Iranian Journal of Operations Research
Vol. 4, No. 2, 2013, pp. 108-126

On Search for all d-MCs in a Network Flow

M. Forghani-elahabad[1], N. Mahdavi-Amiri[*, 2]

A number of problems in several areas such as power transmission and distribution, communication and transportation can be formulated as a stochastic-flow network (SFN). The system reliability of an SFN can be computed in terms of all the upper boundary points, called d-MinCuts (d-MCs). Several algorithms have been proposed to find all the d-MCs in an SFN. Here, some recent studies in the literature on search for all d-MCs are investigated. We show that some existing results and the corresponding algorithms are incorrect. Then, correct versions of the results are established. By modifying an incorrect algorithm, we also propose an improved algorithm. In addition, complexity results on a number of studies are shown to be erroneous and correct counts are provided. Finally, we present comparative numerical results in the sense of performance profile of Dolan and Moré showing the proposed algorithm to be more efficient than some existing algorithms.

***Keywords*:** *Reliability, Stochastic-flow network, Upper boundary points, Minimal cut (MC).*

Manuscript was received on 20/08/2012, revised on 25/03/2013 and accepted for publication on 22/04/2013.

1. Introduction

Network reliability theory has been applied extensively in various real-world systems such as computer and communication [1, 17], power transmission and distribution [5, 19], transportation [9, 24]. Reliability evaluation approaches exploit a variety of tools for system modeling and calculation of reliability indices such as bounding methods [7, 16, 21], Monte-Carlo method [5, 6, 27], and factoring method [14]. Clancy et al. [5] utilized both the Monte-Carlo simulation and the state space decomposition method. In [6], Cook and Ramirez-Marquez employed the Monte-Carlo estimation to approximate the probability that the system capacity level is more than a demand level *d*. Crespo et al. [7] developed a method to compute upper bounds and utilized the upper bounds to obtain accurate estimates of failure probabilities. Doulliez and Jalnoulle [9] applied the Ford-Fulkerson flow augmenting method [10] and gave an iterative state-space decomposition algorithm to decompose the state space. Jane et al. [16] first proposed an exact decomposition approach to calculate system reliability and then modified it to present a practical bounding algorithm for obtainment of lower and upper bounds in computing the two-terminal reliability in large networks.

Among most popular tools, some network-based algorithms are proposed in terms of upper boundary points, called *d*-MinCuts (*d-MC*s); see [10–13, 15, 18, 22, 25, 26, 28–32]. Once all the *d-MC*s are determined, system reliability can be calculated by some exact methods such as inclusion-exclusion [20] and sum of disjoint products [3], or approximating methods such as the Monte-Carlo simulation [5, 6, 27]. Thus, determination of all *d-MC*s is an important step in computing system reliability. The search for all *d-MC*s is an NP-hard problem [4]. In [25], Xue

[*] Corresponding author.
[1] Faculty of Mathematical Sciences, Sharif University of Technology, Tehran, Iran, Email: forghanimajid@gmail.com.
[2] Faculty of Mathematical Sciences, Sharif University of Technology, Tehran, Iran, Email: nezamm@sina.sharif.edu.

proposed an algorithm using discrete function theory, modular decomposition, and system enlarging to generate all the *d-MC*s for a multistate system having multistate components. Jane et al. [15] first defined the notion of a *d-MC* candidate and then proposed an algorithm that first finds all the *d-MC* candidates obtained from each *MC* and then checks every candidate for being a *d-MC*. Several algorithms were proposed (see [10−13, 15, 18, 22, 25, 26, 28−32]) to obtain *d-MC* candidates in order to determine *d-MC*s. Generally, these algorithms consist of two stages, first gathering all the *d-MC* candidates and then determining all the *d-MC*s by testing the candidates. It is thus apparent that by having a smaller number of *d-MC* candidates in the first stage and by improving upon the testing time for each candidate in the second stage, the efficiency of an algorithm is increased.

Lin [18] showed that maximal elements of all the obtained *d-MC* candidates from all *MC*s form the set of all the *d-MC*s and then proposed an algorithm, which is easy to understand and implement. This algorithm generates all the *d-MC* candidates by an enumeration process and then uses a comparative method to specify the set of maximal elements, *d-MC*s, among all the *d-MC* candidates. Yeh [28] first presented an algorithm which was merely based on the definition but unfortunately had a defect [22]. Then, by using some network properties, Yeh [29] proposed another algorithm (denoted by Algorithm 1 here). However, the new algorithm turned not to be practical. Moreover, we show here that the use of some results presented in [29] for solving the *d-MC* problem is quite expensive. Presenting more results, Yeh [30] proposed another algorithm. However, as will be demonstrated here, some results and the computed time complexity of the proposed algorithm in [30] are incorrect. In [31], an attempt was made to improve the proposed algorithm in [30] but, because of using an incorrect result, an incorrect algorithm (denoted by Algorithm 2 here) was proposed which appears to lose some *d-MC*s [26]. Yan and Qian [26] exemplified the fault of Algorithm 2 and proved new results to decrease the number of obtained *d-MC* candidates, to find some *d-MC*s without the need for testing to eliminate the duplicate *d-MC*s. As a result, an improved algorithm, being more efficient than the proposed ones in [28−31], was proposed (see [26] for a comparative study). Later, by pointing out the defect of Algorithm 2, Yeh [32] presented a new technique to decrease number of obtained *d-MC* candidates and a novel approach to avoid production of duplicate *d-MC*s. Then, using new techniques, he proposed an algorithm (denoted by Algorithm 3 here) and showed its efficiency in comparison with the ones in [15, 18, 26, 28−31]. Forghani and Mahdavi-Amiri [12] proposed a simple algorithm for solving the *d-MC* problem. They also extended the algorithm to networks with budget constraint [11] and multi-commodity networks [13].

Here, by investigating the proposed algorithms and the given results in [29−31], we show some existing results and Algorithm 2 to be incorrect and then give their correct versions. Then, a modified version of Algorithm 2 is proposed as an efficient algorithm to find all *d-MC*s in a stochastic-flow network. Moreover, the corresponding complexity results in [29−31] are shown to be incorrect and their correct versions are presented. We also show the efficiency of our proposed algorithm in comparison with Algorithm 3 using the performance profile of Dolan and Moré [8].

The remainder of our work is organized as follows. Section 2 describes the required notations, nomenclature, and assumptions. A brief introduction is provided in Section 3. In Section 4, some existing results, their corresponding algorithms and complexity results are investigated, certain flaws are detected and corrections are provided. An improved version of Algorithm 2 is given in Section 4 and its efficiency is shown using the performance profile of Dolan and Moré on the results obtained over one hundred randomly generated test problems. Finally, we conclude in Section 5.

2. Notations, Nomenclature, and Assumptions

2.1. Notations

Here, we use the notations, nomenclature, and assumptions given in [29]. The notations are:

