SPML : Block 2 Planning Programming Assignment Report

Chantal van Duin (s1004516) & Nayeong Kim (s1006313)

October 2018

1 Introduction

This report will look into planning problems and relaxation heuristics, which heuristic is best for which planning problem and how different heuristics for different problems compare. More specifically, we will research which heuristic is the best to use in terms of time complexity for low, moderate and highly complex planning problems.

1.1 Chosen domains

1.1.1 Blocks

Blocks is a representation of a world where blocks can be positioned in various ways in terms of the position of other blocks and a table and whether blocks currently being hold by hands. There is an initial state of the positions of the blocks start of with and a goal state of how the blocks of the initial state should be positioned in the end. Actions can be performed on the block to change the positions. This representation of the world is converted as a STRIPS domain where the set of predicates is

P = { (on, 2), (ontable, 1), (clear, 1), (handempty, 0), (holding, 1) }

In this, every parameter in action should be Block type

O consists of the operators

O = { (pick-up), (put-down), (stack), (unstack) }

- pick-up : ((x), {(clear, x), (ontable ,x), (handempty)}, {(ontable, x), (clear, x), (handempty)}, {(holding, x)})
- put-down : ((x), {(holding, x)}, {(holding, x)}, {(clear, x), (handempty), (ontable, x)})
- stack : ((x, y), {(holding, x), (clear, y)}, {(holding, x), (clear, y)}, {(clear, x), (handempty), (on, x, y)})
- unstack : ((x, y), {(on, x, y), (clear, x), (handempty)}, {(holding, x), (clear, y), (handempty)}, {(clear, x), (handempty), (on, x, y)})

1.1.2 Movie

Movie domain represents a representation of a world where there is information containing running time of movie (e.g., if it is longer than 2 hours or not) and seat locations. In this world, a movie can be rewound and each person in a seat can get the food and snacks for the movie. To get to the goal state, the actions with movie information and the snack list stored with seat locations in the initial state must be performed. The representation of this world is converted as a STRIPS domain where the set of predicates is

P = { (movie-rewound, 0), (counter-at-two-hours, 0), (counter-at-other-than-two-hours, 0), (counter-at-zero, 0), (have-chips, 0), (have-dip, 0), (have-pop, 0), (have-cheese, 0), (have-crackers, 0), (chips, 1), (dip, 1), (pop, 1), (cheese, 1), (crackers, 1) }

O consists of the operators

O = { (rewind-movie), (rewind-movie-2), (reset-counter), (get-chips), (get-dip), (get-pop), (get-cheese), (get-cracker) }

- rewind-movie : ((), {(counter-at-other-than-two-hours)}, {(movie-rewind)}, {(counter-at-zero)})
- rewind-movie-2 : ((), {(counter-at-two-hours)}, {(movie-rewind)})
- reset-counter : ((), {}, {(counter-at-zero)}, {})

and operators for getting the food and snacks for the movie

- get-chips : ((x), {(chips, x)}, {(have-chips)}, {})
- get-dip : ((x), {(dip, x)}, {(have-dip)}, {})
- get-pop : ((x), {(pop, x)}, {(have-pop)}, {})
- get-cheese : ((x), {(cheese, x)}, {(have-cheese)}, {})
- get-crackers : ((x), {(crackers, x)}, {(have-crackers)}, {})

1.2 Relaxation heuristics

1.2.1 Blind heuristic

Using the blind heuristic is equivalent to using blind search or uninformed search, meaning that the heuristic provides no additional information about the states beyond what was provided in the given problem definition. Consequently, the successor nodes are generated and the only made distinguish between states is whether the state is a goal state or not. Since the blind heuristic provides no additional information about which non-goal state is more promising to reach goal state, the blind heuristic for a state usually corresponds to one constant value for example 0 or 1.

1.2.2 H_{add}, H_{ff} and H_{max}

H_{add}, H_{ff} and H_{max} are heuristic functions of relaxed problem tasks. They are used to estimate the cost of the cheapest path from a state to the goal state and ignore the bad effects of actions to reach that goal. The difference between them is in the way this estimation is computed and what assumptions are made to do so.

