CIS700 Homework4 Report

[redacted]

1 WikiHow Article

The wikiHow article I chose is about making papyrus paper from papyrus plant. I have learned from a course on Math History that Ancient Egyptians use papyrus plant to make papyrus paper, and I am curious how exactly it is achieved.

The article has four parts, where the first three parts are consecutive steps in making papyrus paper from papyrus plant, and the last part is making papyrus paper from newspaper. I only translated the first three parts into PDDL, which are preparing, making and finishing papyrus paper. I turned each part into a problem.

2 Details

The main steps of turning papyrus plant into the final paper products are: cutting plant and getting the stalk, cutting stalks into strips, processing strips and weaving into sheet, and finally processing sheet into papyrus paper. Each of the steps is turned into one or more actions, for example, processing strips in broke down to:

```
(:action soak_strips ;
      :parameters (?p - player ?loc - location ?strip - strip)
      :precondition (and (at ?p ?loc) (has_water ?loc) (inventory ?p ?strip))
      :effect (and (clean ?strip) (not (dry ?strip)))
)

(:action dry_out_strips ;
      :parameters (?p - player ?strip - strip)
      :precondition (and (inventory ?p ?strip) (clean ?strip))
      :effect (dry ?strip)
)
```

where soaking strips and drying out strips are two separate actions. Soaking strips requires the player to be at a location that has water, and the inventory must contain strip. The result of soaking is that strip becomes clean but wet. Drying out strips requires the player to have strip in their inventory and the strip must be clean. Intuitively, the result of drying out strip is that strip becomes dry.

The three problems I designed are: get strips, process strips, and finally get papyrus.

2.1 Problem 1: Get Strips

The first problem is to get strips from papyrus plant. The initial state is that the player is at the garage, and several tools are placed in the garage, and the papyrus plant is at the nursery:

```
(:init
      (connected garage west nursery)
      (connected nursery east garage)
      (connected garage in work_station)
      (connected work_station out garage)
```

```
        (at npc garage)
        (at scissors garage)
        (at shell garage)
        (at wooden_board garage)
        (at plant nursery)
        (has_water work_station)
        (has_hard_surface work_station)
)
```

The goal is for the player to have strip at the inventory:

```
(:goal (and (inventory npc strip)))
```

The solutions is as follows:

```
Time: 0.019362688064575195s
plan:
get scissors npc garage
go west npc garage nursery
get plant npc nursery
cut_plant npc scissors plant stalk
peel_stalk npc stalk
cut_stalk_into_strips npc scissors stalk strip
```

2.2 Problem 2: Process Strips

The second problem is to process the strips the player has obtained from problem one. The initial state assumes that the player has scissors and strip in the inventory, several useful tools are in the garage, and the work station has both water and hard surface:

```
 (:init
        (connected garage west nursery)
        (connected nursery east garage)
        (connected garage in work_station)
        (connected work_station out garage)
        (at npc garage)
        (at scissors garage)
        (at shell garage)
        (at wooden_board garage)
        (inventory npc strip)
        (inventory npc scissors)
        (has_water work_station)
        (has_hard_surface work_station)
)
```

The goal is for the player to obtain a roughly processed sheet through processing strips and weaving:

```
(:goal (and (inventory npc sheet) (flat sheet)))
```

The solutions is as follows:

```
Time: 0.007283449172973633s
plan:
get wooden_board npc garage
go in npc garage work_station
soak_strips npc work_station strip
dry_out_strips npc strip
weave_strips_into_sheet npc work_station strip sheet
press_sheet npc work_station wooden_board sheet
```

2.3 Problem 3: Get Papyrus

The third problem is to finally process the sheet and get the papyrus paper. The initial state assumes the player has already gotten sheet made out of papyrus plant, and the sheet has already been roughly processed to be flat:

```
(:init
      (connected garage west nursery)
      (connected nursery east garage)
      (connected garage in work_station)
      (connected work_station out garage)
      (at npc garage)
      (at shell garage)
      (inventory npc scissors)
      (inventory npc sheet)
      (flat sheet)
      (has_hard_surface work_station)
)
```

The goal is for the player to obtain the final product, which is the papyrus:

```
(:goal (inventory npc papyrus))
```

The solutions is as follows:

```
Time: 0.006060361862182617s
plan:
get shell npc garage
go in npc garage work_station
flatten_sheet npc work_station sheet
polish_sheet npc work_station sheet shell
cut_sheet npc garage sheet scissors papyrus
```

3 Thoughts

It is somewhat tedious to specify the type of parameters for non-general actions. For example, for a very general action GET, the parameter contains a very general type ITEM. In contrast, for specific action like weaving strips into sheet, we have to specify the type of parameters to be STRIP and SHEET, because this action has very narrow definition, and thus we have to declare the subtype of each involved parameter. In general I feel like PDDL is more powerful when it is describing more general problems and actions, where one type of parameter could be refered to many objects.

That being said, I think PDDL is a legit way to create interesting challenges, as it is perfect for setting preconditions, which is the gist of puzzles in text-adventure-style game.

We might be able to use GPT-3 to generate predicates and object types for PDDL. For actions, we could either let GPT-3 to generate preconditions or results but not both, as it would be hard to force GPT-3 generate solvable problem. In general, I think it's impossible for GPT-3 to generate full PDDL document, but we could use have some components generated and then complete the file manually.