$G(V,E,W)$	a stochastic-flow network with the set of edges $E=\{e_i \mid 1\le i\le m-n\}$, the set of nodes $V=\{e_i \mid m-n+1\le i\le m\}$, and $W(e_i)$ denoting the max-capacity of e_i, for $1\le i\le m$.
s,t	source node $s\notin V$ and sink node $t\notin V$.
n,m	numbers of nodes in V and elements in $(V\cup E)$, respectively.
p,σ	numbers of *MC*s and *d-MC* candidates in $G(V,E,W)$, respectively.
e_i	ith element in $(V\cup E)$.
$C(e_i)$	capacity level of edge e_i under system-state (vector) $C=(x_1,x_2,\ldots,x_m)$.
$0(e_i)$	a system vector in which the capacity level is 1 for e_i and 0 for other edges.
C_i	ith *MC* in $G(V,E,W)$.
C_{ij}^d	$C_{ij}^d=(x_1,x_2,\ldots,x_m)$ is jth *d-MC* candidate (system-state vector) generated from C_i in $G(V,E,W)$, where $x_k=C_{ij}^d(e_k)\le W(e_k)$, $\sum x_k=d$, for $e_k\in C_i$, and $x_l=W(e_l)$, for $e_l\notin C_i$.
$G(V,E,C_{ij}^d)$	network corresponding to $G(V,E,W)$ with $C_{ij}^d(e_k)=x_k\le W(e_k)$, for $e_k\in C_i$, and $C_{ij}^d(e_k)=x_k=W(e_k)$, for $e_k\notin C_i$, if $C_{ij}^d(e_k)=(x_1,x_2,\ldots,x_m)$.
$R(V,E,C_{ij}^d)$	residual network corresponding to $G(V,E,C_{ij}^d)$ after sending d units of flow from node s to node t.
$W(C)$	max-flow from source node to sink node in $G(V,E,C)$, where C is a system vector.
ν	max-flow from source node to sink node in $G(V,E,C)$.
$U(C)$	$U(C)=\{e\in(E\cup V) \mid C(e)<W(e)\}$ is the set of unsaturated elements under system-state vector C.
$K_{C_i}(C)$	capacity of *MC*, C_i, under system-state vector C, i.e., $K_{C_i}(C)=\sum_{e\in C_i}C(e)$.
$\lvert\bullet\rvert$	number of elements; e.g., $\lvert N\rvert$ is the number of nodes in N.

3. Preliminaries

The following theorem is an important result concerning search for all *d-MC*s, originated by Jane et al. [15].

Theorem 1. [15] If C is a *d-MC*, then there exists at least one minimal cut (*MC*), C_i, so that

$$\begin{aligned}&(1)\ K_{C_i}(C)=d,\\ &(2)\ 0\le C(e)\le W(e),\ \forall e\in C_i,\\ &(3)\ C(e)=W(e),\ \forall e\notin C_i,\end{aligned}\qquad(1)$$

We observe that for every *d-MC*, say C, there exists at least one *MC*, C_i, satisfying the relations stated in Theorem 1. Thus, in order to determine all *d-MC*s in a network flow, one can first obtain all the system-state vectors satisfying the relations for at least one *MC*, say C_i, as a *d-MC* candidate, and

then search for *d-MC*s among them. It is easily verified that the number of obtained *d-MC* candidates in Theorem 1, corresponding to each *MC*, is bounded by

$$\sigma = Min\left\{\binom{m+d-1}{d}, \quad \prod_{j=1}^{m}\left(1+Min\{W(e_j), d\}\right)\right\}. \tag{2}$$

The proposed algorithms in [12, 14, 18, 22, 26, 28−32], using Theorem 1, first obtain the *d-MC* candidates in the network and then using the definition of *d-MC*, specify the *d-MC*s by testing *d-MC* candidates. Thus, generally, an algorithm in search for all *d-MC*s consists of the following two stages:

1. Obtaining all *d-MC* candidates based on Theorem 1.
2. Determining all *d-MC*s by checking out *d-MC* candidates, using the definition of *d-MC*.

Obviously, every *d-MC* candidate is not necessarily a *d-MC* and having a smaller number of *d-MC* candidates containing all *d-MC*s in the first stage can lessen the work in stage 2. On the other hand, in stage 2, checking any *d-MC* candidate C for a *d-MC* requires testing the following two conditions:

$$(a)\ W(C) = d,\ \ (b)\ W(C + 0(e)) > d, \text{ for all } e \in U(C).$$

It is clear that speeding up the procedures for verifications of (*a*) and (*b*) in stage 2 can ameliorate a proposed algorithm.

4. Corrections

Here, we point out some incorrect published results. We note that an incorrect result is embraced within quotations as '.'. First, in Section 4.1 we will show flaws in some presented results in [29], [30] and [31] and present their corrections. Then, in Section 4.2, we state an incorrect algorithm (the proposed algorithm in [31]) and present a modified version of it. Some incorrect complexity results (Theorem 7 in [29], Theorem 6 in [30], and Theorem 5 in [31]) are rectified in Section 4.3. Section 4.4 shows the efficiency of the modified proposed algorithm using numerical results in the sense of the performance profile introduced by Dolan and Moré [8].

4.1. Incorrect Results

It should be kept in mind that an algorithm on search for all *d-MC*s in a network flow consists of two general stages, obtaining *d-MC* candidates and specifying *d-MC*s by checking out the candidates. To get an efficient algorithm, it has commonly been tried to decrease either the generated *d-MC* candidates in the first stage or the time needed for checking the specifications in the second stage. Here, we are going to show that some presented results in [29], [30] and [31] are not generally correct.

In [29], in order to decrease the number of generated *d-MC* candidates, Yeh stated a result, Theorem 3 in [29], and then utilized it in Step 1.1 of Algorithm 1 in [29]. While the use of this result is claimed to decrease the number of obtained *d-MC* candidates, but, as will be shown later, in reality the number of candidates is not changed and the complexity of the algorithm is increased. To explain this defect, it should first be noted that theorems 2 and 3 in [29] have typographical errors (C_i is printed instead of C). Also, since, in Theorem 3 in [29], the *MC*, C_i, is determined to obtain S_i, then

condition (4) in this theorem should be written separately after the first three conditions. Hence, Theorem 2 below can be a more appropriate statement of Theorem 3 in [29]. Below, note that $C_{ij} \propto d(C_i)$ means C_{ij} is an obtained *d-MC* candidate from C_i and $C_{ik} = d(C_i)$ means that C_{ij} is an obtained *d-MC* from C_i. Also, a min s-t cut is an *MC* with a minimal capacity.

Theorem 2. If C is a *d-MC*, then there exists at least one *MC*, C_i, such that

$$\begin{aligned} &(1)\ K_{C_i}(C) = d, \\ &(2)\ C(e) \le W(e), \forall e \in C_i, \\ &(3)\ C(e) = W(e), \forall e \notin C_i. \end{aligned} \tag{3}$$

Also, if S_i is a set of all the min s-t cuts in $G(V, E, C_{ij})$, for all the obtained *d-MC* candidates from C_i, say C_{ij}, then

$$(4)\ \sum C(e) > \sum C_{ij}(e), \qquad \forall e \in C_i \cap C^*,\ \forall C^* \in S_i \backslash C_i \neq \emptyset, \text{ where } \exists e^* \in C_i \backslash C^* \text{ such that } W(e^*) > C_{ij}(e^*).$$

Yeh [29] claimed that using Theorem 2 in Algorithm 1 would lead to a decrease in the number of *d-MC* candidates, which unfortunately turns not to be correct, as explained below. For convenience, we first rewrite Algorithm 1 in [29] as Algorithm 1 here.

Algorithm 1 (The proposed algorithm in [29]).

Step 0. Let $j = 0,\ C_{ij} = \emptyset,\ S_i = \emptyset$, and $Vc = \{a | a \in V$, and (v, a) or $(a, v)\ in\ C_{ij}$, where $v \in V\}$.
Step 1. Let $j = j + 1$ and use the Implicit Algorithm to find a *d-MC* candidate C_{ij} of the following mathematical model:

$$\begin{aligned} &(1)\ K_{C_i}(C) = d, \\ &(2)\ C(e) \le W(e), \forall e \in C_i, \\ &(3)\ C(e)\ W(e), \forall e \notin C_i, \\ &(4)\ \sum C_i(e) > \sum C_{ij}(e), \text{ where } e \in C_i \cap C^*, \text{ for all } C^* \in S_i \backslash C_i \neq \emptyset. \end{aligned} \tag{4}$$

If no such *d-MC* candidate exists, then halt.
Step 2. If $C_{ij} = C_{i,j-1} + 0(e_x) - 0(e_y)$, then calculate $W(C_{ij})$ by using Algorithm 2 in [29], otherwise use the max-flow algorithm.
Step 3. Use the max-flow algorithm to find $W(C_{ij})$. If $W(C_{ij}) = d$, then C_{ij} is not a *d-MC* and go to Step 1.
Step 4. Check C_{ij} and find a min s-t cut set S_i by using Algorithm 3 in [29]. Go to Step 1.

