Report on Assignment 1

Submitted To:

Levi Santana de Lelis, PhD

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

In Assignment 1, experiment has been done with two different search approaches named bottom-up search and Breadth First Search[1] to find the correct expression generated from a given Domain Specific Language(DSL) for a given set of input-output. The DSL for the experiment is:

$$S \rightarrow x \mid y \mid \cdots \mid 1 \mid 2 \mid \cdots \mid (S + S) \mid (S * S) \mid \text{if } B \text{ then } S \text{ else } S$$
$$B \rightarrow (S < S) \mid B \text{ and } B \mid \text{not } B$$

This report explains and compares the results of the two approaches. Rest of this report is organized in the following order. Section 2 provides the results obtained from the experiment. In section 3, comparison and a brief analysis of the results has been provided. Some general observations on the optimality and completeness of both the algorithm has been provided in section 4. Finally, section 5 concludes the report.

2 Results

In the experiment, there were three different sets of input-output. Both the approaches has been implemented and then tested for each of the input-output set. In table 1, number of program generated, number of program evaluated and time to find a suitable program has been provided. The programs have been executed on google Colab and program execution time has been measured using *time*[2] module of python.

Table 1. Result of the experiment

	Input-output set 1			Input-output set 2			Input-output set 3		
	Generated	Evaluated	Time (sec)	Generated	Evaluated	Time (sec)	Generated	Evaluated	Time (sec)
BUS	19	8	0.003	714	602	1.163	1655	337	0.609
BFS	234	22	0.082	5797[1]	264[1]	48.23[1]	9554[1]	676[1]	143.70[1]

In the experiment, the suitable expression for the first input-output set is same in both of the approaches. For case 2 and 3, the expressions have been obtained in BUS approaches. In case of BFS, the search bound was increased but still no suitable expression was found.

[1] BFS was executed after increasing the search bound to 5 but no suitable expression was found. The generated programs, evaluated programs and execution time are for searching up to bound 5. Not for finding solution.

Later on, in addition to increasing the bound the integer operations were decreased based on the output of BFS. After reducing the integer operations same output were found using the BFS. That is in the experiment, same results were produced in both the approaches.The expressions which have been found by the search approaches are provided below.

1. Expression for Set I: (if $(x < y)$ then x else y)
2. Expression for Set II: (if $(10 < (x * x))$ then (if $(x < 10)$ then x else y) else y)
3. Expression for Set III: (if $(x < y)$ then $(y * -1)$ else $(x + y)$)

It is to be mentioned that, there was several constraints for both bottom-up and Breadth First search approach.due to the constraints, the desired expression wasn't found withing the search bound. The bound has to be increased to find the correct expressions.

3 Result Comparison and Analysis

3.1 Number of Generated Expressions

From the result of the experiment we can see, the BFS approach generates more expression than the BUS approach. This is because BUS starts with all the terminal nodes and in each iteration it produces expression with all possible combination. In our experiment we put a restriction on generating new expressions. The searching starts with the abstract Syntax Tree(AST) of size 1 and then in each iteration the AST size is increased by 1. Thus BUS searches for the small AST first and then goes to the larger AST. For this reason this approaches can find the required expression by generating fewer number of expression.

On the other hand, in BFS approach, the search starts with the starting symbol of the DSL. Then in each iteration it can only replace one non-terminal symbol. In our case we also restricted the program to replace only the left most non terminal symbol. Another property of BFS is to expand the lower level expressions first. For example, before expanding an expression of level 3, all the expression of level 2 is to be expanded and evaluated. Thus to get a complete expression of larger size, several iteration is required. Also for the BFS, a huge number of partial program is generated in the search process. But in case of BUS, all the generated programs are complete. For this reason, BUS produces less number of program to find the correct expression.

3.2 Number of Evaluated Expressions

In terms of number of evaluated expressions, We got two different scenario. For case I and III, where the number of terminal symbol was 2 and number of integer values was 2. BFS evaluated more expressions then BUS. From the discussion of previous subsection, we know

BFS produces more expression than BUS while searching an expression. As more expressions are generated in BFS, it is expected that BFS approach will evaluate more expressions.

But in case II, where the number of terminal symbol was 2 and number of integer values was 1, BUS evaluated more expressions. One reason of Case II can be we couldn't complete the execution of BFS to find the correct expression. And there are huge number of partially completed expression in BFS, the generated programs weren't suitable for evaluation. If we can complete the execution of BFS for CASE II, we will be able to get the complete scenario.

