Computing the probability mass function of the maximum flow through a reliable network

Megha Sharma
Diptesh Ghosh

W.P. No. 2009-10-01
October 2009

INDIAN INSTITUTE OF MANAGEMENT
AHMEDABAD – 380015
INDIA

Computing the probability mass function of the maximum flow through a reliable network

Megha Sharma[1]
Diptesh Ghosh[2]

Abstract

In this paper we propose a fast state-space enumeration based algorithm called TOP-DOWN to compute the probability mass function of the maximum s-t flow through reliable networks. The algorithm computes the probability mass function in decreasing order of maximum s-t flow values in the network states. This order of enumeration makes this algorithm attractive for reliable networks in which the link reliabilities are high, e.g., in telecommunication networks. We present our computational experience with the TOP-DOWN algorithm and a limited comparison with a path-based exact algorithm and show that the TOP-DOWN algorithm solves problems much faster and is thus able to handle much larger problems than such algorithms.
Keywords: Network Flows; Reliable Networks; Maximum Flows; Exact Computation

1 Introduction

A network N is defined as $N = (V, A, s, t)$, where $V = \{1, 2, \ldots, n\}$ is a finite set of nodes and $A \subseteq V \times V$ is a set of arcs connecting nodes in V, $s \in V$ is a pre-defined source node from which flow in the network originates, and $t \in V, t \neq s$ is a pre-defined terminal node at which flow in the network terminates (see e.g., Ahuja et al. [3]). An arc $a \in A$ conventionally has two properties; a strictly positive capacity value describing the maximum amount of flow that can be sent along that arc, and a cost value describing the cost of sending a unit of flow through the arc. Flows originating at s and terminating at t are called s-t flows. A common network flow problem is the maximum flow problem, in which one is required to determine the maximum amount of s-t flow that can be sent through a given network.

While modeling practical networks (for e.g., telecommunication networks, logistic networks etc.), one needs to account for the fact that network components become non-functional from time to time. This is done by adding a reliability value p_i to each arc $a_i \in A$, which is the probability that the arc is functional at a given point in time. Networks which incorporate such reliability values for their arcs are called reliable networks. A reliable network is thus represented as $N = (V, A, s, t, P)$, where $P : A \rightarrow [0, 1]^{|A|}$.

A reliable network exists in one of $2^{|A|}$ network states, where each state is defined by a partition of arcs in A into functional and non-functional arcs. The probability of the network being in any particular state is a function of the reliability values of the arcs. Maximum s-t flow values may differ from state to state, and hence become random variables in reliable networks.

There is considerable literature on the performance evaluation of reliable networks in terms of maximum s-t flow. Such networks are often referred to in the literature as stochastic flow networks. Various measures are used to quantify the performance of a stochastic flow network. The most popular of these measures is the expected maximum flow through the network (see, e.g., Ball et

[1]P&QM Area, IIM Ahmedabad. Email: meghas@iimahd.ernet.in
[2]P&QM Area, IIM Ahmedabad. Email: diptesh@iimahd.ernet.in

al. [5]). Another popular measure is obtained by defining a threshold value of flow, and finding the probability that the network will be able to carry at least that much *s-t* flow (see, e.g., Lee [7], Aggarwal et al. [1]). The problems of computing both of these measures are NP-hard (see, Provan and Ball [9]), so both exact and approximate approaches to compute these measures exist in the literature. However, all evaluation methods for computing expected maximum *s-t* flows need to compute (or estimate) the probability mass function of the maximum *s-t* flow in a given reliable network. This paper presents an algorithm for the exact computation of the probability mass function for the maximum *s-t* flow.

Exact approaches for computing the probability mass function of the maximum *s-t* flow in order to evaluate the performance of stochastic flow networks can be broadly classified into two streams; path based and cutset based methods (see Patra and Misra [8]), and state-space enumeration based methods (see Alexopoulos [4]). The problem with path based and cutset based methods is that the number of *s-t* paths in a network as well as the number of minimal *s-t* cuts can both be exponential in the number of arcs (see, e.g., Ball et al. [5]), so that constructing the set of all combinations of all paths required in path based algorithms as well as the set of all combination of all minimal cuts in cutset based algorithms are both doubly exponential. Hence these methods are not useful for finding the distribution of the maximum *s-t* flow of reasonably sized networks.

The basic idea behind state-space enumeration based method is to generate all the possible network states and for each network state, to determine the maximum *s-t* flow in that state, and the probability that the network will be in that state. The first state-space enumeration based method for stochastic flow networks was introduced by Douilliez and Jamoulle [6] and was later implemented by Alexopolous [4]. It was implemented for multi-state networks, where the capacity of an arc can take multiple values with pre-defined probabilities, and given a threshold value of flow as input, computed the probability that the maximum *s-t* flow through the network is at least as high as that value. Although the algorithm works efficiently for multi-state networks, it reduces to a basic branch and bound routine for bi-state networks whose arcs can only assume two states, a functional state, and a non-functional state. Further, in order to use this algorithm to compute the probability mass function of the maximum *s-t* flow through a network, one needs to repeatedly call this algorithm with different threshold values of *s-t* flow. These multiple calls to the algorithm with different threshold values result in a lot of duplication of computational effort, rendering the algorithm an unattractive choice for determining expected maximum *s-t* flows and for computing the probability mass function of maximum *s-t* flow through a network. In this paper, we present an algorithm called TOP-DOWN for computing the distribution of maximum *s-t* flow through a reliable network. Our algorithm is based on lexicographic depth-first enumeration of the state-space.