- H_{add} : Additive heuristic iterates through numerical values in the relaxed problem graph starting with rough estimates and updates the sum of the costs of predecessor vertices to the goal state(s). Additive heuristic is a pessimistic assumption as it assumes that sub-goals of a problem are independent of each other which can lead to overestimation of the actual cost to reach goal since possible reuse of paths (positive synergies)are ignored and overcounting can take place. H_{add} is not admissible nor consistent but is more informative than h_{max}.
- H_{ff} : H_{ff} is similar to the additive heuristic except is a cost-sensitive heuristic that cannot result in overcounting. Similar to the additive heuristic it iterates through numerical values in the relaxed problem graph however it also marks all the predecessor of an action or goal state but only marks the predecessor of a vertex with the smallest h_{add} value. The heuristic value is the summed cost of all the actions corresponding to the marked vertices. H_{ff} is not admissible or consistent but is usually significantly better than H_{add} and can be computed in linear time.
- H_{max} : Maximum heuristic iterates through numerical values in the relaxed problem graph starting with a rough estimates and finds the cost of the most expensive (maximum) fluent in the set of costs from start node to goal state(s). Maximum heuristic is an optimistic assumption as it assumes that when the most expensive fluent is reached, other fluents can be reached in parallel. It is an admissable and consistent heuristic.

2 Hypothesis

We expect that heuristic blind performs better on the tasks of easy complexity than the relaxation heuristics H_{add}, H_{ff} and H_{max}. For the moderate complexity tasks we expect that H_{ff} will perform better than the blind heuristic and H_{max} with the performance of H_{add} being a close second. For the high complexity problems, we expect that H_{ff} will perform significantly better than the blind heuristic, H_{max} and H_{add}.

3 Methods

To test which heuristic is best for which planning problem, we will test the planning complexity of 3 different problems (tasks) of 2 different domains for the heuristics blind, H_{add}, H_{ff} and H_{max}. To test the complexity of the problems in their domain using various heuristics and search algorithms, the provided program *pyperplan.py* was used. To measure the performance in terms of time complexity of the heuristics we will use the wall-clock search time that is given as part of the output using the *pyperplan.py*.

3.1 Implementation research question method

We chose to use 3 different problems for each domain with each problem having a different complexity level : one easy solvable problem, one with moderate difficulty level to solve and one difficult solvable problem. For domain Blocks to test this order of task corresponding to problem complexity level we chose to use task 1 for easy, task 18 average and task 35 for high complexity level. For domain Movie we chose to use task 1, task 15 and task 30. To take the inconsistency of the heuristics into account, 10 trials will be run for each task and heuristic. We choose to use one searching algorithm for every heuristic and trial to ensure that only the changing variable kind of heuristic is tested and not the different searching algorithm is what is influencing the planning complexity. We will choose to use the searching method A-star for this because we need to use a weighted searching algorithm

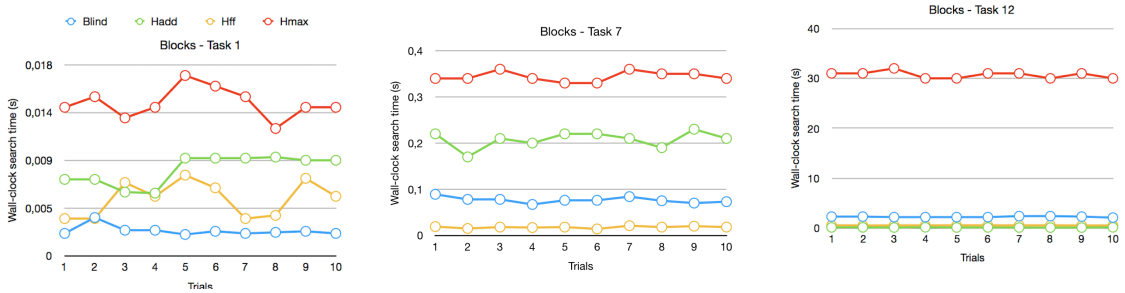
since the relaxation heuristics use weights to choose the most likely node based by attaching weights to nodes. Furthermore, A-star is a method that we are familiar with and is known to work well in general for solving most planning problems.

3.2 Hypothesis reasoning

The reason why we expect the blind heuristic to work better in terms of time complexity for tasks of low complexity is that it does not have to take time to compute the most likely successful node to reach the solution. The solution for low complexity problems can easily be found without using costs functions, which leads to the expectation that blind heuristic would have the lowest time complexity in comparison to the relaxation heuristics that need time to compute the costs of each node resulting in a higher time complexity.
For moderate complexity task we expect that relaxation heuristics will perform better than the blind heuristic. Since the solution is now not as easily found, the time spent on computing the weights will result in finding the solution faster than by using blind search which the blind heuristic uses. Within the relaxation heuristics we suspect that H_{add} and H_{ff} will have the smallest time complexity as they give a more realistic estimate of the cost of the cheapest path to goal than the cost estimation of H_{max} which it only finds the maximum fluent of set of costs from start node to goal. Since the problem is not extremely complex, we expect that H_{add} can still find the solution in a time close to H_{ff}, which computes the solution in linear time contrary to H_{add}.
For the highly complexity task we suspect to see a similar result as of the result of moderate complexity task however with the differences between the heuristics being more severe. We expect to see a bigger difference between H_{add} and H_{ff} as the solution is not as easily found making the characteristic finding the solution in linear time of H_{ff} have a bigger effect, resulting in H_{add} taking a noticeable longer time to find the solution.

