ISSN No. 0976-5697

Volume 5, No. 7, September-October 2014

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Novel Boolean Expression based Algorithm to find all possible Simple Paths between two nodes of a Graph

Sankhadeep Chatterjee
Department of Computer Science & Engineering
Hooghly Engineering & Technology College
Hooghly, India

Debarshi Banerjee
Department of Computer Science & Engineering
Hooghly Engineering & Technology College
Hooghly, India

Abstract: A novel algorithm has been proposed to find all possible simple paths between any two given nodes of a graph which is a NP-hard problem. First a novel approach to represent a graph using Boolean operators has been structured. The unique Boolean expression is used to find all possible paths between any two nodes. The analysis of Boolean expression based representation of a graph reveals that the problem is in NP-hard. Further a necessary and sufficient condition is given to show that the problem is not NP-complete. A detail theoretical analysis and experimental results has been given in support of its ingenuity.

Keywords: Boolean Transformation, Counting problems, Graph enumeration, NP-completeness, Satisfiability, Undecidability.

I. INTRODUCTION

Finding all possible paths between any two given nodes of a graph is a form of counting problems which are computationally equivalent with $#P$-complete problems [1]. The indication of intractability and its presence has been reported in literature. The problem statement is strictly bounded by finding out all possible paths between any two nodes in graph having no cycles. The presence of cycle would make the problem ambiguous to some extent. The problem of counting maximal independent sets which is similar in context of problem intractability has been reported to be in #P-complete [2]. A generalised counting problem in form of Satisfiability has been proposed by Creignou et al [3]. An efficient approach on counting problems has been reported in literature [4]. An association with decision problems has been established by using a witness function. Wrathall [5] has established the idea behind polynomial-time hierarchy which can be successfully used to classify those problems which seems to have a polynomial time algorithm but no such algorithm is known. A novel quantum mechanical observable based mechanism to count number of paths has been established by Markopoulou et al. [6]. The number of paths is calculated to be an expectation value of specially engineered quantum mechanical observable. The expectation value gives good estimation of number of paths and seems promising in terms of time complexity. Provan et al. [7] has proposed a probabilistic approach to tackle the problem of finding whether two given node of a graph is connected or not thus in the other words finding any possible path between the given nodes. Bu et al. [8] has reported a study of a new problem which tries to find out new binary strings apart from the given one which can be satisfied by a set of given operations. The study has revealed an $O(m^2n)$ algorithm to verify whether a string can be represented by another set of literals. The study draws a vital conclusion on satisfiability problems, generally tackled in problems dealing with Boolean expressions. Okamoto et al. [9] has proposed multiple algorithms to solve different problems based on counting number of independent sets in a chordal graph. Study has revealed that problem of counting the number of maximal independent sets and counting the number of minimum maximal independent sets are #P-complete. Efficient algorithms to solve problems like counting the number of independent sets of a fixed size and others have been proposed and found to be solvable in polynomial time. Another probabilistic approach to estimate number of connected pairs has been reported in literature [10]. The study has found enormous application in the field of reliability and survivability in communication networks. Survivability criteria has also been studied by Frank et al. [11] and reported to be useful in network designs which are more resistible to anonymous attacks. The notion of satisfiability and reduction procedures has been reported in the classical paper by Cook [12]. Tovey [13] has proposed a simplified version of NP-complete satisfiability problem. The paper has explained an important conclusion of solvability of SAT in linear time if no variable appears in more than two clauses. Johnson [14] presented a classical approach to describe the concept of NP-completeness. The concepts covered are found to be extremely supportive in establishing the fact that the problem of interest in this paper may be undecidable. Study of NP functions by Faliszewski et al. [15] has revealed that decreasing ambiguity may be done in terms of elimination of solution. The backtracking [16] nature of one of the algorithms proposed in this paper can be observable in other optimization problems and even in artificial intelligence field [17]. Thus the immense variety and interest of researchers in the classical fields of NP-completeness prompted the authors to study and analyse one of the famous and not so well studied problem of finding out the all possible paths from one given node to another given node in a graph. In addition to analysing the algorithms and results, an attempt has been made to point out an efficient way to find whether the problem is NP-complete or not is also included.

