An Empirical Comparison of the Additive and Landmark Cut Heuristics

Min Chen
MSc Artificial Intelligence
min.1.chen@kcl.ac.uk
20002189

Hendrik Devries
MSc Artificial Intelligence
hendrik.devries@kcl.ac.uk
20107976

Toledano Diego
MSc Advanced Computing
diego.toledano@kcl.ac.uk
20113058

Kaiyu Huang
MSc Advanced Computing
kaiyu.huang@kcl.ac.uk
20094641

Kyriacos Mosphilis
MSc Artificial Intelligence
kyriacos.mosphilis@kcl.ac.uk
20100053

Abstract

In this paper, we compare two different configurations of the Fast Downward planner with the Additive heuristic and the LM-Cut heuristic. In order to do so, we carry out experiments for the two heuristics with five different domains and thirty test instances each. We present the collected data such as the plan cost or the overall coverage and analyse it accordingly. We find that outcomes of the Additive heuristic or the LM-Cut heuristic used in combination with the Fast Downward planner depend on many factors.

Part I

Introduction

When it comes to solving problems and to find a plan for them, we (or an AI agent) need to know how well we are performing, how close we are to the goal and how to select our next action. To do so, we are using heuristics. Heuristics output numerical values based on several factors of the given state of the problem. While some heuristics are as simple as giving us our relative distance to the goal, others are more complex. Hence, some heuristics are more feasible to solve a problem while some are not. Choosing a heuristic depends then on the given problem.

As seen later in this paper, the two heuristics we are comparing are the Landmark-Cut heuristic and the Additive heuristic. Those two heuristics are intrinsically linked since the Landmark-Cut heuristics compute a strong admissible estimate of the Additive heuristics. Hence, since those two heuristics are linked and that one of them is outputting another version of the other, we have had to compare them and figured out if tweaking the Additive heuristics resulting in the Landmark-Cut heuristics is more efficient than the simpler Additive heuristics.

This experiment is then conducted by using those two heuristics with the Fast Downward planner and testing which one solves chosen problems faster. We are then using five domains that work differently but all originate from the ICP 2008 competition. Those five domains are then applied with thirty different problems each. Doing this we ensure that we have a good variety of domains. To handle our experiment without being dependent on our different hardware configurations, we are using the cloud computing service offered by Google. By doing this we can ensure that all our tests will be using the same hardware and that we can run multiple instances at the same time without running into some issues. All our instances are then run using the same specifications.

Landmark-Cut Heuristic

The landmark cut heuristic works as an admissible heuristic on delete-free STRIPS tasks, we can guarantee the optimal solution by using landmark cut heuristic with search algorithms like A*.

Like all other landmark heuristics, the main challenge for the landmark cut heuristic is to find a reasonable landmark. To do this, the landmark cut heuristic uses three concepts:

- Precondition choice function (PCF), for each action in the search graph, the PCF choose only one of its preconditions.
- Justification graph, of which the vertices are states in the search graph, each edge is action from one of its preconditions selected by PCF to its postcondition. The justification graph is a simplified problem because some preconditions of actions are ignored by PCF.
- Cut, a subset of edges (i.e., actions) in the justification graph, contains edges that must be included in all paths from the initial state to the goal.

The landmark cut algorithm works as follows.
For a state I, we want to compute its heuristic value h^{LM-Cut}.

We initialise $h^{LM-Cut} = 0$, then iterate:

1. Compute h^{max} value for all states in the original search graph.
2. If the h^{max} value of the goal state $h^{max}(G) == 0$, then exit the iteration and return the $h^{LM-Cut}(I)$.
3. Call the PCF function to select preconditions with maximal h^{max} value, then compute and generate the justification graph.
4. Compute a cut for the corresponding landmark L that satisfies $cost(L) > 0$.
5. Increase $h^{LM-Cut}(I)$ with $cost(L)$ and decrease the cost of all actions in the landmark L by $cost(L)$.

