Artificial Intelligence (CS323)

Project Report - Trisem 1 (2020 - 2021)

Code Readability using AI

B18CSE030 - Manan Shah

Instructor: Dr. Yashaswi Verma

Department of Computer Science and Engineering

Contents

1 Introduction

One of the most important qualities of a good code is the readability of code, it varies from developer to developer because writing code is a subjective topic (*Hosk's dynamic blog* 2014). As stated in *Cost and efforts of software maintenance* (2019) the amount of budget spend on maintenance by an organization is more than 65 % of the amount spend on a software system, one of the task involved during maintenance phase is reading the existing code base which leads to the fact that former developers must have written readable code.

1.1 Motivation and Constraints in Current System

Any software organization wants engineers who writes correct and clean code. Correctness of code is their main criteria for selecting the candidates during online coding rounds, but there can be thousands of candidates submitting code which passes all the test cases. Manual task of reading thousands of code is not possible due to limitation of resources and biasing nature of humans may give inappropriate results. So, organizations don't give importance to code readability during coding rounds and shortlist on the basis of resume using Applicant Tracking System (ATS) (*Getting your CV shortlisted* 2015). This may not be appropriate to candidates who have written correct and clean code in online coding round but not getting shortlisted because ATS shortlists on the basis of resume formatting and keywords.

1.2 Problem Novelty

As mentioned in *Learning a Metric for Code Readability* (2008) code readability includes indentation, selecting proper identifiers name, comments, etc which are the notions defined according to the natural language but that may become biased towards the candidates having grip on natural language in code. So, I have measured the readability of code on the basis of ease of logic, given a problem it can be solved using various logically equivalent algorithms but solving it using tougher logic decreases its readability. **Logic readability is defined in terms of data flow in a program. The program where the flow of data is smooth will be referred as readable in terms of logic.** Moreover, use of redundant variables in the code decreases its readability. Thus, candidates who have written more logic readable code will be shortlisted.

1.3 Problem Class

During this project, I have tried to measure "Logic readability of the code", a metric (which will be called *cost* or tranformation cost) is proposed which takes into measure how close is given code logic, when the setter solution is taken as baseline. The metric tries to measure the cost required to transform the candidate codes to setter code and then rank candidates code according to readability. Initially, a graph of variable dependency is created from the code of setter and candidates, then the cost to transform is measured by formulating the problem as Constraint Satisfaction Problem (CSP - a class of problems defined in Tsang (2014)) to get the rank of candidates codes. The CSP so formed is a variant of Discrete Variable and Finite Domains CSPs.

2 Problem in Depth: Logic Readability

With the code whose logic readability is required to measure, there also needs to be some baseline with which we will compare the candidates code, so a correct and one of the most logically easiest solution is required (we will call that solution as setter code, i.e. the solution will be given by the setter of question). We also require converting the given code into some sort of representable form, with which we can measure candidate codes logic readability by comparing it with the setter code.

2.1 Graph from Code

I have used graph which is one of the most famous representable form in which various structures are stored such as in Computer Networking problems, Google maps, etc. We require only the logic details from the code into our graph, that is obtained from data flow in the code. With the use of variable dependency, I have tried to capture how the data will flow because variables are the center part of data flow. The graph is defined as $G(V, E)$, where V = the variables in a code and $E = \cup$(directed edge from var_1 to var_2, if value of var_2 is calculated from var_1). Also, graph G has no self loops. Listing 1 shows a function to compute average of n numbers, with Figure 1 showing its graph according to variable dependencies.