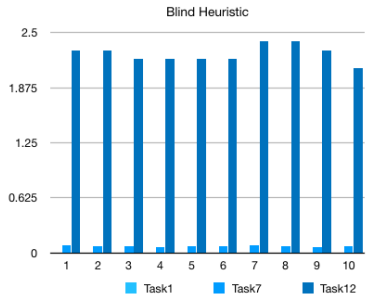
4 Results

4.1 Domain : Blocks

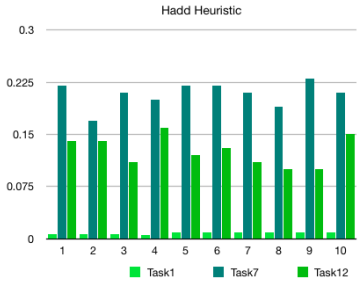


(a) easy task (task 1) (b) moderate task (task 7) (c) complex task (task 12)

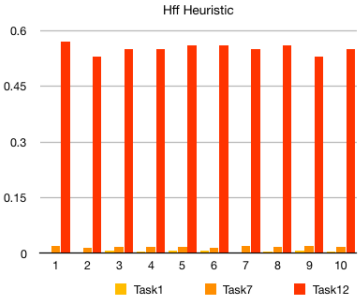
Figure 1: Line graphs of performance of heuristics on tasks of Blocks



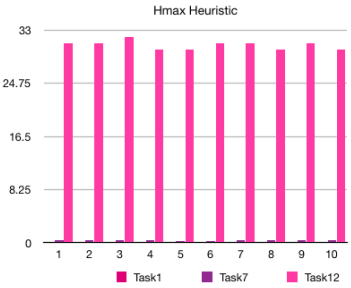
(a) Blind heuristic performance



(b) H_{add} performance



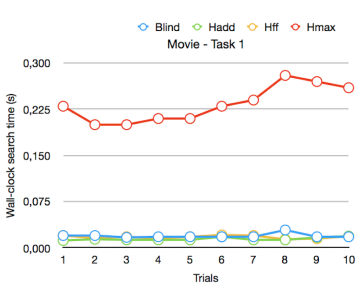
(c) H_{ff} performance



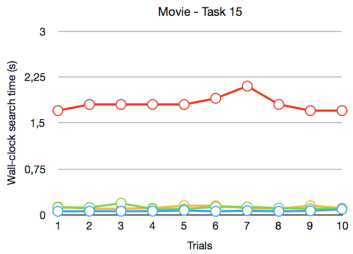
(d) H_{max} performance

Figure 2: Bar graphs of performance of each heuristic on tasks of Blocks

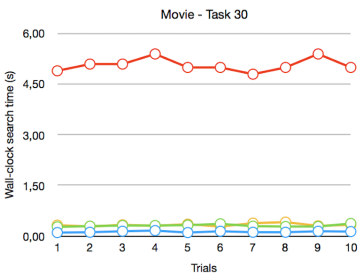
4.2 Domain : Movie



(a) easy task (task 1)

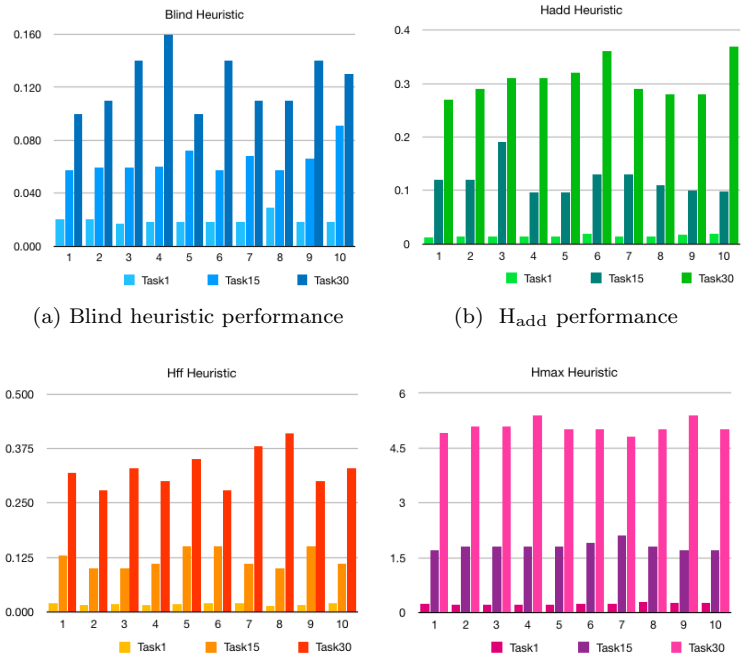


(b) moderate task (task 15)