Focusing on Step 1 of Algorithm 1 (see Theorem 2), it can be found that S_i equals all the min s-t cuts in $G(V, E, C_{ij})$, for all the *d-MC* candidates generated from C_i, i.e., $S_i = \bigcup_{j:\, C_{ij} \propto d(Ci)} S_{ij}$. This means that in order to determine S_i, the following two steps are needed:

1. Generate all the *d-MC* candidates by C_i, namely C_{ij}.

2. Determine all the min s-t cuts in $G(V, E, C_{ij})$, for all the obtained C_{ij} in the first step.

Note that subscript i in S_i corresponds to *MC*, C_i. Thus, in order to determine S_i, all the producible *d-MC* candidates by C_i should be obtained. Therefore, not only the application of Theorem 2 does not decrease the number of obtained *d-MC* candidates, but it rather increases the complexity of the problem as a result of determining all the *d-MC* candidates and all the min *s-t* cuts for every *d-MC* candidate C_{ij} in $G\ (V, E, C_{ij})$. According to theorems 1 and 2, one can realize that inequality (4) in Theorem 2, the fourth restriction in Step 1 of Algorithm 1, does not decrease the number of *d-MC* candidates.

To have an intuitive understanding, consider the experiment conducted by Yeh (Section 7 in [29]) for comparison of algorithms 0 and 1 in [29]. In this experiment, it is seen that the number of obtained *d-MC* candidates by both algorithms 0 and 1 in [29] is the same (they were presented under the same column, N_{Can}). However, considering the above explanations, it seems that the results listed in column N_{Can} of tables 2 and 3 in [29] are correct (i.e., the numbers of *d-MC* candidates for both algorithms 0 and 1 in [29] should be the same). Thus, use of Theorem 2 turns to be more expensive than use of Theorem 1 for obtaining the *d-MC* candidates.

In [30], Yeh employed concept of residual network to lessen the number of usages of the max-flow algorithm [2] and also presented some new results to improve his proposed algorithm in [28]. However, two stated results ('Theorem 6' and 'Corollary 3') are incorrect. Since these results are concerned with complexity results, we discuss them in Section 4.3. Other incorrect results in [30] are 'Lemma 3' and 'Theorem 5'. Here, we show that they need one more assumption to be applicable, in general.

'Lemma 3' [30]: Let C be a *d-MC* candidate in $G(V, E, W)$. If there is a path between the source node and the sink node in $R\big(V, E, C + 0(e_i)\big)$, for $e_i \in E$, then $W(C + 0(e_i)) > d$.

'Theorem 5' [30]: Let C be a *d-MC* candidate in $G(V, E, W)$. If there is a path between the source node and the sink node in $R(V, E, C + 0(e_i))$, for all $e_i \in U(C)$, then C is a real *d-MC*; otherwise, C is not a real *d-MC*.

According to the definition of *d-MC* candidate, it is readily concluded that $W(C) \leq d$, for every *d-MC* candidate, say C. On the other hand, to construct the residual network $R(V, E, C)$, and consequently send d units of flow from *s* to *t*, we need to have $W(C) \geq d$. Thus, for 'Lemma 3' and 'Theorem 5' to be correct, the assumption $W(C) = d$ should be added. To have an intuitive understanding, consider Figure 1 as a network flow. It is obvious that $C_3 = \{e_1, e_3, e_4, e_6\}$ is an *MC* and $C_{31} = (0, 2, 3, 1, 3, 3)$ is a 7-*MC* candidate obtained from C_3. It is straightforwardly seen that $W(C) = 5$, and hence the residual network $R(V, E, C_{31})$ cannot be constructed by sending a flow of 7 units. Therefore, 'Lemma 3' and 'Theorem 5' need an additional assumption, which is $W(C) = d$.

We give the correct form of 'Theorem 5' in [30] as Theorem 3 below. The correct version of 'Lemma 3' can be obtained similarly by adding the assumption $W(C) = d$.

Theorem 3. Let C be a *d-MC* candidate in $G(V, E, W)$ and $W(C) = d$. C is a *d-MC* if and only if there is a path between the source node and the sink node in $R(V, E, C + 0(e))$, for all $e \in U(C)$.

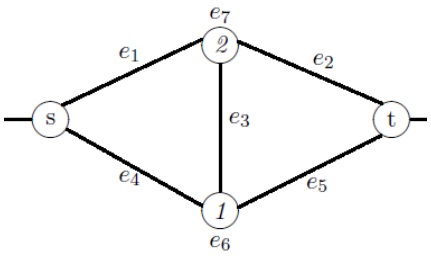

Figure 1. Network flow for examples 1, 2, and 3 with $W = (3, 2, 1, 1, 2, 3, 3)$

To improve the proposed algorithm in [30], Yeh [31] presented new results to propose an improved algorithm (Algorithm 2 here), which may lose some *d-MC*s. In fact, in order to decrease the number of generated *d-MC* candidates and avoid generation of duplicate *d-MC*s, Yeh presented 'Corollary 3' and 'Theorem 4', both of which turn to be incorrect. Here, their faults are exemplified and their correct versions are presented.

'Corollary 3' [31]: Let C be a *d-MC* (candidate) in $G(V, E, W)$. If $\sum C^*(e) \leq \sum C(e)$, for all $e \in U(C)$, then *d-MC* candidate $C^*(\neq C)$ is not a real *d-MC*.

The following example invalidates 'Corollary 3'.

Example 1. Consider Figure 1 as s network flow. Note that $C_1 = \{e_1, e_4\}$ is an *MC* and it is clear that $X = (2, 2, 1, 0, 2, 3, 3)$ and $X^* = (1, 2, 1, 1, 2, 3, 3)$ are 2-*MC* candidates generated from C_1. Since $X \neq X^*$, $U(X) = \{e_1, e_4\}$, and $\sum_{U(X)} X^*(e) \leq \sum_{U(X)} X(e)$, 'Corollary 3' concludes that X^* is not a 2-*MC*, whereas it is easily observed that X^* is indeed a 2-*MC*.

Thus, 'Corollary 3' is not necessarily valid. In fact, in 'Corollary 3', it is not necessarily concluded from $\sum_{U(C)} C^*(e) = \sum_{U(C)} C(e)$ that C^* is not a *d-MC*. Hence, the following lemma is a correct version of 'Corollary 3'.

Lemma 1. Let C be a *d-MC* (candidate) in $G(V, E, W)$. If $\sum_{e \in U(C)} C^*(e) < \sum_{e \in U(C)} C(e)$, then *d-MC* candidate $C^*(\neq C)$ is not a real *d-MC*.

Proof. Since $\sum_{e \in U(C)} C^*(e) < \sum_{e \in U(C)} C(e)$, there exists an element in $U(C)$, say e', so that $C^*(e') < C(e') < W(e')$ and consequently, $e' \in U(C^*)$. Hence, $C^* + 0(e') \leq C$ and thus, $W(C^* + 0(e')) \leq W(C) \leq d$. Therefore, C^* cannot be a *d-MC*. □

Yan and Qian [26] illustrated through an example that 'Theorem 4' in [31] had a defect. Here, a correct version of 'Theorem 4' is presented, exemplifying its flaw.

'Theorem 4' [31]: If X is a (real) *d-MC* in $G(V, E, W)$ obtained from *MC*, C, then each *d-MC* $X^*(\neq X)$ must satisfy $X^*(e) \geq X(e)$, for all $e \in C$, and there is at least one component $\varepsilon \in C$ *s.t.* $X^*(\varepsilon) > X(\varepsilon)$.

Utilizing 'Theorem 4' of [31] in Step 6 of 'Algorithm 2' will result in the omission of some *d-MC*s. To see this point, consider the following example.

