uncertainty and partial information. In *Proc. IJCAI/Workshop on Planning with Uncertainty and Partial Information*, 82–87.

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

Bonet, B., and Geffner, H. 2003a. Faster heuristic search algorithms for planning with uncertainty and full feedback. In *Proc. IJCAI-03*, 1233–1238.

Bonet, B., and Geffner, H. 2003b. Labeled RTDP: Improving the convergence of real-time dynamic programming. In *Proc. ICAPS-03*, 12–21.

Bonet, B., and Thiébaux, S. 2003. GPT meets PSR. In *Proc. ICAPS-03*, 102–111.

Bonet, B.; Loerincs, G.; and Geffner, H. 1997. A robust and fast action selection mechanism for planning. In *Proc. AAAI-97*, 714–719.

Haslum, P., and Geffner, H. 2000. Admissible heuristic for optimal planning. In *Proc. AIPS-2000*, 140–149.

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

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2–3):189–211.

Pearl, J. 1983. *Heuristics*. Morgan Kaufmann.

Pearl, J. 1993. From conditional oughts to qualitative decision theory. In *Proc. UAI-93*, 12–22.

Spohn, W. 1988. A general non-probabilistic theory of inductive reasoning. In *Proc. UAI-88*, 149–158.