CMSC722 Final Report

Abstract:

AI Planning is an important field of AI, which is about decision making about the actions to be taken. The AI planners can be divided into two categories: domain specific planners and domain independent planners. In this project, I am working on comparing a domain independent planner(Metric-FF) with HTN or HGN planners on two planning domains: the block world domain and satellite domain. By using several evaluation metrics like the CPU time or the plan length in the experiment, I can learn which planners perform better.

Student Honor Code: Fuxiao Liu

Introduction:

Artificial intelligence planning is a long-standing sub field of artificial intelligence, which deals with sequential decision-making. Industrial applications with mature planning technology can be seen in various fields such as dialogue systems, network security, transportation and logistics, and so on.

The AI planners can be divided into two categories: domain specific planners and domain independent planners. In this project, I am working on comparing a domain independent planner with HTN or HGN planners on two planning domains: the block world domain and satellite domain.

For the first step, I use the source code in C to generate the problems for both domains. Then, as for the domain independent planner, I chose Metric-FF since it's compatible with the new-version PDDL very well and achieved excellent results in the 3rd international planning competition. After this, I wrote the script in python by myself to translate the problem generator's output into PDDL format. Meanwhile, I also wrote the translator script by myself which can change the output's format to be accepted by Phop(HTN). In the second step, which is the experiment part, I generated 10 problems for each problem size. The number of blocks and the number of targets are the problem size for each domain accordingly. For the satellite domain problem, the number of satellites is 5, the max instruments per satellite is 2, the number of modes is 2, the number of observations is 3. In the final step, which is about the evaluation section, the metrics include the plan length, CPU time and the number of nodes expanded. I also calculate the growth rate for each evaluation feature and have plots. Finally, by comparing the results and plots, I can learn which AI planner is better. The following sections have descriptions of the planners, experiments, conclusions, instructions to run the code and reference.

Description of the Planners:

- ***The Domain Independent Planner I chose:***

    - The domain-independent planner I use is Metric-FF, including three versions: Metric-FF v1, Metric-FF v2 and Metric-FF v3. The reason why I use this planner is that its code is public and easy to use and it's also a fast forward planner. What's more, it's compatible with the new-version PDDL very well and achieved excellent results in the 3rd international planning competition.

- ***The HTN or HGN Planner I Use:***

    - I used HTN(Pyhop) instead of HGN. The reason I chose Pyhop is because Pyhop is a simple hierarchical task network planner written in Python and it's easier to run than GTPyhop. What's more, it represents states of the blocks and satellites using ordinary variable bindings, not logical propositions, which is very efficient. In addition, the advantages are we don't need to learn a planning language, and it can do complex reasoning to evaluate the preconditions and generate subtasks. The actions of Block World Dataset includes pickup, unstack, putdown, stack and the methods contain is_done(), status(), all_clear_blocks(), m_take(), m_put(), move_blocks() and so on. For example, m_take() first checks when the hand is clear then takes the block from either the floor or other blocks into hand. This method may include several actions. The states include the initial state and the goal state. The task is to transfer the initial state to the goal state, like moving the blocks. Of course, there are some middle states. For the satellite dataset, the actions contain find instruments, take images, turn_to, switch_on, switch_off and calibrate. The methods include capture_image(), activate(), calibrate_instrument(). The method capture_image() needs to check whether the power is available and whether it has been switched on and then capture images. The details of the methods and actions are in the code.

Experiments:

- ***Block World Dataset:***
  - ***Pyhop(Experiment Design):***
    - Parameter:
      - Size: number of blocks
      - 10 problems each size
      - Evaluation Metric: CPU Time, and Plan Length (Due to limitation of the space, for each problem size, I only show the average CPU Time and Plan Length)
  - ***Experiment Results (Chart)***

Size	*4*	*6*	*8*	*10*	*12*	*15*	*18*	*20*	*22*	*25*
CPU(s)	0.0007	0.001	0.002	0.002	0.003	0.004	0.0055	0.0072	0.011	0.012
PL	6	10	16	22	29	37	44	53	63	69

*PL means average plan length and CPU means average CPU time, and Expand means the average number of nodes expanded.

