AUTOMATIC PLANNING - REPORT - LAB 4

Manu S. Joseph(mansj125)

Lab 4.1. Theory: Task expansion

Question 1)

Expansion of the first task (<*make-delivery(one-ring)*>)

Current State:
undelivered(one-ring),
destination(one-ring, mount-doom),
delivery-speed(one-ring, slow),
at(hobbit, bag-end),
holding(hobbit, one-ring)

Based on the current state the method *make-delivery(pkg, from, to, carrier)* will be called and therefore the subtasks <*walk(carrier, from, to), deliver(pkg, carrier, to)*> will be called in the order specified.

Doing task *walk(carrier, from, to*) with the variable values as *walk(hobbit, bag-end, mount-doom)*. This task is expanded based on the specified values for the variable as this the only one that satisfies the preconditions.

Doing task walk(hobbit, bag-end, mount-doom),

Current State:
undelivered(one-ring),
destination(one-ring, mount-doom),
delivery-speed(one-ring, slow),
at(hobbit, mount-doom),
holding(hobbit, one-ring)

After the task is done the next task *deliver(pkg, carrier, to)* is tried out. Based on the precondition only one of the expansion that can successfully occur is *deliver(one-ring, hobbit, mount-doom)*

Doing task deliver(one-ring, hobbit, mount-doom),

Current State:
destination(one-ring, mount-doom),
delivery-speed(one-ring, slow),
at(hobbit, mount-doom),

This is the finial state of the problem is give above.

The plan generated is,
1. <make-delivery(one-ring)>
2. walk(hobbit, bag-end, *mount-doom*)
3. deliver(*one-ring*, hobbit, *mount-doom*)

Question 2)

Expansion of the first method (<*make-delivery(death-star-plans)*>)

Current State:
undelivered(death-star-plans),
destination(death-star-plans, hidden-rebel-base),
delivery-speed(death-star-plans, fast),
at(farm-boy, desert-far-away),
holding(farm-boy,death-star-plans),
at(old-ship, desert-far-away)

The tasks <*enter-vehicle(carrier, vehicle, from), travel(vehicle, from, to),leave-and-deliver(carrier, pkg, vehicle, to)*> will be executed in the given order.

The task *enter-vehicle(carrier, vehicle, from)* is done first. Based on the preconditions only expansion that will be successful is *enter-vehicle(farm-boy, old-ship, desert-far-away)*.

Doing the task enter-vehicle(farm-boy, old-ship, desert-far-away),

Current State:
undelivered(death-star-plans),
destination(death-star-plans, hidden-rebel-base),
delivery-speed(death-star-plans, fast),
in(farm-boy, old-ship,),
holding(farm-boy,death-star-plans),
at(old-ship, desert-far-away)

After this task the next task performed as per the order specified is, *travel (vehicle, from, to)*. Based on the preconditions the values for the variable that will be successfully explored is, *travel(old-ship, desert-far-away, hidden-rebel-base)*

Doing task travel(old-ship, desert-far-away, hidden-rebel-base),

Current State:
undelivered(death-star-plans),
destination(death-star-plans, hidden-rebel-base),
delivery-speed(death-star-plans, fast),
in(farm-boy, old-ship,),
holding(farm-boy,death-star-plans),
at(old-ship, hidden-rebel-base)

After this the next task performed is the method as per the order specified is, *leave-and-deliver(carrier, pkg, vehicle, to)*. Based on the preconditions the values for the variable that will be successfully explored is, *leave-and-deliver(farm-boy, death-star-plans, old-ship, hidden-rebel-base)*.

Expansion of method *leave-and-deliver(farm-boy, death-star-plans, old-ship, hidden-rebel-base)*

The tasks in the method *<leave-vehicle(carrier, vehicle, to), deliver(pkg, carrier, to)>* will be executed in the specified order.

