UNIVERSITY OF ALBERTA

Project Report

Title: Dynamic Update of DSL for Heuristics Synthesis

Submitted To:

Dr. Levi Santana de Lelis

Assistant Professor, Faculty of Science - Computing Science

Course: CMPUT 659: Program Synthesis in XAI

Submitted By:

Md Rayhan Kabir

Student ID: 1674524

Department of Computing Science

University of Alberta.

Email: *mdrayha1@ualberta.ca*

1 Introduction

Program synthesis is a very interesting and important area of research. Different synthesis techniques has been defined by many researchers to make it efficient. In this project, our goal is to synthesize a strong heuristics for a grid system, where the start and goal state is given, and the heuristics will guide the search to find the best route to reach the goal state from the starting state by exploring least number of cells of the grid. In our case we will use an enumerative search technique named bottom-up search to synthesize a strong heuristics. Often in bottom-up search, the generated expressions are called program. So, in this whole report, "heuristics", "program", and "expression" denotes same meaning.

In bottom-up search all possible combination of the terminal nodes and production rules of the DSL are generated and they are evaluated against each other to find whether a heuristic is better than the other. This can be very challenging. Because in bottom-up search, the number of generated programs increases exponentially. So if we generate all the heuristics, it can be very time consuming.

There are several approaches of bottom-up search where the search is guided towards a better program. Among them using probabilistic grammar[1] is noteworthy. In probabilistic grammar, certain probability are learnt in the run time for the production rules and search is guided using that probabilities. This is a good approach but in our case for a stronger heuristics, most of the production rules and the terminal symbols are used in almost all the strong heuristics. Thus assigning certain probability to a production rule might fail in our case. Thus we decided to give importance to the whole program which shows a better evaluation score by adding that to our DSL.

In this project, we implemented a methodology where the DSL will be updated dynamically based on the evaluation score of the generated heuristics. That is if a heuristics has a better evaluation score, we add that heuristics to our DSL and restart the search.This provides a great advantage. That is we can combine strong heuristics among themselves and with other non terminal nodes in the first few iterations of the search. Thus we can get a heuristics with larger AST size by generating few programs. The advantage of this approach is, we don't have to evaluate all the programs to reach a program of large AST size.

2 Problem Description

In our project we used a 2-d grid map where one cell of the grid is a starting cell and another cell of the grid is a goal state. The goal to solve the problem is to find the best route in the grid from start state to the goal state by searching. The goal of this project is to find a good heuristics that can guide the search towards the goal state with exploring less number of cells in the grid.

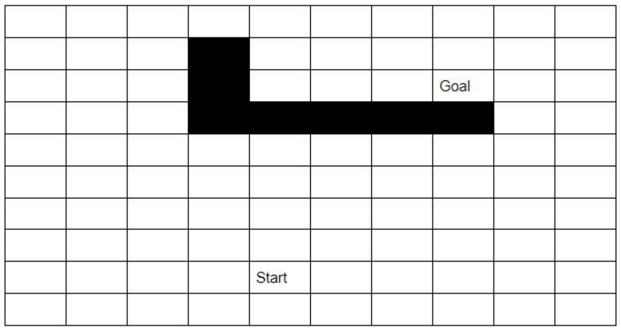

Fig. 1. Sample Map

In figure 2.1, we can see a sample map where the start state and goal state is given. There can be walls in the cells. Walls are to block a path. That is if there is a wall in a path between start and goal state, that path is not valid. We have to synthesize a heuristics that will guide the search for the best path from start state to the goal state.

In our experiment, we generated 50 pairs of start and goal states and evaluated each heuristics using evaluation metrics to decide whether a heuristic is better than the other or not. The brief description of the DSL, evaluation metrics are provided in the following sections.

2.1 DSL

The DSL that we will use in our project is provided bellow: Here, the Abs(S) returns the

$$S \rightarrow S{+}S \mid S{-}S \mid S{*}S \mid \text{Max}(S) \mid \text{Min}(S) \mid$$
$$\text{Abs}(S) \mid x \mid y \mid x\_goal \mid y\_goal \mid 0.5 \mid 2$$

absolute difference of two values. We used some restrictions which seems logical. For example, if the two values of Max and Mean are same, then it will always return the same value. Thus we decided to restrict our DSL from generating programs like that.

2.2 Evaluation Function

In our project, we have used two different evaluation function. The first one is using Manhattan distance with A^* search. Manhattan distance is the sum of distance of start state to goal state in X axis and the distance of start sate and goal state in Y axis.

Another evaluation function that we used in our project is: it keeps the track of the h-value from start state to the goal state along the path and observe whether the distance is increasing or decreasing.

