Comparison of Metric-FF and HTN Planners

Aayushi Roy

CMSC722: Artificial Intelligence Planning

April 25, 2023

Abstract

Given the wide variety and number of planning domains, it is necessary to consider how problems in these domains might be solved most efficiently. In this paper, I compare a domain independent planner, specifically Metric-FF, with Hierarchical Task Network planners for two established domains: Blocksworld and Satellite. I conducted 100 experiments per domain to compare planners' performance in terms of CPU time and plan length. I found the HTN planners to, on average, perform better in terms of CPU time for both domains, and the Blocksworld HTN planner to generate plans with lesser length than Metric-FF. The Satellite HTN planner and Metric-FF were comparable in terms of plan length on the Satellite domain. I discuss the strengths and weaknesses of each planner and what future experiments may entail.

Honor Pledge

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment.

x__*Aayushi Roy*___

Introduction

The aim of this project was to compare a domain-independent planner with HTN planners. These planners were used to solve problems in the famous Blocksworld domain and the metric version of the Satellite domain.

This project consisted of four main parts. First, I evaluated existing independent domain planners, specifically Metric-FF and FastDownward, in the context of CPU time and plan length with the purpose of choosing one for the rest of the project. Second, I wrote Hierarchical Task Network methods in GTPyhop to generate domain-specific plans for the Satellite domain, specifically focusing on generating plans that reduced the fuel used. Third, I randomly generated 100 problems per domain (10 problems per size) to perform my experiments on. Finally, I performed experiments comparing the performance of Metric-FF and the HTN planners in terms of CPU time and plan length. In the following sections, I describe in detail each of these parts and my motivation behind key decisions.

Description of Planners

Domain-Independent Planner

I decided to use Metric-FF as my domain independent planner. I wanted to explore both Metric-FF and Fast Downward (22.12 with the lama-first configuration) as possible planners and compare them in terms of performance before ultimately deciding on one.

Metric-FF seemed to have a slightly better CPU time across problems, but I found that Metric-FF had an upper size bound of ~18 with Blocksworld problems, timing out on a problem of size 19. On the other hand, Fast-Downward was able to solve the same problem of size 19 in 0.47 seconds. For the metric Satellite domain, Metric-FF was able to handle problems of up to size ~90. However, Fast Downward does not support fluents and numerical comparisons, and thus requires the domain to be modified to be used. After removing metric components, Metric-FF was able to solve problems of size ~200 in about 4 seconds whereas Fast-Downward took up to 35 seconds.

Despite Metric-FF having a seemingly poorer performance on the Blocksworld domain given the size limit, I felt it was valuable to use the metric version of the Satellite problem that includes the data capacities and fuel usage, and thus decided to use Metric-FF rather than Fast-Downward for this project. The graphs below illustrate my comparison of the two planners.

