Assignment 8: STRIPS Planner
CS 730/830, Spring 2024
Due at 11:30pm on Wed Mar 27

Overview

You will write a state-space progression planner for classical sequential STRIPS domains. Given a domain description and a problem instance (an initial state and a goal specification), your program will return the shortest sequence of actions that will achieve the goal. More precisely, your program will use weighted A* search and, when used with an admissible heuristic, return a solution within a factor of w of optimal, where w is the weight used in the search. Everyone will implement a lame inadmissible heuristic and graduate students will implement reasonable admissible and inadmissible heuristics.

To gain familiarity with STRIPS, you should start by encoding a new planning domain and at least two problem instances for it (one simple and one that requires a plan of at least 6 steps). As part of your write-up, submit your new domain and problems along with a transcript (try the `script` program) of the reference solution solving them.

Input

Your program should read both parts of the input from standard input. A domain description specifies the predicates of the domain (all on one line), a set of constants that apply to any problem instance, the number of action schemata, and the action definitions. Each part of an action definition is on its own line in the following order: positive preconditions, negative preconditions, negative effects, positive effects. Predicates in preconditions and effects can take constants or variables as arguments. Comment lines start with #. For example:

```
# blocks world

predicates: On(x, y) Clear(x) OnTable(x)
constants: A B
3 actions

Move block src dest
# note that dest <> table
pre: On(block, src) Clear(block) Clear(dest)
preneg: OnTable(block)
del: On(block, src) Clear(dest)
add: On(block, dest) Clear(src)

ToTable block src
pre: On(block, src) Clear(block)
preneg: OnTable(block)
del: On(block, src)
add: OnTable(block) Clear(src)

FromTable block dest
pre: OnTable(block) Clear(block) Clear(dest)
preneg:
del: OnTable(block) Clear(dest)
add: On(block, dest)
```

A problem instance description starts with a line listing all additional constants beyond those listed in the domain. The initial state and the positive and negative goal specifications are each one line:

```
constants: C
initial: OnTable(A) On(B, A) Clear(B) OnTable(C) Clear(C)
goal: OnTable(C) On(B, C) On(A, B)
goalneg:
```

Your program should accept two command line arguments:

weight the real-valued weight your program should use for its weighted A* search ($f'(n) = g(n) + w \cdot h(n)$). It should be ≥ 1.0.

heuristic one of `h0` ($h(n) = 0$) or `h-goal-lits` (number of goal literals that are not satisfied). Graduate students must also support `h1` (the h_1 max heuristic) and `h1sum` (the h_1 sum heuristic).

You are welcome to reuse code that you wrote for previous assignments.

Output

Your program should write only the following to standard output: a list of instantiated actions, one per line, with each action labeled with the time it is to be executed, followed by the number of search nodes expanded and generated (one per line, in that order). For example:

```
0 Move B A C
1 FromTable A B
10 nodes generated
4 nodes expanded
```

If you do not find a plan, print `No plan found.` instead of the list of actions.

Supplied Utilities

We supply:

sample input an example domain and problems.

`strips-plan-validator` takes your program's name as its first argument and your program's arguments as its remaining arguments. Expects a problem on standard input. Runs your program on the problem, reads its output, and verifies that the resulting plan is legal.

`strips-plan-reference` a sample solution. It does some sanity checking on its input, so you might try using it to double-check any test instances you write.

Write-up

Electronically submit your solution using the instructions on the course web page, including your source code as well as a transcript of your program running with the validator and a brief write-up answering the following questions:

1. Describe any implementation choices you made that you felt were important. Clearly explain any aspects of your program that aren't working. Mention anything else that we should know when evaluating your work.

2. Which of your heuristics are admissible?

3. What can you say about the the time and space complexity of your program?

4. What suggestions do you have for improving this assignment in the future?

Evaluation

For grad students, the heuristics will be worth something like 4 points.

Design Suggestions

Develop the heuristics last.

When comparing against the reference solution, be sure you find equally short plans when using an admissible heuristic and $w = 1$.

For a faster and simpler search, start by grounding all the predicates with all the constants. Then perform all operations on these grounded representations. Now you don't have to worry about unifying against the current state as you go!

It is probably a good idea to detect duplicate states.

Another possible speed-up is to index the actions so that you can quickly select those that might be applicable.