Lazy optimization planner

By Nathan Wallheimer and Michael Goldman

June 11, 2017

Planner Description

The *Lazy Optimization Planner* is based on the *Fast-Downward* planner. Our contribution was to add a new feature to the iterated search component in *Fast-Downward*, that allows the next search to start from different starting points in the first solution (which we shall call: the **base plan**). This is done according to a threshold parameter. The threshold parameter is a number in the range $[0, 1)$, that defines a percentile of the base plan cost. When the next search starts, it uses the same operators of the base plan until the cost is larger than:

$$threshold \star (base\ plan\ cost)$$

With this new feature, one can improve the total cost of an existing plan by optimizing only the last parts of it.
The *Lazy Optimization Planner* uses this feature in configuration with some of *Fast-Downward's* search algorithms and heuristics. In more detail, the following configuration is used:

```
./fast-downward.py --overall-time-limit=30m --overall-memory-limit=2G $DOMAIN $PROBLEM
--heuristic h1="lmcount(lm_factory=lm_rhw(lm_cost_type=ONE),pref=true)"
--heuristic h2="ff()"
--heuristic h3="goalcount()"
--heuristic h4="ipdb(max_time=40)"
--search "iterated([
lazy(alt([
tiebreaking([h1,h2]),
tiebreaking([h1,h3]),
tiebreaking([h4,h2])
]),preferred=[h1,h2],cost_type=ONE),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=5,threshold=0.9),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=5,threshold=0.8),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=5,threshold=0.7),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=5,threshold=0.6),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=5,threshold=0.5),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=4,threshold=0.4),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=4,threshold=0.3),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=4,threshold=0.2),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=ONE,w=4,threshold=0.1),
lazy_wastar([h1,h2,h4],preferred=[h1,h2],cost_type=PLUSONE,w=2,threshold=0)
],cost_type=ONE,continue_on_fail=true,repeat_last=true)"
```

This configuration has 2 "phases". The first phase is for finding a solution fast, which is done by alternating on multiple tiebreaking open lists. The second phase is for improving the base plan by decreasing the threshold gradually on a *weighted $A^\star$ search*.

Notes:

- The threshold option was implemented only for the lazy search algorithms (more implementation details will be discussed further).
- Any configuration of the *Fast-Downward* planner works also in the *Lazy Optimization Planner*.

Planner Usage

To use the planner, build it by running the *build.py* file, and then send the domain and problem files as arguments to the *run* script:

```
./build.py
...
./run domain.pddl problem.pddl
```

Results

While experimenting with the planner we have gathered some results.

- It is better to find a lot of small improvements by decreasing the threshold gradually, than to invest time in large improvements that take time.
- For some problems, the planner fails to make large improvement in total, while for others it reduces the cost greatly. We assume that the reason is actions that take place in the beginning of the base plan.
- The iterated search consumes less memory in total when the searches in the list use high thresholds.

Implementation Details

To implement the new feature, we changed the successor operators generator for the lazy search algorithm. Whenever the search has not reached the cost threshold yet, the only operator returned by the generator is the next operator in the base plan.

The new code added by us is surrounded by "*NEW CODE*" comments and can be found in the follwing files:

1. *search_engine.h*, *search_engine.cc*
2. *iterated_search.h*, *iterated_search.cc*
3. *lazy_search.h*, *lazy_search.cc*

Discussion

The *Lazy Optimization Planner* showed positive results. However, in some problems it suffers from "bad" actions that take place in the beginning of the base plan and don't get improved.

During our experiments with the planner, we have had some ideas that could improve it:

- Generalize the idea, such that not only the last part of the base plan gets improved, but a constant number of parts in it. Possibly the split can be done according to landmarks in the base plan.
- Use eager search when the threshold is high, to get better improvements and pump the heuristics cache.
- Study the connection between the first phase algorithms that are used to find a base plan, and the second phase algorithms, that are used to improve it.

The *Lazy Optimization Planner* showed that for an efficient plan, one does not have to explore the search space from the initial state over and over again. It is also beneficial to use new initial states from a pre-computed base plan. We think that it is worth putting more effort in this direction, and hopefully more results will follow.