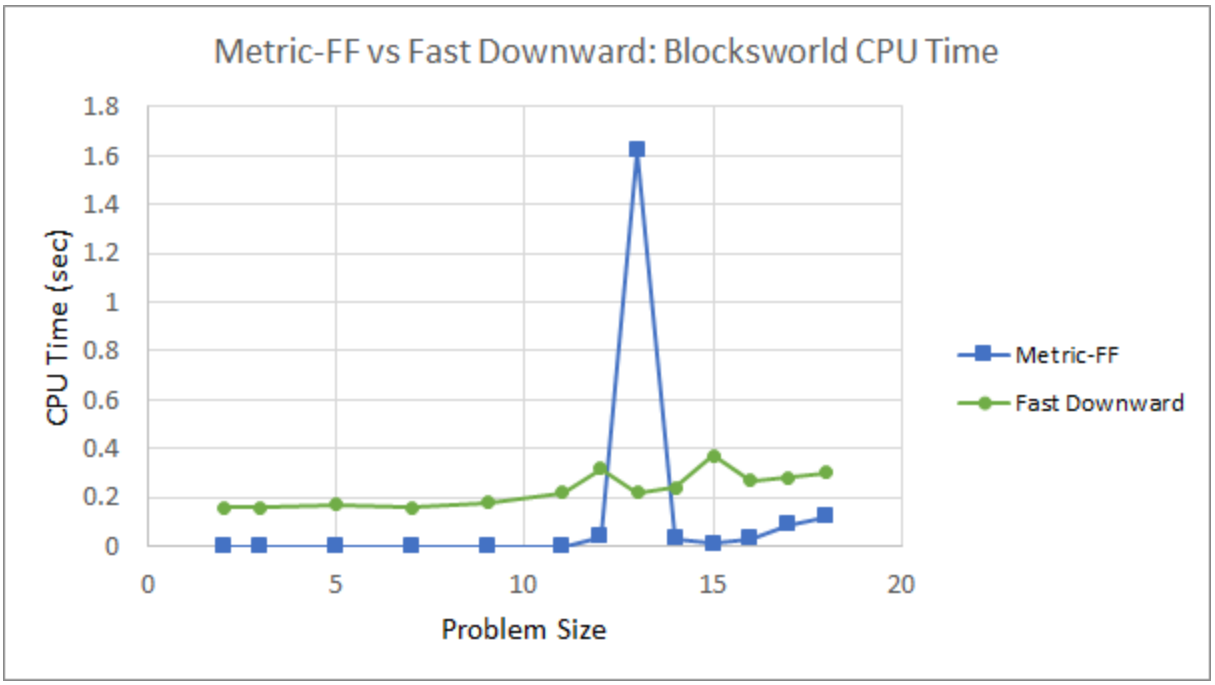


Figure 1. Here, you can see Metric-FF consistently performing better (aside from the one problem of size 13) than Fast Downward by about 0.2 seconds.