Listing 1: C++ function to compute average of numbers in a given array

```
/*
    During the demo of project some assumptions are made on variable declaration,
    defination and computation, to extract variable dependencies from a code.
*/
double function(double arr[] , int n ){

    double sum ;
    sum = 0 ;
    int i ;
    i = 0 ;

    for(;i<n;i++){
        sum = sum + arr[i];
    }

    sum = sum / n;
    return sum;
}
```

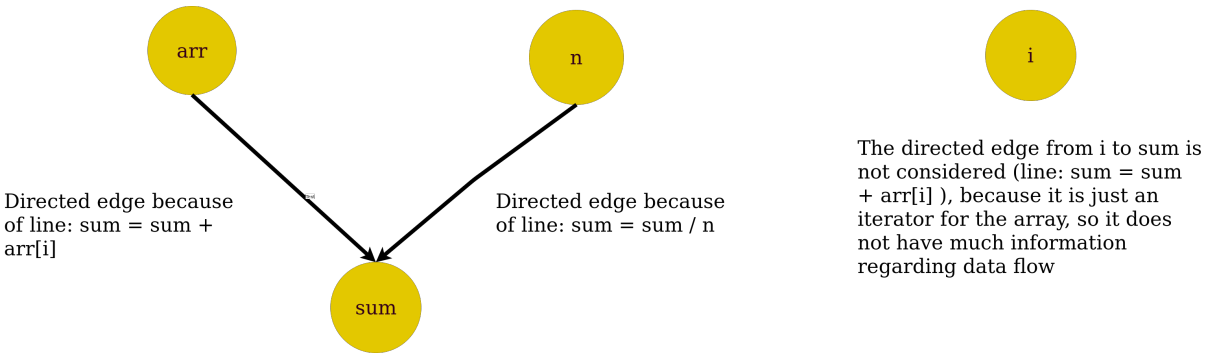

Figure 1: Graph representation of code in Listing 1.

2.2 Transformation of Candidate Graph

After getting a setter and candidate graph from these codes, we need to reduce candidate graph into a setter graph (setter graph is one of the most logically readable code) as it is considered as our baseline for evaluating other candidates. I formulate the reduction as a CSP. The reason for this formulation is because the path from candidate graph to setter graph is not important but the cost of transformation is important. The number of nodes in setter and candidate graph are s and c respectively.

2.2.1 Standard CSP formulation

According to the parameters defined in Tsang (2014) which are required for CSP, the given task is formulated as a CSP with following parameters:

Variables: $\{X_1, X_2, ..., X_c\}$
Domain: {0,1}
Constraint: Modified Candidate graph must be equal to Setter graph

X_i denotes whether node i of candidate graph is present or not. $X_i = 0$ denotes node i is not present while $X_i = 1$ denotes node i is present. Initially none of the variables are assigned any value, at each level a variable is assigned a value either 0 or 1. After all the variables are assigned, the modified candidate graph is formed using the value of variables and compared with setter graph.

2.2.2 Modify Graph from Variables value

The search methods of assigning variables are defined in the later section. After assigning all the variables, we need to modify our candidate graph. The variables having value 1 don't change the graph but for variables X_i with value 0, means that we need to remove node i. The modification for each node as be stated as: **"If there is a path $a \rightarrow b \rightarrow c$ and the value of variable denoting b is 0 then it is modified as $a \rightarrow c$"**. This modification is done for all incoming edges and outgoing edges for the removed node considering all paths. A modification on a graph is shown in figure 2