Example 2. Note that in Figure 1, $C_1 = \{e_1, e_4\}$ and $C^* = \{e_2, e_3, e_4\}$ are *MC*s and $X = (1, 2, 1, 1, 2, 3, 3)$ and $X^* = (3, 1, 1, 0, 2, 3, 3)$ are 2-*MC* candidates generated from C and C^*,

respectively. It is obvious that X and $X^*$are 2-*MC*s, whereas it can be observed that if one utilizes 'Theorem 4', then either X or X^* is omitted. Indeed, any one of the 2-*MC* candidates X or X^*, generated earlier, will result in the omission of the other one, by using 'Theorem 4'.

In fact, in 'Theorem 4', it cannot be necessarily concluded from the hypothesis that $X^*(e) \geq X(e)$, for all $e \in C$. Thus, the following lemma is presented as a correct version of 'Theorem 4'.

Lemma 2. If X is a *d-MC* in $G(V, E, W)$, obtained from *MC*, C, then for each *d-MC* $X^*(\neq X)$, there is at least one element $e \in U(C)$ such that $X^*(e) > X(e)$.

Proof. To reach a contradiction, let us suppose that $X^*(e) \leq X(e)$, for all $e \in U(C)$. Since for all $e \notin U(C)$, $X^*(e) \leq W(e) = X(e)$, we have $X^*(e) \leq X(e)$, for all $e \in V \cup E$. Now, since $X^* \neq X$, there exists at least one element, say e', in $V \cup E$ so that $X^*(e') < X(e')$. Hence, $X^* + 0(e') \leq X$ and consequently, $W\big(X^* + 0(e')\big) \leq W(X) = d$, which contradicts the fact that X^* is a *d-MC* and therefore, the proof is complete. □

In addition to 'Corollary 3' and 'Theorem 4', 'Theorem 2' in [31] has a (probably typographical) error. First, its flaw is demonstrated through an example and then a correct version is proposed and proved.

'Theorem 2' [31]: Assume that C is a *d-MC* candidate in $G(V, E, W)$ with $W(C) = d$. If $C^*(e) \geq C(e)$, for all $e \in V \cup E$ and *d-MC* candidate $C^*(\neq C)$, then C is a real *d-MC*.

Example 3. Consider Figure 1 as a network flow. It is clear that $C_1 = \{e_1, e_3, e_5\}$ and $C_2 = \{e_1, e_4\}$ are *MC*s. It can be verified that $X = (1, 2, 0, 1, 1, 3, 3)$ and $X^* = (1,2,1,1,2,3,3)$ are 2-*MC* candidates generated from C_1 and C_2, respectively. Since $W(X) = 2$ and $X^*(e) \geq X(e)$, for all $e \in V \cup E$, 'Theorem 2' concludes that X is a 2-*MC*, which is not true. As a result, 'Theorem 2' is not necessarily valid.

Now, if one uses "$C^* \leq C$, for each *d-MC* candidate C^*" instead of the assumption "$C(e) \leq C^*(e)$, for all $e \in V \cup E$ and *d-MC* candidate $C^*(\neq C)$" in 'Theorem 2', then 'Theorem 2' will be correct. The correct result is given as Theorem 4 below.

Theorem 4. Assume that C is a *d-MC* candidate in $G(V, E, W)$ with $W(C) = d$. If $C \geq C^*$, for every *d-MC* candidate C^*, then C is a *d-MC*.

It should be noted that Theorem 4, being a correct version of 'Theorem 2', is a special case of the following theorem, proved in [18].

Theorem 5. [18] Every maximal element among all *d-MC* candidates is a *d-MC*.

In Theorem 5, the assumption $W(C) = d$ and the existence of the inequality $C^* \leq C$, for every *d-MC* candidate C^*, are not necessary, while being needed in Theorem 4. Another significant point to consider is that it is possible that C and C^* are *d-MC*s, but the inequalities $C \leq C^*$ and $C^* \leq C$ do not hold.

4.2. On Algorithm 2

Here, for convenience, we first rewrite the proposed algorithm in [31], Algorithm 2 here, and illustrate its defect in Section 4.2.1 and then provide a correcting modified version in Section 4.2.2.

4.2.1. Incorrect Algorithm

To improve the proposed algorithm in [30], Yeh [31] presented a new result, 'Theorem 4', to decrease the number of obtained *d-MC* candidates. However, as shown before, the presented result was incorrect, and consequently Algorithm 2 turns to be incorrect as well. Yan and Qian [26] illustrated the fault of Algorithm 2 through an example and then, without modifying the incorrect result in [31], presented a new result to decrease the number of obtained *d-MC* candidates by finding Lower Capacity Limits (LCLs) for some elements in $V \cup E$. Also, Yeh [31] pointed out the defect of Algorithm 2 and presented a new technique for finding the LCLs and a novel approach to avoid generation of duplicates. Here, in order to modify Algorithm 2, we first give Algorithm 2 as proposed in [31] and then show its defect again over the same example given in [31].

Algorithm 2 (The proposed algorithm in [31]).

Input: All *MC*s C_1, C_2,..., C_p of a stochastic-flow network $G(V, E, W)$ with given unreliable nodes and $|C_i| \leq |C_j|$, for $i > j$.
Output: All *d-MC*s.
Step 1: Let $i = j = 1$, $Q = \{$(the capacity of e)$|\ 0 \leq$ (the capacity of e) $\leq W(e)$, for all $e \in (E \cup V)\}$, and arrange the *MC*s according to their element numbers and the number of element capacities that are not less than *d*.
Step 2: Find a feasible solution, say X, generated from C_i, using Theorem 1. If no such solution exists, then go to Step 7.
Step 3: If $W(X) = d$, then X is not a *d-MC* and return to Step 2 to find the next feasible solution.
Step 4: If $C_{ij}^d(e_k) = W(e_k)$, then go to Step 6.
Step 5: If there is no path from node s to node t through e_i in $R(V, E, C_{ij}^d + 0(e_i))$, then C_{ij}^d is not a *d-MC* and return to Step 2 to find the next feasible solution. Otherwise, go to Step 6.
Step 6: If $k < m$, then $k \leftarrow k + 1$ and go to Step 4. Otherwise, C_{ij}^d is a real *d-MC*, and let $Q = Q \cup$ {the capacity of e is greater than $C_{ij}^d(e)$, for all $e \in U(C_{ij}^d(e)\}$.
Step 7: If $i < p$, then $i \leftarrow i + 1$, $j = 1$ and go to Step 2. Otherwise, halt.

As pointed out in Section 4.1, making use of 'Theorem 4' in Algorithm 2 may miss some *d-MC*s from the desired set of *d-MC*s. Here, by considering Example 2 in [31] (for which Yeh obtained all the 2-*MC*s by Algorithm 2), this shortcoming will be explained completely. For an easy access, the example network is shown as Figure 2. The order of the findings of *d-MC* candidates generated from C_i in Step 2 of the algorithm not being specified, then in Example 2 in [31], one can assume that the 2-*MC* candidate $C_{11} = (3, 1, 1, 1, 1)$ is obtained before the other 2-*MC* candidates. In that case, since $U(C_{11}) = \{e_2, e_5\}$ and C_{11} is a 2-*MC*, Step 6 in the algorithm adds two new constraints, $x_2 > 1$ and $x_5 > 1$, to the set Q (see Step 6 in the proposed algorithm in [31]). This results in missing the other 2-*MC*s, $C_{12} = (3, 2, 1, 1, 0)$ and $C_{13} = (3, 0, 1, 1, 2)$. In the next iteration, if $C_{21} = (2, 2, 1, 0, 2)$ is considered, then the algorithm gives that $x_1 > 2$ and $x_4 > 0$, and consequently the algorithm misses the other 2-*MC*s (the complete solution is given in Table 1).

