David Tran **davtr766**
Adam Rohdin **adaro815**

Task 1:

Explain the way you represent the domain and motivate your choice of predicates, objects and operators.
We choose to model the 'Shakey problem'.
In the shakey domain, there are rooms, light switches, boxes and objects. There are a few restrictions to shakey:

- Shakey can carry small objects, but only two at a time because he has only two grippers.
- For Shakey to be able to find a small object in a room, the light must be on in that room.
- Shakey can not carry boxes, but can push them around. Boxes can only be pushed through wide doors, not narrow ones.
- To switch the light in a room on or off Shakey must climb onto a box that is positioned below the switch (otherwise he can not reach the switch). This may in turn require moving the box into position, possibly from another room.

We define the predicates and operators for this domain based on shakeys abilities and the domain restrictions.
The types are as following:

● box	;There is one large box
● door	;There are 3 doors
● shakey	;There is only one shakey
● object	;There are several blocks
● room	;Room

We have the following predicates:

● (onTopBox ?s - shakey)	;Shakey can be on top of boxes
● (holdRight ?o - object)	;Shakey can carry objects in right hand
● (holdLeft ?o - object)	;Shakey can carry object in left hand
● (lightPos ?r - room)	;Lightswitch can be in different rooms
● (isLit ?r - room)	;Is the room lit?
● (boxPos ?b - box ?r - room)	;Box position
● (objectPos ?o - object ?r room)	;Object position
● (shakeyPos ?s - shakey ?r room)	;Shakey position.
● (lightPos ?l - lightswitch ?r - room)	;Lightswitch position
● (wideDoor ?r1 ?r2 - room)	;Doors can be narrow or wide
● (adjacent ?r1 ?r2 - room)	;Is shakey adjacent with p.

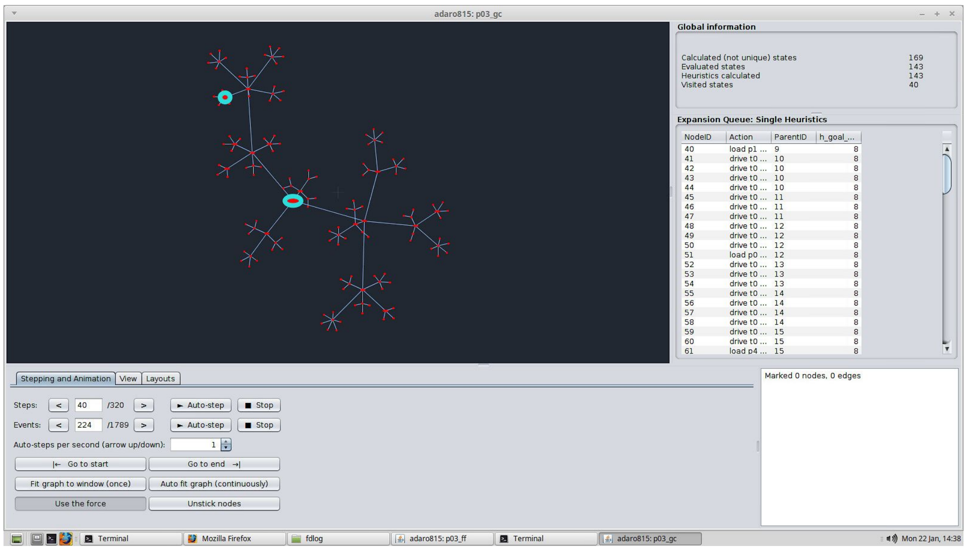
And lastly our actions:

● action move	;Movement when not holding/pushing for Shakey.
● action push	;Shakey pushing a box from ?r1 to ?r2 position.
● action pickUpLeft	;Shakey pick up small object with left grabber.
● action pickUpRight	;Shakey pick up small object with right grabber.

● action dropLeft	;Shakey drop object with left grabber.
● action dropRight	;Shakey drop object with right grabber.
● action climbOnBox	;Shakey climb on top of box.
● action turnLightOn	;Shakey turns on the light in the room.
● action turnLightOff	;Shakey turns off the light in the room.

Task 2 P03:

How do the graphs differ, given the same problem and search method but different heuristics? Include screenshots in your report and discuss what you see!
The graph for the goal-count heuristic picks the node from the expanded queue but that has not been visited before.