Initially the task *leave-vehicle(carrier, vehicle, to)* is executed. Based on the preconditions the values for variables that will be successfully executed is *leave-vehicle(farm-boy, old-ship, hidden-rebel-base)*

Doing task *leave-vehicle(farm-boy, old-ship, hidden-rebel-base)*

Current State:
undelivered(death-star-plans),
destination(death-star-plans, hidden-rebel-base),
delivery-speed(death-star-plans, fast),
at(farm-boy, hidden-rebel-base),
holding(farm-boy,death-star-plans),

Then the task *deliver(pkg, carrier, to)* will be executed. Based the preconditions the values for the variables are *deliver(death-star-plans, farm-boy, hidden-rebel-base)*

Doing task *deliver(death-star-plans, farm-boy, hidden-rebel-base)*

Current State
destination(death-star-plans, hidden-rebel-base),
delivery-speed(death-star-plans, fast),
at(farm-boy, hidden-rebel-base),

The final state of the problem is give above.

The plan generated is,

1. <make-delivery(death-star-plans)>
2. enter-vehicle(farm-boy, old-ship, desert-far-away)
3. travel(old-ship, desert-far-away, hidden-rebel-base)
4. leave-and-deliver(farm-boy, death-star-plans, old-ship, hidden-rebel-base)
5. leave-vehicle(farm-boy, old-ship, hidden-rebel-base)
6. deliver(death-star-plans, farm-boy, hidden-rebel-base)

Lab 4.2. An HTN Version of the Emergency Services Logistics Domain

The domain file was created with the following primitive tasks based on the actions that were used in the previous labs,

- *pickup-crate* : Pickup a crate from a location using a UAV
- *fly-to* : Fly UAV from one location to another
- *deliver-crate* : Deliver a crate in the UAV to a person

The following predicates are used

- *at* : shows that a object is at a specific location
- *crate-content* : depicts which content a crate is associated to
- *heli-free* : depicts if a UAV is free and not carrying any crates
- *in* : denotes the crate that has been pickup by a UAV
- *needs* : denote what type of crate a person needs

The were also the following type predicates defined

- *uav* : denotes an UAV
- *location* : denotes a location
- *crate* : denotes a crate
- *content* : denotes the content of a crate

The following methods were also defined

- *pickup* : pickup a crate if the UAV and the crate is in the same location else flies to the crate location and picks up the crate
- *deliver* : deliver a crate if the UAV and the person is in the same location else flies to the person's location and delivers a crate.
- *deliver-all* : calls pickup and deliver methods to pickup and deliver crates to people who need the crate

Comments have been added in the domain file for more information. The domain file is present in the folder *lab4.2*.

A problem file was defined to test the defined domain. The problem involved 1 UAV, 3 locations, 2 contents, 2 crates and 2 people. The need predicate was used to defined pseudo goals that involved delivering 1 crate to each person. It was seen that JSHOP2 was able to find plans for this problem.

The problem file along with the plan generated is present in the folder *lab4.2*

Lab 4.3. Carriers and Numeric Representation

The domain file was created with the following primitive tasks based on the actions that were used in the previous labs,

- *pickup-crate* : Pickup a crate of a particular type from a location using a UAV
- *fly-to* : Fly UAV from one location to another
- *deliver-crate* : Deliver a crate of a particular type in the UAV to a location
- *load-crate-on-carrier* : Place the crate of a particular type that is in an UAV into a carrier
- *fly-carrier* : Fly a carrier from one location to another
- *take-crate-from-carrier* : Take a crate of particular type in the carrier into an UAV

The following predicates are used

- *at* : shows that a object is at a specific location
- *crates* : depicts number of crates of a particular type at a location
- *heli-free* : depicts if a UAV is free and not carrying any crates

- *in* : denotes the crate of a particular type has been pickup by a UAV
- *need* : denote what type and quantity of crate needed in a location
- *space* : denotes the remaining space in a carrier
- *contains* : denotes the amount of crate of a particular type present in a carrier

The were also the following type predicates defined

- *uav* : denotes an UAV
- *location* : denotes a location
- *carrier* : denotes a carrier
- *content* : denotes the content of a crate