Table 1. Final obtained results of solving Example 2 in [31] by Algorithm 2

MC	2 -*MC* candidate	A 2 -*MC*?	The new constraint
$\boldsymbol{C_1 = D_2 = \{e_2, e_5\}}$	$C_{11} = (3,1,1,1,1)$ No other 2-*MC* candidate can be generated from C_1	Yes	$x_2 > 1,\ x_5 > 1$
$\boldsymbol{C_2 = D_1 = \{e_1, e_4\}}$	$C_{21} = (2,2,1,0,2)$ No other 2-*MC* candidate can be generated from C_2	Yes	$x_1 > 2,\ x_4 > 0$
$\boldsymbol{C_3 = D_3 = \{e_1, e_3, e_5\}}$	None	–	–
$\boldsymbol{C_4 = D_4 = \{e_2, e_3, e_4\}}$	None	–	–

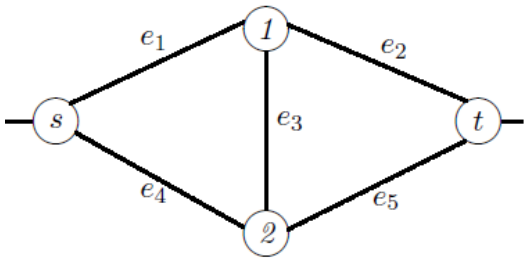

Figure 2. A network flow with $W = (3,2,1,1,2)$

Note that, without writing the complete and exact solutions for Example 2 in [31], Yeh claimed that the proposed algorithm in [31] (Algorithm 2 here) determined all 2-*MC*s in this example, whereas it is now apparent that this is not correct.

Note that Yeh [31] stated how the *MC*s must be rearranged before being used in the algorithm ("all *MC*s are rearranged in increasing order of their element numbers. If there is a tie, then the one with element capacities greater than or equal to d is arranged first [31]."). Since $D_3 = \{e_1, e_3, e_5\}$ has 2 elements e_1 and e_5 with their capacities being greater than or equal to $d = 2$ and $D_4 = \{e_2, e_3, e_4\}$ has just one element e_2 with the capacity $d = 2$, D_3 is arranged before D_4 and thus, $C_3 = D_3$ and $C_4 = D_4$ are obtained. Yeh incorrectly set $C_3 = D_4$ and $C_4 = D_3$.

4.2.2. A Modified Version of Algorithm 2

Two modified versions of Algorithm 2 have been proposed by Yan and Qian [26] and Yeh [32]. As mentioned earlier, due to the use of 'Theorem 4', Algorithm 2 misses some d-MCs. Note that 'Theorem 4' was presented to decrease the number of obtained d-MC candidates, which unfortunately turned not to be correct. To this aim, Yan and Qian [26] presented a new result and used it to obtain all the LCLs at the beginning of their algorithm. The proposed approach in [26] works correctly, but needs extra work at the beginning of the algorithm. Also, Yeh [32] presented a new technique for finding all the LCLs, but as will be shown below, his technique does not work properly. In fact, by obtaining LCL, $L(e)$ corresponding to every element $e \in V \cup E$, one can use the following system, instead of the system (1), in Theorem 1 for generating all the *d-MC* candidates:

$$\begin{aligned}&(1)\ K_{C_i}(C) = d,\\ &(2)\ L(e)\ \le C(e) \le\ W(e), \forall e \in C_i,\\ &(3)\ C(e) =\ W(e), \forall e \notin C_i,\end{aligned} \tag{5}$$

Establishing a theorem (Theorem 1 in [32]), Yeh [32] proposed a technique, given below as Technique 1, for determining all the LCLs and *d-MC*s with only one unsaturated arc.

Technique 1: Find all the LCLs and *d-MC*s with one unsaturated arc.

Step 0. Let $i = 1$.
Step 1. Let $L^* = 0$, $U^* = M_i$, and $x_j = M_j$, for all $j = i$.
Step 2. Let the value of x_i be the greatest integer less than or equal to $(L^* + U^*)/2$.
Step 3. If $V(X) = d$, then let $L_i = x_i + 1$, and go to Step 6.
Step 4. If $V(X) > d$, then let $U^* = x_i$, and go to Step 2.
Step 5. If $V(X) < d$, then let $L^* = x_i$, and go to Step 2.
Step 6. If $i < m$, then let $i = i + 1$, and go to Step 1, else stop.

Now, using an example, we show that Technique 1 does not work properly. Consider Figure 1 as a network flow and find its 2-*MC*s. Employing Technique 1 to find $L(e_7)$, we obtain:

Step 1. $L^* = 0,\ U^* = 3$, and $X = (3, 2, 1, 1, 2, 3, x_1)$.
Step 2. $x_1 = 1$ and $X = (3, 2, 1, 1, 2, 3, 1)$.
Step 3. Since $W(X) = 2$, we have $L(e_7) = 1 + 1 = 2$, which is the desired result.

Next, to find $L(e_4)$, we have:

Step 1. $L^* = 0$, $U^* = 1$, and $X = (3, 2, 1, x_4, 2, 3, 3)$.
Step 2. $x_4 = 0$ and $X = (3, 3, 3, 0, 1, 2, 2)$.
Step 4. Since $V(X) = 3 > 2$, we have $U^* = 0$.
Step 2. $x_4 = 0$ and $X = (3, 2, 1, 0, 2, 3, 3)$.
Step 4. Since $V(X) = 3 > 2$, we have $U^* = 0$.
⋮

As seen, the algorithm does not terminate.

Hence, the proposed technique by Yeh [32] (Technique 1 above) may turn to work improperly. In fact, Technique 1 is faced with a problem when we have U^*=0. Thus, to modify Technique 1, it is only needed to correct Step 2 as follows.

Step 2. If U^*=0, then go to Step 6, else let the value of x_i be the greatest integer less than or equal to $(L^* + U^*)/2$.

Using this technique, Yeh proposed another algorithm (denoted by Algorithm 3 here) to solve *d-MC* problem and demonstrated its efficiency in comparison with the proposed algorithms in [15, 18, 22, 26, 28-31] with respect to time complexity. Next, we first state the proposed algorithm in [32] as Algorithm 3, and then propose a new improved algorithm as a modified version of Algorithm 2.

Algorithm 3 (the proposed algorithm in [32]).

Step 0: Let $i = 1$, and find $L(e)$, for all $e \in V \cup E$, using Technique 1, and let Ω={ the related *d-MC* generated from Technique 1}.
Step 1: Find all feasible solutions, say X_{ij}, for j=1, 2, …, J, generated from C_i, satisfying the following system:

(1) $K_{C_i}(X_{ij}) = d$,
(2) $L(e) \leq X_{ij}(e) \leq W(e), \forall e \in C_i$,
(3) $X_{ij}(e) = W(e), \forall e \notin C_i$,

If no such solution exists, then go to Step 6.
Step 2: If $\sum_{e \in X} W(e) = d$, then all *d-MC* candidates generated from C_i are *d*-MC, and go to Step 5.
Step 3: If $W(X_{ij}) \neq d$, then X_{ij} is not a *d-MC* where j=1,2,…, J. If none of them is a real *d-MC*, then return to STEP 1.
Step 4: Verify whether *d-MC* candidate X_{ij} is a real *d-MC* for j=1,2,…, J, by using Theorem 3. If none of them is a real *d-MC*, then return to STEP 1.
Step 5: According to Lemma 4, for j=1,2,…, J, if there is $0 < k < i$ such that $U(X_{ij}) \subseteq C_k$, then X_{ij} is a duplicate *d-MC*. Find non duplicates and add them to Ω.
Step 6: If $i < p$, then let $i = i + 1$, and go to Step 2, else Ω is a *d-MC* set, and stop.

Note that we have considered Algorithm 3 with the correct version of Technique 1. Computing the complexity results, Yeh [32] demonstrated that Algorithm 3 is more efficient than the other previous proposed ones in [15, 18, 22, 26, 28-31].