II. PROPOSED METHODOLOGY

A. Boolean Transformation –

Boolean transformation of a graph is based on the notion of simple Boolean logical operators such as '$\wedge$' (logical AND) and '$\vee$' (Logical OR). The transformation is done by analysing the branching of the graph at different nodes. The expression is generated by the experience one may have if one start a journey from the given source to a given destination. For instance following the graph depicted in figure 1 it is observed that starting a journey from node '1' there is only one choice at beginning but at '2' we get two choices. This situation is the most generalized version of a 'OR' condition where '3' and '4' are in a relation 'OR' if we look at them from '2'. Thus the 'OR' condition is defined as; - 'A' and 'B' are in a 'OR' relation if they have been produced from same root node. In figure 1 root '2' produces '3' and '4' hence they are in 'OR' relation. The root node in both or all cases is related with its child with 'AND' relation. In figure 1 '2' is related with a 'AND' relation with '3' and '4'. Similarly '1' is related with '2' with 'AND'. Combining all these basic relations the whole graph can be represented as follows for the problem of our interest –

$$G_1 = \Big(1 \wedge \big(2 \wedge (3 \wedge (5)) \vee (4 \wedge (5))\big)\Big)$$

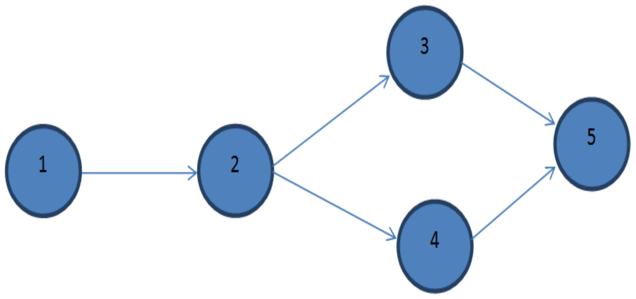

Figure 1. At node '2' we get two different choices to be traversed.

The generalised representation of a graph can be derived using the following recursive algorithm –

Boolean_Transform(S): S is the input node. holder[1:n] holds the existing connections of the current node. numberOfConnection holds total number of existing connections of the current node.

```
1.   Initialize holder[n] ← 0, numberOfConnection ← 0
2.   Do for i = 0 to n-1
3.        if G[S][i] = 1 then
4.   numberOfConnection ← numberOfConnection + 1
5.             holder[i] ← 1
6.        end if
7.   end for
8.   m ← numberOfConnection
9.   print " ( "
10.  print S + 1
11.  if numberOfConnection = 0 then
12.       print " ) "
13.       return
14.  end if
15.  print " ∧ "
16.  if numberOfConnection> 1 then
17.       print " ( "
18.       return
19.  end if
20.  Do for j = 0 to n-1
21.       if holder[j] = 1 then
22.            Boolean_Transform(j)
23.            if numberOfConnection> 1 then
24.                 print " ∨ "
26.  numberOfConnection ← numberOfConnection – 1
27.            end if
28.       end if
29.  end for
30.  if m > 1 then
31.       print " ) "
32.       return
33.  end if
34.  print " ) "
35.  return
36.  end
```

The algorithm takes $O(b^d)$ where the branching factor is b and d is the depth of the traversal. The Boolean transformation of the graph gives a Boolean expression where the variables take value either '0' or '1'. For a particular combination of '0's and '1's the evaluated value of the expression would be TRUE. As the expression is produced for a given source and destination, a 'TRUE' means selecting the nodes corresponding to the '1's would take us to the destination otherwise not. For instance the Boolean expression of figure 1 has four variables; of which each can take exactly two possible values thus there are all total 2^4 or 16 many possible combinations exist. If solution (1, 0, 1, 0, 1) is taken which is indicating a traversal through '1' to '3' to '5' and putting these solution set in 'G_1' we get a 'FALSE' thereby indicating an impossible path to reach '5' from '1' where as for solution set (1, 1, 0, 1, 1) the result comes to be 'TRUE' which indicates a possible path from '1' to '5'. Thus satisfying the Boolean expression is sufficient to find a possible simple path.