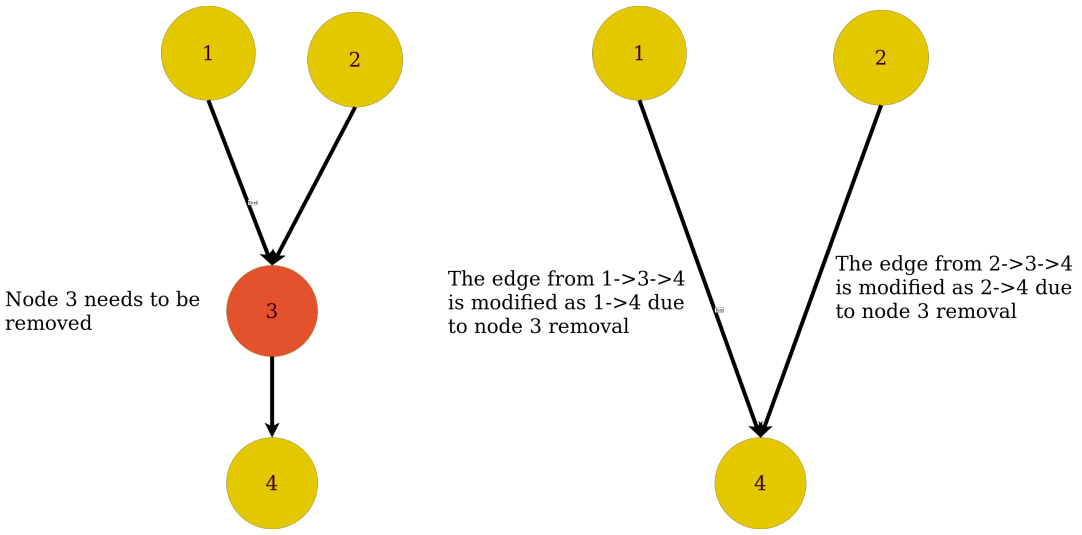

Figure 2: Graphical representation of removing a node

The modified graph is then compared with setter graph. The comparison is done on the basis of structure of graphs. The name of the variables are not taken into consideration, only the structure is considered. As shown in figure 3, the left and the middle graph are equivalent though having different variable names. The middle and right graph are not equivalent because there structures are not same despite having some similar variables.

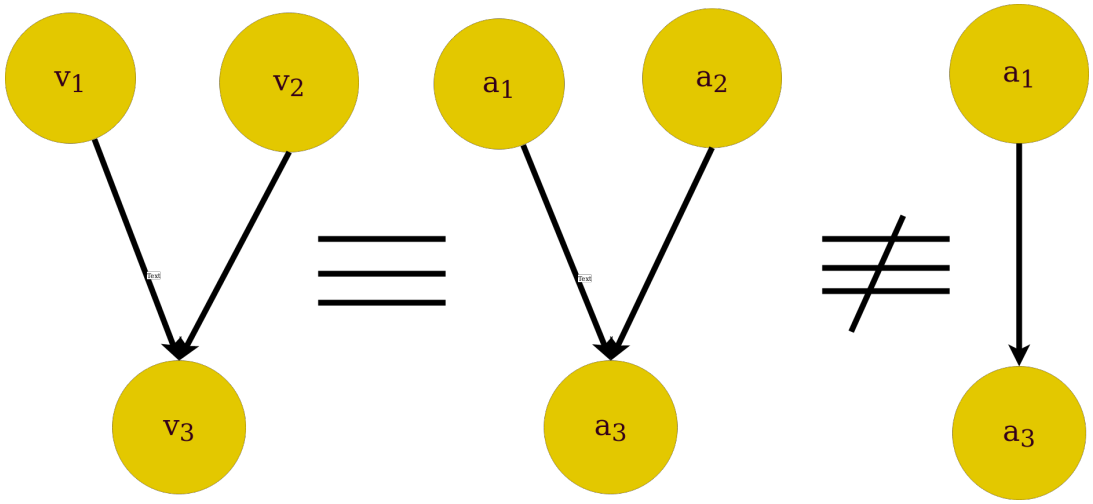

Figure 3: Equivalence/Non-equivalence in graphs

2.2.3 Cost of Transformation

When we get a modified equivalent candidate graph, we need to get the cost of this transformation. The nodes which are removed in the shallow level indicates the use of variables in earlier section of the candidate code and the arithmetic used in earlier section of code needs to remembered for a longer time when compared to arithmetic used in ending section. So, the nodes which are removed in shallow level must have higher cost then the nodes removed in deeper level. Let C(i) be the cost required to remove node i is defined as:

$$C(i) = \begin{cases} d_c - l_i, & \text{if } X_i = 0 \\ 0, & \text{if } X_i = 1 \end{cases} \tag{1}$$

where d_c is the depth of candidate graph and l_i is the level of node i. The total cost of transformation is:

$$cost = \sum_{i \in C} C(i) \tag{2}$$

cost is our transformation cost. Figure 4 shows the calculation of C(i) for a node removal in a candidate graph. **"The graph having more cost of transformation implies that the code is having more cost and eventually less logically readable, when compared to codes having less transformation cost"**. After we get the cost of each candidate code, then we can order there code logic readability and shortlist accordingly.