Additive Heuristic

The additive heuristic is a simple heuristic which just sums up all costs to reach a given fact S to get the heuristic value $h^{Add}(S)$. The additive might overestimate the cost because one or more actions might be counted for more than one time, so it is an inadmissible heuristic.

Domain Discussion

To test the heuristics, we had to select various domains to test them and show which is the better one. The domains that were selected are (from IPC 2008):

- Sokoban
- Peg Solitaire
- Transport
- Wood Working
- Scanalyzer

The reasoning behind choosing these domains is as follows. All the domains are different because if we chose similar ones, then the results would not be fair, e.g. If two coordination problems existed and LM-Cut is particularly good for such problems, it would create a bias towards the research. This is because a heuristic should be good across various kinds of domains and not in favour of certain domains. Sokoban is a Japanese puzzle video game that is PSPACE-complete making it significantly more difficult than NP-hard problems and Peg Solitaire (also known as Solo Noble), which is a board game, is NP-Complete. Transport is a typical coordination problem that can be found in everyday life. Moreover, Wood Working and Scanalyzer are both industrial examples, whilst the one (Wood Working) is much simpler than the other (Scanalyzer) since it must deal with one wood at a time and not complicated batches.

Hypothesis

The landmark selected by the Landmark-Cut heuristic is based on the cut which is close to the target of the problem domain.

Since we need to compare the processing speed and processing accuracy of the problem domain with Landmark-Cut and Additive. We assume that the result of Landmark-Cut is better, and the number of nodes expanded is less than Additive heuristic, but the dealing speed is slower than additive (i.e., h^{LM-Cut} performs better than Additive heuristic in terms of accuracy, which means Landmark-Cut as the optimal planning with less plan cost. However, on the other hand, Additive heuristic can get results faster than LM-Cut).

Experimentation Discussion

To test all the domains, we decided to run the complete and optimal A* search and Greedy Search with both heuristics and Enforced Hill-Climbing with the Additive Heuristic. The experimentation part was split among the members of the team and tested on Google Cloud with the usage of multiple instances of C2 Compute-Optimised engine using Cascade Lake processors (Intel codename) with 16GB of RAM

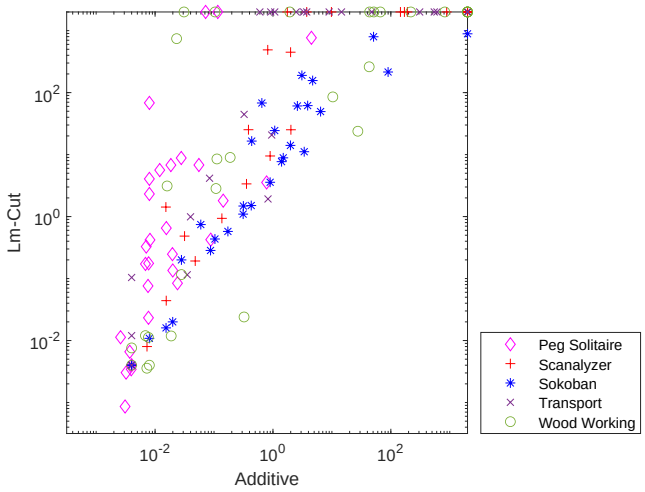

Figure 1: A* (Search Time)

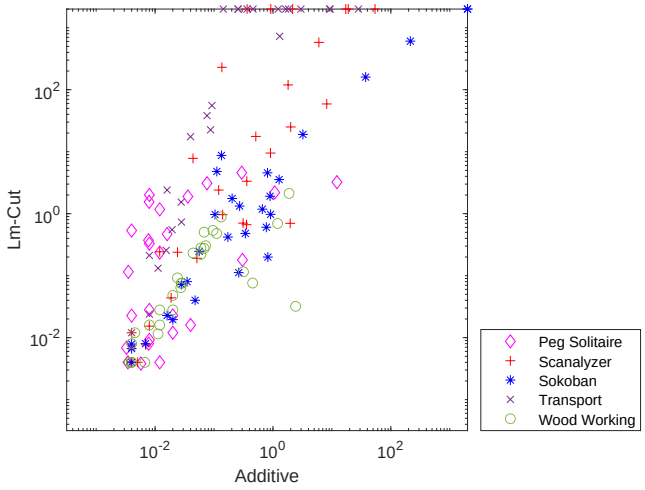

Figure 2: Greedy Search (Search Time)

