Iranian Journal of Mathematical Sciences and Informatics
Vol. 17, No. 2 (2022), pp 253-271
DOI: 10.52547/ijmsi.17.2.253

Logical s-t Min-Cut Problem: An Extension to the Classic s-t Min-Cut Problem

Mohammad Valizadeh, Mohammad Hesam Tadayon*

Faculty of Information Technology, Iran Telecommunication Research Center (ITRC), P.O.Box:14155-3961, Tehran, Iran

E-mail:valizadeh80@gmail.com
E-mail:tadayon@itrc.ac.ir

Abstract. Let G be a weighted digraph, s and t be two vertices of G, and t is reachable from s. The logical s-t min-cut (LSTMC) problem states how t can be made unreachable from s by removal of some edges of G where (a) the sum of weights of the removed edges is minimum and (b) all outgoing edges of any vertex of G cannot be removed together. If we ignore the second constraint, called the logical removal, the LSTMC problem is transformed to the classic s-t min-cut problem. The logical removal constraint applies in situations where non-logical removal is either infeasible or undesired. Although the s-t min-cut problem is solvable in polynomial time by the max-flow min-cut theorem, this paper shows the LSTMC problem is NP-Hard, even if G is a DAG with an out-degree of two. Moreover, this paper shows that the LSTMC problem cannot be approximated within $\alpha logn$ in a DAG with n vertices for some constant α. The application of the LSTMC problem is also presented in test case generation of a computer program.

Keywords: Logical s-t min-cut, LSTMC, Complexity, Inapproximability, Flow graph, Test case generation.

2000 Mathematics subject classification: 68R10, 68Q17, 68Q25.

*Corresponding Author

Received 15 December 2018; Accepted 28 August 2019
©2022 Academic Center for Education, Culture and Research TMU

[Downloaded from ijmsi.ir on 2024-10-12]
[DOI: 10.52547/ijmsi.17.2.253]

1. Introduction

Given a weighted digraph $G = (V, E)$, the min-cut problem states how we can partition V into two nonempty sets S and T in order to minimize the total weights of the edges from S to T. The min-cut problem has a variant called the s-t min-cut, which requires that the two special vertices s and t be on opposite sides of the cut. The value of the min-cut in a digraph is equal to the minimum, taken over all pairs of vertices s and t, of the s-t min-cut [1]. The min-cut problem has lots of applications in various fields. A matching in a graph is a set of edges, no two of which share a vertex and a maximum matching is a matching of maximum cardinality. Matching problems occur in many practical applications such as scheduling. We can use the min-cut approach to find a maximum matching in an unweighted bipartite graph in polynomial time [2]. The problem of determining the connectivity of a network arises in the network reliability field [1]. Karger showed a connection between the min-cut and network reliability [3]. Picard and Querayne studied the applications of the min-cut problem in weighted graphs, including partitioning items in a database [4]. The s-t min-cut problem can be rephrased as follows. How a target vertex t can be made unreachable from a source vertex s in a digraph G where the sum of the weights of the removed edges is minimum. This definition of the s-t min-cut problem is more appropriate in the context of reachability. In some situations, in order to make the target vertex unreachable from the source vertex, we cannot remove *all* outgoing edges of any vertex of G together. If a removal of the edges of a digraph follows the latter constraint, we say the removal is logical. The notion of the logical removal is borrowed from the nature of the control flow graph of computer programs. The label of any edge of the control flow graph G of a program indicates a logical expression and the OR of the labels of all outgoing edges of any vertex of G is always True. In such digraphs, the removal of an edge e of G is equivalent to making False the label of e. Thus, we cannot remove all outgoing edges of any vertex of G together. Adding the logical removal constraint to the classic s-t min-cut problem, we call it the logical s-t min-cut or the LSTMC problem, in short.

Related Work. Minimum cut is one of the most basic problems in computer science and has various applications in different contexts [5, 6]. The LSTMC problem is a variant of the s-t min cut problem having the constraint of logical removal. The s-t min-cut problem is dual of the s-t max-flow problem. G. B. Dantzig is credited with development of the general max-flow problem in 1951 [7]. Ford and Fulkerson [8, 9] developed the first known algorithm in 1955. After that, new algorithms have been designed using more efficient methods to compute the max-flow. However, up to this time, no one has studied the max-flow/min-cut problem by considering the logical removal constraint.

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

Contribution. Let $G = (V, E)$ be a weighted DAG with n vertices, s and t be two vertices of G, and $LSTMC$ be an instance of the logical s-t min-cut problem. We show $LSTMC$ problem is NP-Hard. Moreover, we show $LSTMC$ problem is NP-Hard, even if G is a binary DAG. We demonstrate the $LSTMC$ problem cannot be approximated within $\alpha logn$ for some constant α. Let $OPTR$ be the following problem: how to remove some edges of G where all paths starting from s pass through t and the removal is both minimal and logical [10]. We show that both problems of $LSTMC$ and $OPTR$ are reducible to each other. Then, based on this reduction, we show an application of the LSTMC problem in test case generation of a computer program.

Organization. The remaining part of the paper is organized as follows. The next section presents the necessary concepts and notations. Section 3 shows basic properties of the LSTMC problem. Section 4 discusses the computational complexity of the LSTMC problem in both acyclic and binary digraphs. Section 5 studies the inapproximability of the LSTMC problem. Section 6 provides application of the LSTMC problem in test case generation of a computer program. Finally, Section 7 concludes the research findings and proposes future works.