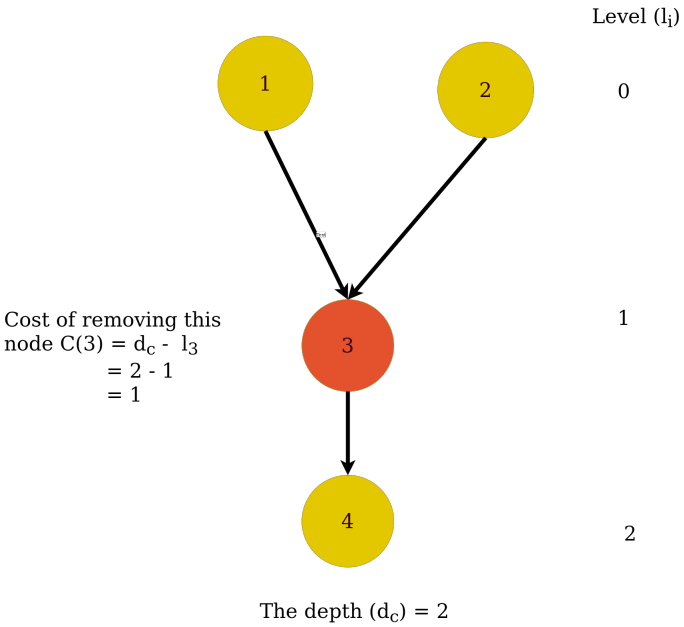

Figure 4: Graphical explanation of C(i)

2.3 Search Algorithms

Search is required to find out the values of $X_1, ...X_c$ such that we get equivalent graph with setter graph. Following are the search parameters:

- **The state** of a search is defined as the values assigned to $X_1, ..X_c$.
- **Initial state** is empty assignment.
- **Successor** function is to assign an unassigned variable a value from its domain (i.e. 0 or 1).
- **Goal test** is to check whether the modified candidate graph is equivalent to setter graph.
- **The search space** for this is all the 2^c combinations of assigning variables.

We can use BFS or DFS for this task, but DFS is advantageous because of its memory advantage over BFS. Can there be a way to predict failure which are sure to occur when we will perform further assignment?

2.3.1 Nodetrack

This search uses DFS with the use of an additional condition such that it considers best and the worst possible assignment, checks whether it is possible to further reach to a goal state from the present state. Best assignment means that it assumes that all the unassigned variable are assigned the value 1 and checks whether the number of nodes which can be achieved in the modified graph is greater then s (number of successor nodes). Worst assignment means that it assumes that all the unassigned variable are assigned the value 0 and checks whether the number of nodes which can be achieved in the modified graph is lesser then s. Let $assg$ = number of assigned variable at this level.

We must stop our search when the boolean value of *backtrack* is true and then perform backtracking from this state.

$$nodes = \sum_{i \in \text{assigned variables}} X_i \tag{3}$$

$$backtrack = (c - assg + nodes < s) \text{ or } (nodes > s) \tag{4}$$

nodes is the count of total number of nodes which are assigned value 1. The former condition in (4) represents the best assignment condition, if from present state the maximum number which are possible is less then number of nodes in setter graph then we cannot reach goal state. The later condition in (4) represents the worst possible condition, if present state already has more than setter nodes then also we cannot reach the goal state. Thus, Nodetrack tries to predict earlier failures with the use of node counts and helps to backtrack. In this search we haven't specified the order of assignment nor the value which must be tried, can we do that?