Global information

```
Calculated (not unique) states      169
Evaluated states                    143
Heuristics calculated               143
Visited states                      40
```

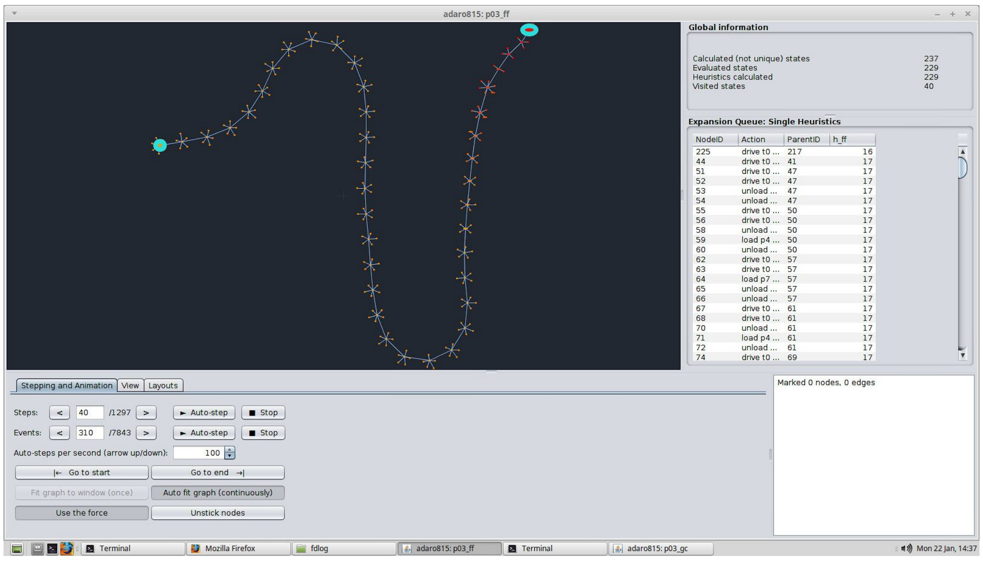
Expansion Queue: Single Heuristics

NodeID	Action	ParentID	h_goal_...
40	load p1 ...	9	8
41	drive t0 ...	10	8
42	drive t0 ...	10	8
43	drive t0 ...	10	8
44	drive t0 ...	10	8
45	drive t0 ...	11	8
46	drive t0 ...	11	8
47	drive t0 ...	11	8
48	drive t0 ...	12	8
49	drive t0 ...	12	8
50	drive t0 ...	12	8
51	load p0 ...	12	8
52	drive t0 ...	13	8
53	drive t0 ...	13	8
54	drive t0 ...	13	8
55	drive t0 ...	14	8
56	drive t0 ...	14	8
57	drive t0 ...	14	8
58	drive t0 ...	14	8
59	drive t0 ...	15	8
60	drive t0 ...	15	8
61	load p4 ...	15	8

Marked 0 nodes, 0 edges

Steps: 40 /320
Events: 224 /1789
Auto-steps per second (arrow up/down): 1

The graph for the fast forward heuristic expands the node with the lowest heuristic value in the queue that has not been visited.



Global information

```
Calculated (not unique) states      237
Evaluated states                    229
Heuristics calculated               229
Visited states                      40
```

Expansion Queue: Single Heuristics

NodeID	Action	ParentID	h_ff
225	drive t0 ...	217	16
44	drive t0 ...	41	17
51	drive t0 ...	47	17
52	drive t0 ...	47	17
53	unload ...	47	17
54	unload ...	47	17
55	drive t0 ...	50	17
56	drive t0 ...	50	17
58	unload ...	50	17
59	load p4 ...	50	17
60	unload ...	50	17
62	drive t0 ...	57	17
63	drive t0 ...	57	17
64	load p7 ...	57	17
65	unload ...	57	17
66	unload ...	57	17
67	drive t0 ...	61	17
68	drive t0 ...	61	17
70	unload ...	61	17
71	load p4 ...	61	17
72	unload ...	61	17
74	drive t0 ...	69	17

