Journal of Mathematical Extension
Vol. 15, No. 1, (2021), 41-60
ISSN: 1735-8299
URL: https://doi.org/10.30495/JME.2021.1443
Original Research Paper

Complexity and Approximability of the Marking Problem

M. Valizadeh
Iran Telecommunication Research Center

M. H. Tadayon*
Iran Telecommunication Research Center

Abstract. Let G be a digraph with positive edge weights as well as s and t be two vertices of G. The marking problem (MP) states how to mark some edges of G with T and F, where every path starting at source s will reach target t and the total weight of the marked edges is minimal. When traversing the digraph, T-marked edges should be followed while F-marked edges should not. The basic applications and properties of the marking problem have been investigated in [1]. This paper provides new contributions to the marking problem as follows: (i) the MP is NP-Complete even if the underlying digraph is an unweighted binary DAG; (*ii*) the MP cannot be approximated within $\alpha \log n$ in an unweighted DAG and even in an unweighted binary DAG with n vertices, where α is a constant. Moreover, a lower bound to the optimal solution of the MP is provided. We also study the complexity and challenges of the marking problem in program flow graphs.

AMS Subject Classification: 68R10; 68Q17; 68Q25; 68W25; 90C59
Keywords and Phrases: Marking problem, reachability, approximability, complexity, feasibility

1. Introduction

Let G be a weighted digraph and s and t be two vertices of G. The reachability assurance (RA) problem states how to label the edges of G where every path

Received: May 2019; Accepted: November 2019
*Corresponding author

starting at s finally reaches t, while the sum of the weights of the labeled edges, called the reachability cost, is minimal. Various practical problems can be modeled with the RA problem. For instance, in the context of graph-based test case generation, the main problem is to generate test cases in order to cover the vertices or edges of a given digraph [2].

The common solution to the RA problem is path finding, in which a path p is sought from s to t, and then every edge of p is labeled as T, which implies that those edges should be followed [3, 4]. The lower bound of the reachability cost of this solution is the shortest path weight. Although the path finding solution is efficient, it is generally not effective, especially when the size of G grows.

Reachability assurance is a serious challenge to software testing [2]. In order to test a computer program, we should provide some input to the program where every statement of the program is reached at least once. Let G be the flow graph of a given program and s be the start vertex of G. To extract such a test data using the path finding approach, we should find a path from s to each vertex of G, and then satisfy the labels (Boolean expressions) of all edges of the path. If G is a small digraph, this may not be difficult. However, if G is large enough, this goal can become very hard to achieve, which implies that it might not be solvable using the current SAT solvers [5].

We provided the MP approach to solve the reachability assurance problem effectively [1]. We revealed the reachability cost of the MP approach is far less than that of the path finding approach [1]. We demonstrated the fundamental properties of the MP and presented its application in software testing [1]. In this paper, we scrutinize the complexity and approximbility of the MP in general and restricted digraphs.

This article is structured as follows. Section 2 presents the required notation and terminology. Section 3 discusses the computational complexity of the marking problem in unweighted binary DAGs. Moreover, this section examines approximability of the marking problem in both general and restricted digraphs. Section 4 deals with the marking problem in program flow graphs. Finally, Section 5 concludes the research findings and proposes future work.

2. Preliminaries and Notation

Let $G = (V, E)$ be a digraph. The set of reachable vertices from vertex v is denoted as $reach(v)$. The set of outgoing edges of a vertex v is represented as $oe(v)$. The out-degree of a vertex v of G is denoted as $od(v)$ and the out-degree of G is the maximum out-degree of the vertices of G.

Digraph G is said to be a binary DAG if G has no cycle and the out-degree

of G is 2. A flow graph (FG) is a triple (V, E, s), where (V, E) is a digraph, $s \in V$ is the unique start vertex of the digraph and there is a path from s to each vertex of G [6]. If $G = (V, E)$ is a digraph and $v_i \in V$, then a flow graph can be formed with start vertex v_i by the removal of every vertex of G (and its adjacent edges) which is non-reachable from v_i. A complete flow graph is a quadruple $G = (V, E, s, f)$, where (V, E, s) is a flow graph with the unique start vertex s and $f \in V$ is the unique final vertex of G; there is a path from s to each vertex of G and there is also a path from each vertex of G to f [1]. The edge $e = (v_i, v_j)$ of the flow graph (V, E, s) is a back edge if every path from s to v_i goes through v_j. A flow graph is said to be reducible if the removal of its back edges leads to an acyclic digraph where each vertex can be reached from s.

Let $G = (V, E)$ be a digraph with positive edge weights and $v_i, v_j \in V$. The marking problem (MP) is how to assign marks T and F to some edges of G where every path starting at source v_i will reach target v_j. When the digraph is being traversed and a vertex v_k is visited, these marks find the following interpretations: (a) If some outgoing edges of v_k are marked with T, then we must pass through one of these edges; (b) If some outgoing edges of v_k are marked with F, then these edges must not be chosen; (c) If some outgoing edges of v_k are not marked, then any of these edges may be chosen. The optimization problem involves minimizing the total weight of the marked edges [1].

A solution to the marking problem $MP = (G, s, t)$ is a partial function from domain $E(G)$ to co-domain $\{T, F\}$. In a pure (non-labeled) digraph, marking an edge e with F means removal of the edge while marking e with T signifies the removal of every sibling edge of e. If G denotes the flow graph of a computer program, marking e with T/F is equivalent to making $TRUE/FALSE$ the label (logical expression) of e. Let $G = (V, E, s)$ be a flow graph, $MP = (G, s, t)$ be an instance of the marking problem, and v_i be an arbitrary vertex of G. In an optimal solution to the MP, we have [1] : (a) The outgoing edges of v_i cannot be marked with both marks T and F; (b) If v_i has only one outgoing edge, it cannot be marked with F; (c) At most one outgoing edge of v_i can be marked with F; (d) If G is unweighted, the MP has a solution using only T-mark.

The MP is a problem in reachability context. It is interesting to know whether such a reachability problem can be reduced to and solved by a corresponding problem in unreachability context, such as cut problem. This topic has been discussed in [7].