B. On the question of NP-completeness of the problem of finding out all possible Simple Paths between two nodes of a Graph –

Proof – The above algorithm transforms any instance I of the original problem (L) into Î which is an instance of the problem of finding the set of values for $X_1, X_2, \ldots X_n$ such that it satisfies the Boolean expression $G(X_1, X_2, \ldots X_n)$. The solution of the Boolean expression is clearly the *satisfiability* problem which is already a well-known NP – complete problem. The conversion of *satisfiability* to the problem of our interest can be done easily by traversing the Boolean expression in polynomial time. Hence *satisfiability* $\propto$ L that is a sufficient condition to show that the problem of our interest is in NP – hard. Now if it is possible to show that there exists a nondeterministic polynomial time algorithm to solve the L then it would be sufficient to show it is NP-complete. However it seems quite awkward to find such an algorithm that can conclude in polynomial time whether all possible paths have been found or not. To make it sure that we haven't left any other possible path, all possible combinations must be tested which would lead us to an algorithm of exponential complexity. Hence the only possible way to show the problem L is not in NP is to prove that the problem is undecidable. Undecidability of the problem would be sufficient to conclude that the problem is NP-hard but not NP-complete.

FindAllPath(Boolean Expression): Start is the first variable of the given Boolean Expression. Goal is the destination given. substring denotes part of original Boolean Expression.

```
1.   If Start = Goal then
2.        Print path
3.   Else
4.        Count substrings with AND relation
5.        If Count >= 1 then
```

```
6.              Do for each substring
7.                   FindAllPath (substring)
8.              End for
9.         End if
10.   End if
```

III. Result & Discussion

The algorithm Boolean_Transform recursively transforms the given graph into an equivalent Binary expression. The procedure followed by the algorithm is depicted in Figure 2 and Figure 3 which are depicting a typical graph and the search tree when it's fed to the Boolean_Transform. The search is initiated at node '1' then it traverses to '2' and '5'. As there is no other connection it returns to '2' and looks for other possibilities at '2'. Unavailability of other choices returns the control to '1' again. It looks for other choices available at '1' and finds '3' next; hence start traversing from '3'. Finally we come up with the tree depicted in Figure 3. The Binary expression generated is as follows –

$$G_2 = (1 \wedge ((2 \wedge (5)) \vee (3 \wedge (5)) \vee (4 \wedge (3 \wedge (5)))))$$

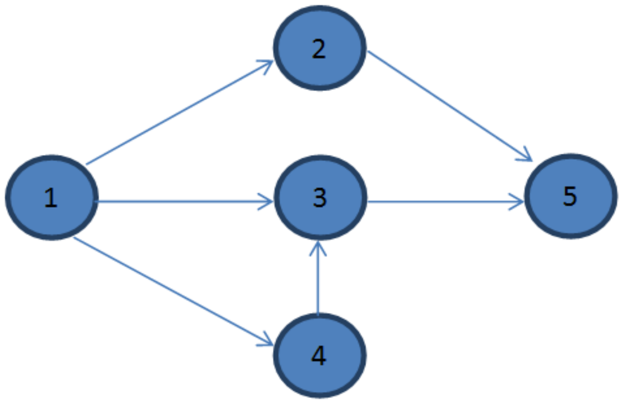

Figure 2: A typical graph used to test Boolean_Transform

FindAllPath algorithm takes a Boolean expression as input and it recursively operate on the given expression to find all paths possible to reach the given Goal or destination node. It is important to notice that the Boolean_Transform generates the equation for a particular node. The next algorithm to handle the Boolean expression is thus designed for a particular goal.

Figure 3 depicts the test results for the algorithm Boolean_Transform algorithm for a set of randomly selected graphs. It depicts the number of edges in the graph. Figure 4 shows the number of iterations required by corresponding problem instances to be solved by the algorithm Boolean_Transform. The analysis of Figure 3 and Figure 4 reveals that with increasing number of edges in a graph the iterations required by the algorithm increases though for same

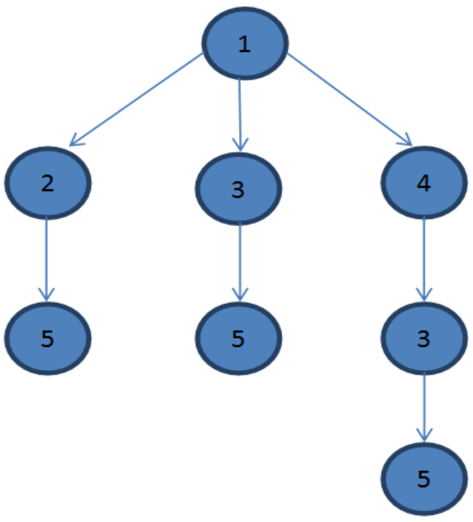

Figure 3: Tree generated after calling Boolean_Transform on the graph depicted in Figure 2

