B(E)3M33UI — Exercise HTN: Hierarchical Task Net (HTN) Planning

Radek Mařík

May 15, 2018

1 HTN planning

The goal of this task is to become familiar with hierarchical planning by writing a planning definition in Pyhop. The planning goal will be control a group of soldiers in a predefined domain.

Basic ideas of HTN planning:

- Complex plans often have identifiable structure
- Structure can often be captured in the form of hierarchies of abstract sub-plans
- Sub-plans are often (nearly) independent of one another

We will use the **Pyhop** planner (`https://bitbucket.org/dananau/pyhop/src`), which is a simple HTN planner written in python (planning algorithm is similar to that one used in SHOP).

Installation: Download the package from course web page that contains: `pyhop.py` and examples: `travel_dana_nau.py,blocks.py`.

You either implement the blocks example of soldiers example.

1.1 Travel example

A very simple example of traveling from home to park (consult lectures for more info). It shows how to define operators (description of what basic actions do) and methods (description of a possible way to perform a compound task by performing a collection of sub-tasks).

Only operators can make changes to the state of the world! Pyhop will allow you to change state variables even in methods but this is not good practice.
Definition of initial state using dict:

```
state1 = pyhop.State('state1')
state1.loc = {'me': 'home'}
state1.cash = {'me': 20}
state1.owe = {'me': 0}
state1.dist = {'home': {'park': 8}, 'park': {'home': 8}}
```

Definition of goal state:

```
goal = ('travel', 'me', 'home', 'park')
```

'travel' is a method, and variables $a = me$, $x = home$, $y = park$

Operators (a = agent, i.e. me)

- **walk** ?a from ?x to ?y (pre-condition: ?a at ?x, effect: ?a at ?y)
- **call_taxi** to location ?x (pre-condition: None, effect: taxi at ?x)
- **ride_taxi** from ?x to ?y (pre-condition: ?a and taxi at ?x, effect: ?a and taxi at ?y)
- ...

Methods: description how to decompose problem to sub-task

```
def travel_by_foot(state, a, x, y):
    if state.dist[x][y] <= 2:
        return [('walk', a, x, y)]
    return False

pyhop.declare_methods('travel', travel_by_foot, travel_by_taxi)
```

Task 1: Get familiar with the travel example: `travel_dana_nau.py`

Task 2: Modify the example such that you can ride a bike if distance is less than 10 km.

1.2 OPTION 1: Blocks

Another simple example but providing nice domain to demonstrate and study planning. The domain consists of:

- a set of blocks (A, B, C) and a table
- the blocks can be stacked one on one to form towers
- the stacking is performed by a robot arm
- one block at a time can be held and moved

The goal: move block from initial position to the final position, cf. Figure 1.

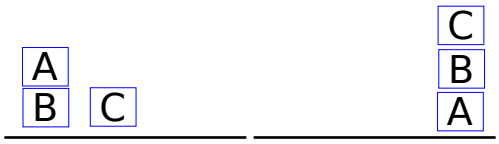

Figure 1: Left: initial position, Right: goal position

We can use four actions (operators):

- unstack(A,B) – pick up clear block A from block B
- stack(A,B) – place block A using the robot arm onto clear block B
- pickup(A) – lift clear block A with the empty arm
- putdown(A) – place the held block A onto a free space on the table

And the predicates:

- pos(A,B) – block A is on block B
- clear(A) – block A has nothing on it
- holding(A) – the arm holds block A

Task 3: Implement operators and methods in: `blocks.py`

1.3 OPTION 2: Soldiers

Task 4: The goal is to propose and implement a control for a group of soldiers. The soldiers must accomplish several objectives by a cooperative behavior under some constraints.

An environment is modeled by a graph of non-oriented edges. If there is an edge between two nodes, a soldier can move between these two vertices. There is also a group of three soldiers in some vertices. In four vertices, one can also find monsters. **The mission objective is to destroy all the monsters.**

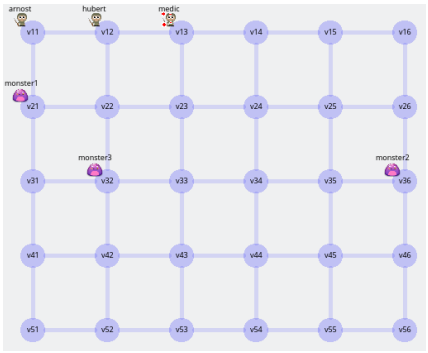

Figure 2: Example map with soldiers and monsters

We begin the implementation with **only one soldier and one monster**. Then we will add some additional constraints.

Task 5: **Move soldiers:** Plan a trip of a soldier from one node to another. e.g. from V11 to V16.

Hints:

- Map is represent using python `dict()`, hence the edge between two vertices can be
- Implement operator `goto(soldier,to)` that moves soldier from vertex to another if there is a link between them
- Implement a method `moveto(soldier,to)` that recursively calls operator goto.

Task 6: **Kill monsters:** Add method kill-monster(soldier). The soldier goes to a monster and kill it (deletes it from the world. If there is no monster, it does not do anything.

Hints:

- Define and use operator `killmonster(soldier,monster)`.
- A natural condition is that the soldier and the monster must be at the same location (node), so the soldier must move to a location occupied by monster

We can see that using simple decomposition we can easily create a plan to move soldiers and kill monsters. There are possible extensions so that task gets more interesting.

Task 7: We add the following constraint: A monster can be killed by at least two soldiers - one is holding a supporting fire in a distance of exactly one edge from the monster and one is directly attacking the monster in the same node. (Soldier must attack the monster whenever it gets to the same node as the monster).

Task 8: You are encouraged to use one of these (voluntary) constraints:

- During the attack, the attacking soldier is always injured by the monster. An injured soldier cannot fight with the monster, but can hold the supporting fire.
- One of the soldiers is medic, which has a capability of healing an injured soldier. The plan must contain an operator HEALING(medic, soldier), which can be applied only if medic and soldier are in the same node. For simplicity, the healing takes infinitely short period.
- The medic cannot attack, but can hold the supporting fire.
- For safety reasons, maximum number of soldiers in one node is two.
- Each soldier must see at least one another soldier in the maximum distance of one edge.
- The soldiers know the number and positions of all the monsters.

2 Have fun!

Complete the exercise as a homework, ask questions on the forum, and upload the solution via Upload system!