2.3.2 Valorder

Valorder is a Nodetrack search with the use of ordering. In CSP the one of the main goal is to reduce the search space. This is can be done by ordering (e.g. Minimum Remaining Value, Least Constraining value as mentioned in Tsang (2014)). I propose that we can fix which variable must be assigned next and which value must be tried on the variable to reduce the search space. BFS, DFS and Nodetrack doesn't make use of setter graph except the last comparison when all the variables are assigned, we can use setter graph and use its structure and compare with the structure of candidate graph. The structure comparison is done on the basis of the number of outgoing edges from setter and candidate node. The variable which must be assigned next must be done in increasing manner to compare such structure similarity with setter graph. The heuristic of assigning value: **"When both the referred setter and candidate node have same number of outgoing edges then value 1 must be tried else value 0"**. The C++ code snippet for this is ordering is given below:

Listing 2: C++ code snippet which returns the value which must be tried on next variable

```
/*  currentSetterNode is global variable which keeps the reference of setter node
    currentCandidateNode is global variable which keeps the reference of candidate node
    return what must be expected value assigned to next candidate node */

int nextExpectedValue{

    // terminal case
    if(currentSetterNode > s || currentCandidateNode > c){
        return rand()%2;
    }

    // when number of outgoing edges of both referenced nodes are same then current
    candidate node may be there
    if(sum(outEdgesSetter[currentSetterNode]) == sum(outEdgesCandidate[
    currentCandidateNode])){
        currentSetterNode = currentSetterNode + 1;
        currentCandidateNode = currentCandidateNode + 1;
        return 1
    }

    // when number doesn't matches then current candidate node may not be there
    currentCandidateNode = currentCandidateNode + 1;
    return 0
}
```

3 Background Survey

Code readability is not considered by various organization in there coding rounds with the use of any automated programs, they either use manual checking or skip it for coding rounds. In *Learning a Metric for Code Readability* (2008), A technique for the construction of an automatic software readability metric based on local code features is proposed but it considers the identifiers, keywords, indentation, spaces, commas, loops, assignments, line length and such natural language features. But *Learning a Metric for Code Readability* (2008) don't take into consideration of unused variable or redundant variables in a code, which the proposed "Logic Readability" considers.

4 Discussion

4.1 Novelty

The organization for shortlisting candidates for software engineers or similar profiles do not consider the readability of code and uses ATS. In this project, I have proposed a manner in which they can take into account "Logic Readability" of candidates code so that they can shortlist candidates accordingly. The uses of "Logic Readability" which is different from Natural Language readability which may become bias towards candidates who have a good hold on english. Also, to introduce Natural Language readability in a code requires time to think what name must be assigned to a variable in code and that is subjective which can again become bias.

For measuring the readability of a code in by solution, I have made a graph from code which tries to model the code into a graph. The model depicts the data flow which will happen in code. Then, I propose a metric (*cost*) which tells how different candidate graph is when compared to a setter graph (baseline). This transformation of graph consider the problem as CSP where the propose search method **Valorder** defines what variable must be assigned next and what value must be tried so that number of nodes searched during the search decreases. The Valorder uses the structure of setter graph at each state to assign an expected value to a node, it is an ordering scheme.

4.2 Scalability and Feasibility

The things required for this type of comparison is a good baseline (setter code) and a good parser. The good baseline means that it requires a one of the most logically readable setter code which is not very tough task for online coding rounds as the question statement are kept in view to the time that will be allotted to solve a given question. The second thing which is required is a good parser, it must be able to capture maximum variable dependency in a graph irrespective of the format of declaration and arithmetic of variables. Building a parser is not an easy task but it is required to be built for one time, so building it may require some resources initially but then it is reused. So the solution can be scaled with a good baseline for each question and a good parser which is required to built only one time.