2. Preliminaries and Notations

Let G be a digraph. We use $V(G)$ and $E(G)$ to denote the vertex and edge set of G, respectively. A *path* on a digraph is an alternating series of vertices and edges, beginning and ending with a vertex, in which each edge is incident with the vertex immediately preceding it and the vertex immediately following it. A *simple path* is a path in which all vertices are distinct. We denote the outgoing and incoming edges of a vertex v of G as $oe(v)$ and $ie(v)$, respectively. Let v_i and v_j be two vertices of G and $e = (v_i, v_j)$ be an edge of G. We refer to v_i and v_j as the *tail* and *head* of the edge e, respectively. Let G' be a subgraph of G and e be an edge of G. The edge e is called an *incoming edge* of G' if we have $tail(e) \notin V(G')$ and $head(e) \in V(G')$. In contrast, the edge e is called an *outgoing edge* of G' if we have $tail(e) \in V(G')$ and $head(e) \in V(G')$. Two distinct edges e_1 and e_2 of G are called *sibling* if the tail of the two edges is the same. The out-degree of G is the maximum out-degree of the vertices of G. A digraph G is called a *binary digraph* if the out-degree of G is two. A digraph G is called a *binary* DAG if it is acyclic and the out-degree of G is two. An induced subgraph H of G is a subset of the vertices of G together with any edges whose endpoints are both in this subset. If the vertex set of H is the subset S of $V(G)$, then H can be written as $G[S]$ and is said to be induced by S. Flow graph (FG) is a triple (V, E, s) where (V, E) is a digraph, $s \in V$ is the unique source vertex of the digraph, and there is a path from s to each vertex of G [11]. If $G = (V, E)$ is a digraph and $v_i \in V$, then we can construct

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

a flow graph with the source vertex V_i, by removal of any vertex of G (and its adjacent edges), which is not reachable from v_i. The function $FG(G, v_i)$ is used for this purpose. Hence, we have $FG(G, v_i) = (V', E', v_i) = G[V']$ such that $V' = \{v \in V | v \in reach(v_i)\}$. A *control flow graph* is the flow graph of a computer program that associates an edge with each possible branch in the program, and a node with sequences of statements [12]. We denote the logical s-t min-cut problem in the underlying digraph G as the triple (G, s, t). In this paper, by LSTMC, we refer to the generic logical s-t min-cut problem and by LSTMC, we refer to a specific (an instance of) logical s-t min-cut problem.

Symbol	Description
$LSTMC$	Logical s-t min-cut problem
$OPTR$	Optimal reach problem
FG	Flow graph
$FG(G, V_i)$	The function converting the digraph G to a flow graph with the source vertex v_i by removal of any vertex of G which is not reachable from v_i
CFG	Complete flow graph
HS	Hitting set problem
DFS	Depth-first-search traversal of a digraph starting from a given vertex
$ie(v)/oe(v)$	Incoming/Outgoing edges of the vertex v of a digraph
$ie(G'/oe(G')$	Incoming/Outgoing edges of the subgraph G' of a digraph

Table 1. Notations.

3. Properties of Logical s-t Min-Cut Problem

Proposition 3.1. *Let $G = (V, E)$ be a digraph, s and t be two vertices of G, and (G, s, t) be an instance of the $LSTMC$ problem. We have that (G, s, t) can be transformed to (G', s, t) where $G' = FG(G, s) - oe(t)$.*

Proof. It is clear from definition of the LSTMC problem. □

Note that the removal of all outgoing edges of t is not a logical removal. However, Proposition 3.1 states that the outgoing edges of t have no effect on the LSTMC problem. So, prior to computing the answer to the LSTMC

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

problem, we can remove them safely. Proposition 3.1 also states that all non-reachable vertices from s have no effect on the LSTMC problem. Since we can transform any logical s-t min-cut problem (G, v_i, v_j) in the digraph G to the corresponding logical s-t min-cut problem (G_1, s, v_j) in the flow graph G_1 with the source vertex $s = v_i$ such that $G_1 = FG(G, v_i)$, in the following, we usually consider a flow graph instead of a digraph in studying the LSTMC problem.

Proposition 3.2. *Let $G = (V, E, s)$ be a flow graph, t be a vertex of G, and (G, s, t) be an instance of the LSTMC problem. If there exists a vertex k in G such that t is not reachable from k, then the LSTMC problem has an answer.*

Proof. It is obvious that the LSTMC problem has an answer (not necessarily optimal) if and only if there exists a logical removal of edges of G such that the removal makes t unreachable from s. Now, we find a simple path p from s to k. By considering the path p as a subgraph of G, an answer to the LSTMC problem is to remove all outgoing edges of the subgraph p. Removing all outgoing edges of p, if we start moving from the vertex s in G, as we have only one path to move, we finally reach the vertex k, which never reaches t. As the edges of the path p are not removed, the removal is logical. Indeed, each removed edge has at least one un-removed sibling edge in p. □

Proposition 3.3. *Let the flow graph $G = (V, E, s)$ be acyclic, t be a vertex of G, and (G, s, t) be an instance of the LSTMC problem. The LSTMC problem has an answer if and only if there exists a vertex k in G such that t is not reachable from k.*

Proof. By Proposition 3.2, the if-part of the proposition holds. Now, suppose that t is reachable from every vertex of G. If we start moving from the vertex s, then, since G is acyclic, we finally reach the vertex t after passing through at most $|E|$ edges of G. Now, let $E_1 \subset E$ be an arbitrary logical removal of the edges of G. If we start moving from the vertex s in the digraph $G - E_1$, then, since the removal is logical, the vertex s has at least one un-removed outgoing edge called e_1. Thus, by passing through e_1, we can exit from s and reach a vertex v_1 of G. As the removal is logical, the vertex v_1 has at least one un-removed outgoing edge such that we can use it to exit from v_1 and reach a vertex v_2 of G. If we repeat this scenario, we finally reach the vertex t, which is the final vertex of G. Hence, by any logical removal of the edges of G, we cannot make t unreachable from s, implying the LSTMC problem has no answer. □

Note that the acyclicity condition cannot be dropped in this proposition as the simple example in 1 shows.

Definition 3.4 (Complete Flow Graph)**.** A quadruple $G = (V, E, s, f)$ where (V, E) is a digraph, $s \in V$ is the unique source vertex of G, $f \in V$ is the unique

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

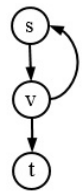

FIGURE 1. The vertex t is reachable from every vertex of the flow graph G. However, since G is cyclic, we can make t unreachable from s by removal of the edge (v, t), which is also a logical removal. Note that the sibling edge (v, s) of the edge (v, t) is not removed.

final vertex of G, there is a path from the source vertex to each vertex of G, and there is a path from each vertex of G to the final vertex.

Corollary 3.5. *Let $G = (V, E, s, t)$ be an acyclic complete flow graph with the source and final vertices s and t, respectively. We have the LSTMC problem (G, s, t) has no answer, implying it is infeasible to make t unreachable from s by a logical removal.*

Proof. By Definition 3.4, we have t is reachable from every vertex of G. So, by Proposition 3.3, we have the LSTMC has no answer. □

4. Computational Complexity of Logical s-t Min-Cut Problem

This section demonstrates the LSTMC problem in a weighted digraph is NP-Hard, even if the digraph is a binary DAG. The decision problems of the LSTMC and Hitting Set are shown in the following tables. We use the latter problem to prove the NP-Completeness of the LSTMC problem.