The following methods were also defined

- *pickup-single* : pickup a single crate of specific type if the UAV and the crate is in the same location else flies to the crate location and picks up a single crate of the required type
- *deliver-single* : deliver a crate if the UAV has the required crate type and is in the correct location. Otherwise if the UAV has the crate of required type files the UAV to the required location and delivers it.
- *pickup-multiple* : If the UAV, carrier and the crates are all in the same location pickup's up the required number of crates and places it in the carrier. If the carrier or the UAV is in a different location than that of the crates then flies the carrier and UAV to location of the crates and places the required number of crates in the carrier.
- *deliver-multiple* : If a carrier has a set of crates that are required at a location and if the carrier and UAV is in the same location as the one that needs the crates, then takes the crates from the carrier and delivers it. Otherwise if the carrier has the required crates but is in UAV or carrier is in different location than the one that needs the crates then fly the UAV and carrier to the place that needs the crates and delivers the crates.
- *pickup* : if the required number of crates is 1 then calls the *pickup-single* method otherwise if the required number of crates is greater than 1 calls *pickup-multiple.*
- *deliver* : if the required number of crates is 1 then calls the *deliver-single* method otherwise if the required number of crates is greater than 1 calls *deliver-multiple.*
- *deliver-all* : calls pickup and deliver methods to pickup and deliver crates to people who need the crate.

Comments have been added in the domain file for more information. The domain file is present in the folder *lab4.3*.

Two problem files were defined to test the implementation.

- *Small problem* - With 1 UAV, 1 carrier , 3 locations, 2 contents with 6 crates each.
- *Large Problem* - With 2 UAV's, 2 Carriers, 9 locations, 2 contents with 40 crates each.

It was seen that JSHOP2 was able to find plans for this problem.

The problem file along with the plan generated is present in the folder *lab4.3*

Lab 4 Report

Question 1) *Compare building the HTN domain to what you did previously in PDDL. Which parts were easier, if any? Which parts were harder? Why?*

Answer:
When compared with PDDL I felt that writing domains for HTN were much easier because I felt it was much more natural to think of the problem as tasks and subtask to be performed instead of figuring all the procedures that might be needed for a domain. Also writing the PDDL version in the earlier labs helped a lot in identifying the tasks that needed to be created and the operations that needed to be performed. I felt the hard part of using SHOP was when there was a need to debug any bugs that are encountered while running. Since predicate, constants or operators need not be explicitly defined I found that small mistakes like spellings mistakes caused large debug time.

Question 2) *How does the run time of JSHOP2 scale with larger problems? Clarification 2021-05-06: Please test this with larger problems that require at least a minute to run using a modified problem generator. Include actual timing and plan length results for different problem sizes in your report!*

Answer:
The problem that was run in 1 minute was found to contain - 20 UAV's, 10 carriers, 300 locations, 1000 crates and 800 goals. The problem took a total of 59.31 seconds to generate and had a total of 4034 actions. The problem file is named - *uav_problem_u20_r10_l300_p500_c1000_g800_ct2*

Different variation of the problem was generated with increased amounts of corresponding components. More details are given below,

1. *Problem with Increased UAV's* - The problem contained 30 UAV's, 10 carriers, 300 locations, 1000 crates and 800 goals. It was run in 133.188 seconsds.
2. *Problem with Increased Crates* - The problem contained 20 UAV's, 10 carriers, 300 locations, 1200 crates and 800 goals. It was run in 52.446 seconds.
3. *Problem with Increased Goals* - The problem contained 20 UAV's, 10 carriers, 300 locations, 1000 crates and 1000 goals. It was run in 67.286 seconds.
4. *Problem with Increased Locations* - The problem contained 20 UAV's, 10 carriers, 400 locations, 1000 crates and 800 goals. It was run in 72.106 seconds.

It can be seen from these results that in all cases other than raising the number of UAV's there is not significant increase in the run time of JSHOP2.

The problem along with their outputs are given in the folder - *lab4.3*

Question 3) *Is JHOP2 faster/slower than the competition planners from previous labs, given your own particular PDDL and HTN formalizations of the domain? How much?*

Answer:

It can be seen from the results above and from the previous lab results that JSHOP is much faster than the competition planners from previous labs.

The largest problem that the fastest tested temporal planner could solve within 1 minute contained 25 UAV's, 5 Carrier, 20 locations, 50 crates, 30 people and 50 goals. The largest problem that sequential satisfying planners could solve within 1 minute contained 2 UAV's, 99 locations, 70 people, 80 crates and 70 goals.

It can be seen from the above results that the problem that JHOP2 can solve in 1 minute is much larger. Hence from these results it can be assumed JHSOP2 is around 5-10 times faster than he competition planners from previous labs.