The solution will be feasible for comparing readability of smaller codes and coding rounds have smaller codes as mentioned earlier. Smaller code means the code having lesser variables, for codes having 20 variables then the solution is feasible because in worst case it will be $O(2^{20})$ which takes nearly 1 second. However, for question involving use of many variables may require more time and the solution may be infeasible for that type

of questions. Generally, The parser will also be feasible as it can capture the variable dependency in one scan of any code. So, feasibility will depend on the number of variables that will be used in a question.

The solution is engineering based solution because of the solution automating the task of logic readability with being Scalable and Feasible upto a certain extent.

4.3 Results and Observation

The results are calculated for codes which are written for calculating average of n numbers. Listing 1 shows the setter code. Listing 3 and 4 shows the code of Candidate 1 and Candidate 2 respectively for which the results are calculated. There is a comparison of cost between two candidates and comparison of nodes expanded in for various search for each individual Candidate.

Listing 3: C++ code for calculating average of n numbers by Candidate 1

```
// code has passed all test cases then cheking its logic readability
double fun(double arr[] , int n ){
    double array[n] ;
    int range ;
    int sum ;
    range = n ;
    int i ;
    sum = 0 ;
    i = 0 ;
    for(;i<n;i++){
        array[i] = arr[i] ;
    }

    i = 0 ;
    for(;i<n;i++){
        sum = sum + array[i];
    }

    double average ;
    average = sum / range;
    return average;
}
```

Listing 4: C++ code for calculating average of n numbers by Candidate 2

```
// code has passed all test cases then cheking its logic readability
double fun(double arr[] , int n ){

    int sum ;
    sum = 0 ;
    int i ;
    i = 0 ;

    for(;i<n;i++){
        sum = sum + arr[i];
    }

    double average ;
    average = sum / n;
    return average;
}
```

4.3.1 Result and Observation of Transformation Cost

Table 1: The transformation cost of candidate codes

Candidates	Cost of Transformation
Candidate 1	4
Candidate 2	1

The Logic Readability of Candidate 1 code (Listing 3) is lesser when compared to Candidate 2 code (Listing 4) because there is use of variables which can be avoided in Candidate 1's code when we try to understand there codes. The cost of Candidate 2 code is lesser then Candidate 1 code as mentioned in Table 1, which means Candidate 1 code is less readable. So, the transformation cost shows the logical readability of Candidate codes when Setter code is taken as baseline.

4.3.2 Result and Observation of Nodes Expanded

Table 2: Number of nodes expanded in Candidate 1 code

Search Algorithms	Number of nodes expanded
BFS	224
DFS	71
Nodetrack	57
Valorder	11

Table 3: Number of nodes expanded in Candidate 2 code

Search Algorithms	Number of nodes expanded
BFS	56
DFS	21
Nodetrack	19
Valorder	9

For both the candidates the number of nodes expanded in Valorder is least when compared to other three searches. This shows that by the use of structure property of setter graph, ordering the variable which must be assigned next and what value must be tried with that variable, we can reduce the number of nodes that will be expanded during the search.

5 Algorithmic Analysis

5.1 Optimality of Solution

The optimality for Logic Readability is the algorithm which helps to clearly differentiate which code has more redundant variables. To get redundant variables we need to capture all the variables and check whether there is more better way to write the code without use of that variable. The solution proposed in Section 2 takes into account all the variable dependencies and tries to reduce the candidate graph into setter graph such that each node has a varying cost according to its depth. So, the solution considers both the aspects which makes it optimal.

Another form in which optimality can defined is the optimality of Search Algorithms (the search tells what value must be assigned to variables) such that search returns the correct assignment. As Valorder, in the worst would end up checking the whole search space it is guaranteed to return optimal solution.