Now, we are ready to present our improved algorithm. By using the correct version of 'Theorem 4', as presented in the previous section, a simple technique is provided to find all the LCLs. Although Lemma 2 is the correct form of 'Theorem 4', but it is not applicable to Algorithm 2. The following lemma, directly obtained from Lemma 2 and useful for Algorithm 2, is established.

Lemma 3. If X is a *d-MC* in $G(V, E, W)$ obtained from *MC*, C, and $|U(C)| = 1$, cosidering $U(C) = \{e'\}$, then each *d-MC* $X^*(\neq X)$ satisfies $X^*(e') > X(e')$.

Proof. Since $U(C)$ has only one element, the proof directly follows from Lemma 2. □

According to Lemma 3, one can find the LCLs when a *d-MC* candidate with only one unsaturated arc is obtained. For instance, in Figure 1, $C_1 = \{e_1, e_4\}$ is an *MC* and $X_{11} = (3,2,1,0,2,3,3)$ and $X_{12} = (2,2,1,1,2,3,3)$ are two 3-*MC*s obtained from C_1. Since $U(X_{11}) = \{e_4\}$, by using Lemma 3, it is deduced that $L(e_4) = 0 + 1 = 1$ and similarly, since $U(X_{12}) = \{e_1\}$, we find that $L(e_1) = 2 + 1 = 3$.

It is easily verified that the probability of obtaining the *d-MC* candidates with only one unsaturated arc from an *MC*, say C, is increased as the capacity of C, $K_C(W)$, is decreased. Thus, to enhance the effectiveness of Lemma 3 in the modified algorithm, all *MC*s are arranged in a non-decreasing order of their capacities.

A notable common weak point of the proposed algorithms in [28−31] was the production of duplicate *d-MC*s. Yan and Qian [26] compared all the *d-MC*s obtained by the algorithm for eliminating the duplicates. However, the time complexity of the proposed algorithm increased

to $O(mp^2\sigma^2)$. To liquidate this deficiency, Yeh [32] presented a novel technique based on the following lemma stated in [32].

Lemma 4. [32] Let X be a *d-MC* generated from *MC*, C_i. If $U(X) \subseteq C_j$, then X is also a *d-MC* generated from *MC*, C_j.

In accordance with Lemma 4, if X_{ij} is a *d-MC* candidate obtained from C_i and there is $0 < k < i$ such that $U(X_{ij}) \subseteq C_k$, then X_{ij} was obtained from C_k previously and it is now a duplicate. Therefore, to avoid production of the duplicate *d-MC*s, we should gather the *d-MC*s obtained from C_i, say X_{ij}, for $i = 1, 2, \ldots, p$, such that $U(X_{ij}) \nsubseteq C_k$, for $0 < k < i$.

Using Lemma 3 to find LCLs and applying Lemma 4 to avoid production of the duplicates, a modified version of 'Algorithm 2' is proposed as Algorithm 4 below.

Algorithm 4 (a modification of the proposed algorithm in [31], Algorithm 2 here).

Input: All *MC*s.
Output: All *d-MC*s.
Step 1: Let $i = j = k = 1$, $Q = \phi$, $L(e) = 0$, for all $e \in V \cup E$, and arrange all *MC*s in a non-decreasing order of their capacities.
Step 2: Find a feasible solution, say X_{ij}, generated from C_i, satisfying the following system:

(1) $K_{C_i}(X_{ij}) = d$,
(2) $L(e) \leq X_{ij}(e) \leq W(e), \forall e \in C_i$,
(3) $X_{ij}(e) = W(e), \forall e \notin C_i$,

If no such solution exists, then go to Step 7.
Step 3: If $U(X_{ij}) = \{e\}$ has only one element, then set $L(e) = X_{ij}(e) + 1$.
Step 4: If there is $0 < k < i$ such that $U(X_{ij}) \subseteq C_k$, then X_{ij} is a duplicate *d-MC*, and there is no need to test it, let $j = j + 1$ and go to Step 2.
Step 5: If $W(X_{ij}) = d$, then X_{ij} is not a *d-MC*, let $j = j + 1$ and go to Step 2.
Step 6: If $X_{ij}(e_k) = W(e_k)$, then go to Step 8, else go to Step 7.
Step 7: If there is no path from node s to node t through e_k in $R(V, E, X_{ij} + 0(e_k))$, then X_{ij} is not a *d-MC*, let $j = j + 1$ and go to Step 2.
Step 8: If $k < m$, then let $k = k + 1$ and go to Step 6, else X_{ij} is a *d-MC*, let $Q = Q \cup \{X_{ij}\}$, $k = 1$, $j = j + 1$ and go to Step 2.
Step 9: If $i < p$, then let $i = i + 1$, $j = 1$, $k = 1$ and go to Step 2, else stop (Q is the set of all the *d-MC*s with no duplicates).

Now, Example 2 in [31] is solved again by Algorithm 4. For brevity, the final solution is given in Table 2. As seen in Table 2, three LCLs were obtained by Algorithm 4 and, without losing any 2-*MC*, the number of candidates decreased to 8, which is exactly the number of 2-*MC*s. Moreover, the algorithm did not find any duplicate 2-*MC*.

4.3. On Complexities

Here, the complexity results of the proposed algorithms in [29], [30], and [31] are closely investigated.

4.3.1. Time Complexity of Algorithm 1

According to Lemma 2 in [30], the time complexity of Algorithm 2 in [29] is $O(n)$. Moreover, according to lemmas 5 and 6 in [29], the time complexity of Algorithm 3 in [29] is $O(m|C_i|) = O(m^2)$. Considering Algorithm 1 here, it is seen that it calculates $W(C_{ij})$ in steps 2 and 3 and checks the second condition given in the definition of *d-MC* in Step 4. Thus, to determine each *d-MC*, both algorithms 2 and 3 in [29] are implemented and consequently the time complexity for steps 2 to 4 is $O(m^2 n)$. According to Lemma 5 in [29], the time complexity of finding S_i in $G(V, E, C_{ij})$ for each C_{ij} is $O(n)$. Now, let p be the number of *MC*s in $G(V, E, W)$ and σ be the upper bound of the obtained *d-MC* candidates from each *MC* through verification of relations (1)−(3) in Step 1. Therefore, $p\sigma$ is an upper bound for the number of all solutions, the C_{ij}, obtained in Step 1. As a result, since we need to find S_i in Step 1, for each C_{ij}, the time complexity of Algorithm 1 is $O(m^2 n^2 p\sigma)$, and hence the following theorem is a correct version of 'Theorem 7' in [29].

Theorem 6. The time complexity of Algorithm 1 is $O(m^2 n^2 p\sigma)$.

In addition, according to Theorem 6, it is clearly concluded that the complexity result given by 'Corollary 3' in [30], $O(m^2 p\sigma)$, is incorrect and the correct time complexity of the algorithm that combines theorems 3 and 4 in [30] (Algorithm 1 here) is $O(m^2 n^2 p\sigma)$.

4.3.2. Time Complexities of the Proposed Algorithms in [30], [31], and Algorithm 4

We first compute the time complexity of Algorithm 2 (the proposed one in [31]) in detail, and then find the time complexity of the proposed algorithm in [30] and Algorithm 4 using the obtained result.

Table 2. Final results of Example 2 in [31] obtained by Algorithm 4

MC	2 -*MC* candidates	Is a 2-*MC* ?	Is a duplicate?	The new LCLs
$\boldsymbol{C_1 = D_2 = \{e_2, e_5\}}$	$C_{11} = (3,1,1,1,1)$ $C_{12} = (3,2,1,1,0)$ $C_{13} = (3,0,1,1,2)$	Yes Yes Yes	No No No	– $L(e_5) = 1$ $L(e_2) = 1$
$\boldsymbol{C_2 = D_1 = \{e_1, e_4\}}$	$C_{21} = (2,2,1,0,2)$ $C_{22} = (1,2,1,1,2)$	Yes Yes	No No	– $L(e_1) = 2$
$\boldsymbol{C_4 = D_4 = \{e_2, e_3\ e_4\}}$	$C_{41} = (3,2,0,0,2)$ $C_{42} = (3,1,1,0,2)$ $C_{43} = (3,1,0,1,2)$	Yes Yes Yes	No No No	– – –
$\boldsymbol{C_3 = D_3 = \{e_1, e_3\ e_5\}}$	None	–	–	–

● **Time complexity of the proposed algorithms in [30] and [31]:** In [31], Yeh in Theorem 5 erroneously claimed that the time complexity of his proposed algorithm, Algorithm 2 here, was $O(mnp\sigma)$. We will show that the time complexity of Algorithm 2 is indeed $O((m^2 + n^2\sqrt{m})p\sigma)$.