In this paper, we assume that arc failures are independent of each other. We also assume that the network has arcs with high reliability, between 0.9 and 1.0. This is a reasonable assumption for many real life applications, e.g. telecommunication networks. In such networks, a relatively small proportion of network states contribute a major part to the whole distribution. This is shown in Table 1 on a network with 25 arcs each having a reliability of 0.9. Notice that probability of the network being in a state in which 19 or more arcs are functional is more than 99% although those states account for only 0.73% of the total number of states.

Taking advantage of this fact, the TOP-DOWN algorithm starts with the highest possible maximum flow and computes the probability that the network would allow this flow. Once, the probability of the highest flow is computed, the algorithm computes the probability of the next highest flow and so on, until it satisfies a stopping criterion. In the process, given a suitable stopping rule, it is capable of generating the complete probability mass function of the maximum *s-t* flow in the network. If not allowed adequate time, for networks with high arc reliabilities, it quickly generates a large portion of the probability mass function.

The paper is organized as follows. In Section 2, we present our algorithm to compute the distribution of maximum *s-t* flow through the network exactly. In Section 3, we present the results of our algorithm on randomly generated test bed and compare its performance with that of a path

Table 1: Network characteristics for a reliable network with 25 arcs of reliability 0.9

Number of functional arcs	Probability of occurrence	Percentage of total number of states
0	$< 5 \times 10^{-5}$	$< 5 \times 10^{-5}\%$
1	$< 5 \times 10^{-5}$	0.0001%
2	$< 5 \times 10^{-5}$	0.0009%
3	$< 5 \times 10^{-5}$	0.0069%
4	$< 5 \times 10^{-5}$	0.0377%
5	$< 5 \times 10^{-5}$	0.1583%
6	$< 5 \times 10^{-5}$	0.5278%
7	$< 5 \times 10^{-5}$	1.4326%
8	$< 5 \times 10^{-5}$	3.2233%
9	$< 5 \times 10^{-5}$	6.0885%
10	$< 5 \times 10^{-5}$	9.7417%
11	$< 5 \times 10^{-5}$	13.2841%
12	$< 5 \times 10^{-5}$	15.4981%
13	$< 5 \times 10^{-5}$	15.4981%
14	$< 5 \times 10^{-5}$	13.2841%
15	0.0001	9.7417%
16	0.0004	6.0885%
17	0.0018	3.2233%
18	0.0072	1.4326%
19	0.0239	0.5278%
20	0.0646	0.1583%
21	0.1384	0.0377%
22	0.2265	0.0069%
23	0.2659	0.0009%
24	0.1994	0.0001%
25	0.0718	$< 5 \times 10^{-5}\%$

based method. Finally in Section 4, we summarize our findings and point out directions for future research.

2 The TOP-DOWN Algorithm

In order to describe our algorithm formally, we first introduce some terminology that is used in the rest of the paper.

A *network state* (or simply, state) of a reliable network $N = (V, A, s, t, P)$ is a representation of its functional part at any point in time. It is represented by a network $N_S = (V, A_S, s, t)$ where $A_S \subseteq A$ is the set of arcs which are functional in that state. A reliable network attains one of the possible $2^{|A|}$ states at any point in time. The probability of observing N in state N_S is given by

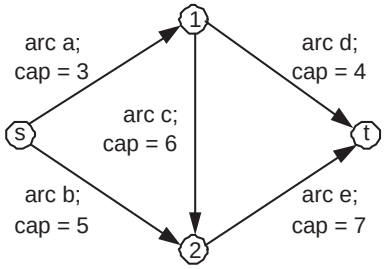
$$\prod_{a_i \in A_S} p_i \prod_{a_j \in A \setminus A_S} (1 - p_j),$$

where p_i is the reliability of arc a_i for each $a_i \in A$.