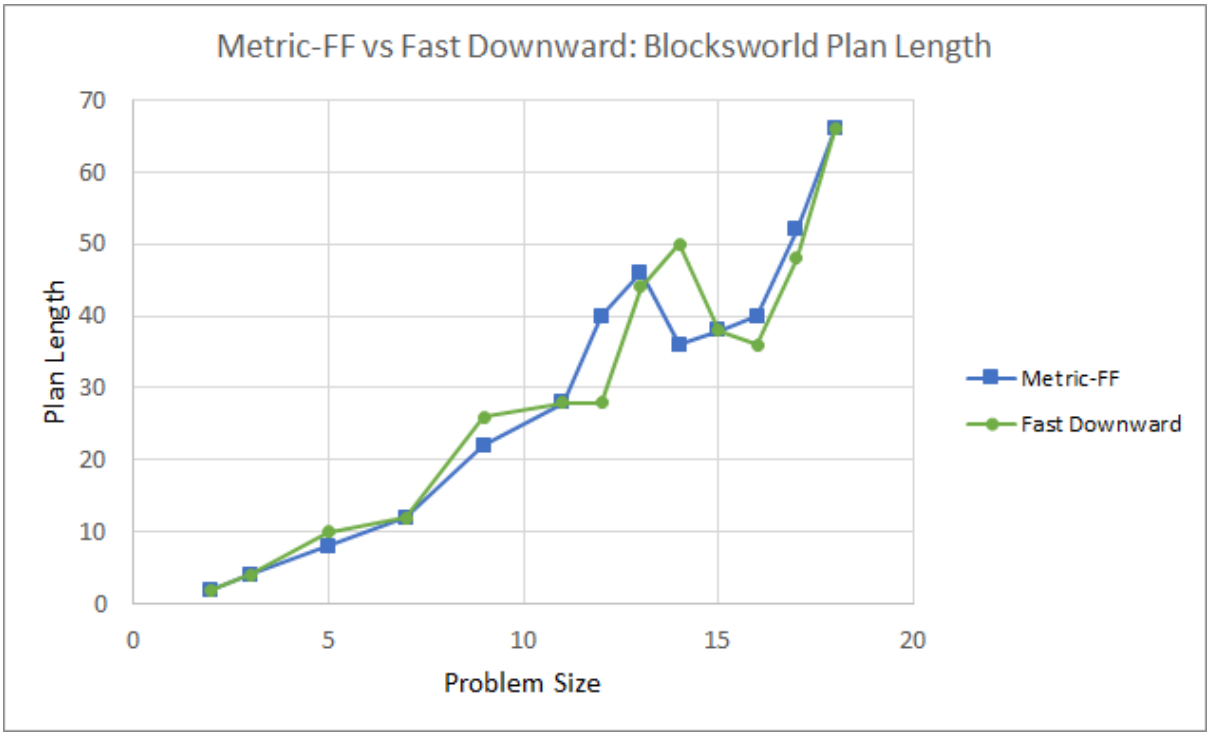


Figure 2. I also wanted to compare plan lengths across planners. They match up fairly well.

Hierarchical Task Network Planners

For the Blocksworld domain, I used the HTN planner written in GTPyhop by Professor Nau. Specifically, I used the goal-task-planning version, or blocks_gtn. I did not make any changes to the methods provided.

For the Satellite domain, I also used a goal-task-planning approach for writing my HTN methods. I have four task methods: m_take_image, power, m_calibrate, and point. I also have one multigoal method: m_img_move, and I wrote two helper functions to aid with the task methods. Below, I describe each method in detail.

Task Methods

m_take_image

The purpose of this task is to initiate taking an image of a destination given a specific mode. In the interest of minimizing the fuel usage metric, I first find the best instrument for this task. Then, I call a series of other task methods (power, m_calibrate, point) that ensure the preconditions for the final action (take_image) are met.

power

This task focuses on providing power to the specified instrument. It first checks to make sure the instrument isn't already on to ensure we avoid unnecessary switching off and recalibration. Then, if there is power available on the instrument's satellite, we simply give it power by calling action switch_on. If the satellite does not have power, this indicates another instrument on the satellite currently has power, and we must find it (find_powered_instrument), switch it off, and give our instrument of interest power (switch_on).

m_calibrate

To avoid confusion with the action calibrate, this task method is preceded by the letter m. This task focuses on calibrating the instrument via the calibrate action. If the preconditions for the calibrate action are not met (the satellite is not pointing at the calibration target), we must first call the action turn_to.

point

This task is focused on turning the satellite to a specified direction via the turn_to action, but first checks to make sure the precondition of enough fuel for the turn is met.

Multigoal Method

m_img_move

This multigoal method realizes that there are two types of goals. The first is that an image of a destination has been taken in a specific mode. The second is that a certain satellite is pointing in a certain direction.

Completion of the first goal can be checked for by seeing if the images taken in the goal state have been already taken in the current state (using the have_image dictionary). If there remains an image to be taken, I call the m_take_image task method to start the process.

The second goal can be checked for by simply checking the direction the satellite is currently pointing in, and if it is the one in the goal state. If it is not, I call the task method point, which eventually calls turn_to unless there is not enough fuel.

I also pass in mgoal in the return statement, and check for an empty list as signifying completion of the tasks.

Helper Functions

find_instrument

This returns the "best" instrument (and associated satellite) for taking the image. The quality of the instrument is determined by the one that will require the least fuel to calibrate and turn to take the image, and it of course must support the requested image mode and have enough data for taking the image. Called by m_take_image.

find_powered_instrument

Finds the instrument that is currently switched on on the satellite. Called by power.