Input: A weighted digraph $G = (V, E)$, two vertices s and t of G, and a real value w_1.
Question: Can t be made unreachable from s by removal of some edges of G such that the removal is logical and sum of weights of the removed edges is at most w_1?

TABLE 2. Decision problem of logical s-t min-cut.

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

Input: A ground set $\{a_1, a_2, \ldots, a_m\}$, a collection of n subsets s_i of that ground set, and an integer k_1.
Question: Does there exist the subset A of the ground set such that $\lvert A\rvert \leq k_1$ and for each $i = 1 \ldots n$, we have $s_i \cap A \neq \emptyset$?

TABLE 3. Decision problem of hitting set (HS).

Algorithm $HS2LSTMC(s_1, s_2, \ldots, s_n)$

```
G' = (V', E'), V' = {s, t, k}, E' = φ
  for each element a_j of the union of the input sets do
    V' = V' ⋃ {a_j}
    E' = E' ⋃ {(a_j, k), (a_j, t)}
    w(a_j, k) = 1, w(a_j, t) = 1 // w(e) indicates the weight of the edge e.
  end for
  for each s_i do
    V' = V' ⋃ {s_i}
    E' = E' ⋃ {(s, s_i)}, w(s, s_i) = ∞
    for each element a_j of s_i do
      E' = E' ⋃ {(s_i, a_j)}, w(s_i, a_j) = 0
    end for
  end for
return G'
```

/*Reduction of the hitting set (HS) problem to the LSTMC problem*/

Theorem 4.1. *The LSTMC problem is NP-Complete even if the underlying digraph is acyclic.*

Proof. Let $G = (V, E)$ be a weighted DAG, s and t be two vertices of G, and (G, s, t) be an instance of the LSTMC problem. First, we show the LSTMC problem is NP. We can verify a given answer to the decision problem of the LSTMC in polynomial time as follows. Let $E_1 \subset E$ be a given answer to be verified. We remove every edge of E_1 from G and call G' the new digraph. If each removed edge e of G has at least one un-removed sibling edge and the vertex t is unreachable from s in G', then E_1 is an answer, otherwise, it is not an answer. It is obvious that this verification can be performed in polynomial time. Also, we can check the sum of the weights of elements of E_1 in polynomial time. Now, we should show the LSTMC is NP-Hard. We reduce the hitting-set problem to the LSTMC problem. The hitting-set is a classic NP-complete problem proved by Karp in 1972 [13]. Let $S = \{s_1, s_2, \ldots, s_n\}$ be the given sets and $\{a_1, a_2, \ldots, a_m\}$ be the union of all the sets. Given the number k_1, the decision problem of the hitting set problem states whether

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

there exists a set A with k_1 elements such that every element of S (every set s_i where $i = 1 \dots n$) contains at least one element of A. We denote the hitting set problem as $HS(S)$. We construct the weighted DAG G' from the set S by Algorithm $HS2LSTMC$ (see Figure 2 (a) as an example). This algorithm considers s as the source vertex of the DAG G'. For each set s_i of HS where $i = 1 \dots n$, the algorithm considers the new vertex s_i and adds an edge with infinite weight from s to each s_i. Then, for each element a_j of the union of the input sets where $j = 1 \dots m$, the algorithm considers the new vertex a_j and adds an edge with zero weight from each s_i to any a_j where $a_j \in s_i$ in HS. Finally, the algorithm considers two final vertices called t and k, and add two edges from each a_j where $j = 1 \dots m$ to the both final vertices. It is clear that G' can be constructed in polynomial time.

Now, we demonstrate that $HS(S)$ has an answer with k_1 elements if and only if the LSTMC problem (G', s, t) has an answer with some logically removed edges such that the sum of the weights of the removed edges is k_1.

HS $\rightarrow$ LSTMC. Let A be a set with k_1 elements such that each element of S (each set $s_i \in S$ where $i = 1 \dots n$) contains at least one element of A. We intend to show (G', s, t) has an answer with some logically removed edges such that the sum of weights of the removed edges is k_1. For each element $a_j \in A$ where $1 \leq j \leq m$, we remove the outgoing edge (a_j, t) of a_j from G'. This removal is logical, as the edge (a_j, t) has the un-removed sibling edge (a_j, k) in G'. As the weight of any edge (a_j, t) is 1, the cost of this removal is k_1. Furthermore, for each set $s_i \in S$ in $HS(S)$ where $i = 1 \dots n$, we remove all outgoing edges of the vertex s_i from G' except those edges, which are in the form of (s_i, a_p) where $a_p \in s_i \bigcap A$. This removal is also logical, as A contains at least one element of any element of S (any s_i) which is not removed from G'. Since the weight of any edge (s_i, a_p) of G' is zero, the cost of the latter removal is zero. Now, if we start moving from s in G', first we reach a vertex s_i where $1 \leq i \leq n$. Then, moving from s_i, we reach a vertex a_p where $a_p \in A$ $(1 \leq p \leq m)$, as we have not removed any edge (s_i, a_p) of G' such that $a_p \in A$. Finally, as we have removed the outgoing edge (a_p, t) of any a_p where $a_p \in A$, we reach the vertex k, which never reaches t, implying t becomes unreachable from s. Note that the sum of weights of the total removed edges is k_1. Also, note that k is a vertex of the digraph G' whereas k_1 is the number of the elements of the set A.