using a Linux distribution, Ubuntu 20.04, as if we had equivalent powered computers to provide robustness and consistency to the collected data and flexibility among the group work. All the searches had a time limit of 1000 seconds (~ 16.667 minutes) and 4 Gigabytes of memory (RAM, that is). All the experiments were automated by a script that we created. The unfinished domains (the ones that reached the time limit) will be given shown as 2000 (the edge of the graph) and not taken into consideration while doing the appropriate calculations. One of our main limitation was our time limit of 1000 seconds (about 16 and a half minutes) we have chosen this limit because in real life, problem should be solved fast. We also chose to double the amount of RAM and shorten the time limit from the original since it has been 12 years since the competition and hardware has gotten better and cheaper.

Part II

Analysis

Among all our data, we have found that the best way for us to figure out our heuristics detailed was to compare these three statistics, the average plan cost of solutions, the average number of nodes expanded to find a valid plan, and the time is taken to do it. By doing so, we have found that that, on the one hand, we have an Additive heuristic which is faster on average than the Landmark-Cut heuristic as we can see the faster average time taken to find a solution in figure 1 with a faster pace to explore nodes – Node Expanded/Time Taken - While on the other hand, we have a Landmark-Cut heuristic which gives better results than an Additive heuristic in a way that the LM-CUT heuristic is cheaper to execute as we can see on figure 1 with the average Plan Cost.

There are two definitions of experimental record values. Note that the processing speed is related to the expansion node and processing time, so this corresponds to all three statistics mentioned above.

Plan Cost: represents our average consumption of instances of each problem domain. It is 2965.7178 for LM-Cut and 3620.9178 for Additive. This proves that LM-Cut has less average cost than Additive.

Process speed: dividing node by time (just like speed = distance/time), it can tell us the running speed of our LM-Cut and additive methods. For LM-Cut it is 1738629.13/406.7952=4273.96667, and for Additive it is 1150548.24/226.1759=5086.96215 (because 4273.96667 < 5086.96215, therefore it proves that LM-Cut has slower speed of process to the Additive heuristic). We compare the processing speed of the two and we can see that although the Additive method has more Nodes Expanded than LM-Cut, it also has less search time.

On a per-domain basis, we have found that as seen in figure 1, some domains work better with the LM-CUT Heuristic while others work better with the Additive heuristics. That has been found by figuring out which Search algorithm works better with the LM-CUT heuristics and which one works better with the Additive heuristics. To find this out, we have compared results from the average plan costs, the average pace to explore nodes and the overall coverage. For example, with the Woodworking domain, A* works better with the additive heuristics, this is due to the additive heuristics have a better coverage a better pace than the LM-CUT heuristics. Another example is Eager Greedy with the Peg-Solitaire domain, here, the average plan cost and the average pace are better for the LM-CUT heuristic than for the additive heuristic.

Consequently, here are the result per domains:

- Sokoban works good with both and slightly better with LM-Cut
- Peg Solitaire works good with both
- Transport works better with Additive
- Wood Working works good with both
- Scanalyzer works good with both