Experiments

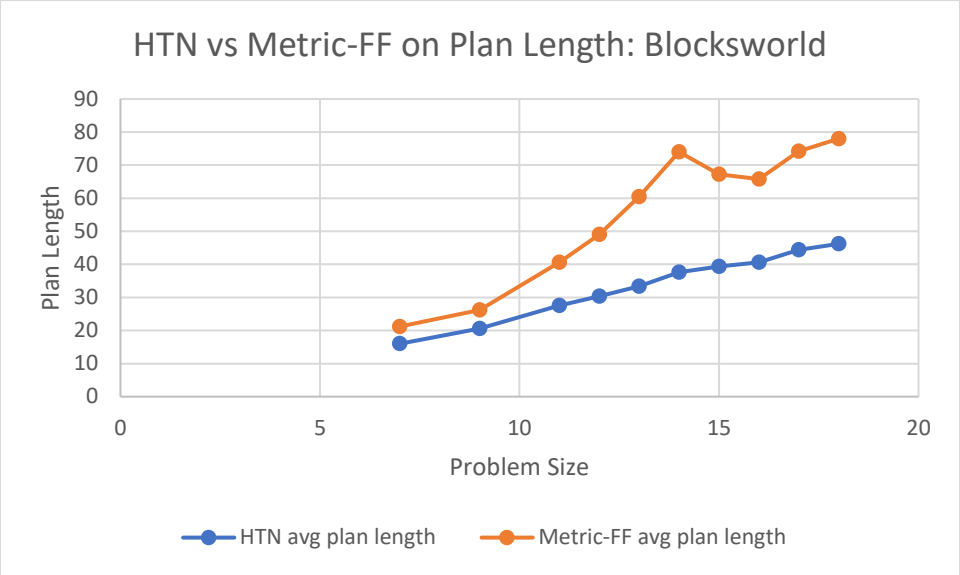
I decided to focus on two performance metrics: average CPU time and average plan length. In deciding whether to also include average number of nodes expanded, I had trouble understanding how Metric-FF indicates the nodes expanded, and what that really means. In future experiments, I would like to dig deeper into how Metric-FF expands nodes and analyze how nodes expanded compares between the two types of planners.

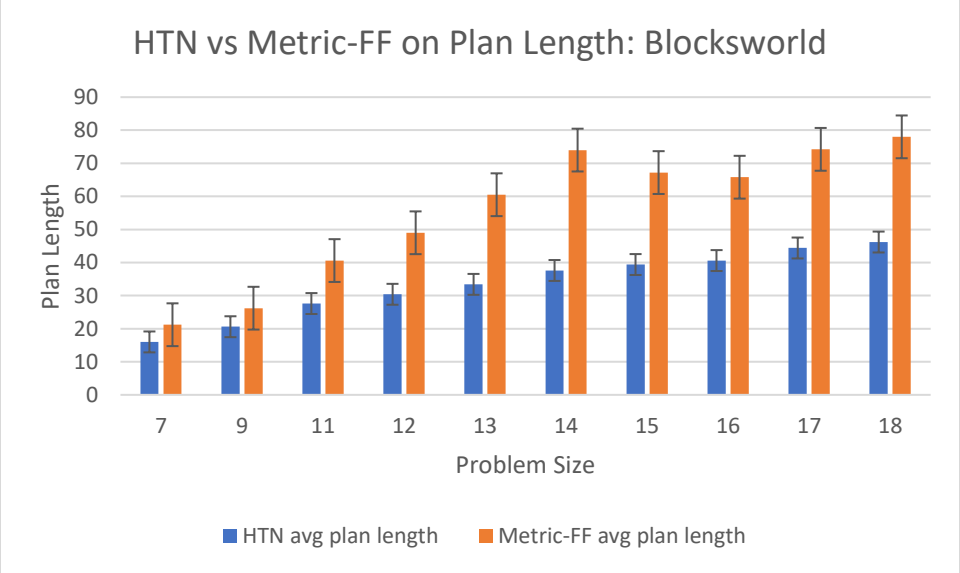
I laid out my reasoning for my upper bound on problem sizes when discussing Metric-FF in the Description of Planners section. Since Metric-FF can only support or run on problems up to a certain size, it was necessary to stick to that as an upper bound (18 for Blocksworld, 92 for Satellite) to allow for fair comparison. For Blocksworld, problem sizes incremented mostly by 1, whereas Satellite incremented by 10 for the most part. There were a few problem sizes that were not in those increments (for example, 91 and 92 in Satellite) just to push the limit and get as much data as possible on the upper limit size.