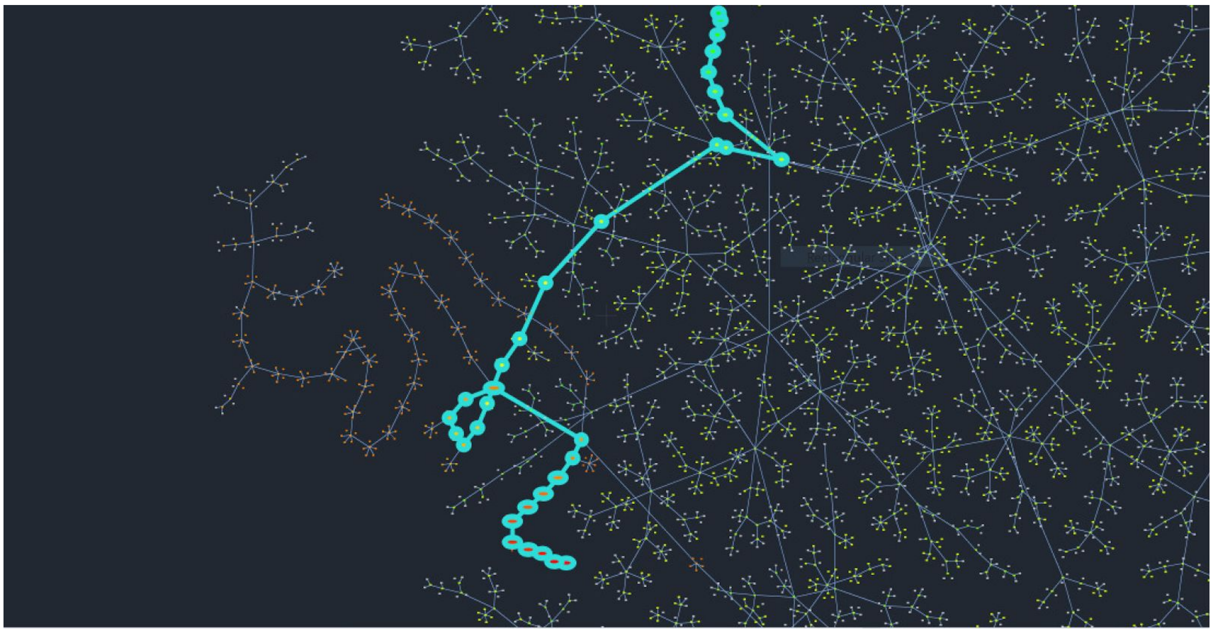
Marked 0 nodes, 0 edges

Steps: 40 /1297
Events: 310 /7843
Auto-steps per second (arrow up/down): 100

David Tran **davtr766**
Adam Rohdin **adaro815**

Do the different configurations use different actions? Zoom in on the edges to see which actions are used and discuss the differences.
The difference between between fast-forward and goal-count actions is that goal-count queues all nodes in order of breadth first search to a certain depth in the tree meanwhile fast-forward moves much deeper in the tree before checking other optional paths which is similar to depth-first search.

Step the FF visualisation to the end through auto-stepping (hint: set the Auto-steps per second to a value greater than 1 but small enough so that you can follow the search process). Don't use the go to functionality.
At the beginning, the FF visualisation expanded nodes along a long path before it started expanding at other places. Are the nodes in this path used in the final plan? Hint: the nodes are numbered incrementally as they are expanded so this can be used to find the answer.
The nodes in this path are partially used in the final plan in the beginning and then deviates from the long path.



Task P02:

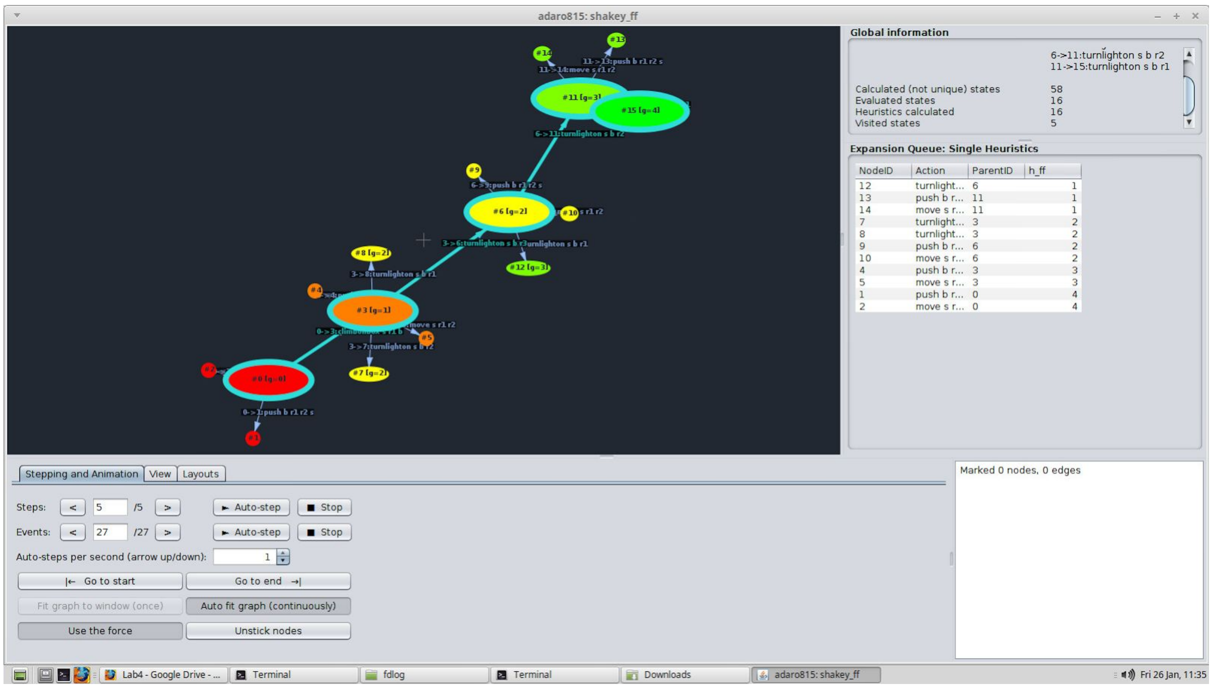
When the planner adds a new action to a branch in the search tree, this doesn't necessarily cause the heuristic function to decrease. In which time step *does* the planner first find a new lower value for the main heuristic function in each example (FF and GC, respectively)?
We found out that for problem P02 with the fast-forward heuristic, the value decreases at step 2 whereas the goal-count with fast-forward operators first decreases at step 20.

Step both visualisation to time step 27 (the last time step with the configuration using FF as heuristic). How many goal predicates are left to solve according to the goal

count heuristic (see the description of *Eager greedy search, Goal count heuristic*) in the state that has progressed the longest?
The amount of goal predicates left to solve after step 27 is 16.

Follow the marked path from the start node (the initial state) to the last node (the first found goal state) in the highlighted path in the graph (the plan). Does the solution ever increase the value of the goal count heuristic between one state and the next?
Yes it does, for example between the edge:
4-18-64 the heuristic value went from 17 to 18 and back to 17 and from edge
147-197-353 the heuristic value went from 13 to 15 and back to 14.

Run one of the configurations on your own domain and problem. Is the graph similar to any of those that you have examined above? Can you explain why or why not?



Global information

```
                                   6->11:turnlighton s b r2
                                   11->15:turnlighton s b r1

Calculated (not unique) states     58
Evaluated states                   16
Heuristics calculated              16
Visited states                     5
```

Expansion Queue: Single Heuristics

NodeID	Action	ParentID	h_ff
12	turnlight...	6	1
13	push b r...	11	1
14	move s r...	11	1
7	turnlight...	3	2
8	turnlight...	3	2
9	push b r...	6	2
10	move s r...	6	2
4	push b r...	3	3
5	move s r...	3	3
1	push b r...	0	4
2	move s r...	0	4

```
Steps:   <  5   /5   >      Auto-step   Stop
Events:  <  27  /27  >      Auto-step   Stop
Auto-steps per second (arrow up/down):  1
```

Marked 0 nodes, 0 edges

For our shakey domain, the specified problem is that shakey has to turn on the lights in every room. The graph is similar to the fast-forward search graph because there is a straightforward solution to the specified problem.

Questions:

The statistics from using the different configurations to solve the given problems. Was any configuration better than the other? Was it better on everything or just on some problems?
According to the statistics from using different configurations to solve the problems some configurations performed slightly better for the specific problems. For instance in task 2 when using P02, the time step for finding a heuristic function with decreasing value is found at step 2 using the fast-forward heuristic meanwhile for the goal-count heuristic the decreasing value is found at step 26. However, this may not imply that fast-forward was necessarily better than goal-count, it just happens that the fast-forward heuristic was more

suited in the P02 problem.

Are your findings applicable to all the infinite many possible domains and problems? If so why? Else, why not?
No, it is not possible to solve every problem efficiently with the given heuristic. Because every domain and problem is different and some heuristic works better for some problems compared to other. Therefore, not all domains and problems have a straightforward solution, it may need to require backtracking to previous goals to be able to clear subsequent goals.