The reason behind using a second evaluation function is running A^* is very costly in terms of computation time. For this reason we decided to use the second evaluation function in our project.

2.3 Evaluation Metrics

In both of the evaluation function, we calculated the total number of cells processed while finding the optimal path for all 50 start-goal pair. We used this metrics to decide whether a heuristic is better than the other or not. If a heuristic explores less than another, it means this heuristics is guiding the search better to find the path to the goal state as it is exploring less number of cells in the grid. In our first evaluation function we directly used the number of cells explored but in the second heuristics we used some regularization to ensure the values are consistent. The other metrics that we used is cost. Cost indicates total cost to find the optimal route from start state to goal state for all 50 start-goal pair. We observed this cost value but decided a heuristics is better than the other using total number of cell processed.

3 Our Approach

From previous discussions we understand that running bottom up search(BUS) to find a good heuristics can be very very costly and time consuming as we have to evaluate all the generated heuristics. In addition to this, our evaluation functions are also costly as we have to evaluate the heuristics for all 50 start-goal state pairs. Thus it will be nearly impossible to find a good heuristics easily using BUS without any guidance.

We used a very simple but effective approach to guide our search for a good heuristics. In each iteration we generated all possible combination using the production rules of the DSL. Then we search for a better heuristics among the generated programs. Initially, we set the minimum-explored-cell-value to infinite. Then after generating the heuristics, if any heuristic has explored-cells is less than the minimum-explored-cell-value, we update the minimum-explored-cell-value to the explored-cells value of the current heuristic and save the current heuristics in a separate list. Here, to evaluate the heuristics by the evaluation metrics provided by the path based evaluation. We continue this to all the heuristics that is generated in a given iteration. In this way we evaluate all the heuristics that are generated in a single iteration. After evaluating all the heuristics, we get heuristics those showed better performance than their previous heuristics. Thus, we assume these heuristics have better chance to be a part of stronger heuristics. So we add those in our DSL and restart our search.

We run a 10 iteration to search for a stronger heuristics. We recorded the heuristics those are added to the DSL and their evaluation metrics provided by both the evaluation functions. While generating programs, We also checked for weakly equivalent programs. We decide a heuristics is weakly equivalent to a previous heuristics by generating output by the current heuristics. After generating outputs by the current heuristics we check whether same outputs were provided by any other heuristics previously or not. If any previous heuristics provided the same output then we consider the current heuristics as weakly equivalent and discard it.

4 Result and Discussion

We run BUS using the mentioned method and recorded the number of programs generated in each iteration. We provided the number of program generated in each step in table 1. In the table, in our approach in each iteration we generated all possible combination of the programs but in the approach of increasing by size of AST, we allowed to increase the size of the AST of program by 1 in each step.

Table 1. Number of program generated in each step

Iteration	In our approach	Increasing by size
1	6	6
2	85	–
3	180	85
4	324	64
5	260	1271
6	945	2663
7	360	25008
8	411	72653
9	15915	529497
10	1194	907519

From table 1, we can observe something interesting. We see in size based approach, the number of generated program becomes very high within a few iteration and if we move forward using this approach it will be very hard while the size of the program increase. But in our approach, the number of generated program is low because we could find heuristics that can outperform another withing one or two iterations and the number of program in first few iterations where the number of generated program remain smaller. In the next discussion we are going to provide the explanation, how our approach is finding a heuristics with a larger size of AST within few iterations. We want to comment one thing here which is very important for the data shown in table 1. As our start and goal states are generated randomly for different run the heuristics that are added to the DSL sometimes varies. Thus the number of generated program also varies. In a run of the synthesizer we got around 52k programs generated and evaluated in step 9 which took almost 1 hour 12 minutes to evaluate and find a strong heuristic. But in all run, we ended up getting the same heuristics. We should make further analysis to detect how to keep the number of generated programs stable. Though it is a big question whether the number of generated programs should be stable or not as we update our DSL dynamically based on the performance of the heuristics.

For better understanding of how the synthesizer reached to a heuristics with larger AST size within few iteration, at first let's see the programs that are added to the DSL during the BUS. The programs and their evaluation metrics are provided in table 2.We said before, we generate the 50 pairs of start and goal state pair randomly. Thus for different runs, the evaluations metrics can be changed.