To collect results, I used a mix of scripts in the experiment.py files in the HTN files and a Google Colab notebook. These results can be found in the codebase. I generated Blocksworld problems via this site, using a seed of 1 and generating 10 problems per size at a time: http://users.cecs.anu.edu.au/~jks/cgi-bin/bwstates/bwcgi and Satellite problems using the C files Professor Nau provided (also found in the repo).

Results

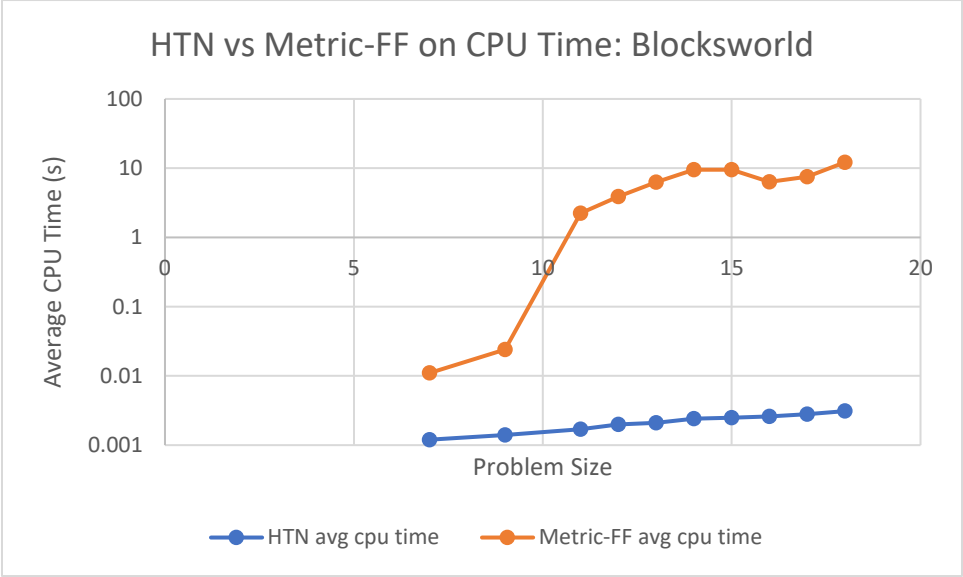
Blocksworld – Plan Length





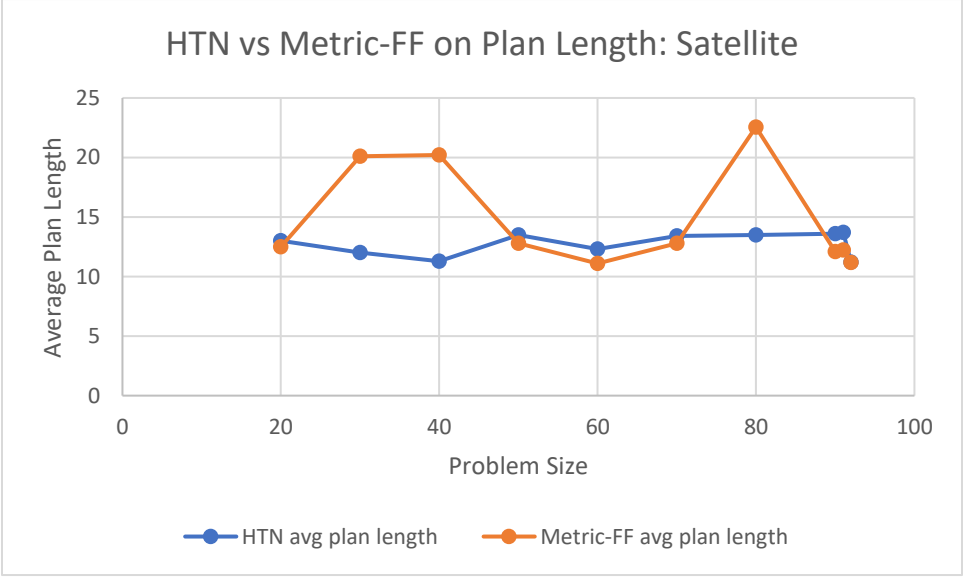
Here, we see that Metric-FF produces longer plans as the size of the problem scales. The error bars begin to separate as the problem size increases, indicating that the HTN planner does perform better in terms of plan length.