LSTMC $\rightarrow$ HS. Let the edge set $E_1 \subset E(G')$ with the total weight k_1 be an answer to (G', s, t), implying the removal of E_1 from G' makes t unreachable from s and the removal is logical. The answer to the LSTMC problem cannot remove any outgoing edge of s, as the weight of any outgoing edge of s is infinite. The answer to the LSTMC problem may remove some outgoing edges of some s_i's such that $1 \leq i \leq n$. However, since the weight of every outgoing

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

edge of any s_i is zero, the cost of this removal is zero, too. As the removal is logical, at least one outgoing edge of every vertex s_i of G' $(i = 1 \dots n)$ is not removed. We call A_1 the heads of the un-removed outgoing edges of every vertex s_i of G' $(i = 1 \dots n)$. Hence, there are some paths from s to some a_j's in $G'(1 \leq j \leq m)$ such that $a_j \in A_1$. The answer to the LSTMC problem definitely removes the outgoing edge (a_j, t) of any vertex a_j of A_1 $(1 \leq j \leq m)$, otherwise, we have that t is reachable from s in G', which is a contradiction. Let $E_2 \subseteq E_1$ be those removed edges of G', which are in the form of (a_j, t), namely the removed edges of G' with the head t. Note that A_1 is the tail set of E_2. Since the total weight of E_1 is k_1, we have that the total weight of E_2 is also k_1, because the weight of any removed outgoing edge of any s_i $(1 \leq i \leq n)$ is zero. So, the size (the number of elements) of A_1 is k_1. Now, we claim that A_1 with the size k_1 is an answer to $HS(S)$. Suppose that A_1 does not hit one of the elements of S such as s_l $(1 \leq l \leq n)$, implying that A_1 contains no elements of s_l in $HS(S)$. It means that in the digraph G', we have not removed the outgoing edge (a_p, t) of any a_p $(1 \leq p \leq m)$ where $a_p \in s_l$ in $HS(S)$. Since the removal is logical, at least one outgoing edge of s_l is not removed in G'. Moreover, since A_1 contains no elements of s_l in $HS(S)$, the head of any outgoing edge of the vertex s_l in G' does not belong to A_1. Hence, there exists at least one un-removed edge from s_l to an a_p in G' $(1 \leq p \leq m)$ such that the outgoing edge (a_p, t) of a_p is not removed in G'. Thus, there exists the path $s.s_l.a_p.t$ in G', implying t is reachable from s in G', which is a contradiction. □

Example 4.2. Let $S = \{s_1, s_2, s_3\}$ such that $s_1 = \{1, 2, 3\}$, $s_2 = \{1, 4\}$, and $s_3 = \{2, 5\}$. Figure 2 (a) shows the weighted DAG G' of the LSTMC problem corresponding to the hitting set problem $HS(S)$. In this example, the set A, namely the union of all elements of S, is $A = \{1, 2, 3, 4, 5\}$. We have $|S| = 3$ and $|A| = 5$. This example shows how an arbitrary instance of the hitting set problem can be solved by the help of a specific instance of the logical s-t min-cut problem in a weighted DAG. If we compute an answer to (G', s, t) and consider those removed edges of the answer which are in the form of (a_j, t) called $E_1(1 \leq j \leq m)$, then the tail set of E_1 is an answer to $HS(S)$. An answer to the LSTMC problem is the edge set $E_1 = \{(s_1, 2), (s_1, 3), (s_2, 4), (s_3, 5), (1, t), (2, t)\}$. So, the tail set of the subset $E_2 = \{(1, t), (2, t)\}$ of the edge set E_1, namely the set $\{1, 2\}$, is an answer to the $HS(S)$.

Theorem 4.3. *The LSTMC problem is NP-Complete even if the underlying digraph is a binary DAG.*

Proof. Let the weighted digraph $G = (V, E)$ be a binary DAG, s and t be two vertices of G, and (G, s, t) be an instance of the LSTMC problem. By Theorem 4.1, we have the LSTMC problem is NP-Complete. This theorem intends

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

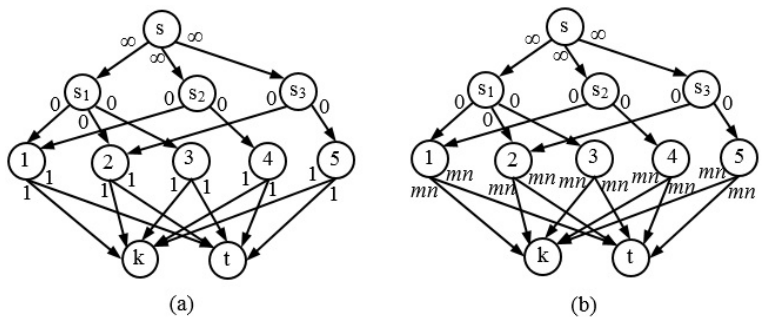

FIGURE 2. (a) The weighted DAG G' of the LSTMC problem corresponding to the hitting set problem $HS(S)$ such that $S = \{s_1, s_2, s_3\}$, $s_1 = \{1, 2, 3\}$, $s_2 = \{1, 4\}$, $s_3 = \{2, 5\}$, $n = |S| = 3$, $m = |A| = 5$, and A is the union of all elements of S. (b) The modified instance of G', called G'', such that the weight of all outgoing edges of every vertex a_j of A is $m * n$ where $1 \leq j \leq m$.

to show that the out-degree of a DAG has no effect on the computational complexity of the logical s-t min-cut problem. The proof is the same as the proof of Theorem 4.1, except that in the reduction of the $HS(S)$ to the LSTMC, we consider a new binary DAG, G'', instead of the DAG G'. In the DAG G' constructed by Algorithm $HS2LSTMC$, the out-degree of s and any s_i where $i = 1 \ldots n$ ($n = |S|$) is generally more than 2. However, the out-degree of any a_j ($j = 1 \ldots m$) is 2 where m is the number of elements of the union of all elements of S. Therefore, to transform G' to a binary DAG, we should change the structure of the outgoing edges of s and s_i's ($i = 1 \ldots n$). We transform the DAG G' to the binary DAG G'' in polynomial time as follows. We demonstrate the idea on an example with $n = 4$ and $m = 5$. The idea is easily extendable to arbitrary values of n and m. The Figure 3 (a) shows the DAG G' of an instance of $HS(S)$ with $n = 4$ and $m = 5$.