Table 2. Heuristics that are added to the DSL

Heuristics	Path based evaluation		Manhattan distance with A^* search	
	Cost	Expanded value with regularization	Cost	Expanded
(y + y)	177	-1667	5852	662935
(y + x)	175	-1685	5735	746752
(abs((y - y_goal)))	78	-3202	5717.5	331801
(abs((x - x_goal)))	70	-3361	5717.5	316046
(abs((y - y_goal)) + abs((x - x_goal)))	53	-4022	5779	31740
max(abs((y - y_goal)), abs((x - x_goal)))	29	-4654	5717.5	181960
((abs((y - y_goal)) + abs((x - x_goal))) + max(abs((y - y_goal)), abs((x - x_goal))))	39	-4887	6004.5	11843

From table 2, we can see the heuristics that are added to the DSL during run-time. Here after first iteration we found (y+y) as the first program that is to be added in the DSL. We found another program(y+x) in the same iteration which outperforms (y+y). So we add both of them to the DSL, we restart the search and withing two iteration we can find two more programs abs(y-y_goal) and abs(x-x_goal) each of which are better than their previous one. So we add those to our DSL and restart search. Thus in the next iteration this two heuristics that are added to the DSL will directly be combined with the other non-terminals of the DSL and also among themselves directly. Thus we can combine the heuristics having better performance among themselves in first few iteration and combining them we found more strong heuristics. In this way our approach can generate stronger heuristics with larger size by generating fewer number of programs in BUS.

From the Result section we can clearly observe that adding heuristics to the DSL provides a huge improvement to search a heuristics. This method guide the search towards a better heuristics by providing the important sub-programs that can be a part of a stronger heuristics. From table 1, we can see, in first few iteration, (abs((y - y_goal))) and (abs((x - x_goal))) were added to the DSL as they showed improvement to search for the optimal path to the goal state. After adding to the DSL and restarting the search, in one iteration it can find the next best heuristics max(abs((y - y_goal)), abs((x - x_goal))) in just 1 iteration. by combining the two newly added heuristics with a production rule. So in this way this approach guides the search towards stronger heuristics.

In our approach, we are allowing BUS to generate programs of all size in a single iteration. That is we are not controlling the program generation by any metrics. For example, there are several approaches to control the number of generated programs in each iteration. One way is described previously, increase the number of size of the program by 1 in each iteration. The program generation can also be controlled using program cost or other metrics. But here we are not using any metrics. This approach has a potential danger. The number of generated programs becomes sky high within a few iterations. But in our case it is unlikely. Because we are searching for a heuristics. And from the experiment we can see that, Stronger heuristics are found by combining heuristics those provides better evaluation metrics by combining previously added heuristics within 1 or 2 iteration. So it is highly unlikely that to find the next better heuristics, we have to go for several iteration where the number of generated program will be very large.

Improvement over run time Our approach provides a huge improvement over the run time to find a strong heuristics. Previously we mentioned the evaluation using A^* search is very costly. Thus we used a path based evaluation. Still if we generate the heuristics based on size, in 10 iterations, we have nearly 9.8 million programs to evaluate. But in our approach, in 10 iteration, we evaluated only 195K programs and got a strong heuristic of size 19. And to evaluate these program we needed around 26 minutes time. In size based program generation, we didn't evaluate the program. We just generated the programs to get the number and found to get e heuristics of size 10, we had to evaluate around 10 million programs which will reach to sky high if we generate the program of size 19 in size based approach. From this discussion, we can clearly understand that, our approach made a huge improvement in the run time to find a strong heuristics.

5 Drawback and Future Works

One drawback of this approach is, we run the program several times. As the start and goal state are generated randomly in each run, we found there is a difference by which the heuristics are augmented in the DSL. for example we found in one run abs((x - x_goal)) was added to the DSL and in another run it wasn't. Thus for different run the number of generated program varies which causes the variation in the run time. We have to analyze further to find out by which we can make this stable.

Our approach can be improved by adding complementary heuristics to the DSL. By complementary heuristics, we mean the h-value decreases for a portion of the path from start to goal state by a heuristics, for rest of the portion of the path, h-value by another heuristics decreases. We call those two heuristics complementary to each other. So, by using both of them together, we will have a stronger heuristics which can guide the search. So, it is a good idea to search for the complementary heuristics and add them to the DSL. We are interested to explore this idea in future.

6 Conclusion

Different approaches of BUS can be used to find a strong heuristics generated from a DSL. But current approaches are not feasible as the AST size of a strong heuristics is typically large. We proposed a method by which we can get a heuristics with larger AST size in a small amount of time by evaluating a very small amount of programs compared to the other approaches. We used a DSL to generate heuristics which should guide the search to find best optimal path from start sate to the goal state in a 2-D grid system. From our experiment, we found our proposed approach can find a strong heuristic with larger AST with less time by evaluating really a small number of heuristics. Though this approach works in our DSL, further study is needed whether this approach works on other DSL or on the DSL of other domains.

References

1. Barke, S., Peleg, H., Polikarpova, N. (2020). Just-in-time learning for bottom-up enumerative synthesis. Proceedings of the ACM on Programming Languages, 4(OOPSLA), 1-29.