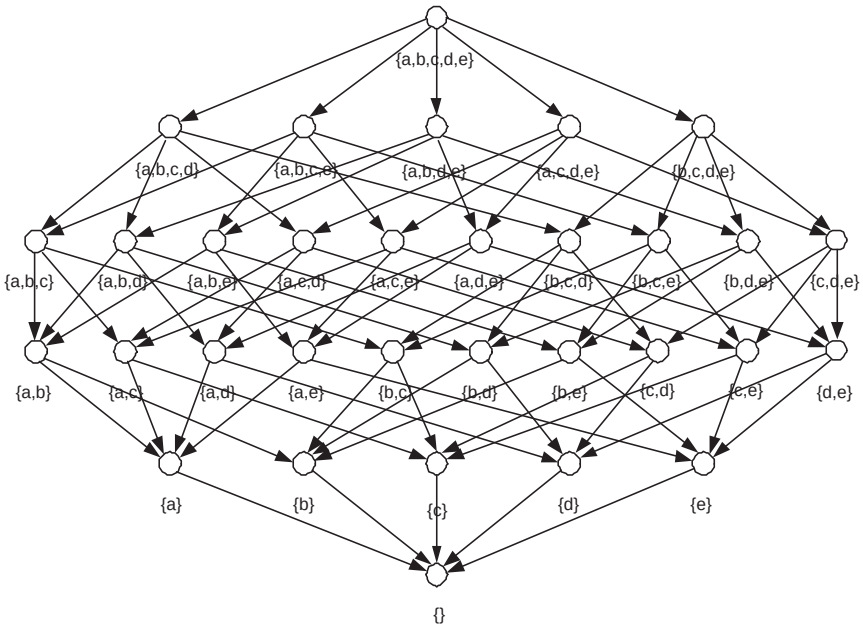
A *Hasse diagram* for the network N based at state $N_K = (V, A_K, s, t)$ is a directed graph with $2^{|A_K|}$ nodes, where a node, say j, represents a state $N_j = (V, A_j, s, t)$ of the network such that $A_j \subseteq A_K$, and an arc connects the node representing state $N_{S_1} = (V, A_{S_1}, s, t)$ to the node

representing $N_{S_2} = (V, A_{S_2}, s, t)$ if and only if A_{S_2} is obtained by deleting one arc from A_{S_1}. Network state N_K is called the *base* of the Hasse diagram.

Example 1 Figure 1 shows a network with five arcs and its Hasse diagram based at the state $(\{1, 2, s, t\}, \{a, b, c, d, e\}, s, t)$. In the network, node s represents the source node and node t represents the terminal node. In the Hasse diagram, the node corresponding to the state $N_S = (V, A_S, s, t)$ is labeled by the arc set A_S. ∎



(a) A five arc network



(b) The Hasse diagram of the network based at $(\{1, 2, s, t\}, \{a, b, c, d, e\}, s, t)$

Figure 1: A network and its Hasse diagram

The TOP-DOWN algorithm to compute the probability mass function of maximum s-t flow through the network is a state-space enumeration algorithm. It outputs a list called pmf which contains tuples of the form (f_i, p_i) where f_i is a flow value, and p_i is the probability that the input reliable network will be in a state that allows a maximum s-t flow of f_i. The basic idea behind this algorithm is the following. Given a reliable network $N = (V, A, s, t, P)$, TOP-DOWN finds out the maximum s-t flow possible through (V, A, s, t). It then identifies all network states for which the maximum s-t flow has this value. It does this by searching the Hasse diagram for the network, based at state (V, A, s, t). During the search, a child state is generated from a parent state by removing an arc from the arc set of the parent state, and backtracking is achieved be replacing the arc that was removed. Since the Hasse diagram is a graph and not a tree, the TOP-DOWN algorithm performs a lexicographic depth-first search on states represented in the Hasse diagram.

For this, the arcs in the reliable network are arranged as follows. Each arc is deleted in turn from the network (V, A, s, t) and the maximum s-t flow is computed in the resulting network. This flow is taken as a score corresponding to the arc deleted. The arcs are then ordered in non-decreasing order of the scores. Let the ordering of the arcs after this arrangement be $(a_{[1]}, a_{[2]}, \ldots, a_{[|A|]})$. If during the search, a network state $N_j = (V, A_j, s, t)$ is reached by removing arc $a_{[k]}$ from its parent state, then while examining child states of N_j, TOP-DOWN only looks at states that can be obtained by removing one of the arcs $a_{[k+1]}$ through $a_{[|A|]}$ from N_j. This prevents any network state from being examined more than once by the TOP-DOWN algorithm.

The TOP-DOWN algorithm operates by maintaining several lists. The pmf list is a list of tuples of the form (f_i, p_i) which stores the (partial) probability mass function that the algorithm computes. In each tuple, f_i denotes a value of maximum s-t flow, and p_i denotes the probability that the network will be in a state for which the maximum s-t flow is f_i. While examining states with a maximum s-t flow of f_i, the *enum* list stores network states in which the maximum s-t flow is f_i, and $lowerflow$ list stores a subset of the network states in which the maximum s-t flow is lower than f_i. Periodically TOP-DOWN removes states from $lowerflow$ in which the maximum s-t flow is the highest among the states in $lowerflow$, and puts them in a list called *candidate* and clears *enum*.