We replace the outgoing edges of s with the binary DAG given in Figure 3 (b). In Figure 3 (b), as the weight of any edge of the binary DAG is infinite, none of these edges are removed. Hence, replacing the outgoing edges of s with the binary DAG given in Figure 3 (b) does not alter the answer to the LSTMC problem. Also, for each vertex s_i ($i = 1 \ldots n$) with the out-degree greater than two, we replace the outgoing edges of s_i with a binary DAG similar to the one given in Figure 3 (c). In Figure 3 (a), as the out-degree of only the vertex s_1 is greater than two, the conversion is just performed for s_1. Note that two edges $(s_1, a_1), (y_2, a_1)$ in Figure 3 (c) are the representative of the edge (s_1, a_1) in Figure 3 (a). Also, the edges $(y_1, a_2), (y_2, a_3)$ in Figure 3 (c) are the representatives of the edges $(s_1, a_2), (s_1, a_3)$ in Figure 3 (a), respectively. Now, we claim

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

that replacing the outgoing edges of s_1 with the binary DAG given in Figure 3 (c) does not alter the answer to the LSTMC problem. To prove this claim, we should show that the removal of any subset of the outgoing edge set of s_1 in Figure 3 (a) is also feasible with the same cost in Figure 3 (c). As a rule of thumb, the removal of multiple outgoing edges $\{e_1, e_2 \ldots\}$ of an s_i $(1 \leq i \leq n)$ with the out-degree greater than 2 in Figure 3 (a) can be performed by the removal of the representatives of $\{e_1, e_2, \ldots\}$ in Figure 3 (c), unless the removal of the representatives is not logical. In this special case, instead of the removal of two outgoing edges of a vertex v in Figure 3 (c), we remove the incoming edge of v. Hence, we have that the removal of the edge (s_1, a_1) in Figure 3 (a) is equivalent to the removal of two edges (s_1, a_1) and (y_2, a_1) in Figure 3 c. The removal of (s_1, a_2) or (s_1, a_3) in Figure 3 (a) equals the removal of (y_1, a_2) or (y_2, a_3) in Figure 3 (c), respectively. The removal of $\{(s_1, a_1), (s_1, a_2)\}$ in Figure 3 (a) equals the removal of $\{(s_1, a_1), (y_2, a_1), (y_1, a_2)$ in Figure 3 (c). The removal of $\{(s_1, a_2), (s_1, a_3)\}$ in Figure 3 (a) equals the removal of $\{(y_1, a_2), (y_2, a_3)\}$ in Figure 3 (c). The removal of $\{(s_1, a_1), (s_1, a_3)\}$ in Figure 3 (a) equals the removal of $\{(s_1, a_1), (y_1, y_2)\}$ in Figure 3 (c). In the latter removal, we have considered the removal of (y_1, y_2) instead of $\{(y_2, a_3), (y_2, a_1)\}$, because the removal should be logical, implying we cannot remove every outgoing edge of the vertex y_2 in Figure 3 (c).

Hence, replacing the outgoing edges of s and any s_i with the binary DAG's given in Figure 3 (b)-(c) does not alter the answer to the LSTMC problem, implying we have $(G', s, t) = (G'', s, t)$. Therefore, we can consider G'' instead of G' in the proof of Theorem 4.1 and the theorem holds. This proof was provided for $n = 4$ and $m = 5$. However, for any values of n and m, only the height of the binary DAG's given in Figure 3 (b)-(c) is increased polynomially. So, the idea naturally generalizes to different values of n and m. □

Proposition 4.4. *The LSTMC problem is NP-complete even if the underlying digraph is a binary DAG and all weights are non-zero and finite.*

Proof. We should show that in Theorem 4.3, the zero and infinite weights on the edges of the digraph are not restrictive. The zero and infinite weights on the edges of the digraph can be substituted with the natural numbers as follows. Let $N = (n*m)+1$ where $n = |S|$ and m is the number of elements of the union of all elements of S in the $HS(S)$. We can replace the weight 0 of the outgoing edges of s_i's $(1 \leq i \leq n)$ with 1, the weight 1 of the outgoing edges of a_j's $(1 \leq j \leq m)$ with N, and the weight ∞ of the outgoing edges of s with N^2. We can remove at most $(n*m)$ outgoing edges of all s_i's, even in the binary mode. Since the new weight of the outgoing edges of any s_i is 1, the total cost of the removal is $(n*m)$, which is always less than N (the new weight of one outgoing edge of any a_j). It implies that any answer to the LSTMC problem does not try to remove some outgoing edges of a_j's instead of that of s_i's, otherwise, it

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

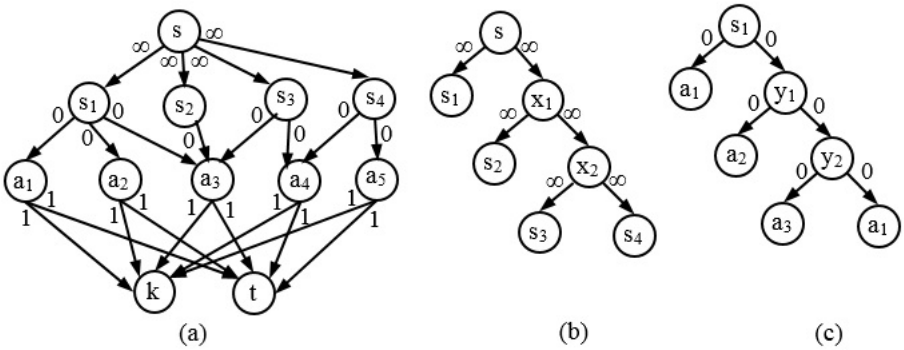

FIGURE 3. (a) The digraph G' of the LSTMC problem corresponding to an instance of the hitting set problem $HS(S)$ with $n = 4$ and $m = 5$. (b) The conversion of the outgoing edges of s to binary mode; (c) The conversion of the outgoing edges of s_1 to binary mode.

will not be a minimum removal, which is a contradiction. Moreover, we can remove at most m outgoing edges of all a_j's, because the removal should be logical. Since the new weight of the outgoing edges of any a_j is N, the total cost of the removal is $(m * N)$, which is always less than N^2, implying any answer to the LSTMC problem does not try to remove some outgoing edges of s instead of that of a_j's. □

Proposition 4.5. *The LSTMC problem is NP-complete in a binary DAG even if the weights of any two sibling edges are the same.*

Proof. In the DAG G' of Theorem 4.1 and its corresponding binary DAG (refer to Figure 3), we have that the weights of any two sibling edges are the same. Hence, the proposition holds. □

5. Inapproximability of Logical s-t Min-Cut Problem

Theorem 5.1. *If the underlying digraph with n vertices is a weighted DAG, then the LSTMC problem cannot be approximated within αlogn for some constant α.*

Proof. We modify the constructed DAG G' from the set S by the Algorithm *HS2LSTMC* in Theorem 4.1 as follows: change the weight of all outgoing edges of every vertex a_j of A to $m * n$ where A is the union of all elements of $S, 1 \leq j \leq m$, $n = |S|, m = |A|$. The modified digraph is called G'' (See Figure 2 (b) as an example). In this case, we can easily see that $HS(S)$ has an answer with k_1 elements if and only if (G'', s, t) has an answer with some logically removed edges where the sum of the weights of the removed edges is mnk_1. The reduction of HS to LSTMC and vice-versa is exactly the same as the

[Downloaded from ijmsi.ir on 2024-10-12]
[DOI: 10.52547/ijmsi.17.2.253]

reduction provided in Theorem 4.1. We call the *hit edges*, the tail set of the edge set (a_j, k) of an answer to the LSTMC problem in G''.