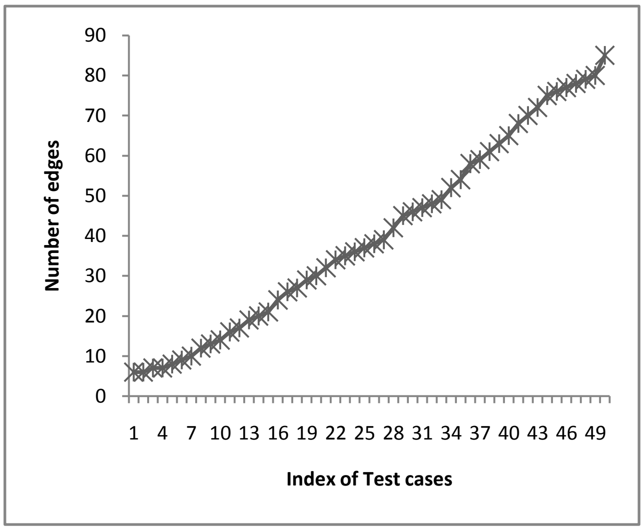

Figure 3:Number of edges in different test cases

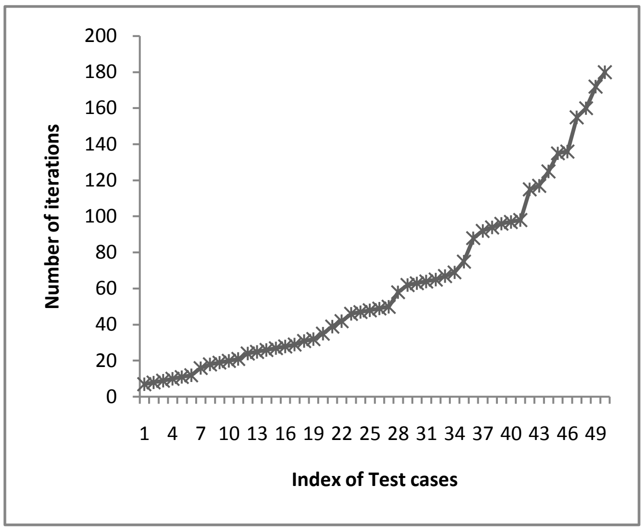

Figure 4: Iterations required in constructing Boolean Expressions in different test cases using Boolean_Transform.

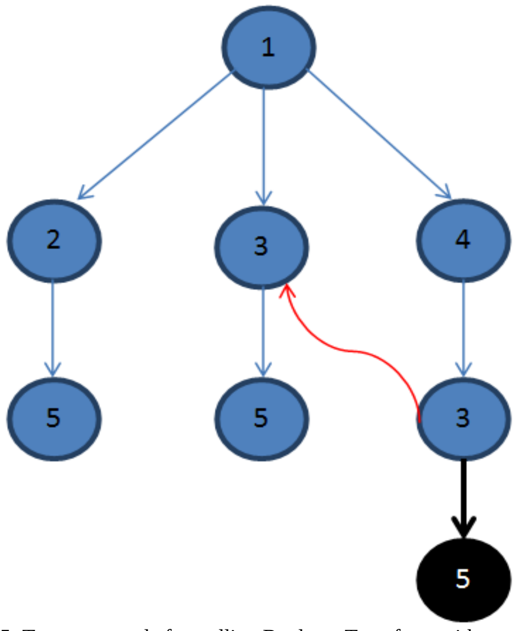

Figure 5: Tree generated after calling Boolean_Transform with pruning on the graph depicted in Figure 2.

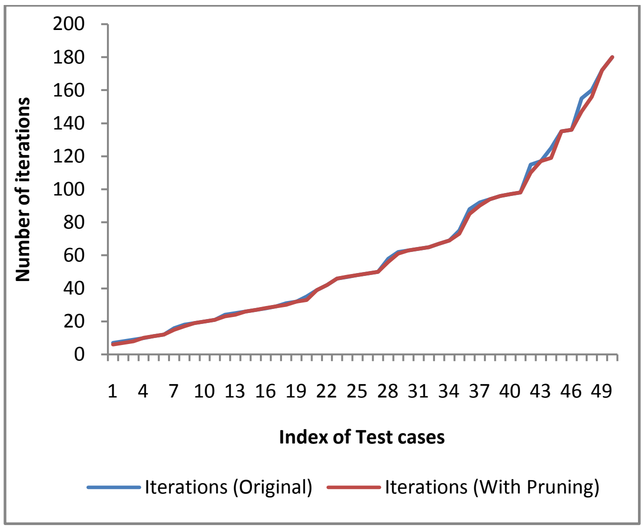

Figure 6: Iterations required in constructing Boolean Expressions in different test cases with original algorithm (Boolean_Transform) and the same with pruning.

