Width and Inference Based Planners: *SIW*, *BFS(f)*, and *PROBE*

Nir Lipovetzky
University of Melbourne
Melbourne, Australia
@unimelb.edu.au

Miquel Ramirez
RMIT University
Melbourne, Australia
@rmit.edu.au

Christian Muise
University of Melbourne
Melbourne, Australia
@unimelb.edu.au

Hector Geffner
ICREA & Universitat Pompeu Fabra
Barcelona, SPAIN
@upf.edu*

Introduction

We entered the planners *SIW*, $BFS(f)$, and $PROBE$ to the *agile-track* of the 2014 International Planning Competition, and an anytime planner for the *satisficing track* that runs both *SIW* and $BFS(f)$. SIW and $BFS(f)$ are planners that make use of a notion of width for classical planning (Lipovetzky and Geffner 2012), while PROBE is a standard best-first search planner that augments the expansion of a node by throwing a probe which either reaches the goal or terminates in low polynomial time (Lipovetzky and Geffner 2011).

The basic building block of *SIW* is the Iterative Width Procedure (*IW*) for achieving atomic goals. *IW* runs in time exponential in the problem width by performing a sequence of pruned breadth first searches. The planner $BFS(f)$ integrates a *novelty* measure borrowed from *IW* with helpful-actions, landmarks and delete-relaxation heuristics in a Greedy Best-Fist search.

In the following sections we introduce the basic notions of the algorithms and the implementation. We assume a STRIPS problem $P = \langle F, I, O, G\rangle$, where F is the set of atoms, I is the set of atoms characterizing the initial state, O is the set of actions, and G is the set of goal atoms.

SIW: Iterated Width Search

The algorithm *Iterated Width*, or *IW*, consists of a sequence of calls *IW*(i) for $i = 0, 1, \ldots, |F|$ until the problem is solved. Each iteration *IW*(i) is a breadth-first search that prunes right away states that do not pass a *novelty* test; namely, for a state s in *IW*(i) *not* to be pruned there must be a tuple t of at most i atoms such that s is the first state generated in the search that makes t true. The time complexities of *IW*(i) and *IW* are $O(n^i)$ and $O(n^w)$ respectively where n is $|F|$ and w is the problem width. The width of existing domains is low for atomic goals, and indeed, 89% of the benchmarks can be solved by *IW*(2) when the goal is set to any one of the atoms in the goal (Lipovetzky and Geffner 2012). The width of the benchmark domains with conjunctive goals, however, is not low in general, yet such problems can be serialized.

The algorithm *Serialized Iterative Width*, or *SIW*, uses *IW* for serializing a problem into subproblems and for solving the subproblems. SIW uses IW to greedily achieve one atomic goal at a time until all atomic goals are achieved jointly. In between, atomic goals may be undone, but after each invocation of IW, each of the previously achieved goals must hold. SIW will thus never call IW more than $|G|$ times where $|G|$ is the number of atomic goals. SIW compares surprisingly well to a baseline heuristic search planner based on greedy best-first search and the h_{add} heuristic (Bonet and Geffner 2001), but does not approach the performance of the most recent planners. Nonetheless, *SIW* is able to compete well in domains amenable to simple serializations where the induced problems can be solved by *IW*(i) with $i \leq 3$.

BFS(f): Novelty Best-First Search

While the blind-search *SIW* procedure competes well with a greedy best-first planner using the additive heuristic, neither planner is state-of-the-art. Since state-of-the-art performance is important in classical planning, we show next that it is possible to deliver such performance by integrating the idea of novelty that arises from width considerations, with known techniques such as helpful actions, landmarks, and heuristics. For this, we switch to a standard *forward-search best-first planner* guided by an evaluation function that combines the notions of novelty and helpful actions (Lipovetzky and Geffner 2012; Hoffmann and Nebel 2001). Additionally, ties are broken lexicographically by two other measures: (1) the number of subgoals not yet achieved up to a node in the search, and (2) the additive heuristic, h_{add}.

PROBE

PROBE is a complete, standard greedy best first search (GBFS) STRIPS planner using the standard additive heuristic (Bonet and Geffner 2001), with just *one change*: when a state is selected for expansion, it first launches a *probe* from the state to the goal. If the probe reaches the goal, the problem is solved and the solution is returned. Otherwise, the states expanded by probe are added to the open list, and control returns to the GBFS loop. The crucial and only novel part in the planning algorithm is the definition and computation of the probes (Lipovetzky and Geffner 2011).

PROBE is built using an early-version of an automated planning toolkit that supports the implementation details of

*firstname.lastname

Copyright © 2014, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

the width-based algorithms. The only difference with respect to PROBE-IPC7 is that the anytime procedure is disabled, as we are only concerned with the first solution.

Implementation Notes

The planners *SIW*, $BFS(f)$ have been implemented using a novel automated planning framework, `lwaptk`[1]. The toolkit is an extensible C++ framework that decouples search and heuristic algorithms from PDDL parsing and grounding modules, by relying on planner "agnostic" data structures to represent (ground) fluents and actions. We consider `lwaptk` to be a valuable contribution as it allows for the construction of a planning system by combining readily available implementations of search algorithms and planning heuristics, without depending on specific parsing modules and grounding algorithms. Systems built in this manner can then be tested on IPC benchmarks by "plugging" the planner into any of several available parsing and grounding strategies and algorithms, most notably those of the FF (Hoffmann and Nebel 2001) planner and the FAST-DOWNWARD framework (Helmert 2006). Alternatively, and more interestingly, the planner can be embedded into complex applications, *directly*, if the "host" application is written in C++, or *indirectly* when the host is written in an interpreted language, such as PYTHON, by wrapping the planner with suitably generated marshalling code.

Experiments

We call the resulting best-first search planner, BFS(f), and compare it with three state-of-the-art planners: FF, LAMA, and PROBE (Hoffmann and Nebel 2001; Richter, Helmert, and Westphal 2008; Lipovetzky and Geffner 2011).[2] Like LAMA, BFS(f) uses delayed evaluation, a technique that is useful for problems with large branching factors (Richter and Helmert 2009).

. . . . I can generate a table with results over the last IPC taking 5min max, (agile track) and keeping just the time score.

On the other hand, runing the 30min anytime-algorithm will take ages..., and we should run lama-anytime as well for compare comparison. we may skip this, unless you think it's really meaningful...

Discussion

. . . .

Acknowledgments

References

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

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.

Lipovetzky, N., and Geffner, H. 2011. Searching for plans with carefully designed probes. In *Proceedings of the Twenty-First International Conference on Automated Planning and Scheduling (ICAPS 2011)*, 154–161.

Lipovetzky, N., and Geffner, H. 2012. Width and serialization of classical planning problems. In *Proceedings of the Twentieth European Conference on Artificial Intelligence (ECAI 2012)*, 540–545.

Richter, S., and Helmert, M. 2009. Preferred operators and deferred evaluation in satisficing planning. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009)*.

Richter, S.; Helmert, M.; and Westphal, M. 2008. Landmarks revisited. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008)*, 975–982.

[1] Source code available from `https://github.com/miquelramirez/lwaptk`.

[2] FF is FF2.3, while PROBE and LAMA are from the 2011 IPC.