(c) complex task (task 30)

Figure 3: Line graphs of performance of heuristics on tasks of Movie



(a) Blind heuristic performance (b) H_{add} performance

(c) H_{ff} performance (d) H_{max} performance

Figure 4: Bar graphs of performance of each heuristic on tasks of Movie

5 Discussion

Before discussing the interpretation of the results, it is important to state that we used different tasks for the domain Blocks than specified in the Results section $\Rightarrow$ task 1,7 and 12 instead of 1, 18 and 35. When running tasks 13 and 14 with the heuristics, we can see that each requires a run time at least one minute, which means that *pyperplan.py* does not terminate for a long time without any output. We suspect that is the result of the complexity of those tasks being too high for our chosen implementation of *pyperplan.py* and our searching algorithm. This point makes it difficult to presume how long it will take to check other heuristics 10 times each since there is no certain *pyperplan.py* will terminate in these cases, so we took task 12 as the difficult step to check the results more efficiently within finite time. In addition, we did not want to change the searching method in order to make sure that we only test one variable : the used heuristic.
First of all, from our results, we can see that the blind heuristic is indeed performing better than the other heuristics for the easily solvable tasks for domain Blocks. These differences seem not to be significant enough - with the exception of H_{max} for both domains - to conclude that the blind heuristic is indeed the best heuristic for easy solvable tasks. Especially for the domain Movie, the differences between the blind heuristic, H_{ff} and H_{add} are minimal. For the tasks of moderate complexity, the H_{ff} seems to be performing slightly better than the other heuristics for domain Blocks while for domain Movie the differences between the blind heuristic, H_{ff} and H_{add} are once again minimal.
In the case of the tasks of high complexity, there seems to be little to no significance to the difference between the performance of the blind heuristic, H_{ff} and H_{add}. However, from the difference

between the performance of H_{max} and the other heuristics, we can say that the performance H_{max} is significantly less in comparison to other heuristics for all tasks.
Furthermore, the actual result we got is not always the result that we expected. For H_{add} performance in domain blocks, task 7 takes more time for computing the solution than task 12 even though task 12 is more complex and contains more objects. We suspect that the characteristic of ignoring reusing or overcounting of addictive heuristic can be the reason for this unexpected outcome for task 7.
To check whether this happens only for task 7, we checked the result of H_{add} performance for task 6 and 8 which have similar complexity with task 7. Both of their run time is shorter than the runtime of task 12. The result of task 7 showcases the reason why H_{add} is not the optimal heuristic for planning problems in some cases. Further research is needed to confirm and support this conclusion more, than just the result of task 7.

6 Conclusion

For our analysis of the result, we can not confirm which heuristic is better in terms of time complexity as the results are not significant enough. The differences between the heuristics were not as big as we expected them to be, making it harder to come to a conclusion supported by significant result. We recommend doing more research on H_{add}, H_{ff} and the blind heuristic with more task of various complexity levels and more domains. We also recommend that further research test not just the time complexity but also the space complexity, optimality and expansion of nodes to further test the performances of the heuristics to conclude which heuristic is best.
From our results we can conclude that H_{max} is not advisable to use as it performs significantly worse than other heuristics even in comparison with the blind heuristic. Looking at our results we do suspect that H_{ff} is best heuristic. From our results we can see that it does not perform significantly less for tasks of low complexity in comparison to other heuristics while also working well for tasks of high complexity. H_{ff}'s performance is comparable to H_{add} but since H_{ff} is consistent, computes solution in linear time and does not over count or reuse paths it is better applicable for more situations than H_{add}. H_{add} can lead to unexpected outcomes such as task 7 for domain Blocks due it's characteristics, we advise to do more research on the characteristic of over counting and reusing paths of H_{add} to further confirm our conclusion on H_{add} .

7 Acknowledgement

Information on heuristics used to write section relaxation heuristics in addition to the slides of BKI212a - Artificial Intelligence : Search, Planning, and ML Planning lectures : `https://ai.dmi.unibas.ch/_files/teaching/fs16/ai/slides/ai36-handout4.pdf`