number of edges it may take different number of iterations which is a clear indication of non-dependency of number of edges upon time complexity. Figure 4 reveals that the increasing number of edges increases the time complexity and reflects a nature of exponential complexity. A modification over Boolean_Transform could have been done by pruning some of the branches which are spawned during the generation of the tree (Figure 1). The scenario is described in Figure 5 which reveals how the generation of one node can be avoided using pruning technique. Though the implementation of such an algorithm would take very small modification of Boolean_Transform, but it would incur extra memory overload. To store the tree generated so far. Thus such algorithm is not a good choice as it takes almost same time complexity as that of the original one as depicted in Figure 6.

IV. Conclusion

The paper proposes a unique way to represent a graph in terms of Boolean variables and operators which is referenced by an algorithm to find all possible path between any two given node. The Boolean Transformation proposes a unique way to prove that the problem discussed is NP-hard. Further the conclusion has been made to show how it can be shown that the problem of finding all possible paths is NP-hard but not NP-complete. The present work will be immensely helpful for the researchers in near future.

V. References

[1] Leslie G. Valiant, The Complexity of Enumeration and Reliability Problems, SIAM J. Comput., 8(3), 410–421. (1977)

[2] Min-Sheng Lin, Sheng-Huang Su, Counting maximal independent sets in directed path graphs. Information Processing Letters 114, 568-572. (2014)

[3] Nadia Creignou, Miki Hermann, Complexity of Generalized Satisfiability Counting Problems information and computation 125, 112 (1996)

[4] D.C. Kozen, The design and analysis of algorithms, Counting problems and #P, Springer-Verlag, pp. 138–143. (1992)

[5] C. Wrathall, Complete sets and the polynomial-time hierarchy, Theoretical Computer Science, 3 (1) , pp. 23–33. (1976)

[6] Fotini Markopoulou, Simone Severini, A Note on Observables for Counting Trails and Paths in Graphs. Journal of Mathematical Modelling and Algorithms 8, 335-342. (2009)

[7] J. Scott Provan and Michael O. Ball, The Complexity of Counting Cuts and of Computing the Probability that a Graph is Connected, SIAM J. Comput., 12(4), 777–788. (12 pages) (1983)

[8] Tian-Ming Bua, ChenYuanb, PengZhanga, Computing on binary strings, Theoretical Computer Science. (In press)

[9] Yoshio Okamotoa, TakeakiUnob, Ryuhei Ueharac, Counting the number of independent sets in chordal graphs. Journal of Discrete Algorithms 6, 229-242. (2008)

[10] A.T Amin, K.T Siegrist, P.J Slater, The expected number of pairs of connected nodes: Pair-connected reliability. Mathematical and Computer Modelling 17, 1-11, (1993)

[11] H. Frank, I.T. Frisch, Analysis and design of survivable networks, IEEE Trans. on Communication Technology, COM-18 (5), pp. 501–519. (1970)

[12] Cook, S. A., The complexity of theorem-proving procedures. In Proceedings of the third annual ACM symposium on Theory of computing (pp. 151-158). ACM. (1971, May)

[13] Tovey, C. A., A simplified NP-complete satisfiability problem. Discrete Applied Mathematics, 8(1), 85-89. (1984)

[14] David S Johnson, The NP-completeness column: An ongoing guide. Journal of Algorithms 6, 145-159. (1985)

[15] Piotr Faliszewski , Lane A. Hemaspaandra, The consequences of eliminating NP solutions. Computer Science Review 2, 40-54. (2008)

[16] Bitner, J. R., &Reingold, E. M., Backtrack programming techniques. Communications of the ACM, 18(11), 651-656. (1975)

[17] Sint, L., & de Champeaux, D., An improved bidirectional heuristic search algorithm. Journal of the ACM (JACM), 24(2), 177-191. (1977)