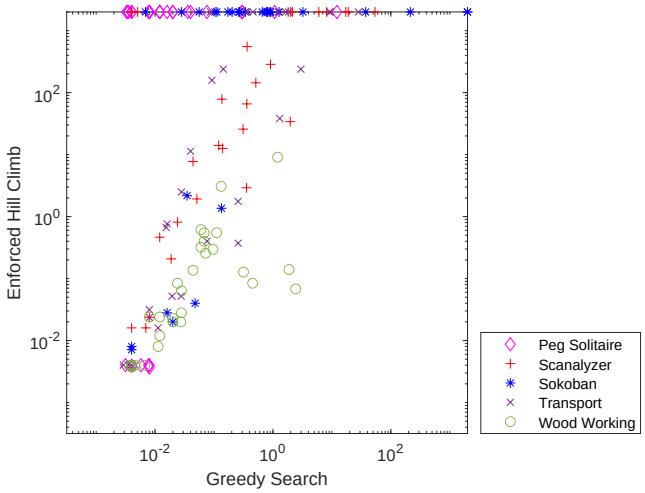

Figure 3: Additive Heuristic (Search Time)

Transport works exceptionally better with the use of the Additive heuristic; even though is it inadmissible. This can be seen in the table and at the graphs (numbers here) where the additive is able to solve almost all (Greedy Search solves all of them) the problems. With that in mind, it is still just an inadmissible heuristic as the plan costs are exceptionally big (cf. table 1).

Peg-Solitaire, Scanalyzer and Wood Working work good with both heuristics, since they almost solved the same amount of problems with the same cost for each plan. It is also good to mention that LM-Cut would be better to use here since it provides the best solutions for each problem and that Greedy Search solved the problems in an exceedingly small amount of time.

Finally, Sokoban was good with both heuristics but performed better with the LM-Cut and giving the best plan for all the problems with a reasonable duration. Finally, all the results can be found in the graphs and in the table more explicitly.

Furthermore, we tested the Additive heuristic with the Enforce Hill Climb (EHC) search, since it is an inadmissible heuristic and that is an incomplete search, which had a remarkably interesting outcome. As seen in the graph (put number here), EHC performs bad compared to Greedy Search since most of the experiments for Peg-Solitaire and Sokoban had failed (due to time restrictions). On the other hand, for both Scanalyzer and Transport it performed well giving reasonable results. But for the Wood Working domain, EHC outperformed all the other search using both heuristics giving an outstanding 100% coverage.

Note: The main reason that some problems-instances never finished is because they reached the time limit. In addition, this is the main reason why the coverage of LM-Cut is smaller than the Additive heuristic.

Conclusion

According to the experimental results, LM-Cut has less plan cost, but slower searching speed (this is reflected by pro-