3. Complexity and Approximability of the Marking Problem

The decision versions of the marking and hitting set problems are presented in Tables $1-2$. We will use the hitting set problem to prove the complexity and approximability of the marking problem. The MP is an NP optimization problem [1].

Table 1. Decision version of the marking problem (MP).

Input: A digraph $G = (V, E)$ with positive edge weights plus vertices s and t of G and a real value w_1. **Parameter:** w_1 **Question:** Is it possible to mark some edges of G with $\{T, F\}$, where every path starting at s will reach t and the total weight of the marked edges is at most w_1?

Table 2. Decision version of the hitting set (HS) problem.

Input: A ground set $\{a_1, a_2, \cdots, a_m\}$, a collection of n subsets s_i of that ground set and an integer k_1. **Parameter:** k_1 **Question:** Is there any subset A of the ground set, such that $\|A\| \leq k_1$ and for each $i = 1, \cdots, n$, $s_i \cap A \neq \phi$?

Theorem 3.1. *If the underlying digraph is a weighted DAG, then the marking problem cannot be approximated within $\alpha \log n$ for some constant α where n is the number of vertices of the digraph.*

Proof. We reduce the hitting set problem to the marking problem as follows. Suppose $S = \{s_1, s_2, \cdots, s_n\}$ are the given sets and $\{a_1, a_2, \cdots, a_m\}$ is the union of all the sets. Given the number k_1, the hitting set problem states whether or not there exists a set A with k_1 or fewer elements such that every element of S (every set s_i where $i = 1, \cdots, n$) contains at least one element of A. The hitting set problem instance is denoted as $HS(S)$. The DAG G is constructed from the given set S as follows (Figure 1). Consider s as the start vertex of G. For each set s_i of HS where $i = 1, \cdots, n$, consider the corresponding vertex s_i and add an edge with an infinite weight from s to each s_i. Then, for each element a_j of the union of the input sets where $j = 1, \cdots, m$, consider the corresponding vertex a_j and add an edge with weight 1 from each s_i to any

a_j where $a_j \in s_i$ in HS. Furthermore, consider two final vertices called k and t and add two edges with weights $n * m$ from each a_j where $j = 1, \cdots, m$ to both final vertices. Finally, each vertex s_i where $i = 1, \cdots, n$ is connected to vertex k with weight 1. Clearly, G can be constructed in polynomial time. It is now shown that $HS(S)$ has a solution with k_1 or fewer elements if and only if $MP = (G, s, t)$ has a solution with the total weight $n + mnk_1$ or less of marked edges where $n = |S|$.

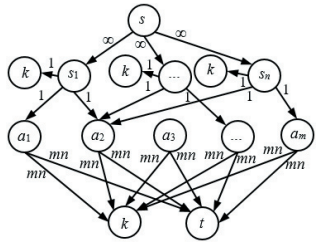

Figure 1. Digraph G of the marking problem corresponding to the hitting set problem $HS(S)$ where $n = |S|$ and m is the size of the union of all the given sets.

HS $\rightarrow$ MP.
Suppose A is a set with k_1 or fewer elements such that every element of S contains at least one element of A. It will now be indicated that $MP = (G, s, t)$ has a solution with a total weight $n + mnk_1$ or less of marked edges. For each element $a_j \in A$, edge (a_j, t) is marked with T. Furthermore, as A contains at least one element of any element of S, for each set $s_i \in S$, one and only one outgoing edge of s_i called (s_i, a_p) is marked with T where $a_p \in s_i \cap A$. As the weight of every outgoing edge of s is infinite in G, no outgoing edges of s are marked. Now when moving from vertex s of G, vertex s_i where $i = 1, \cdots, n$ is reached first. Since one outgoing edge of every element of S has been marked, by starting from s_i, vertex a_p where $a_p \in A$ will be reached. Finally, as the outgoing edge (a_p, t) of any element of A has been marked, t will be reached, with the sum of the weights of marked edges being at most $n + mnk_1$.

MP $\rightarrow$ HS.
Suppose function $f_1 : E_1 \rightarrow \{T, F\}$ is a solution to $MP = (G, s, t)$ where $E_1 \subset E(G)$. Any solution to the marking problem needs to mark one and only one outgoing edge of every s_i where $i = 1, \cdots, n$. At least one outgoing edge of each s_i must be marked because vertex s_i has a direct edge to vertex k which never reaches t. In addition, at most one outgoing edge of s_i must be marked as either edge (s_i, k) can be marked with F or edge (s_i, a_j) with T, where $a_j \in s_i$ in $HS(S)$ and the weights of both edges are the same. Furthermore, one and only one outgoing edge must be marked of some a_j's where $1 \leqslant j \leqslant m$. Indeed,

depending on which outgoing edge of any element of S is marked, the outgoing edges of the corresponding a_j's, but not all a_j's, must be marked. Again, at least one outgoing edge of each such a_j must be marked, since vertex a_j has a direct edge to vertex k which never reaches t. Also, at most one outgoing edge of each such a_j must be marked because either (a_j, k) can be marked with F or (a_j, t) with T and the weights of both edges are equal. Thus, the solution to MP has used total weight $n + mnk_1$ or less of marked edges where n is the number of marks used in the form of $((s_i, a_i) \rightarrow T$ or $(s_i, k) \rightarrow F)$ for all s_i's and k_1 is the number of marks used in the form of $((a_j, t) \rightarrow T$ or $(a_j, k) \rightarrow F)$ for some a_j's. So, the solution to MP can be considered to be $f_1 = f_2 \cup f_3$ such that $f_2 : (s_i, x) \rightarrow \{T, F\}$ where $(x = a_j$ or $x = k, i = 1, \cdots, n, 1 \leqslant j \leqslant m)$ and $f_3 : (a_j, y) \rightarrow \{T, F\}$ where $(y = t$ or $y = k, 1 \leqslant j \leqslant m)$. The tail set of the domain of function f_3, here called hit edges, can be denoted as A_1. It is claimed that A_1 with the size k_1 is a solution to $HS(S)$. Suppose that A_1 does not hit one of the elements of $S, e.g., s_i$. This means that in digraph G, no outgoing edges have been marked of some a_j's where $a_j \in s_i$ in $HS(S)$. Hence, the path $s.s_i.a_j.k$ in G starts at s but does not reach t, which is a contradiction as f_1 is a solution to the marking problem $MP = (G, s, t)$.