As hitting set problem cannot be approximated within $\alpha logn$ for some constant α [14], we can show that the LSTMC problem cannot be approximated within $\alpha logn$ for some constant α. Suppose size of the optimal solution of an instance of the hitting set problem is B, then size of the optimal solution of the corresponding LSTMC problem in the constructed digraph G'' (Figure 2 (b)) will be mnB. If we can find a logical s-t cut in which the total number of all hit edges is B_1, then the logical s-t cut has a weight mnB_1. Assume $\frac{mnB_1}{B} < \alpha_1 log(n * m)$ for some constant α_1. Note that size of the digraph G'' is $O(n * m)$. Then, we have that $\frac{B_1}{B} < \alpha_1 log(n * m)$. For the hitting set problem with n sets and $m = poly(n)$ elements, it cannot be approximated within $\alpha logn$ [14]. Since m is bounded by some polynomial in n, we can see 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 $\alpha_1 \leq \frac{\alpha}{\alpha_2}$, then $\frac{B_1}{B} \leq \alpha log(n)$. Now, we have a contradiction, which means that the LSTMC problem cannot be approximated within $\alpha logn$ for some constant α. □

6. Application of Logical s-t Min-Cut Problem

Suppose that $G = (V, E, s)$ is the control flow graph of a computer program with the source vertex s. The vertices of the control flow graph G indicate the processing statements of the program and the edges of G indicate the conditional or iteration statements. Let G be a binary DAG. Note that the control flow graph of a computer program can be transformed to a semantically equivalent binary flow graph. The main problem is to generate a set of test cases for the program such that each statement of the program is reached by at least one test case. In the software testing terminology, this method is called the node coverage [12]. The common approach to generate such test cases is to find a path p from s to each vertex t of G and then to satisfy (make True) the label (Boolean expression) of every edge of p. It results in n Boolean equations where n is the length of the path p. By solving these equations, a test case for reaching the vertex t is obtained. When the length of the shortest path from s to t is increased, the number of the equations is increased, too. Thus, it becomes more complex to satisfy all Boolean equations. Now, we propose an alternative and better approach to reach any vertex of G.

In order to reach a vertex t from s in G, we can make True the labels of a set of edges (and not necessarily a sequence of the edges of a path from s to t). Figure 4 (a) shows the acyclic control flow graph G of a computer program with the out-degree of two.

The length of the shortest path from the source vertex $s = v_1$ to the target vertex $t = v_{12}$ in G is 4. So, to reach t from s in G by the common approach of the shortest path, we need to make True the labels of four edges. However,

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

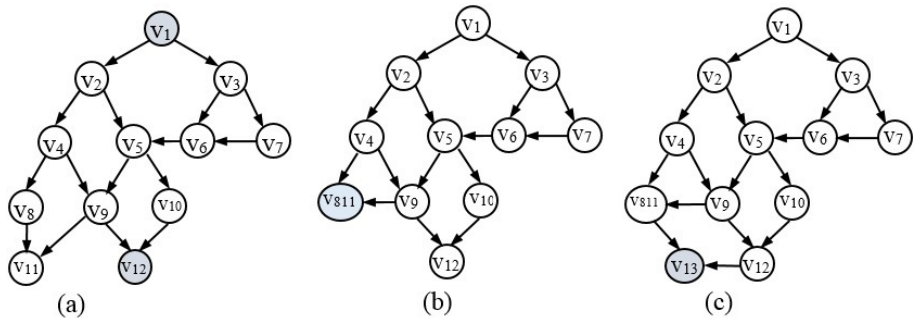

FIGURE 4. (a) The acyclic control flow graph G of a computer program with the out-degree of two. The goal is to guarantee to reach the target vertex, $t = v_{12}$, from the source vertex, $s = v_1$, in G. The important vertices of the digraph are shaded. The vertex t is not reachable from any vertex of the set $K = \{v_8, v_{11}\}$. (b) Using Algorithm *Condense*, the induced subgraph $G[K]$ of G is substituted with the new vertex, $k = v_{811}$, and the head of any edge with the head h where $h \in K$ is changed to k. The resulting digraph is called G'. (c) The conversion of G' to a complete flow graph with the final vertex $f = v_{13}$. The new vertex, v_{13}, and the two edges, $(k, f), (t, f)$, are added to G' and the resulting graph is called G''.

we can guarantee to reach t from s by making True the labels of only two edges (v_4, v_9) and (v_9, v_{12}), because making True the labels of the edges (v_4, v_9) and (v_9, v_{12}) is equivalent to removal of (making False the labels of) their sibling edges namely (v_4, v_8) and (v_9, v_{11}). Note that the Boolean expressions of any two sibling edges in the control flow graph of a computer program are complement of each other. Now, in Figure 4 (a), you can observe that, by removal of the edges (v_4, v_8) and (v_9, v_{11}) from G, any path starting from the source vertex v_1 finally reaches the target vertex v_{12}. Let A be the problem to guarantee to reach the target vertex t from the source vertex s of the binary DAG G by making True the labels of the minimum number of edges. Moreover, let $E_1 \subset E$ be an answer to the problem A, implying that it is guaranteed to reach t from s by making True the label of every edge of E_1 and there is no answer smaller than the size of E_1. If a vertex of G has only one outgoing edge, then according to the semantic of the program, the label of that outgoing edge is always True. As E_1 is a minimum answer, the set E_1 has no such always-true edges, implying that any edge of E_1 has a distinct sibling edge in G. As the out-degree of G is 2, that sibling edge is unique. Hence, instead of making True the label of an edge, e_1, of E_1, we can make False the label of the unique sibling edge e_2 of e_1. It means that the problem A has an answer by making False

[Downloaded from ijmsi.ir on 2024-10-12]
[DOI: 10.52547/ijmsi.17.2.253]

Input: A weighted digraph $G = (V, E)$ and the two vertices s and t of G.
Question: How can we remove some edges of G such that, by following *any* path starting from s, it is guaranteed to reach t and the removal is both minimal and logical?

