BRSU

Planning and Scheduling
Project Report

Bastian Lang

January 17, 2016

1 Why Symba-2

1.1 Track

I chose to work on project 1. Therefore I first checked the tracks of IPC'14 for a track that fits the given tasks. Sequential optimizing promised to be the correct track as it covers classical STRIPS planning and participating planners aim to find optimal plans in terms of least costs.

1.2 Domain

I then checked for a domain that is comparable to the domain of the given task. I found *tidybot* and *transportation* to be similar to the given task.

1.3 Comparing IPC results

Two planners scored best in these two domains: cGamer and Symba-2. Because Symba-2 performed better in the transport-domain, I chose to use this planner.

1.4 Installation and Running

To get the planner to run, one first needs to extract the planner. Executing the file "build" in the extracted folder compiles all the files of the planner. I needed to install missing packages, which could be identified by using Google for the error messages. I needed to install g++-4.8-multilib ("sudo apt-get install g++-4.8-multilib")

To run the planner and find a plan for a given problem, on need to execute the file "plan" in the extracted folder and specify a domain-file, a problem-file and an output-file ("plan domain.pddl problem.pddl output.out").

The planner will probably run into an error. This can be fixed by changing line 29 of the file src/search/downward to "PLANNER="$BASEDIR/downward-4"".

2 Results

2.1 Basic Fetch and Carry

Creating the domain and the problem files I ran into many syntax-errors using a simple editor without any syntax highlighting. After facing some of those problems I started using test problems for every defined action.

Coming up with a scalable line predicate took some time, because Symba is not able to use axioms and derived predicates. In the end I used a two- and a three-objects line predicate and some predicates to be able to say that an object is part of a line and an object is the last in line. I do not allow the robot to take objects out of a line because none of the goal states require building lines unless for the goal state.

Interesting to mention is that none of the created plans made use of the stage and unstage actions. Because there are no costs assigned, moving and moving back does cost the same as staging and unstaging an object.

2.1.1 Problem 1

The planner needed 0.22 seconds to come up with a reasonable solution.
It needed 221460 KB peak memory.
The plan needs 9 steps to reach the goal state.

2.1.2 Problem 2

The planner needed 0.12 seconds to come up with a reasonable solution.
It needed 262788 KB peak memory.
The plan needs 20 steps to reach the goal state.

2.1.3 Problem 3

The planner needed 2.18 seconds to come up with a reasonable solution.
It needed 308296 KB peak memory.
The plan needs 36 steps to reach the goal state.

2.2 Least Costs

Using least costs involves using a fluent "least-cost" and a function "distance ?s1 ?s2". The distances between all locations have to be specified in the problem definition. Then these costs can be added to least-cost as an effect of the move-action.
In the problem definition then has to be specified that minimizing this least-cost fluent is the goal ("(:metric minimize (total-cost))").

2.2.1 Problem 1

The planner needed 0.1 seconds to come up with a reasonable solution. It needed 221464 KB peak memory.
The plan needs 12 steps and has costs of 2 to reach the goal state.

2.2.2 Problem 2

The planner needed 0.16 seconds to come up with a reasonable solution.
It needed 262792 KB peak memory.
The plan needs 20 steps and has costs of 4 to reach the goal state.

2.2.3 Problem 3

The planner needed 0.54 seconds to come up with a reasonable solution.
It needed 270192 KB peak memory.
The plan needs 44 steps and costs of 14 to reach the goal state.

2.3 Limited Carrying Capacity

This task turned out to be tricky. I first tried to use another fluent to store the number of carried items. A precondition in the pick-action should prevent picking up if the limit has been reached. But Symba does not support any more fluents than the one for total-costs.

I then tried to use conditional effects and multiple predicates indicating the number of objects on the robot. But using multiple "when"-statements in the effect of an action did not work as well.

In the end I used some inspiration from the accessible domains and problems from the IPC. I now use a capacity parameters in the actions and some predicates defining which capacity precedes another one.

2.3.1 Problem 1

The planner needed 0.1 seconds to come up with a reasonable solution.
It needed 229800 KB peak memory.
The plan needs 12 steps and has costs of 2 to reach the goal state.

2.3.2 Problem 2

The planner needed 0.12 seconds to come up with a reasonable solution.
It needed 204820 KB peak memory.
The plan needs 20 steps and has costs of 8 to reach the goal state.

2.3.3 Problem 3

The planner needed 5.42 seconds to come up with a reasonable solution.
It needed 365496 KB peak memory. The plan needs 61 steps and costs of 19 to reach the goal state.

2.4 Rewards

PDDL provides the possibility to add rewards to actions and then maximize for it.