The hitting set problem cannot be approximated within $\alpha \log n$ for some constant α [8]. Accordingly, we can see that the marking problem cannot be approximated within $\alpha \log n$ for some constant α. Suppose that the size of the optimal solution of an instance of the hitting set problem is B. Then, the size of the optimal solution of the corresponding marking problem in the constructed digraph G (Figure 1) will be $mnB + n$. If we can find a marking in which the total number of all hit edges is B_1, then the marking has a weight $mnB_1 + n$.

Assume $\frac{mnB_1+n}{mnB+n} < \alpha_1 \log(n * m)$ for some constant α_1. Note that the size of the digraph G is $O(n * m)$. Then, we have $\frac{B_1}{B} < \frac{mnB_1+n}{mnB+n} + o(1) < \alpha_1 \log(n * m)$. For the hitting set problem with n sets and $m = poly(n)$ elements, it cannot be approximated within $\alpha \log n$ [8]. Since m is bounded by some polynomial in n, we can observe that $\frac{B_1}{B} < \alpha_1 \log(n * m) < \alpha_1 \alpha_2 \log(n)$, where α_2 is another constant. If we choose $\alpha_1 \leqslant \alpha/\alpha_2$, then $\frac{B_1}{B} \leqslant \log(n)$. Now we have a contradiction, implying that the marking problem cannot be approximated within $\alpha \log n$. □

Theorem 3.2. *The marking problem cannot be approximated within $\alpha \log n$ for some constant α even if the underlying digraph with n vertices is an unweighted DAG.*

Proof. Let G be a weighted DAG, s and t be two vertices of G, $MPX = (G, s, t)$ be the marking problem instance constructed from $HS(S)$ in Theorem 3.1. Every edge of G is unit-weighted except the outgoing edges of the start vertex

s as well as the outgoing edges of every vertex a_j where a_j is the ground set of HS $(j = 1, \cdots, m)$. Initially, we reduce the MPX to another instance MP of the marking problem as follows: Construct one square digraph M_0 with $(mn|E(G)| + 1) * (mn|E(G)| + 1)$ vertices where each vertex of any row of M_0 is connected to all vertices of its next row (Figure 2.a). Further, construct m square digraphs, M_j $(j = 1, \cdots, m)$, each one with $(mn-1)*(mn-1)$ vertices where each vertex of any row of M_j is connected to all vertices of its next row (Figure 2.b). Now, the unweighted DAG $G' = (V', E')$ is constructed from the weighted DAG G in two stages as follows: Stage I: Remove every outgoing edge of s in G. Then connect s to each vertex of the first row of M_0. Also connect each vertex of the last row of M_0 to every successor of s in G. Stage II: Remove every outgoing edge of every vertex a_j of G where a_j is the ground set of HS $(j = 1, \cdots, m)$. Then, for each a_j, connect the a_j to each vertex of the first row of M_j. Also connect each vertex of the last row of M_j to both vertices k and t of G (Figure 2.b).

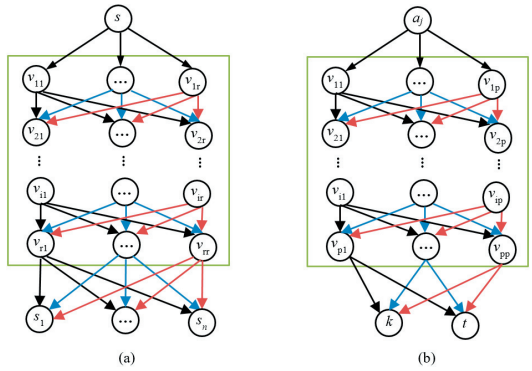

Figure 2. a) The square digraph M_0 with $r * r$ vertices where $r = (mn|E(G)|) + 1$. (b) Each square digraph M_j $(j = 1, \cdots, m)$ with $p * p$ vertices where $p = mn - 1$. Note that G is digraph of the marking problem corresponding to the hitting set problem $HS(S)$ where $n = |S|$ and m is the size of the union of all the given sets.

Note that the structure of DAG G' is similar to the structure of DAG G, except that the subgraph M_0 is added between the vertex s and its successors as well as each subgraph M_j $(j = 1, \cdots, m)$ is added between the vertex a_j and the final vertices k and t of G. Clearly, G' can be constructed in polynomial time and its size is $O(E(G)|) + O((mnE(G)|)^2) + m * O((mn-1)^2)) = O(m^4 * n^4)$. Now, it is demonstrated that MPX has a solution with a total weight $n + mnk_1$ or less of marked edges in the weighted digraph G if and only if MP has a solution

with the total weight $n + mnk_1$ or less of marked edges in the unit-weighted digraph G', implying a one-to-one reduction.

MPX $\rightarrow$ MP.
Let the function $f_1 : E_1 \rightarrow \{T, F\}$ be a solution to MPX in DAG G where $E_1 \subset E$ and the total weight of E_1 is $n + mnk_1$. If function f_1 is applied to G', then the reachability of t from s is assured in G' and total weight of the marked edges is $n + mnk_1$.