Domain	Search Algorithm	Heuristic	Plan cost		Node Expanded		Time taken		Coverage
			Range	Average	Range	Average	Range	Average	
Wood Working	A*	LM-Cut	130.0∼400.0	244.5833	8.0∼696858.0	77464.9167	0.0∼742.023	93.8723	40.00%
		Additive	140.0∼450.0	288.4375	10.0∼834013.0	96374.5625	0.0∼822.966	73.4866	53.33%
	Eager Greedy	LM-Cut	130.0∼660.0	383.75	8.0∼55.0	21.25	0.004∼0.683	0.1651	66.67%
		Additive	155.0∼630.0	361	10.0∼17847.0	1142.85	0.0∼2.4205	0.2266	66.67%
	Lazy EHC	Additive	95.0∼865.0	427	18.0∼187945.0	10990.4333	0.0∼9.0698	0.5263	100.00%
Peg Solitare	A*	LM-Cut	2.0∼12.0	6.7143	6.0∼16708181.0	712541.25	0.0∼766.142	31.3138	93.33%
		Additive	2.0∼24.0	9.9667	6.0∼225076.0	10069.7	0.0∼4.4941	0.1961	100.00%
	Eager Greedy	LM-Cut	2.0∼23.0	8.8	6.0∼122397.0	21590.3	0.0∼4.5836	0.7344	100.00%
		Additive	2.0∼25.0	10.9333	6.0∼574685.0	22869.3667	0.0∼12.175	0.4698	100.00%
	Lazy EHC	Additive	2.0∼12.0	6.8571	7.0∼237.0	72	0.0∼0.004	0.0006	23.33%
Scanalyzer 3D	A*	LM-Cut	13.0∼54.0	28.3333	6.0∼699932.0	49456.3333	0.0∼489.185	66.8927	50.00%
		Additive	13.0∼69.0	34.4545	7.0∼200031.0	18699.0909	0.0∼883.744	63.858	73.33%
	Eager Greedy	LM-Cut	13.0∼90.0	39.6667	6.0∼17479.0	1578.5833	0.0∼579.58	43.9206	80.00%
		Additive	13.0∼139.0	53.1333	7.0∼715.0	120.9	0.0∼51.3954	3.5178	100.00%
	Lazy EHC	Additive	24.0∼92.0	49.2222	888.0∼3028405.0	375263.7222	0.012∼552.614	68.0945	60.00%
Sokoban	A*	LM-Cut	2.0∼76.0	28.4333	49.0∼6956040.0	568913.5	0.0∼888.724	85.9463	100.00%
		Additive	2.0∼86.0	29.7586	68.0∼3675467.0	239502.4828	0.0∼89.8076	6.0065	96.67%
	Eager Greedy	LM-Cut	2.0∼82.0	34.75	56.0∼5743870.0	300287.2143	0.0∼607.448	29.2268	93.33%
		Additive	2.0∼90.0	33.9643	57.0∼7824621.0	349133.75	0.004∼214.04	9.3757	93.33%
	Lazy EHC	Additive	2.0∼29.0	12.625	150.0∼79776.0	14546.75	0.0∼2.1597	0.4494	26.67%
Transport	A*	LM-Cut	54.0∼632.0	427.9091	6.0∼27921.0	5164	0.0∼44.5065	6.5897	36.67%
		Additive	54.0∼2069.0	849.8696	7.0∼8746051.0	410848.7391	0.0∼605.21	67.1377	76.67%
	Eager Greedy	LM-Cut	54.0∼5093.0	1762.7778	9.0∼8039.0	1611.7778	0.0∼726.967	48.1335	60.00%
		Additive	126.0∼6349.0	1949.4	8.0∼24947.0	1786.8	0.0∼28.1774	1.9011	100.00%
	Lazy EHC	Additive	54.0∼1878.0	837.6667	11.0∼1855922.0	242568.619	0.0∼238.529	32.9035	70.00%

Table 1: Experiment results

cess speed = node/time) and the longer total search time. This is not entirely consistent with our expectations of the experimental results before the experiment (The hypothesis is LM-Cut always has less planning cost and faster search speed, but more searching time).

For LM-Cut and additive algorithms, the effect of their combination with A* and Eager Greedy algorithms also depends on some characteristics of the problem domain. This is obtained by analyzing the processing goals of the problem domain and the specific structure of the problem domain.

In addition, we also compared Enforce Hill Climb (EHC) search with Additive heuristics with Eager Greedy search with Additive heuristics and found that EHC + Additive search results are extremely poor.

For domains like Sokoban and Peg-Solitaire of which solutions are relatively short and plan costs are relatively low, we usually care more about the efficiency of problem solving rather than finding an optimal solution, so that we might tend to use an inadmissible heuristic. But in domains like wood working which has a long plan and relatively high plan cost, it will be better to use an admissible heuristic to solve the problem.

After our analysis, we believe that for domains with similar functions such as wood working, Sokoban and Transport, they have similar actions in the domain, which also means that the layout of their domains may be the same, which may be the reason these three factors have comparable results (the reason additive is better than LM-Cut). After reviewing their three problem domains, it was confirmed that our idea was correct. After our analysis and comparison of the data, the result of the experiment is that additive algorithm works better in Wood Working, Transport works and Sokoban, LM-CUT works better in Peg Solitaire, while LM-Cut and Additive work equally well in Scanalyzer domain.

References

Bonet, B., and Geffner, H. 2001. Heuristic search planner 2.0. *AI Magazine* 22(3).

Helmert, M. 2009. Lm-cut: Optimal planning with the landmark-cut heuristic. *APA*.