5.2 Time Complexity Analysis of Valorder

The branching factor is 2 (domain of each variable) and the depth of the search tree is c (the depth is number of variables to which value will be assigned). The size of search space is $O(2^c)$. However, Valorder tries to predict which value must be assigned, say Valorder is able to predict the value of c' variables correctly. Due to some correct assignment it will not explore the full state space. The time complexity for Valorder Search can be given as $O(2^{c-c'})$. In the best case when Valorder performs correct assignment to all nodes, the time to reach the goal state will be $O(1)$, while in the worst case still it will end searching the full state space.

5.3 Faster Non-Optimal Search

As in worst case the Valorder Search will explore the whole search space, this might make it infeasible for some bigger questions in coding rounds (in terms of variables). Rather then trying to find exact reduction of candidate graph to setter graph, Local Search Methods can be used which tries to reduce candidate graph to setter graph as much as it is possible in the given time constraints. Thus, with the use of Local Search we can get some reduction and some cost related to that transformation in given time, but the cost returned by it will not be optimal because the reduction is not exact. So, some error will be there in the cost of transformation but can be used in some fields of practical life because of its time complexity advantage.

6 Worked Example

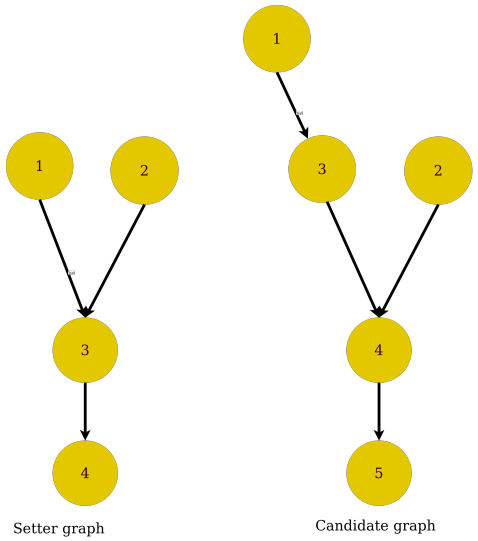

Figure 5: The graphs of code for worked example

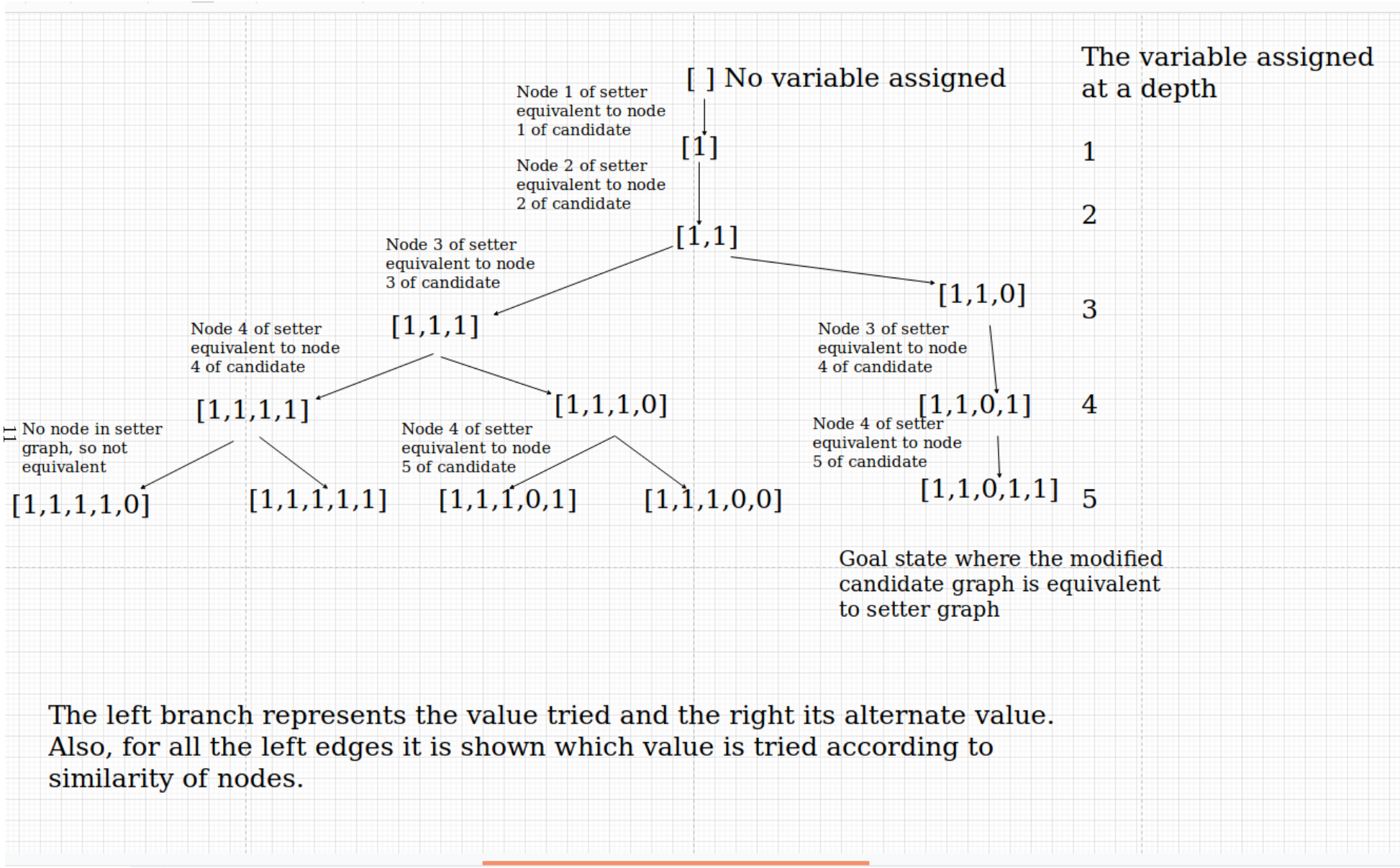

Figure 6: The state space exploration for graphs shown in figure 5

The graphs which are considered for workout example are shown in figure 5, I have assumed those are the graphs which we will get from a given setter and candidate code, to run Valorder search.