MP $\rightarrow$ MPX.
Let $s_1, \cdots, s_n$ be the successors of s in G. DAG G' has been constructed in a way that the reachability cost of any vertex s_i from s is at least $(mn|E(G)|)+1$, where $1 \leqslant i \leqslant n$. For instance, consider a path from s to s_i in G' and mark every edge of the path with T. As the length of p in G' is $(mn|E|+2)$, then the $(mn|E| + 2)$ edges are marked with T. Also consider the edge set connecting the last row of M_0 to s_i and mark them with T. In this case, the $(mn|E| + 1)$ edges are marked with T. So, the reachability cost of any s_i from s in G' is at least $(mn|E(G)|) + 1$, implying that it is impossible to reach an s_i from s in G' with a total weight $(mn|E(G)|)$ of marked edges $(i = 1, \cdots, n)$. Note that $(n + mnk_1) < mn|E(G)|$, because $|E(G)| > (m + n)$ and $k_1 < m$. So, no solution to the MP marks the edges of the subgraph M_0 in G'. In addition, each subgraph M_j $(j = 1, \cdots, m)$ in G' has been constructed in a way that the cost of reaching t (or k) from an a_j in G' is mn, which is equal to the cost of reaching t (or k) from an a_j in G.

Let the function $f_2 : E'_1 \rightarrow \{T, F\}$ be a solution to MP in DAG G', where $E'_1 \subset E'$. Further, let the total weight of E'_1 be $n + mnk_1$. If the function f_2 is applied to G, the reachability of t from s is assured in G and the total weight of the marked edges is $n + mnk_1$.

As the reduction of MPX to MP is one-to-one, by Theorem 3.1, it follows that Theorem 3.2 holds. Note that the size of digraph G in the problem MPX is $O(m * n)$, whereas the size of G' in the problem MP is $O(m^4 * n^4)$. However, in the proof of Theorem 3.1, the term $\alpha_1 \log(n*m)$ is just replaced with $\alpha_1 \log(n^4 * m^4)$ which equals $\alpha'_1 \log(n * m)$ where $\alpha'_1 = 16\alpha_1$. $\square$

Theorem 3.3. *The marking problem is NP-Complete and cannot be approximated within $\alpha \log n$ for some constant α even if the underlying digraph with n vertices is an unweighted binary DAG.*

Proof. In the unweighted DAG G' constructed in the proof of Theorem 3.2, the out-degree of the vertices s, s_i's $(i = 1, \cdots, n)$ and a_j's $(j = 1, \cdots, m)$ is larger than 2. In order to prove Theorem 3.3, we should construct a new unweighted DAG G' from the weighted DAG G in polynomial time and size such that out-degree of all vertices of G' is 2.

This is demonstrated by an example using $n = 3$ and $m = 5$. Figure 3.a displays the DAG G of $HS(S)$ with $n = 3$ and $m = 5$. First, the weight of every outgoing edge of each vertex s_i $(i = 1, \cdots n)$ in G is changed to the out-degree of s_i. So, the weight of every outgoing edge of s_1 in Figure 3.a will be 3.

Then, the outgoing edges of s in G are substituted with the binary DAG given in Figure 3.b. The reachability cost of any s_i from s in Figure 3.b is greater than $mn|E(G)|$, implying that no solution to the $MP = (G, s, t)$ marks the edges of Figure 3.b. So, replacement of the outgoing edges of s with the binary DAG presented in Figure 3.b will not change the solution to MP in G.

Thirdly, for each vertex s_i of G $(i = 1, \cdots, n)$, its outgoing edges are substituted with the binary DAG given in Figure 3.c. Let $i = 1$. In order to guarantee reaching t, we should reach an a_j from each s_i (in this case s_1) where $a_j \in s_i$ in $HS(S)$ and $j = 1, \cdots, m$. The binary DAG represented in Figure 3.c is constructed in a way that the reachability cost of any a_j from an s_i is the same and equals the weight of any outgoing edge of s_i in G. Figure 4.b illustrates how to convert outgoing edges of an s_i to binary mode when s_i has more outgoing edges, e.g., 4 outgoing edges including 3 vertices a_j's and 1 vertex k. For arbitrary values of n and m, only the height of the binary DAG given in Figure 4.b grows polynomially.

Finally, for each vertex a_j of G $(j = 1, \cdots, m)$, its outgoing edges are substituted with the binary DAG given in Figure 4.a. The binary DAG presented in Figure 4.a is constructed such that the reachability cost of the vertex t (or k) from each vertex a_j is the same and equals mn.

Thus, substituting the outgoing edges of s, s_i's and a_j's with the unit-weighted binary DAGs given in Figure 3.b, Figure 3.c (or Figure 4.b) and Figure 4.a does not change the cost of marking to reach t, implying that $(G, s, t) = (G', s, t)$. Hence, digraph G' can be considered instead of G in the proof of Theorem 3.2, so the theorem holds. Obviously, G' can be constructed in polynomial time whose size is $O(E(G)|) + O((mn|E(G)|)^2) + m * O((mn)^2)) + n * O(m^2) = O(m^4 * n^4)$. Also, the term $\alpha_1 \log(n * m)$ is replaced with $\alpha_1 \log(n^4 * m^4)$ which equals $\alpha'_1 \log(n * m)$ where $\alpha'_1 = 16\alpha_1$. □

Lemma 3.4. *Let G be an unweighted binary DAG and s and t be two vertices of G. The lower bound of the optimal solution to the marking problem (G, s, t) is $min - cut(s, k)$ where k is the representative of all vertices unreachable to t.*

Proof. Let K be the set of all vertices of G which are not reachable to t. We condense all vertices of K and their adjacent edges in a new vertex called k and call G' the new digraph. Consider any edge $e = (v_i, v_j)$ of G such that $v_i \notin K$ but $v_j \in K$. We replace the head of every such an edge e with the vertex k in G'. Now, we claim that $M = min - cut(s, k)$ in G' is a lower bound

to the optimal solution of the marking problem $MP = (G, s, t)$ in G. Since G is an unweighted binary DAG, the marking problem MP has an answer using only T-mark [1]. Suppose the number of the T-marked edges of the optimal solution to the MP is N where $N < M$. For each T-marked edge e, consider its sibling edge e'. We call E' the set of all edges e'. Marking all edges of E' with F (equivalently, the removal of all edges E' from G') assures the reachability of t from s or equivalently the unreachability of k from s where the size of E' is less than M, which is a contradiction, since M is the size of minimum cut. Note that min-cut (s, k) in G' is not an answer to the marking problem $MP = (G, s, t)$ in G as the min-cut (s, k) does not guarantee the reachability of t from s in general. □