Blocksworld – CPU Time

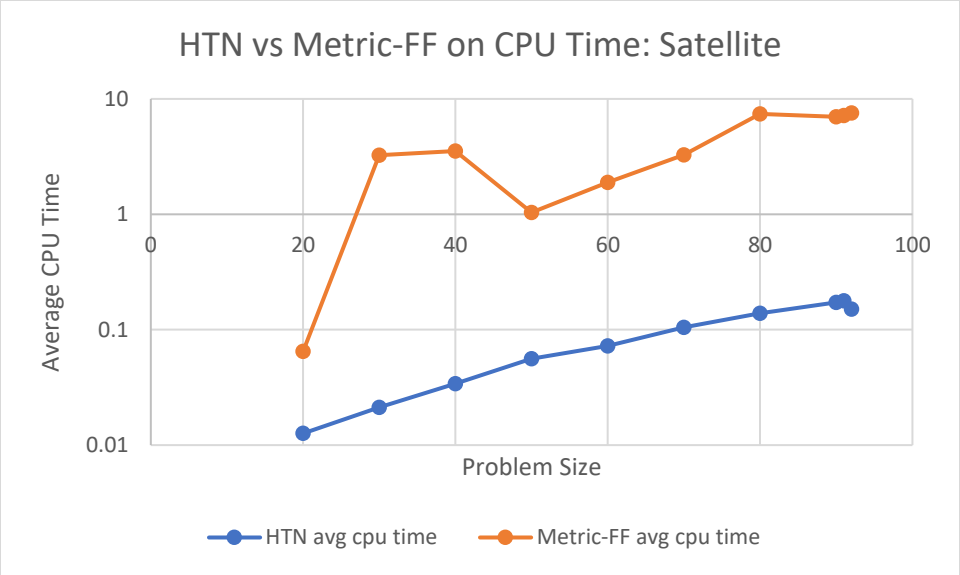


This was converted to a log scale as the CPU time for the HTN planner was so low. We can see that the HTN planner performs better for Blocksworld on both metrics this way.

Satellite – Plan Length



Satellite – CPU Time



This was converted to a log scale as the CPU time for the HTN planner was so low.

Conclusions:
Overall, these results indicate that the HTN planners, on average, perform better in terms of CPU time for both domains, and the Blocksworld HTN planner to generate plans with lesser length than Metric-FF. The Satellite HTN planner and Metric-FF were comparable in terms of plan length on the Satellite domain. From these results, one might have a slight preference for the HTN planners especially in the context of CPU time. As I discuss below, I think these results are promising but require further work before a strong conclusion may be drawn.

In the future, I would like to explore a larger range of problems, which may mean choosing a different independent domain planner. I think it would be interesting to compare several independent domain planners against the HTN planners to gain a better understanding of overall strengths and weaknesses rather than make a generalizing statement as to which type of planner is better with such a small sample size.

Supplemental Material

All code is in the "final project" directory. I have also committed my work to Github: https://github.com/aroy2530/722-final-project if that makes it easier to view. Navigate to GTPyhop folder to see HTN work and experiments and problems. The Metric-FF and Satellite problems notebook has Metric-FF related work and experiments, and the generation of Satellite problems.