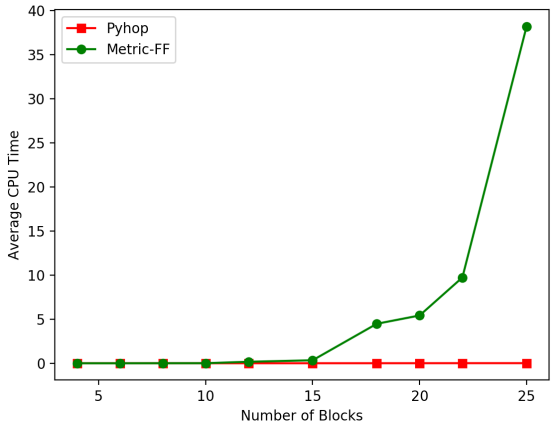
  - ***Metric-FF(Experiment Design):***
    - Parameter:
      - Size: number of blocks
      - 10 problems each size
      - Evaluation Metric: CPU Time, and Plan Length (Due to limitation of the space, for each problem size, I only show the average CPU Time and Plan Length)
  - ***Experiment Results (Chart)***

Size	*4*	*6*	*8*	*10*	*12*	*15*	*18*	*20*	*22*	*25*
CPU(s)	0.00	0.00	0.005	0.008	0.17	0.34	4.48	5.42	9.73	38.2
PL	6	13	25	28	45	48	53	69	84	93
Expand	7	14	28	105	237	1085	1660	23267	67421	162685

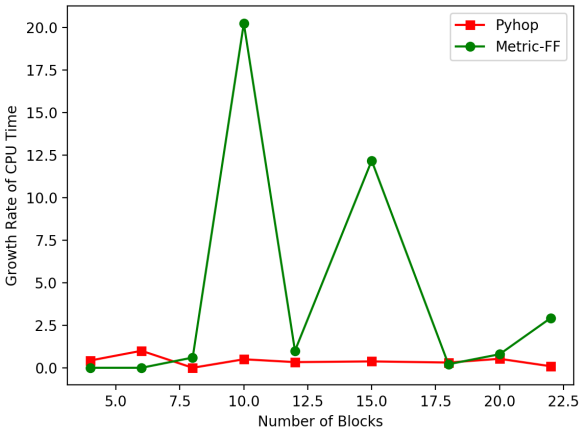
* PL means average plan length and CPU means average CPU time, and Expand means the average number of nodes expanded.

  - ***Experiment Results (Plots):***

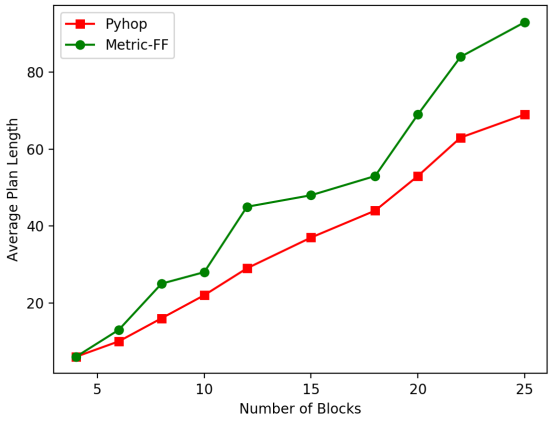
- ***Average CPU Time/Number of Blocks***



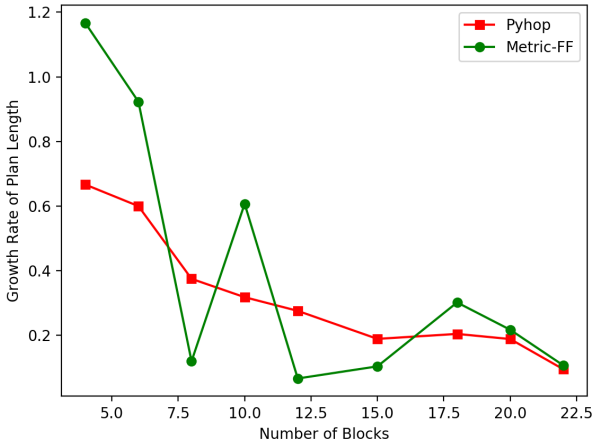
- ***Growth Rate for the CPU Time/Number of Blocks:***