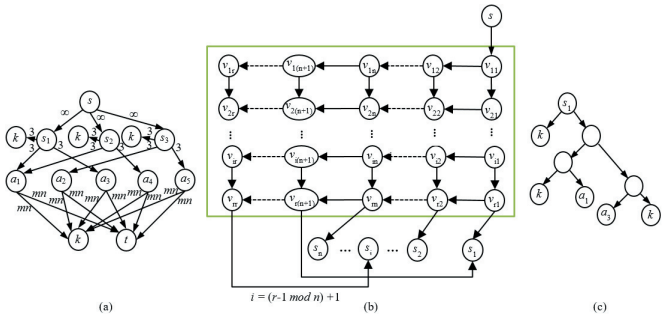

Figure 3. (a) Digraph G of a marking problem corresponding to a hitting set problem with $n = 3$ and $m = 5$; (b) The conversion of outgoing edges of s to binary mode where $r = (mn|E(G)|) + 1$; (c) The conversion of outgoing edges of s_1 to binary mode.

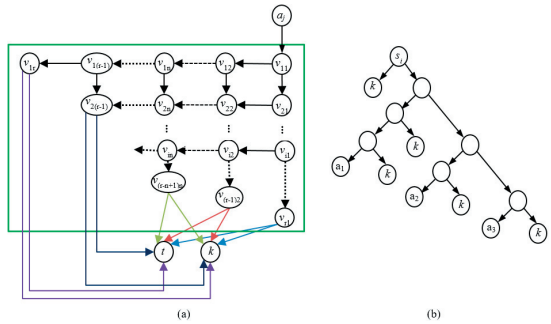

Figure 4. (a) Conversion of outgoing edges of each a_j $(j = 1, \cdots, m)$ to binary mode, where $r = mn$. (b) The conversion of outgoing edges of an s_i to binary mode where s_i has 4 outgoing edges including 3 vertices a_j's and 1 vertex k.

4. Marking Problem in Program Flow Graphs

4.1 Complexity of the marking problem in program flow graphs

There are two types of the reachability problem: syntactic reachability vs. semantic reachability [2]. The semantic reachability problem is studied in the context of program flow graphs. An edge of a program flow graph has a label indicating a Boolean expression. Unlike the pure flow graphs, we cannot mark any two arbitrary edges of a program flow graph with T/F as their labels (Boolean expressions) might be contradictory. The main complexity of the semantic reachability problem originates from the feasibility problem. An arbitrary path of a program flow graph is not necessarily feasible and it might be semantically infeasible. The feasibility problem was proven to be undecidable by Goldberg et al. [9] and DeMillo and Offutt [10]. Some partial solutions have been presented by Gallagher et al. [11], Goldberg et al. [9], R. Jasper et al. [12], and Offutt and Pan [13].

This section indicates that the reachability problem is NP-Complete even when the label of each edge of the underlying flow graph is a Boolean variable. The decision versions of the one-variable feasible path problem (1-VAR-FP) and 3-DNF-SAT problem are shown in Tables 3-4. We use the later problem to prove NP-Completeness of the 1-VAR-FP problem.

Table 3: 1-VAR-FP problem.

Input: A program flow graph $G = (V, E, s)$, where the label of each edge of G is a Boolean variable, and a vertex t of G. **Question:** Is there any feasible path from s to t in G?

Table 4: 3-DNF-SAT problem.

Input: A Boolean expression consisting of m clauses in 3-DNF, as disjunctive normal form, where any clause consists of three distinct literals. **Question:** Is there any Boolean assignment to all literals where the truth value of the Boolean expression becomes $FALSE$?

Theorem 4.1.1. *The 1-VAR-FP problem is NP-Complete.*

Proof. The 1-VAR-FP problem is NP; when given an answer P_1, we can verify in polynomial time whether P_1 is both a path and feasible. It is enough to verify that the sequence of the edges of the path P_1 do exist in G and there are no two edges in P_1 having the complement labels. Since the label of each edge of G is a Boolean variable, this verification can be performed linearly.

Now, we should show that 1-VAR-FP problem is NP-Hard. To do this, we reduce 3-DNF-SAT problem to 1-VAR-FP problem. The 3-DNF-SAT problem is the dual of 3-CNF-SAT problem (also known as 3-SAT), which is one of the 21 classic NP-complete problems proved by Karp in 1972 [14]. The 3-CNF-SAT problem states that determination of satisfiability of a given Boolean expression in 3-CNF is NP-complete. By duality, determination of unsatisfiability of a given Boolean expression in 3-DNF is NP-complete. We can convert the given Boolean expression $BE = c_1 \vee c_2 \vee \cdots \vee c_m$ where $c_i = x_{1i} \wedge x_{2i} \wedge x_{3i}$ $(1 \leqslant i \leqslant m)$ to an acyclic program flow graph by Algorithm 1

Algorithm 1. SAT2AcyclicFG (BE)

---

1. $G = (V, E, s), V = \{v_1, v_2\}, E = \phi$
2. **for each** $c_i = x_{1i} \wedge x_{2i} \wedge x_{3i}$ **do**
3. $\quad V = V \cup \{v_{1i}, v_{2i}, v_{3i}\}$
4. $\quad E = E \cup \{(v_{1i}, v_{2i}), (v_{2i}, v_{3i}), (v_{3i}, v_1)\}$
5. $\quad label(v_{1i}, v_{2i}) = x_{1i}, label(v_{2i}, v_{3i}) = x_{2i}, label(v_{3i}, v_1) = x_{3i}$
6. **end for**
7. **for** $i = 1$ to $(m - 1)$ **do**
8. $\quad E = E \cup \{(v_{1i}, v_{1(i+1)}), (v_{2i}, v_{1(i+1)}), (v_{3i}, v_{1(i+1)})\}$
9. $\quad label(v_{1i}, v_{1(i+1)}) = NOT(x_{1i}), label(v_{2i}, v_{1(i+1)}) = NOT(x_{2i}),$ $label(v_{3i}, v_{1(i+1)}) = NOT(x_{3i})$
10. **end for**
11. $E = E \cup \{(v_{1m}, v_2), (v_{2m}, v_2), (v_{3m}, v_2)\}$
12. $label(v1m, v2) = NOT(x1m), label(v2m, v2) = NOT(x2m), label(v3m, v2) =$ $NOT(x3m)$
13. $s = v_{11}$
14. **return** G