Suppose the TOP-DOWN algorithm starts the lexicographic search at a network state N_i in the *candidate* list, which allows a maximum s-t flow of f_i. There are two outcomes possible when TOP-DOWN evaluates the maximum s-t flow in a state N_j. Either the maximum s-t flow allowed in the state is f_i, or it is lower. In the former case, the TOP-DOWN algorithm adds the state to *enum* and continues to search the appropriate child states of N_j. In the latter case, the algorithm does not search the child states of N_j and copies N_j to $lowerflow$ for possible future processing.

Once TOP-DOWN has finished exploring all states in *candidate*, it computes the cumulative probability of occurrence of all states in *enum*. This is the probability that the network will be in a state which allows a maximum s-t flow of f_i. The algorithm then checks the network states in $lowerflow$, picks states which admit the highest value of maximum s-t flow among them and transfers them to another list called *candidate*. Assuming that the termination condition for the TOP-DOWN algorithm is not reached, it performs another iteration, exploring states in *candidate*.

An algorithm such as the one described above can be terminated through various stopping criteria. For example, it can be terminated once all the states of the network have been evaluated. In this case, the algorithm outputs the complete probability mass function of the maximum s-t flow through the reliable network. It can also be terminated once it has run for a pre-specified amount of time. We use the following criterion to terminate the TOP-DOWN algorithm. We specify a value p, such that $0 \leq p \leq 1$ to TOP-DOWN and terminate the algorithm when it has obtained the top $100p\%$ of the distribution of maximum s-t flows. We believe that such a stopping rule makes the TOP-DOWN algorithm attractive when evaluating large networks in the absence of adequate computation time. The remaining portion of the probability mass function of the maximum s-t flow can be estimated by other means, for example, through sampling. We formally describe the TOP-DOWN algorithm in Algorithm 1.

An enhancement that significantly improves the performance of the TOP-DOWN algorithm is the use of a warmstart mechanism (see Sharma and Ghosh [11]) in Step 2 of the TOP-DOWN algorithm to compute the maximum s-t flow in states that it explores during the lexicographic search. The basic idea behind warmstart is to not compute the maximum s-t flow afresh for all network states, but to use the residual network obtained after computing the maximum s-t flow in another state with a large number of common arcs to determine the maximum flow in a state. Consider two network states $N_i = (V, A_i, s, t)$ and $N_j = (V, A_j, s, t)$ such that $A_i = A_j \cup \{a\}$. Assume that the maximum s-t flow has been computed in N_i as f_i, and the residual network after this computation is N_i^r. If there is no flow through arc a in N_i^r, then the maximum s-t flow in state N_j is clearly f_i and the residual network N_j^r is obtained by deleting arc a from N_i^r. If a transmits a flow, say f_a in N_i^r, then the arc a is removed from N_i^r. f_a is added as a positive excess flow to the tail node of

Algorithm 1 The TOP-DOWN algorithm

Input: A reliable network $N = \{V, A, s, t\}$; p, $0 \leq p < 1$.

Output: The top $100p\%$ of the probability mass function of the maximum s-t flow in N.

Steps:

Step 1: For each arc in $a_i \in A$, compute $t_i \leftarrow$ maximum s-t flow in $(V, A \setminus \{a_i\}, s, t)$. Arrange arcs in A in non-increasing order of t_i values. (This is the ordering of arcs used in the lexicographic search in Step 2.) Set $pmf \leftarrow \{\}$, $lowerflow \leftarrow \{\}$, $f_i \leftarrow$ maximum s-t flow allowed in N and $candidate \leftarrow (V, A, s, t)$. Go to Step 2.

Step 2: For each state $N_S \in candidate$, perform a lexicographic depth-first search on the Hasse diagram starting at the node representing N_S in the Hasse diagram. Child states are generated by deleting a single arc from the arc set of the parent state following the lexicographic search order. The maximum s-t flow is computed for the child state. If it is equal to f_i, then the child state is copied to the *enum* set and its children are explored by the search. If it is less than f_i, then the state is added to $lowerflow$. Go to Step 3.

Step 3: If $\sum_{i:(f_i,p_i)\in pmf} p_i \geq p$, go to Step 4. Else, set $p_i \leftarrow$ the probability of N being in any of the states in *enum*, and $pmf \leftarrow pmf \cup \{(f_i, p_i)\}$. Set $f_i \leftarrow$ maximum value of the maximum s-t flow allowed in any state in $lowerflow$; $candidate \leftarrow \{N_S :$ maximum s-t flow in $N_S = f_i\}$, $enum \leftarrow \{\}$. Go to Step 2.