The exploration of search space is show in figure 6. X_1, X_2, X_3 and , X_4 are assigned 1 because of structure similarity. Then X_5 is assigned 0 because no node of setter is present (terminal condition mentioned in code snippet of Listing 2). But the assignment do not give equivalent graph so we need to continue our search. When the search assigns X_3 the value 0 then X_4 and X_5 are assigned 1 because of structure similarity and the state passes our goal test so we stop our search. The number of states which will be explored during search are 4 (before we get goal state).

7 Summary and Conclusions

The project defines what is Logic Readability of a code, proposes a metric (*cost*) to measure Logical Readability of a code using a baseline code. It converts both the code into graph using variable dependency in code to capture the data flow. Then proposed Valorder search to get the reduction of one graph into another graph when framed as a CSP. The cost of transformation supported the defined logic readability of the code. Moreover, Valorder search showed relatively less exploration of search space by using structure similarity of graphs. The solution is scaleable but feasible for coding questions which require use of few variables (around 20 for normal processors) with a good parser for creating graph.

References

Cost and efforts of software maintenance (2019). Geeks for Geeks. URL: `https://www.geeksforgeeks.org/cost-and-efforts-of-software-maintenance/` (visited on 10/24/2020).

Getting your CV shortlisted (2015). URL: `https://talentvis.com/article/detail?id=Winning-Over-Algorithms-and-Getting-Your-CV-Shortlisted`.

Hosk's dynamic blog (2014). CRMbuisness. URL: `https://crmbusiness.wordpress.com/2014/12/08/why-code-readability-is-important/` (visited on 10/24/2020).

Learning a Metric for Code Readability (2008). URL: `https://web.eecs.umich.edu/~weimerw/p/weimer-tse2010-readability-preprint.pdf` (visited on 10/24/2020).

Tsang, Edward (2014). *Foundations of Constraint Satisfaction.*