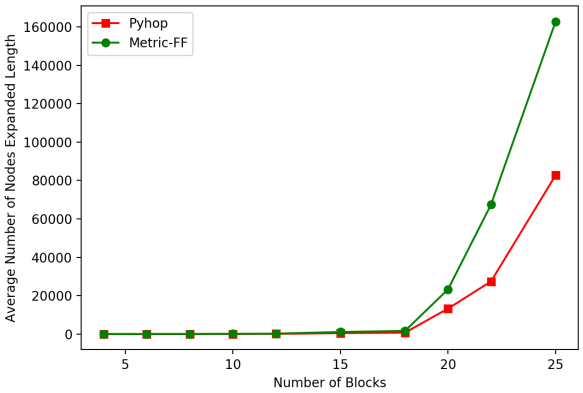
- ***Average Plan Length/Number of Blocks***



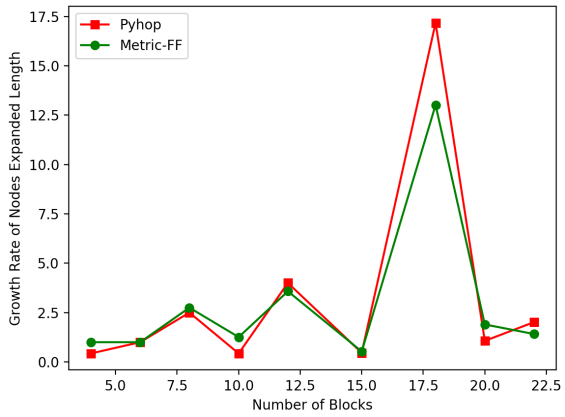
- ***Growth Rate for the Plan Length/Number of Blocks:***



- ***Average Number of Expanded Nodes/Number of Blocks:***



- ***Growth Rate for the Expanded Nodes/Number of Blocks:***



- ***Experiment Discussion:***
  - The plots above show the performance comparison between Pyhop(HTN) and Metric-FF. The Pyhop is in the red line and the Metric-FF is in the green line. For each plot, the x-axis represents the number of blocks, the y-axis represents average CPU

time/plan length/number of nodes expanded accordingly. I also show the growth rate for each of them.
    - For the problems which can't be solved within the 60s, I skipped them to generate new problems.
    - After comparing them in the plots, I find the Metric-FF performs better in terms of CPU time and the Pyhop has shorter plan length than the independent planner. Then as for the number of nodes expanded, Metric performs better. It makes sense that the planning process of HTN starts by decomposing the initial task network and continues until all compound tasks are decomposed, that is, a solution is found. The CPU Time is longer but the shortened plan length can be found. As for the growth rate, I didn't find much difference except that when the problem size is small, the value of the growth rate is larger. This might be because I only test 10 problems for each problem size and more problems are needed.

- ***Satellite Dataset***
    - ***Pyhop(Experiment Design):***
        - Parameter:
            - Size: number of target(1-10)
            - 10 problems each size
            - Other parameters:
                - number of Satellite: 5
                - Max instruments per satellite: 2
                - Number of mode: 2
                - Number of of target: 1-10
                - Number of observation: 3
            - Evaluation Metric: CPU Time, and Plan Length (Due to limitation of the space, for each problem size, I only show the average CPU Time and Plan Length)
        - ***Experiment Results (Chart)***

Size	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*15*
CPU(s)	0.004	0.0066	0.0075	0.0055	0.005	0.006	0.009	0.01	0.01	0.008	0.004
PL	13	14.4	13	15	13	16	14	15	15	13	10

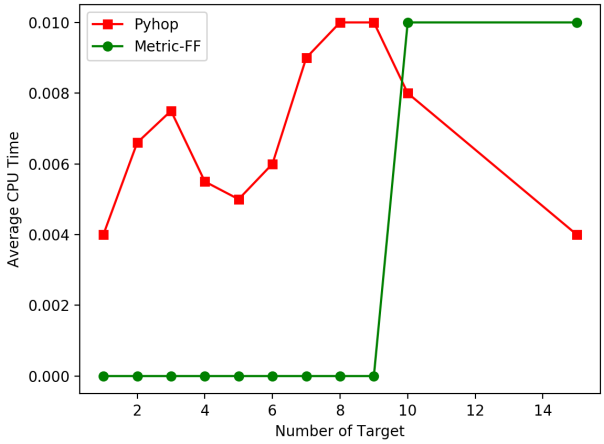
* PL means average plan length and CPU means average CPU time, and Expand means the average number of nodes expanded.