Step 4: Choose $(f_m, p_m) \in pmf$ such that $f_m = \max\{f_i : (f_i, p_i) \in pmf\}$. Let $\underline{p} = \sum_{i:(f_i,p_i)\in pmf} p_i - p_m$. Replace (f_m, p_m) with $(f_m, p - \underline{p})$ in pmf. Output pmf and terminate.

a and as a negative excess flow to the head node of a. Then using an augmenting path algorithm, an attempt is made to redirect the excess flow from the tail node to the head node. This can lead to one of two situations; either all the excess flow can be re-routed, or only a portion f, $0 \leq f < f_a$ can be re-routed. If all the excess flow is re-routed, then then the maximum s-t flow in N_j remains f_i, and the residual network after the re-routing is the correct residual network N_j^r. If only f units of flow can be re-routed, then this re-routing is first achieved. Next a flow of $(f_a - f)$ is routed from the tail of a to s, and the same amount of flow is routed from t to the head of a. The maximum s-t flow in N_j is $f_i - (f_a - f)$. The residual network for state N_j is the residual network obtained after all the re-routing is executed in N_i^r. From our experiments we have observed that this warmstarting process is particularly effective when evaluating the maximum s-t flow in network states for large networks when both states have a lot of functional arcs in common.

We illustrate the working of the TOP-DOWN algorithm on the reliable network shown in Figure 1(a) in the Example 2.

Example 2 Consider the network shown in Figure 1(a). Assume that all arcs have a reliability of 90%. On this network, the TOP-DOWN algorithm first arranges the arcs in the sequence (b, e, a, d, c). Using this sequence of arcs, the lexicographic search tree corresponding the Hasse diagram for the network based at (V, S, s, t), where $V = \{1, 2, s, t\}$ and $A = \{a, b, c, d, e\}$, is shown in Figure 2.

TOP-DOWN starts by evaluating the maximum flow in the the state (V, A, s, t) as 8 units. It then tries to find the states in the Hasse diagram which have a flow of 8 units. The search tree it generates is shown in Figure 3. The numbers on each of the nodes denote the maximum s-t flow possible in the states that they represent. The nodes colored black correspond to the states that have a flow of 8 units. The states corresponding to these nodes are added to the *enum* list. The nodes colored gray correspond to the states which are connected to states having flow of 8 units, but

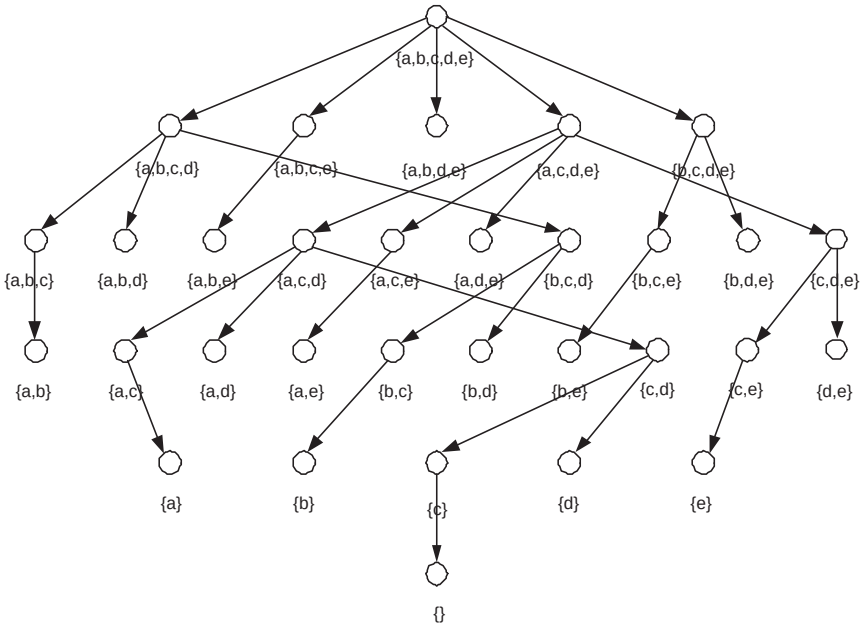
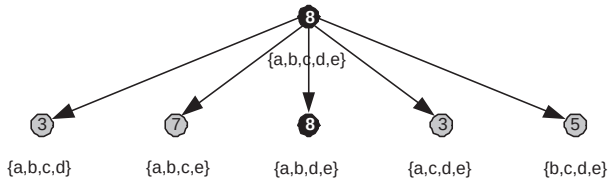


Figure 2: The lexicographic search tree corresponding to the network of Figure 1(a)



pmf after Step 1 is $\{(8, 0.6561)\}$

Figure 3: The search tree generated in the first iteration of the TOP-DOWN algorithm

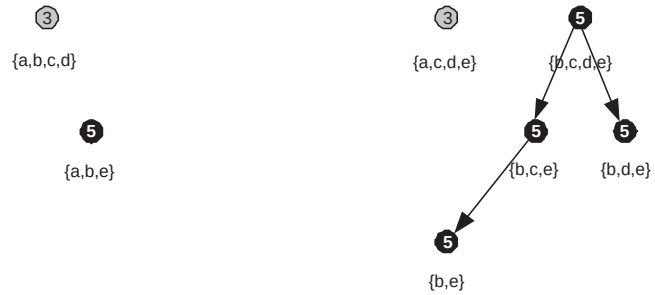
which have a flow of less than 8 units. These are put in $lowerflow$ at the end of the iteration. At the end of the iteration state $(V, \{a, b, c, e\}, s, t)$ is removed from $lowerflow$ and added to *candidate*.

