CS 687 Fall'21 – Project 4 Report

Chien Wei, Hsiung
Sai Kishore, Salaka

Introduction:

This project contains a Partial Order Planner that takes in the Start and Goal states, along with a bunch of actions that can be performed in a specific order to achieve the solution, i.e., reach Goal from Start. The task here is to develop the core logic to implement the Planner along with possible heuristics that optimize the Partial Order Planning.

Implementation:

- A few implementations were straight-forward, such as `before-p()`, `link-exists()`, `inconsistent-p()`, `all-effects()`, `all-operators()`, `promote()`, `demote()`, etc.
- In `operator-threatens-link-p()`, the negated operator's effects are searched in the link-to's effects to determine a threat.
- In `pick-precond()`, all the operator and precondition pairs that don't currently have a link in the plan are considered.
- In `choose-operator()`, `hook-up-operator()` is performed for all the operators in the plan first. If the solution isn't found, then the hook-up is performed for all the operators that aren't in the plan, while adding the respective operator, link and ordering to the plan.
- In `hook-up-operator()`, appropriate link, ordering are added. If the operator isn't already in the plan, the ordering for the Start and Goal for the new operator. Then, the threats in the plan are identified and resolved accordingly.
- In `threats()`, a threat is identified in all the links and operators with the provided operator and link.
- In `all-promotion-demotion-plans()`, all combinations of the threats are taken and are either promoted or demoted to find a consistent plan (if present).

Heuristics:

All possible heuristic opportunities were addressed. Here are the notable heuristic implementations:

- In `reachable()`, the suggested implementation was done, i.e., maintaining lists of reachable and non-reachable, and move the items using transitivity. This improved the performance a lot (from a few minutes to 10s of seconds).
- In `operator-threatens-link-p()`, the simple scenarios such as the operator being before or after the link or the operator being the same as to/from of the link; were handled first to reduce the computation.
- In `pick-precond()`, the following implementations were considered:
    - [Very slow] A random pair is chosen – This took nearly 30-40 minutes to find a solution for the 3-blocks problem.
    - [Faster] The precondition with the least number of related operations is chosen – This was implemented as per the suggestion. It took about 30 seconds to provide a solution to the 3-block problem. A lot better than the previous implementation.

- [Fastest] The sum of the number of operators related to the precondition and the sum of the number of preconditions of the operator – While performing random testing to achieve better performance, this combination provided the best result. It took about 3 seconds to provide a solution to the 3-block problem.

- In `all-promotion-demotion-plans()`, the one threat scenario was handled separately to reduce the computation.

Experiments:

To extend the project, 2 problems from AIMA Planning chapter were propositionalized and tested with the implementation.

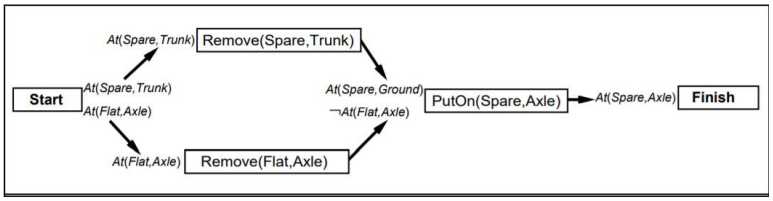
1. The simple spare tire problem [Fig. 11.3]:
   The problem description didn't have variables and was propositionalized as-is and didn't have a problem with the code. The result was as mentioned in AIMA.
2. A STRIPS problem involving transportation of air cargo between airports [Fig. 11.2]:
   As a challenge to try a difficult problem, the air cargo transport was taken. Initially, propositionalizing the problem as-is with all permutations of the cargos, planes, and airports didn't work. The issue was with the `Cargo(), Plane(), Airport()` methods, as they were causing irrelevant plans. As variables aren't being used and explicitly the entities were mentioned in the action names, there wasn't a lot of use with the predicates. Skipping those predicates provided a valid plan later.

Results:

The actual output logs are provided in the lisp file attached in the submission. Here's the plan for 4 problems solved with the submitted code:

1. **Two and Three block Blocks World Problem:**
   The implementation was able to find the solution as provided in the description, and in a very less time, 2-block got the solution immediately and 3-block got the solution in ~3 seconds.

2. **Spare Tire Problem:**
   The implementation found the exact solution to this problem as mentioned in AIMA.



3. **Air Cargo Transport Problem:**
   The solution for this was interesting. The problem is as below:
   Start: Cargo C1 and Plane P1 are at SFO; Cargo C2 and Plane P2 are at JFK.
   Goal: Cargo C1 is at SFO; Cargo C2 is at JFK.
   Expected Solution: P1 loads C1 from SFO, flies to JFK, and unloads C1 at JFK; P2 loads C2 from JFK, flies to SFO, and unloads C2 at SFO.

Achieved Solution: P1 loads C1 from SFO, flies to JFK, and unloads C1 at JFK; P1 (which is at JFK currently) loads C2 from JFK, flies to SFO, and unloads C2 at SFO.