TABLE 4. Optimal reach problem (OPTR), denoted by the triple (G, s, t).

the labels of a set of edges of G. As G indicates the flow graph of a computer program, making False the label of an edge e of G is equivalent to the removal of e from G. Also, according to the semantic of a computer program, we cannot remove (make False the labels of) all outgoing edges of a vertex of G together, implying that the removal should be logical. Therefore, the problem A can be rephrased as the following problem called $OPTR$ (Optimal Reach) [10].

Note that both problems of A and $OPTR$ are equivalent. In Figure 4 (a), as mentioned above, the answer to the problem A is to make True the labels of the edges (v_4, v_9) and (v_9, v_{12}) of G. Hence, the answer to the problem $OPTR$ is to remove (make False the labels of) the edges (v_4, v_8) and (v_9, v_{11}) of G.

Up to this point, the test case generation problem has been transformed to the problem $OPTR$ by using the idea of minimum logical removal. However, the goal of the problem $OPTR$ is to guarantee to reach the target vertex t from the source vertex s in G, which is the opposite of making t unreachable from s (the LSTMC problem). Lemma 6.1 demonstrates that the problem $OPTR$ can be reduced to an LSTMC problem. This application shows a case in which, non-logical removal of the edges of a digraph is *infeasible*.

Lemma 6.1. *Let $G = (V, E, s)$ be an acyclic weighted flow graph and t be a vertex of G. Also, let $OPTR = (G, s, t)$ be an instance of the optimal reach problem. Moreover, let $K \subset V$ be the set of the vertices of G such that t is reachable from no vertices of K. Furthermore, let Algorithm Condense(G, t) condense all vertices of K as well as their adjacent edges in one vertex called k and return the new flow graph G'. Finally, let $LSTMC = (G', s, k)$ be an instance of the LSTMC problem. We have that any answer to the LSTMC problem (G', s, k) is an answer to the $OPTR$ problem (G, s, t) and vice-versa.*

Proof. Algorithm *Condense* substitutes the induced sub-graph $G[K]$ of G with a new vertex named k and considers the incoming edges of $G[K]$ as the incoming edges of k and calls G' the new flow graph (See Figure 4 (b) as an example). For each vertex v of G, if there exists only one incoming edge of K with the tail v in G, then, the corresponding edge $e' = (tail(v), k)$ is added to G' and the weight of e' is considered same as the weight of e. Otherwise, if there

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

exists multiple incoming edges of K with the same tail v in G, then, although only one corresponding edge $e' = (tail(v), k)$ is added to G', the weight of e' is considered as the sum of weights of all incoming edges of K with the tail v (lines 5-13). Hence, the cost of removal of any subset of the incoming edges of K is the same in the both digraphs G and G'. As the outgoing edges of t have no effect on the reachability of t from s, we also remove every outgoing edge of t in G' (line 3). Now, we claim that any answer to the LSTMC problem (G', s, k) is an answer to the $OPTR$ problem (G, s, t) and vice-versa. It means that in order to guarantee to reach the target vertex t from the source vertex s in the DAG G it is enough to make k unreachable from s in G' by the approach of the minimum logical removal such that $G' = Condense(G, t)$.

The latter claim can be demonstrated as follows. We add a new vertex called f to G' and two edges from k and t to f. We call G'' the new flow graph (See Figure 4 (c) as an example). Since f is reachable from any vertex of the flow graph G'', by Definition 3.4, we have that G'' is a complete flow graph with the source and final vertices s and f, respectively. Thus, by Corollary 3.5, we have that it is impossible to make f unreachable from s in G'' by a logical removal of the edges of G''. On the other hand, as the edges (t, f) and (k, f) have no siblings, they cannot be logically removed. So, by performing any logical removal in G'', we will definitely reach either k or t. Hence, if we make k unreachable from s in G' by a logical removal of the edges of G', then t will be reachable from s and since t becomes the final vertex of G', it is guaranteed to reach t by moving any path starting from s in G'. Conversely, if we guarantee to reach t from s in G' by a logical removal of the edges of G', then, as t has no paths to k, the vertex k becomes unreachable from s. Note that the digraph G' is the same as G except that the vertex k in G' is the replacement of the induced sub-graph $G[K]$ in G. Therefore, to make k unreachable from s in G' by a logical removal of the edges of G', we should guarantee to reach t from s in G' (or G) and vice-versa. It is obvious that G' is computable in polynomial time in the size of G. □

Remark 6.2. In Lemma 6.1, let $E_1 = E(G[K])$. As the edges of E_1 have no effect on the reachability of t from s in G, we have that an answer to the $OPTR$ problem (G, s, t) contains no edge of E_1. That is why we removed $E(G[K])$ in G' and substituted $V(G[K])$ with one vertex called k. Moreover, suppose that $E'_1 \subset E(G')$ is an answer to the LSTMC problem (G', s, k) in G'. If $e'_1 \in E'_1$ be an edge with the head k in G', then the corresponding edges of e'_1 in G are the edges of G with the tail $tail(e'_1)$ and the head h where h is a vertex of K.

Figure 4 (b) shows the corresponding flow graph, $G' = Condense(G, t)$, of the acyclic flow graph G of a program such that $s = v_1$ and $t = v_{12}$. As the outgoing edges of t have no effect on the reachability of t from s in G, they have already been removed from the figure. The set $K = \{v_8, v_{11}\}$ is

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

the set of vertices of G such that t is reachable from no vertices of K. Using Algorithm *Condense*, the induced sub-graph $G[K]$ of G is substituted with the new vertex, $k = v_{811}$ and the head of any edge with the head h where $h \in K$ is changed to k. By Lemma 6.1, an answer to the LSTMC problem (G', s, k) by a logical removal of the edges of G' is also an answer to the $OPTR$ problem (G, s, t). An answer to the LSTMC problem (G', s, k) is to remove the edge set $E'_1 = \{(v_4, v_{811}), (v_9, v_{811})\}$. Hence, in order to guarantee to reach the target vertex $t = v_{12}$ from the source vertex $s = v_1$ in G' by a logical removal, it is enough to remove the edge set E'_1. The corresponding edges of E'_1 in G are the edge set $E_1 = \{(v_4, v_8), (v_9, v_{11})\}$. Therefore, an answer to the $OPTR$ problem (G, s, t) is to remove the edge set E_1 from G.