Figure 4 presents the search tree explored in the remaining iterations that the TOP-DOWN algorithm executes while computing the probability mass function of the network in Figure 1(a). The contents of the pmf list storing the probability mass function are also presented in the figure.

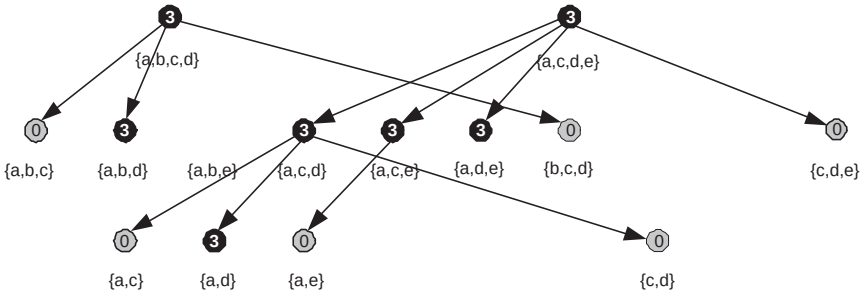
3 Computational experience

We performed computational experiments with the TOP-DOWN algorithm, and present the results of our experiments in this section. We did not find any set of benchmark instances for reliable networks in the literature, and hence generated random layered and random grid networks, two types of random networks suggested as difficult networks for maximum s-t flow problems in Ahuja et al. [2]. Random layered networks are acyclic, while random grid networks include directed cycles. We provide a detailed description of these networks in this section. We also compare the performance of the TOP-DOWN algorithm with PATRA, an algorithm proposed by Patra and Misra [8]. The PATRA algorithm is based on enumerating all the s-t paths in the network and is designed for acyclic networks. So the comparison between TOP-DOWN and PATRA is carried out only on random layered networks.

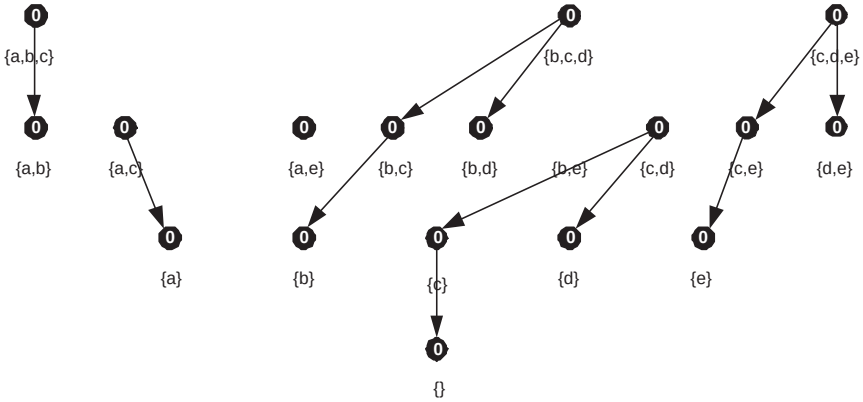
pmf after Step 2 is $\{(8, 0.6561), (7, 0.0656)\}$



pmf after Step 3 is $\{(8, 0.6561), (7, 0.0656), (5, 0.0883)\}$



pmf after Step 4 is $\{(8, 0.6561), (7, 0.0656), (5, 0.0883), (3, 0.1612)\}$



pmf after Step 5 is $\{(8, 0.6561), (7, 0.0656), (5, 0.0883), (3, 0.1612), (0, 0.0288)\}$

Figure 4: The search tree generated in the next four iterations of the TOP-DOWN algorithm

Random layered networks are networks characterized by three parameters, the width (W) of the network, the length (L) of the network, and the outdegree (K) of all nodes in the network except the source and terminal nodes. The number of nodes in such a network is $n = LW + 2$. All nodes other than the source node s and the terminal node t are arranged in L layers, each containing W nodes. The source node s is connected to all the nodes in the first layer, and all the nodes in the last layer are connected to the terminal node t. Each node in a particular layer, except the last layer, is connected to a random set of K nodes in the next layer.

Random grid networks are networks characterized by two parameters, the width (W) of the network and the length (L) of the network. The number of nodes in such a network is $n = LW + 2$. The nodes except the source node s and the terminal node t are arranged in a rectangular $L \times W$ grid. The source node s is connected to all the nodes in the first column of the grid, and all the nodes in the last column are connected to the terminal node t. The node in the ith row and jth column is connected to the nodes in the $i-1$th and $i+1$th rows in column j, and to the nodes in the $i-1$th, ith, and $i+1$th rows in column $j+1$ if such nodes exist.

For both these types of networks, arcs connecting from s or to t are assigned capacities randomly as integers in the range $[50000, 100000]$. For the remaining arcs capacities are assigned randomly as integers in the range $[500, 10000]$. All arcs are assigned reliabilities randomly in the range $[0.9, 1.0]$.