- ***Metric-FF(Experiment Design):***
  - Parameter:
    - Size: number of target(1-10)
    - 10 problems each size
    - Other parameters:
      - number of Satellite: 5
      - Max instruments per satellite: 2
      - Number of mode: 2
      - Number of of target: 1-10
      - Number of observation: 3
    - Evaluation Metric: CPU Time, and Plan Length (Due to limitation of the space, for each problem size, I only show the average CPU Time and Plan Length)
- ***Experiment Results (Chart)***

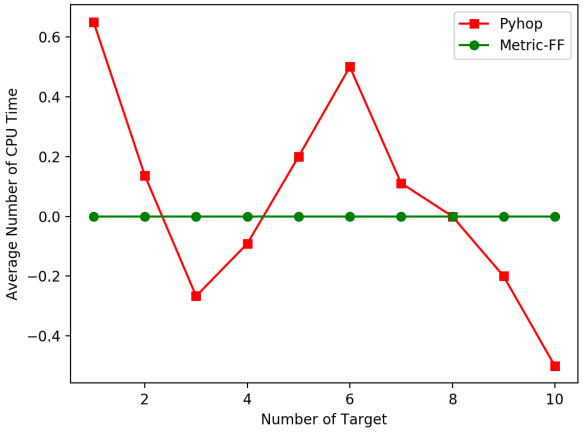
Size	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*15*
CPU(s)	0	0	0	0	0	0	0	0	0	0.01	0.01
PL	9	12	13	13	13	13	15	14	12	16	12
Expand	23	18	30	32	26	31	43	33	25	39	30
Size	*20*	*30*	*50*								
CPU(s)	0.02	0.10	0.29								
PL	11	11	12								

* PL means average plan length and CPU means average CPU time, and Expand means the average number of nodes expanded.

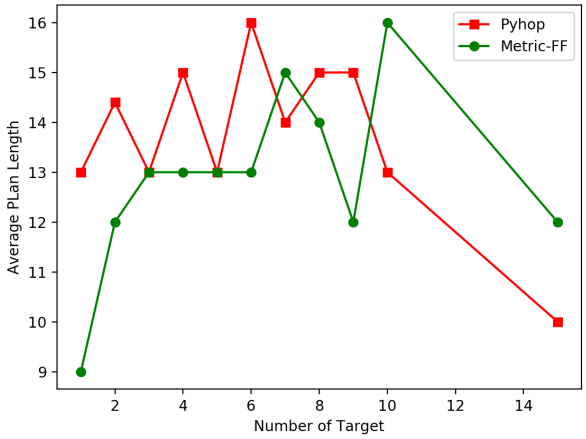
- ***Experiment Results (Plot):***
  - ***Average CPU Time/Number of Target:***



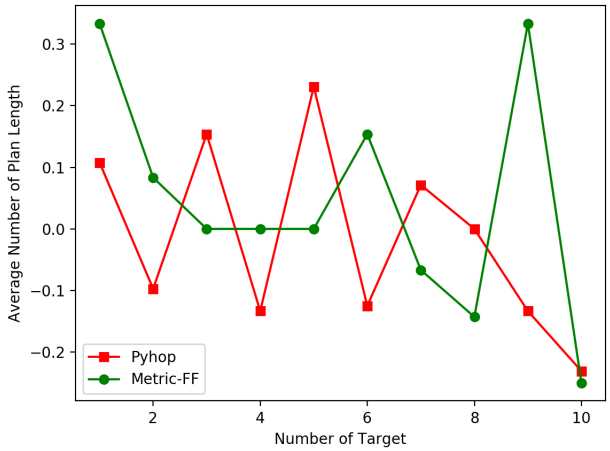
- ***Growth Rate for the CPU Time/Number of Blocks:***



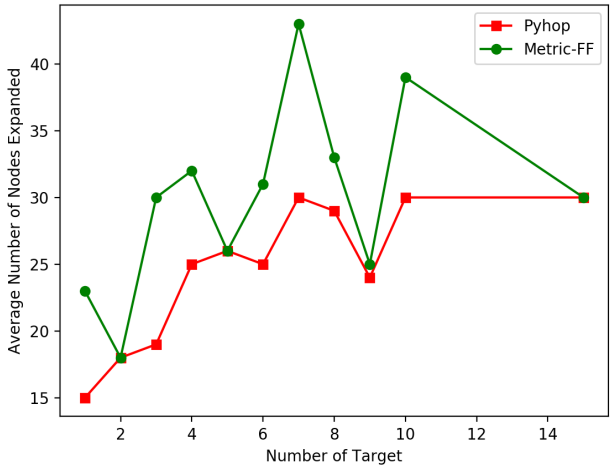
- ***Average Plan Length/Number of Target:***