It is obvious that G (return value of Algorithm 1) is a labeled DAG and the label of each edge of G is a Boolean variable. It is also a flow graph, as each vertex of G is reachable from the start vertex $s = v_{11}$. The digraph G is a program flow graph as the label of each edge of G is a Boolean expression. Each clause c_i of the Boolean expression BE is equivalent to the ith-column of G. We intend to show it is NP-Hard to decide whether there exists a feasible path from $s = v_{11}$ to $t = v_2$ in G. According to the structure of the digraph G, we claim that BE equals $FALSE$ if and only if there exists a feasible path from s to t in G, implying that the 1-VAR-FP problem is NP-Hard. To prove the claim, the digraph G is constructed in a way that for reaching v_1, at least one of clauses of BE must be $TRUE$. The vertices v_1 and v_2 are the final vertices of G, implying that if we start moving from the start vertex $s = v_{11}$, we finally reach either v_1 or v_2, but not both of them.

Now, if BE equals $FALSE$, then none of the clauses of BE is $TRUE$, implying that if we start moving from the start vertex $s = v_{11}$, then we never reach v_1, so we definitely reach $t = v_2$. In contrast, if there exists a feasible path from $s = v_{11}$ to $t = v_2$ in G, then none of clauses of BE is $TRUE$ (otherwise, we reach v_1, so we never reach t, which is a contradiction). Thus, BE will equal $FALSE$. Obviously, an instance of a 3-DNF-SAT expression can be converted to an acyclic program flow graph in linear time. Indeed, if a 3-DNF-SAT problem has m clauses, its corresponding program flow graph will have $3m + 2$ vertices and $6m$ edges. □

Corollary 4.1.2. *The $1 - VAR - FP$ problem is NP-Complete, even if the underlying digraph is a binary DAG.*

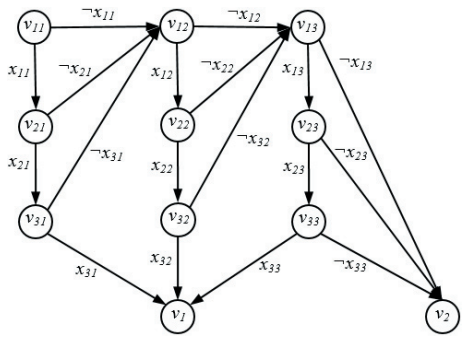

Figure 5. The acyclic program flow graph of the Boolean expression $(x_{11} \wedge x_{21} \wedge x_{31}) \vee (x_{12} \wedge x_{22} \wedge x_{32}) \vee (x_{13} \wedge x_{23} \wedge x_{33})$.

Proof. In the proof of Theorem 4.1.1, we have that the return value of Algorithm 1 is a DAG with an out-degree of 2. Hence, the corollary holds. □

Example 4.1.3. Figure 5 demonstrates the acyclic program flow graph of the Boolean expression $(x_{11} \wedge x_{21} \wedge x_{31}) \vee (x_{12} \wedge x_{22} \wedge x_{32}) \vee (x_{13} \wedge x_{23} \wedge x_{33})$. The symbol $\rightharpoonup$ denotes 'NOT'.

Corollary 4.1.4. *The marking problem is NP-Complete if the underlying program flow graph is an unweighted binary DAG.*

Proof. The corollary holds according to either Theorem 3.3 or Theorem 4.1.1. Interestingly, either minimality or feasibility of the marking problem makes it NP-Complete. □

4.2 A heuristic algorithm to the marking problem in program flow graph's

In order to solve the marking problem in program flow graphs, we can use the heuristic algorithm provided in [1]. However, a feasibility checking function should be added to this algorithm in order to consider only the feasible solutions.

Suppose that $G = (V, E, s)$ is an acyclic flow graph of a computer program and v_i and v_j are two vertices of G. Moreover, suppose that the label of any edge of G is a Boolean variable. We use the function $CFG(v_i, v_j)$ to construct a complete flow graph, H, with the start and final vertices v_i and v_j, respectively, where every path starting at v_i will finally reaches v_j, implying that the reachability of v_j from v_i is guaranteed (Algorithm 2). The digraph H is constructed as follows: (i) consider the induced sub-graph $G[H]$ such that $H = \{k \in V | k \in reach(v_i) \text{ and } v_j \in reach(k)\}$; (ii) remove (mark with F) every outgoing edge of H, namely every e of G where $tail(e) \in H$ and $head(e) \notin H$; (iii) instead of F-marking multiple sibling outgoing edges $e_1, e_2, \cdots$ of H, their another sibling edge e_k should be marked with T, if $head(e_k) \in H$ and the weight of e_k is less than the sum of the weights of F-marked edges. Since the weights of all outgoing edges of a vertex of program flow graphs are the same, we can use only the mark T [1].

Let E_1 and E_2 and E_3 be the subsets of $E(G)$, s and t be two vertices of $G, MP = (G, s, t)$ be an instance of the marking problem, $f_1 : E_1 \rightarrow \{T, F\}$ be an optimal solution to the MP, $f_2 : E_2 \rightarrow \{T, F\}$ be a heuristic solution to the MP and $SP : E_3 \rightarrow \{T\}$ be the T-marked edges of the shortest path from s to t in G. $|E_i|$ denotes the sum of the edge weights, where $E_i \subset E$. According to the definition of the marking problem, we have $|E_1| \leqslant |E_3|$, which means that the shortest path length is an upper bound to the solution of the marking

problem. Thus, the solution of a good heuristic algorithm to the MP would be better than $SP(s,t)$, which means that $|E_1| \leqslant |E_2| \leqslant |E_3|$.