We used nine network configurations of each of the two types of networks in our experiments. A network configuration for a random layered network is a tuple of the form (L, W, K), while that for a random grid network is a tuple of the form (L, W). The details of the configurations used in our experiments are given in Table 2.

Table 2: Structural configurations of test problems

Type of network	Configuration (W, L, K)	Number of nodes	Number of arcs
Layered	(3, 4, 2)	14	24
	(3, 5, 2)	17	30
	(3, 6, 2)	20	36
	(4, 6, 2)	26	48
	(4, 5, 3)	22	56
	(4, 6, 3)	26	68
	(4, 7, 3)	30	80
	(4, 8, 3)	34	92
	(5, 11, 2)	57	110
Grid	(2, 3)	8	18
	(2, 5)	12	30
	(2, 6)	14	36
	(3, 4)	14	43
	(3, 5)	17	54
	(3, 6)	20	65
	(3, 7)	23	76
	(4, 6)	26	94
	(4, 7)	30	110

For each configuration, we generated 20 random instances. The results reported in this paper are results averaged over these 20 instances. For each instance, we kept $p = 1$ for TOP-DOWN so that the algorithm can compute the entire probability mass function. However since most of these instances are quite large, the times taken to generate the complete probability mass functions are prohibitively large. So we set an additional execution time-based stopping criterion that terminated

Table 3: Performance of the TOP-DOWN algorithm on random layered networks

Configuration (W,L,K)	No. of Nodes	No. of Arcs	Fraction of the pmf* generated Mean	SD	Execution time (in sec.) Mean	SD
(3,4,2)	14	24	1.0000	0.0000	23.89	9.57
(3,5,2)	17	30	0.9937	0.0043	600.02	0.00
(3,6,2)	20	36	0.8924	0.0405	600.02	0.00
(4,6,2)	26	48	0.4753	0.0859	600.02	0.00
(4,5,3)	22	56	0.3419	0.0326	600.02	0.00
(4,6,3)	26	68	0.1545	0.0278	600.02	0.00
(4,7,3)	30	80	0.0815	0.0184	600.02	0.00
(4,8,3)	34	92	0.0441	0.0078	600.02	0.00
(5,11,2)	57	110	0.0175	0.0052	600.02	0.00

$\star$: probability mass function

TOP-DOWN after running for 600 seconds. The experiments were run on a Dell machine with Core 2 Quad processors with 3GB 800MHz Dual channel DDR2 SDRAM running Linux. The algorithms were written in C and compiled with a `gcc 4.3` compiler.

Table 3 summarizes the results from our experiments with the TOP-DOWN algorithm on layered networks. The first column of the table refers to the configuration of the instances for which results are tabulated. Columns 2 and 3 provide the number of nodes and the number of arcs in networks with the given configuration. Columns 4 and 5 report data about the fraction of the probability mass function of the maximum s-t flow that the algorithm could compute within the stipulated time, with column 4 reporting the average and column 5 reporting the standard deviation over all 20 instances with that configuration. For example, among the 20 instances with configuration (3,5,2) i.e., with 17 nodes and 30 arcs, the TOP-DOWN algorithm could, on average, compute the top 99.37% of the complete probability mass function of the maximum s-t flow, and the standard deviation of these probability values across the 20 instances was observed to be 0.43%. Columns 5 and 6 provide the average and standard deviation of the execution time taken for the 20 instances. For the same configuration, i.e. (3,5,2), we see that the average of the execution times was 600.02 seconds and the standard deviation was 0, implying that TOP-DOWN could not compute the complete probability mass function of any of the instances within the stipulated time.

From the table, we observe that the TOP-DOWN algorithm obtains the complete probability mass function of the maximum s-t flow in only the set of instances with the smallest number of arcs. As the number of arcs in the problem instance increases, the percentage of the total probability mass function that the algorithm could compute reduces exponentially.

Since random layered networks are acyclic, they could also potentially be solved by the PATRA algorithm. We input these instances to the PATRA algorithm, and ran it with the same stopping rules as the TOP-DOWN algorithm. We found out for the smallest of the configurations, i.e. configuration (3,4,2) with 14 nodes and 24 arcs, the PATRA algorithm could only obtain 58.34% of the complete probability mass function of the maximum s-t flows on average with a standard deviation of 23.07%. The average time taken was 600.63 seconds with a standard deviation of 0.97 seconds. The PATRA algorithm could obtain the complete probability mass function for only two of the 20 instances. For configurations that resulted in larger networks, the PATRA algorithm could not compute all combinations of all the s-t paths that it generated within the stipulated time limit,and hence could not compute any part of the probability mass function of the maximum s-t flow in these networks. Thus the TOP-DOWN algorithm is clearly superior to the PATRA algorithm.