3.3 Required Time to Find Suitable expression

It has been mentioned previously that the execution of BFS for case II and case III weren't finished for the lack of proper resources and time. For this reason, only the result of Case I can be compared in terms of execution time. Here it can be found that the required time to find correct expression for BUS is almost 16 times faster than BFS. The reason behind it is BFS starts from the starting symbol and incrementally expands it by one symbol in each iteration whereas BUS start with the terminal symbol and in each iteration it find expressions with all possible combination and every expression is complete. Thus less time is required for BUS to find the correct expression.

During the experiment, in addition to increasing the bound in BFS, the provided integer operations were reduced and the expansion of the partial expression was restricted based on the suitable expression found using BUS. It has been found that using the smaller set of integer operation i.e. using only the required operation, approximately 936 second and 427 second ware required to find the correct expression for Case II and Case III respectively.

From this discussion we can clearly understand that for the given DSL, BUS is much faster than BFS in terms of required time.

4 General Observation

4.1 Optimality and Completeness of BUS

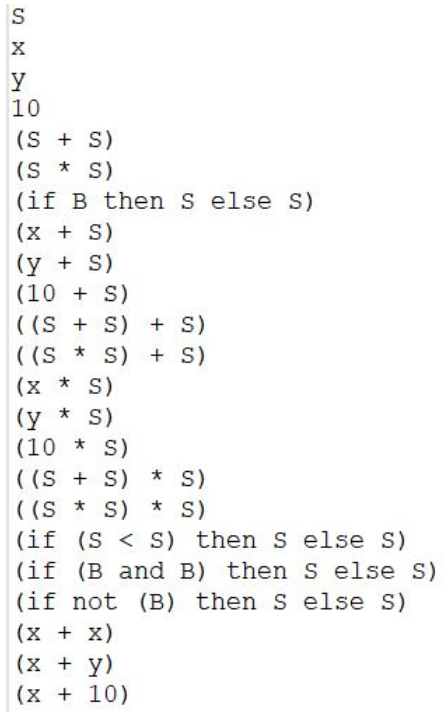
From the experiment, it has been observed that BUS starts searching from the terminal nodes of the DSL and in each iteration the size of the AST increases by 1. Thus it starts from the smallest complete program and gradually forwards towards the larger program. This ensures the optimality of the approach. That is BUS will find the smallest expression and the expression which generates first. From the experiment, the expression found for Case II is **(if (10 $<$ (x $*$ x)) then (if (x $<$ 10) then x else y) else y)** but there are other solution for the input-output set of Case II. For example another solution for Case II is **(if (x $<$ y)) then**

(if (x < 10) then x else y) else (if (y < x) then y else x)). But BUS found the first one as the size of the AST of the first expression is smaller. From this we can say BUS is Optimal.

BUS produces expressions with all possible combination of the terminal symbols and operation. For this reason it is ensured that if there is a solution for the given set of input-output it must be found using BUS. This is the completeness property of BUS.

4.2 Optimality and Completeness of BFS

From the discussion of previous sections we have come to know that BFS starts with the initial symbol of the DSL and gradually expands it to form the complete programs in all possible combination. It is also mentioned that in BFS the evaluation of program or expansion is done level by level. For this reason the smaller programs are evaluated before the larger programs. So for BFS also the smaller expression which suits the given input-output set will be found in the search before the smaller program Thus BFS is also optimal.



```
S
x
y
10
(S + S)
(S * S)
(if B then S else S)
(x + S)
(y + S)
(10 + S)
((S + S) + S)
((S * S) + S)
(x * S)
(y * S)
(10 * S)
((S + S) * S)
((S * S) * S)
(if (S < S) then S else S)
(if (B and B) then S else S)
(if not (B) then S else S)
(x + x)
(x + y)
(x + 10)
```

Fig. 1. A portion of generated programs using BFS

From the experiment we can see the BFS is complete too. In the experiment, the generated expressions were appended in a text file. A small portion of the programs from the text file is provided in Figure 1. From the figure, it can be seen that all possible expressions are produced using BFS from smaller to larger. This if there is a solution for the given input-output set, it can be found by BFS. So we can say BFS is complete.

5 Conclusion

From the experiment, it has been found that Bottom-up search and Breadth First Search both are optimal and complete. It has been also found that for the given DSL, bottom up search is more efficient in terms of generated program, evaluated program and required time to find a solution. Another observation of the experiment is that BUS could take a long time to find a solution as the number of generated program increases exponentially in each iteration. Restricting AST size to increase by 1 in each iteration can improve the performance of BUS to a great extent.

References

1. Kononenko, I., amp; Kukar, M. (2014, March 27). Learning as search. from https://www.sciencedirect.com/science/article/pii/B9781904275213500058
2. Time – time access and conversions¶. (n.d.). Retrieved February 04, 2021, from https://docs.python.org/3/library/time.htmlmodule-time