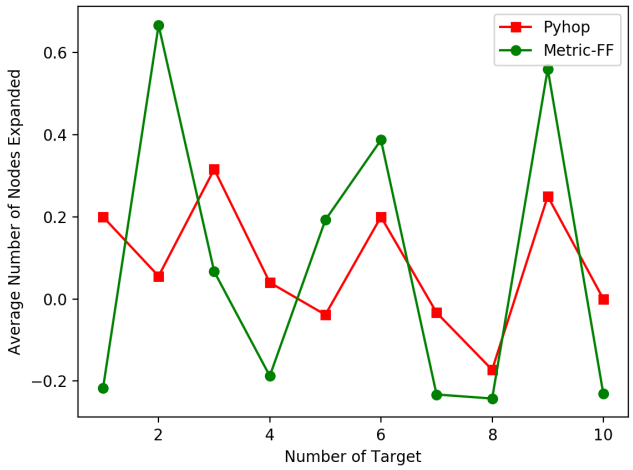
- ***Growth Rate for the Plan Length/Number of Blocks:***



- ***Average Number of Expanded Nodes/Number of Target:***



- ***Growth Rate for the Expanded Nodes/Number of Target:***



- ***Experiment Discussion:***

- The plots above show the performance comparison between Pyhop(HTN) and Metric-FF. The Pyhop is in the red line and the Metric-FF is in the green line. For each plot, the x-axis represents the number of targets, the y-axis represents average CPU time/plan length/number of nodes expanded accordingly. I also show the growth rate for each of them.
- For the problems which can't be solved within the 60s, I skipped them to generate new problems.
- In order to be fair, I fix other parameters including the number of satellites is 5, the max instruments per satellite is 2, the number of modes is 2, the number of observations is 3. It would be better if I can do the comparison experiment for each of them.
- After analyzing the plots, I found Metric-FF performs better in terms of the CPU time when the problem size is smaller than 10. It makes sense since Metric-FF often had a hard time searching for the plans during my experiments. In this aspect, I think Pyhop achieves better results. Also, the number of nodes expanded favors Pyhop rather than domain independent model according to the plot. As for the plan length, Pyhop has a shorter plan length when the problem size is larger than 10. The threshold is not expected so more samples are needed especially for other parameters.

Instructions to Run the Code:

- ***For the supplemental material:***
  - I saved them in the github link. The filename is Code_722_project_fuxiao.zip. You can also view it through the gradescope.
- ***Instructions to run the Pyhop code on the block-world dataset.***
  - Run python test.py
  - The dataset is in the dictionary './data'
  - The translation script is in the file ./test.py
- ***Instructions to run the Pyhop code on the Satellite dataset.***
  - Run tmp.py
  - The dataset is stored in three files: object.txt, init.txt, goal.txt.
  - The translation script is in the file ./tmp.py

- ***Instructions to run the Metric-FF on block-word dataset and Satellite dataset***
    - For example on the satellite dataset:
        - ./ff -o "../code_satellite/domain.pddl" -f "../code_satellite/size30/problem1.pddl"
    - Code for the translation script is in this: link
- ***Plot Code***
    - The code is in this link.

Conclusions:

In this project, I am working on comparing a domain independent planner(Metric-FF) with HTN or HGN planners on two planning domains: the block world domain and satellite domain. By using several evaluation metrics like the CPU time or the plan length in the experiment, I can learn that Metric-FF performs better in terms of CPU time while Pyhop has better results for the number of nodes expanded. Pyhop often has a shorter plan length in block world dataset while it depends on the problem size when it comes to the satellite dataset.

Reference:

- Dana S. Nau. 2021. https://bitbucket.org/dananau/pyhop/src/master/
- Jorg Hoffmann. ECAI 2002. Extending FF to Numerical State Variables
- Dana S. Nau. 2013. https://github.com/ospur/hgn-pyhop