Algorithm 2. $CFG(G, v_i, v_j)$

1. $H = \phi$
2. **for each** $v_k \in V(G)$ **do**
3. **if** ($v_k \in reach(v_i)$ and $v_j \in reach(v_k)$) **then** $H = H \cup \{v_k\}$
4. **end for**
5. $markedEdges = \phi$
6. **for each** $v_k \in H$ **do**
7. **if** exist $e \in outgoingEdges(v_k)$ in G such that $head(e) \notin H$ **then**
8. e' = a sibling edge of e such that $head(e) \in H$
9. $markedEdges = markedEdges \cup \{(e', T)\}$
10. **end if**
11. **end for**
12. **return** $markedEdges$

One trivial way to compute f_2 is to compute $SP(s,t)$ and then mark every edge of the shortest path with T. However, in order to compute f_2, we can consider marking a set of edges instead of marking the sequential edges of a path (the CFG function). In some cases, the value of the function $CFG(s,t)$ is a good initial value for the solution of the MP. However, the value of $CFG(s,t)$ may be greater than the value of $SP(s,t)$. Hence, the minimum value of $SP(s,t)$ and $CFG(s,t)$ should be considered as the initial value for the solution of the MP, namely, the one with the minimal total weight of the marked edges. Algorithm 3 improves this initial value using an iterative improvement technique [1]. As G is a program flow graph, we cannot mark an arbitrary edge set of G with T. So, we should check the feasibility of the return value of the SP and CFG functions. The function FIZ is used for this purpose (Algorithm 4). The provided algorithm can also be used for cyclic flow graphs which are reducible [1].

Lemma 4.2.1. *Algorithm 3 provides a heuristic solution to the marking problem in a given acyclic program flow graph.*

Proof. Obviously, the removal of the outgoing edges of t has no effect on the solution of the marking problem (line 1). In the initialization phase of Algorithm 3, for each pair (v_i, v_j) where $v_j \in reach(v_i)$, the minimum value of $SP(v_i, v_j)$ and $CFG(v_i, v_j)$ is considered as the initial value of the solution of the marking problem (G, v_i, v_j) [lines 3-8]. The function (v_i, v_j) is a T-marking of the shortest path from v_i to v_j in G. The function $CFG(v_i, v_j)$ uses the marking approach to guarantee reaching v_j from v_i in G. It considers the set of every vertex reachable from v_i and to v_j. This vertex set with all edges among

them creates the induced subgraph H. Now, in order to guarantee reaching t from s in G, it is enough to mark the outgoing edges of H with F, or to mark a sibling edge of the F-marked edges with T, if the weight of T-marked edge is less than the total weight of the F-marked edges. The function FIZ checks the feasibility of the answers of the SP and CFG functions before using them.

Algorithm 3. $MP\_DAG(G = (V, E), s, t)$.

---

1. $E = E - outgoingEdges(t)$
2. MP : a matrix $|V| * |V|$ such that each cell is a set of tuples (e, X), where $e \in E$ and $X \in \{T, F\}$
3. **for each** $v_i \in V$ **do**
4. **for each** $v_j \in reach(v_i)$ **do**
5. $SP(v_i, v_j) = \cup\{(e, T)\}_{e \in shortestPath(v_i, v_j)}$
6. $MP(v_i, v_j) = min(FIZ(SP(v_i, v_j)), FIZ(CFG(v_i, v_j)))$
7. **end for**
8. **end for**
9. $vertexList = reverseTopologicalSort(G)$
10. **for each** v_i in $vertexList$ **do** // t is the first element of the list
11. **for each** $v_j \in reach(v_i)$ **do**
12. **if** $(MP(v_i, v_j) \neq \infty$ and $MP(v_j, t) \neq \infty)$ **then**
13. $MP(v_i, t) = min(MP(v_i, t), FIZ(MP(v_i, v_j) \cup MP(v_j, t)))$
14. **end if**
15. **end for**
16. **end for**
17. **return** $MP(s, t)$

Algorithm 4. $FIZ(S)$. //S is a set of tuples (e, x), where $e \in E(G)$ and $x \in \{T, F\}$.

---

1. **for each** (e, x) in S **do**
2. **if** there exist a tuple (e', x') in S where $x' = NOT(x)$ **then**
3. **return** ∞
4. **end if**
5. **end for**
6. **return** $|S|$ //size of S

To find a better solution to the marking problem $MP = (G, s, t)$, an iterative improvement technique is used to guarantee reaching the intermediate vertex v from s (for each vertex v of V where $v \in reach(s)$ and $t \in reach(v)$) and then reaching the vertex t from v. To apply this technique, G is sorted reverse-topologically [line 9] where t and predecessors of t become the first vertices of G for the computation of the MP. For each vertex v_i in the reverse-topologically sorted vertex list of G, the formula provided in line 13 of Algorithm 3 is used to guarantee reaching the intermediate vertex v_j from v_i, for each vertex v_j between v_i and t, and then reaching t from v_j. As the order of v_j is less than the order of v_i in the sorted list, $MP(v_j, t)$ should be computed prior to $MP(v_i, t)$. The feasibility of the intermediate solutions is checked in lines 12-13. When execution of lines 10-16 is finished, $MP(v_i, t)$ is computed for the last vertex of the list, namely for $v_i = s$. Hence, $MP(s, t)$ is a heuristic solution to the given marking problem, as it guarantees reaching t from s. □

Complexity. The complexity of finding single-source shortest paths in the weighted DAG G is $\Theta(|V| + |E|)$ [15]. Thus, the complexity of finding all-pairs shortest paths is $\Theta((|V| + |E|) * |V|)$ which equals $\Theta(|V| * |E|)$ in a flow graph. Suppose that the function *shortestPath* stores the shortest path between every pair of vertices in a matrix $|V| * |V|$ for subsequent access. Moreover, suppose that the function *reach* has already been computed and stored in matrix $|V| * |V|$. Hence, the memory consumption of Algorithm 3 is $\Theta(|V|^2 + |E|)$. The complexity of Algorithm 2 for computing the function CFG is $\Theta(|V| + |E|)$. Further, the complexity of Algorithm 4 for computing the function FIZ is $\Theta(|E|)$. Hence, the complexity of lines 3-8 of Algorithm 3 is $\Theta(|V|^2 * (|V| + |E|))$, which equals $\Theta(|V|^2 * |E|)$ in a program flow graph. Moreover, the complexity of lines 10-16 of Algorithm 3 is $\Theta(|V|^2 * |E|)$ because of the union of two edge sets as well as the feasibility checking. The complexity of other parts of Algorithm 3 is linear or constant. Thus, total complexity of Algorithm 3 is $\Theta(|V|^2 * |E|)$. If G indicates the flow graph of a computer program, then usually $\Theta(|E|) = \Theta(|V|)$. Therefore, in this particular case, total complexity of Algorithm 3 is $\Theta(|V|^3)$.