Since the number of *MC*s is p, then the time complexity of arranging *MC*s in Step 1 is $O(p\log p)$. In Step 2, Algorithm 2 finds a *d-MC* candidate using an implicit enumeration to solve the existing system given in Theorem 1. Although solving the system (1) in Theorem 1 has its own complexity, the authors in [12, 15, 18, 22, 26, 28-32] commonly disregard this complexity when calculating the time complexity of their algorithms. Therefore, we do the same in our analysis. The best time complexity of the max-flow algorithm [2], and so the time complexity of Step 3 is $O(n^2\sqrt{m})$. It is obviously seen that the time complexity of Step 4 is $O(1)$. Since $U(C_{ij}^d)$ is bounded by m (the number of arcs in E) and the time complexity of searching for a path from the source node to the sink node is $O(m)$, the time complexity of Step 5 is $O(m^2)$. The time complexity of updating the set Q, and hence the time complexity of Step 6 is $O(m)$. The time complexity of Step 7 is vividly $O(1)$. Thus, the time complexity of steps 3−6 is $O(m^2 + n^2\sqrt{m})$. Since steps 3−6, the main steps in the algorithm, are executed $p\sigma$ times the time complexity of Algorithm 2 is $O(plogp\ + (m^2 + n^2\sqrt{m})p\sigma) = O((m^2 + n^2\sqrt{m})p\sigma)$, disregarding the computing complexity corresponding to Step 2 as mentioned above. Therefore, the following theorem gives the correct result.

Theorem 7. The time complexity of 'Algorithm 2' is $O((m^2 + n^2\sqrt{m})p\sigma)$.

As stated in [31], the time complexity of the proposed algorithms in [30] and [31] are the same (see Theorem 6 in [30] and Theorem 5 in [31]). Therefore, according to Theorem 7 here, 'Theorem 6' in [30] is incorrect and the following theorem gives the correct result.

Theorem 8. The time complexity of the proposed algorithm in [30] is $O((m^2 + n^2\sqrt{m})p\sigma)$.

● **Time complexity of Algorithm 4:** It is noted that except for steps 3 and 4, Algorithm 4 is similar to Algorithm 2. The time complexity of Step 4 is $O(mp^2\sigma)$ [31] and the time complexity of Step 3 is vividly $O(m)$. As a result, using the above explanations, the time complexity of Algorithm 4 is $O(mp^2\sigma) + O\left((m^2 + n^2\sqrt{m})p\sigma\right) = O(mp^2\sigma)$. Since $O(p) = O(2^{n-2})$ and $O(n) \le O(m) \le O(n^2)$ [23], $O\left((m^2 + n^2\sqrt{m})p\sigma\right) \le O(mp^2\sigma)$, and consequently we have the following result.

Theorem 9. The time complexity of Algorithm 4 is $O(mp^2\sigma)$.

It should be kept in mind that the proposed algorithms in [28−31] may obtain duplicate *d-MC*s, and thus, similar to the proposed algorithm in [26], these algorithms need to compare all the obtained *d-MC*s to remove the duplicates, increasing the time complexity of the algorithms to $O(mp^2\sigma^2)$. Therefore, $O(mp^2\sigma^2)$ is the time complexity of the proposed algorithms in [26, 28−31] in the case of not generating any duplicate. Thus, according to Theorem 9, Algorithm 4 is more efficient than the proposed algorithms in [18, 22, 26, 28−31]. Yeh [32] also showed his proposed algorithm (Algorithm 3 here) is more efficient than the ones proposed in [18, 22, 26, 28−31]. In fact, the time complexity of Algorithm 3 is $O(mp^2\sigma)$ [32] which is the same as the one for Algorithm 4. To show the efficiency of Algorithm 4 in comparison with the other algorithms, in the next section we generate one hundred random test problems and compare algorithms 3 and 4 in the sense of the performance profile of Dolan and Moré [8].

4.4. Numerical results

Here, we make some numerical comparisons between our MATLAB implementation of algorithms 3 and 4. All the experiments were done on the computer, Intel(R) Core(TM) 2 Duo CPU 2.40 GHz, with 2 GB of RAM. Since the time complexity of both algorithms 3 and 4 are the same ($O(mp^2\sigma)$), to show the practical efficiency of our proposed algorithm, we compare these algorithms on one hundred randomly generated test problems and give a summary of the obtained results in the sense of Dolan and Moré's performance profile [8]. To generate random networks, we use Algorithm 5 below in MATLAB R2011b. Since we need a connected network for the *d-MC* problem, Algorithm 5 first generates a path from node 1 as the source node to node n as the sink node, and then adds up the remaining random arcs.

Algorithm 5: Generate a random connected network (using MATLAB).

```
function [M] = CreateNetwork( n, c, e, p)
M = zeros(n, n);
A = 2:n;
prevNode = 1;
for i = 1:n − 1
      t = 1 + floor(rand ∗ (n − i));
      newNode = A(t);
      M(prevNode, newNode) = 1 + floor(rand ∗ c);
      M(newNode, prevNode) = M(prevNode, newNode);
      A(t) = 0;
      A = A(A > 0);
      prevNode = newNode;
end;
edgesCount = n − 1;
for i = 1:n
for j = i + 1:n
  if edgesCount < e && M(i, j) == 0
    if rand <= p
      M(i, j) = floor(rand ∗ c);
      M(j, i) = M(i, j);
      edgesCount = edgesCount + 1;
    end;
  end
end
end;
end.
```

Since the *d-MC* problem is NP−hard and a personal computer has its limitations, we generated only random networks of medium-size with $n = 5, 6, 7, 8, 9$ as the number of nodes. We also considered 16 as the maximum number of arcs for each network and generated 20 random networks associated with each value of n. Moreover, the demand level d in each network was assigned to be a random integer-valued variable between $v/2$ and v with v being the maximum flow of the network.