Algorithm *Condense* (G, t) // $G = (V, E, s)$

1. $K = \{v \in V | t \notin reach(v)\}$
2. $EK = \{e \in E | \exists\, v_1, v_2 \in K$ where $head(e) = v_1$ and $tail(e) = v_2)\}$
3. $G' = (V', E', s), V' = V - K, E' = E - EK - oe(t)$
4. $V' = V' \bigcup k$
5. **for each** incoming edge e of the induced subgraph $G[K]$ **do**
6. $e' = (tail(e), k)$
7. **if** $(e' \notin E')$ **then**
8. $E' = E' \bigcup \{e'\}$
9. $w(e') = w(e)$ // $w(e)$ indicates the weight of e.
10. **else**
11. $w(e') = w(e') + w(e)$
12. **end if**
13. **end for**
14. **return** G'

/*Let $G = (V, E, s)$ be an acyclic weighted flow graph and t be a vertex of G. Let $K \subset V$ be the set of vertices of G such that t is reachable from no vertices of K. This algorithm condenses all vertices of K as well as their adjacent edges in one vertex called k and returns the new flow graph G'. Also, the outgoing edges of t are removed in G'.*/

Remark 6.3. In Lemma 6.1, we demonstrated that the $OPTR$ problem is reducible to the LSTMC problem. By a similar proof, we can show that the LSTMC problem is reducible to the OPTR problem, too. The proof stems from this fact that the roles of the vertices of t and k can be transformed to each other. In other words, in order to guarantee to reach the target vertex t from the source vertex s in the underlying DAG G by a minimum logical removal, we should guarantee not to reach the vertex k in the digraph $G' = Condense(G, t)$. In contrast, to guarantee not to reach t from s in G

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

by a minimum logical removal, we should guarantee to reach the vertex k in $G' = Condense(G, t)$.

Conjecture 6.4. The LSTMC problem is NP-Complete and it cannot be approximated within $\alpha logn$ even if the underlying digraph is an unweighted binary DAG.

This section showed why the LSTMC approach is superior to the current approaches, such as shortest path, in the context of program test case generation. As previously mentioned, in order to generate a test case to reach the target vertex in the flow graph of Figure 4, we need to satisfy 4 edges in the common approach of the shortest path but 2 edges in the $OPTR$ (or, equivalently LSTMC) approach. It implies that in order to find the test case, we have to solve 4 Boolean equations in the shortest path approach but 2 equations in the LSTMC approach. When the size of the underlying digraph becomes bigger, the shortest path approach needs to solve more Boolean equations, implying it becomes more difficult to find an input to satisfy all Boolean expressions. In this state, the LSTMC approach can be used to find a test case easily comparing to the shortest path approach.

7. Conclusion and Future Works

The logical removal constraint applies in situations where non-logical removal is either infeasible or undesired. We introduced the Logical s-t Min-Cut (LSTMC) problem as a cut problem having both constraints of the minimal and logical removal. We presented the basic properties as well as the application of the LSTMC problem. We showed why the LSTMC approach is superior to the current approaches in the context of program test case generation. Although the s-t min-cut problem is solvable polynomially in any digraph, we showed that the LSTMC problem is NP-Hard, even if the underlying digraph is acyclic with an out-degree of two. Moreover, we showed that the LSTMC problem cannot be approximated within $\alpha logn$ in a DAG with n vertices for some constant α. Given the results presented in this research paper, new areas for further works are identified, including:

- To show whether or not the LSTMC problem is NP-Hard in unweighted DAG's, and especially in unweighted binary DAG's.
- Assuming the LSTMC problem is NP-Hard in unweighted DAG's, is it also inapproximable? What about unweighted binary DAG's?
- To provide a necessary and sufficient condition for verifying the existence of an answer to the LSTMC problem in *cyclic* flow graphs.

Acknowledgements

The authors would like to thank Reza Sadraei and Mohsen Amini for the review of Theorem 4.1 and Theorem 5.1, respectively.

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]

References

1. D. R. Karger, Clifford Stein, An $o(n^2)$ Algorithm for Minimum Cuts, *Theory of Computing, Proc.*, 25th ACM Symposium on, (1993), 757–765.
2. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithms, Second ed., *MIT Press and McGraw-Hill*, (2001), ISBN 0-262-03293-7, 665–592.
3. D. Karger, Global Min-Cuts in RNC, and other Ramifications of a Simple Min-Cut Algorithm, *Proceedings of the* 4th *ACM-SIAM Symposium on Discrete Algorithms*, (1993).
4. J. C. Picard, M. Querayne, Selected Applications of Minimum Cuts in Networks, *INFOR.*, **20**, (1982), 394–422.
5. E.Lawler, *Combinatorial Optimization: Networks and Matroids*, Dover Publications, 2001.
6. V. V. Vazirani, *Approximation Algorithms*, Springer, 2004.
7. G. B. Dantzig, Application of the Simplex Method to a Transportation Problem. In T.C. Koopman(ed.): *Activity Analysis and Production and Allocation, New York*, (1951), 359–373.
8. L. R. Ford, Jr., D. R. Fulkerson, Maximal Flow through a Network. *Canad. J. Math*, (1956), 399–404.
9. L. R. Ford, Jr., D. R. Fulkerson, *Flows in Networks*, Princeton University Press, 1962.
10. M. Valizadeh, M. H. Tadayon, M. Bagheri, Marking Problem: a New Approach to Reachability Assurance in Digraphs, *Scientia Iranica*, **25**(3), (2018), 1441–1455.
11. Nobuji Saito, Takao Nishizeki, Graph Theory and Algorithms, *17th Symposium of Research Institute of Electrical Communication, Tohoku University, Sendai, Japan*, October 24-25, (1980), pp. 66.
12. P. Ammann, J. Offutt, *Introduction to Software Testing, First ed.*, Cambridge University Press, 2008, pp. 52, 33.
13. R. M. Karp, Reducibility Among Combinatorial Problems, *In R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum*, (1972), pp. 85–103.
14. U. Feige, A Threshold of ln n for Approximating Set Cover, *J.ACM*, **45**, (1998), 314–318.

[Downloaded from ijmsi.ir on 2024-10-12]

[DOI: 10.52547/ijmsi.17.2.253]