Algorithm 3 is designed for the flow graphs with highly feasible solutions. In fact, we have assumed that there are limited number of edges having contradictory labels. For the flow graphs having frequently contradictory labels, Algorithm 3 may provide no answer. In this case, the algorithm should be modified.

We have evaluated the quality of the heuristic algorithm based on hundreds of program flow graphs generated by various graph generators. In most cases, the heuristic algorithm gives the optimal feasible solution to the marking problem. In order to compute the optimal solution, an exponential-time algorithm

was implemented considering different combinations of edges to be marked. The algorithms were implemented and tested in Java 1.7 with 1 GB of heap memory. The computer used for testing was an ASUS X554L laptop running Windows 8.1 equipped with an Intel Core i5-5200U CPU running at 2.20 GHz with 7.90 GB of usable main memory. The result of benchmarking Algorithm 3 on 3000 marking problem instances in 150 program flow graphs shows that the heuristic algorithm gives the optimal solution in 98.5% of the cases.

5. Conclusion and Future Work

We demonstrated that the marking problem is NP-complete in an arbitrary unweighted binary DAG. We also indicated that the marking problem cannot be approximated within $\alpha \log n$ in an unweighted binary DAG with n vertices. Then, we provided a lower bound to the optimal solution of the marking problem. Further, we investigated the complexity of the marking problem in program flow graphs. Given the results presented in this paper, new areas for further works are:

• Providing an algorithm for the marking problem in program flow graphs having highly infeasible paths.

• Studying the complexity and approximability of the marking problem in special flow graphs.

Acknowledgements
The authors would like to appreciate Mohsen Amini and Reza Sadraei for reviewing the Theorem 3.1.

Abbreviations
RA Reachability Assurance
MP Marking Problem
HS Hitting Set
DAG Directed Acyclic Graph
CFG Complete Flow Graph

References

[1] Valizadeh, M. and Tadayon, M.H. and Bagheri, M. "Marking problem : a new approach to reachability assurance in digraphs", Volume 25, Issue 3, pp. 1441-1455, 2018.

[2] Ammann, P. and Offutt, J. "Introduction to Software Testing", first ed., Cambridge University Press, pp. 29-33, 120, 2008.

[3] Sharma, P. and Khurana, N. "Study of optimal path finding techniques", Int. J. Adv. Technol., vol. 4, no. 2, 2013.

[4] Dellin, C. and Srinivasa, S. "A Unifying Formalism for Shortest Path Problems with Expensive Edge Evaluations via Lazy Best-First Search over Paths with Edge Selectors", ICAPS-2016, London, UK, 2016.

[5] Anand, S. and Burke, E. and Chen, T.Y. and Clark, J. and Cohen, M.B. and Grieskamp, W. and Harman, M. and Harrold, M.J. and McMinn, P. "An Orchestrated Survey on Automated Software Test Case Generation", Journal of Systems and Software, vol. 86, no. 8, pp. 1978-2001, 2013.

[6] Saito, N. and Nishizeki, T. "Graph Theory and Algorithms", 17th Symposium of Research Institute of Electrical Communication, Tohoku University, Sendai, Japan, p. 66, 1980.

[7] Valizadeh, M. and Tadayon, M.H. "Logical s-t Min-Cut Problem: An Extension to the s-t Min-Cut Problem", Accepted to Iranian Journal of Mathematical Sciences and Informatics (IJMSI), 2019.

[8] Feige, U. "A threshold of ln n for approximating set cover", J.ACM 45 314-318, 1998.

[9] Goldberg, A. and Wang, T.C. and Zimmerman, D. "Applications of feasible path analysis to program testing, In 1994 International Symposium on Software Testing and Analysis", pp. 80-94, Seattle, WA, 1994.

[10] DeMillo, R.A. and Offutt, J. "Constraint-based automatic test data generation", IEEE Transactions on Software Engineering, 17(9):900-910, 1991.

[11] Gallagher, L. and Offutt, J. and Cincotta, T. "Integration testing of object oriented components using finite state machines", Software Testing, Verification, and Reliability, 17(1):215-266, 2007.

[12] Jasper, R. and Brennan, M. and Williamson, K. and Currier, B . and Zimmerman, D. "Test data generation and feasible path analysis", In 1994 International Symposium on Software Testing and Analysis, pp. 95-107, Seattle, WA, 1994.

[13] Offutt, J. and Pan, J. "Detecting equivalent mutants and the feasible path problem", Software Testing, Verification, and Reliability, 7(3):165-192, 1997.

[14] Karp, R.M. "Reducibility Among Combinatorial Problems", In R. E. Miller and J. W. Thatcher (editors), Complexity of Computer Computations, New York: Plenum, pp. 85-103, 1972.

[15] Cormen, T.H, and Leiserson, C.E. and Rivest, R.L. and Stein, C. "Introduction to Algorithms", second edition, MIT Press and McGraw-Hill, ISBN 0-262-03293-7, p. 592, 2001.

Mohammad Valizadeh
Ph.D of Software Engineering
Department of Engineering
Iran Telecommunication Research Center
Tehran, Iran
E-mail: valizadeh80@gmail.com

Mohammad Hesam Tadayon
Associate Professor of Engineering
Department of Engineering
Iran Telecommunication Research Center
Tehran, Iran
E-mail: tadayon@itrc.ac.ir