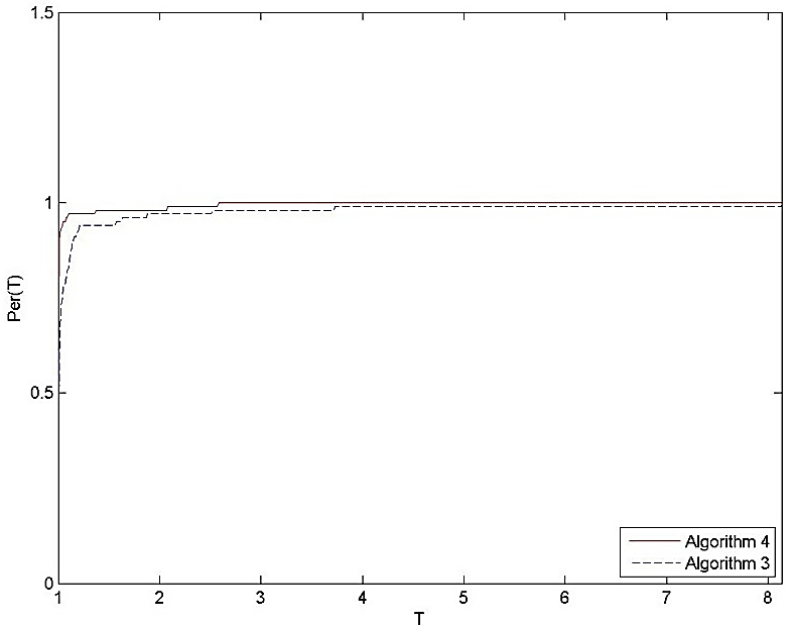

Figure 3. CPU time performance profiles for algorithms 3 and 4

To have an appropriate comparison, we made use of the performance profile of Dolan and Moré [8]. For any time T, the performance profile gives $per(T)$, as the percent of solved problems in time T by the algorithms. Using this profile, an algorithm is preferred to another one when its performance graph falls above the other [8]. Figure 3 gives a diagram of the performance profile. We considered a 100×2 matrix in which column 1 and 2 have all the running times of algorithms 1 and 2, respectively. We divided each row into the minimum number in the row and obtained a new matrix of running times. Then, we considered the maximum number in the new matrix as the length of the horizontal axis. The vertical axis gives the percentage of solved problems in the desirable time. In Figure 3, the horizontal axis shows that Algorithm 4 solved some problems 8 times faster than Algorithm 3, and the vertical axis illustrates that Algorithm 4 solved more than 60 percent of the test problems faster than Algorithm 3. As a result, Algorithm 4 has shown to be more efficient than Algorithm 3.

5. Conclusion

Evaluating reliability of a network flow or determining probability that the maximum flow of network is not less than a specified demand level d can be done in terms of d-MinCuts (d-*MC*s). Several algorithms were proposed in the literature to search for and determine all upper boundary points in network flows. Here, some published studies were investigated and a number of flaws in their results were pointed out. In addition, the correct versions of the results were given and a modified algorithm correcting an existing incorrect algorithm was presented. The efficiency of the modified algorithm was established and compared to other existing algorithms using the Dolan and Moré performance profile [8] on the numerical results obtained over randomly generated test problems. Moreover, the time complexities given for some existing algorithms were shown to be incorrect and the correct time complexities were established.

Acknowledgements

The authors thank Iran Telecommunication Research Center for its support.

References

[1] Aggarwal, K.K., Gupta, J.S. and Misra, K.B. (1975), A simple method for reliability evaluation of a communication system, *IEEE Transactions on Communications*, COM-23, 563-566.
[2] Ahuja, R.K., Magnanti, T.L. and Orlin, J.B. (1993), Network flows: theory, algorithms, and applications, Englewood Cliffs, Prentice-Hall International, New Jersey.
[3] Balan, A.O. and Traldi, L. (2003), Preprocessing min paths for sum of disjoint products, *IEEE Transactions on Reliability*, 52(3), 289-295.
[4] Ball, M.O. (1986), Computational complexity of network reliability analysis: an overview, *IEEE Transactions on Reliability*, 35, 230-239.
[5] Clancy, D.P., Gross, G. and Wu, F.F. (1983), Probability flows for reliability evaluation of multi area power system interconnections, *Electrical Power and Energy Systems*, 5, 100-114.
[6] Cook, J.L. and Ramirez-Marquez, J.E. (2007), Two-terminal reliability analyses for a mobile ad hoc wireless network, *Reliability Engineering and System Safety*, 92(6), 821-829.
[7] Crespo, L.G., Giesy, D.P. and Kenny, S.P. (2009), Reliability-based analysis and design via failure domain bounding, *Structural Safety*, 31(4), 306-315.
[8] Dolan, E.D. and Moré, J.J. (2002), Benchmarking optimization software with performance profiles, *Mathematical Programming*, 91(2, Ser. A), 201-213.
[9] Doulliez, P. and Jalnoulle, E. (1972), Transportation network with random arc capacities, *R.A.I.R.O.*, 3, 45-60.
[10] Ford, L.R. and Fulkerson, D.R. (1962), Flow in networks, Princeton University Press, Princeton.
[11] Forghani-elahabad, M. and Mahdavi-Amiri, N. (2010), Finding all the upper boundary points of a stochastic-flow network with budget constraint, *The CSI Journal on Computer Science and Engineering*, 8 (2 & 4(b)), 42-50.
[12] Forghani-elahabad, M. and Mahdavi-Amiri, N. (2013), A simple efficient approach for the d-MinCut problem, *In Proceedings of International Conference on Operations Research and Optimization*, January 19-23, 2013, University of Tehran, Tehran, Iran, 72-75.
[13] Forghani-elahabad, M. and Mahdavi-Amiri, N. (2013), Multi-commodity Unreliability Evaluation for a Stochastic-Flow network, *In Proceedings of the 6th International Conference of Iranian Operations Research Society*, May 8-9, 2013, Tehran, Iran, 172-73.
[14] Hayhurst, K.J. and Shier, D.R. (1991), A factoring approach for the stochastic shortest path problem, *Operations Research Letters*, 10, 329-334.
[15] Jane, C.C., Lin, J.S. and Yuan, J. (1993), Reliability evaluation of a limited-flow network in terms of minimal cut sets, *IEEE Transactions on Reliability*, R-42, 354-361.
[16] Jane, C.C., Shen, W.H. and Laih, Y.W. (2009), Practical sequential bounds for approximating two-terminal reliability, *European Journal of Operational Research*, 195(2), 427-441.
[17] Levitin, G., Xie, M. and Zhang, T. (2007), Reliability of fault-tolerant systems with parallel task processing, *European Journal of Operational Research*, 177, 420-430.
[18] Lin, Y.K. (2002), Using minimal cuts to evaluate the system reliability of a stochastic-flow network with failures at nodes and arcs, *Reliability Engineering and System Safety*, 75, 41-46.
[19] Lin, Y.K. and Yeh, C.T. (2011), Maximal network reliability for a stochastic power transmission network, *Reliability Engineering and System Safety*, 96, 1332-1339.

[20] Locks, M.O. (1985), Recent developments in computing of system-reliability, *IEEE Transactions on Reliability*, R-34(5), 425-436.
[21] Niu, Y.F. and Shao, F.M. (2011), A practical bounding algorithm for computing two-terminal reliability based on decomposition technique, *Computers and Mathematics with Applications*, 61, 2241-2246.
[22] Salehi-Fathabadi, H. and Forghani-elahabad, M. (2009), A note on "A simple approach to search for all *d-MC*s of a limited-flow network", *Reliability Engineering and System Safety*, 94, 1878-1880.
[23] Shier, D. (1991), Network reliability and algebraic structures, Clarendon Press, New York.
[24] Wu, W.W., Ning, A. and Ning, X.X. (2008), Evaluation of the reliability of transport networks based on the stochastic flow of moving objects, *Reliability Engineering and System Safety*, 93, 838-844.
[25] Xue, J. (1985), On multistate system analysis, *IEEE Transactions on Reliability*, R-34, 329-337.
[26] Yan, Z. and Qian, M. (2007), Improving efficiency of solving *d-MC* problem in stochastic-flow network, *Reliability Engineering and System Safety*, 92, 30-39.
[27] Yang, Q. and Chen, Y. (2011), Monte Carlo methods for reliability evaluation of linear sensor systems, *IEEE Transactions on Reliability*, 60(1), 305-314.
[28] Yeh, W.C. (2001), A simple approach to search for all *d-MC*s of a limited-flow network, *Reliability Engineering and System Safety*, 71(2), 15-19.
[29] Yeh, W.C. (2002a), Search for all *d*-Mincuts of a limited-flow network, *Computers and Operations Research*, 29(13), 1843-1858.
[30] Yeh, W.C. (2002b), A new approach to the *d-MC* problem, *Reliability Engineering and System Safety*, 77(2), 201-206.
[31] Yeh, W.C. (2004), A simple *MC*-based algorithm for evaluating reliability of stochastic-flow network with unreliable nodes, *Reliability Engineering and System Safety*, 83(1), 47-55.
[32] Yeh, W.C. (2008), A fast algorithm for searching all multi-state minimal cuts, *IEEE Transactions on Reliability*, 57(4), 581-588.