The results of our computational experiments with the TOP-DOWN algorithm on random grid networks is presented in Table 4. The structure of the table is identical to that of Table 3. The

Table 4: Performance of the TOP-DOWN algorithm on random grid networks

Configuration (W,L)	No. of Nodes	No. of Arcs	Fraction of the pmf⋆ generated Mean	SD	Execution time (in sec.) Mean	SD
(2,3)	8	18	1.0000	0.0000	0.21	0.03
(2,5)	12	30	0.9907	0.0055	600.02	0.00
(2,6)	14	36	0.8490	0.0422	600.02	0.00
(3,4)	14	43	0.6373	0.0737	600.02	0.00
(3,5)	17	54	0.3393	0.0629	600.02	0.00
(3,6)	20	65	0.1832	0.0220	600.02	0.00
(3,7)	23	76	0.1010	0.0160	600.02	0.00
(4,6)	26	94	0.0408	0.0106	600.02	0.00
(4,7)	30	110	0.0165	0.0057	600.02	0.00

⋆: probability mass function

description of the problems being solved is provided in the first three columns of the table, the fourth and fifth columns present data about the fraction of the probability mass function that the TOP-DOWN algorithm could compute within the specified time limit of 600 seconds, and columns 6 and 7 give details about the execution times taken. Here too we observe that the TOP-DOWN algorithm could obtain the complete probability mass function in the set containing the smallest of the instances, and for larger instances, the percentage of the probability mass function that the algorithm could compute reduces exponentially with problem size.

Since the random grid instances contain cycles, the PATRA algorithm cannot be applied to compute the probability distribution functions of the maximum *s-t* flow in these instances.

4 Summary and directions for future work

In this paper, we examined the problem of computing the probability mass function of maximum *s-t* flows in reliable networks, i.e. those in which arcs are functional with a given probability. Such reliable networks are more realistic models of practical situations than conventional networks. The probability mass function of the maximum *s-t* flow is essential to compute measures to compare reliable networks. We proposed a lexicographic depth-first search algorithm called TOP-DOWN that enumerates the states of the network. Noting that arc reliabilities are high in many practical networks, e.g., telecommunication networks, our proposed algorithm searches the state-space of reliable networks starting from a state in which all arcs are functional. Because of this, even if it is prematurely terminated, TOP-DOWN can output a large fraction of the probability mass function of maximum *s-t* flows in reasonably sized networks. The TOP-DOWN algorithm uses a warmstart mechanism to minimize the time required to compute the maximum *s-t* flows in the network states that it examines. The warmstart mechanism is one that uses the residual network obtained after examining a network state to quickly compute the maximum *s-t* flow in a state with a largely similar arc set. This is particularly useful for evaluating the probability mass functions in high performance networks, like modern telecommunication networks. We report our experience with implementing the TOP-DOWN algorithm and running it on randomly generated networks. Based on our experience, we conclude that the TOP-DOWN algorithm is far more effective than existing algorithms while computing the probability mass functions of reliable networks.

There are several directions in which this research can be taken forward. Our computational experience showed us that the TOP-DOWN algorithm, while much faster than existing algorithms, is not capable of computing the complete probability mass function for large networks. So there

is a necessity for generating hybrid algorithms to compute the complete probability mass function for such networks. One such approach would hybridize the TOP-DOWN algorithm with an efficient simulation algorithm. Another approach could hybridize the TOP-DOWN algorithm with the COMPUTE-RISK algorithm as presented in Sharma and Ghosh [10] which computes the probability mass function starting with states that permit low *s-t* flows. A general direction of future research is in the development of similar algorithms for other network flow problems.

References

[1] K.K. Aggarwal, Y.C. Chopra, and J.S. Bajwa. Capacity consideration in reliability analysis of communication systems. *IEEE Transactions on Reliability*, 31:177–180, 1982.

[2] R.K. Ahuja, M. Kodialam, A.K. Mishra, and J.B. Orlin. Computational investigation of maximum flow algorithm. *European Journal of Operational Research*, 97:509–542, 1997.

[3] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, 1993.

[4] C. Alexopoulos. A note on state space decomposition methods for analyzing stochastic flow networks. *IEEE Transactions on Reliability*, 44:354–357, 1995.

[5] M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors. *Handbooks in Operations Research and Management Science*, volume 7, Network Models. Elsevier Science B.V., The Netherlands, 1996.

[6] P. Douilliez and E. Jamoulle. Transportation networks with random arc capacities. *R.A.I.R.O.*, 3:45–49, 1972.

[7] S.H. Lee. Reliability evaluation of a flow network. *IEEE Transactions on Reliability*, R-29:24–26, 1980.

[8] S. Patra and R. B. Misra. Evaluation of probability mass function of flow in a communication network considering a multistate model of network links. *Microelectronic Reliability*, 36(3):415–421, 1996.

[9] J.S. Provan and M.O. Ball. Computing network reliability in time polynomial in the number of cuts. *Operations Research, Reliability and Maintainability*, 32(2):516–526, 1984.

[10] M. Sharma and D. Ghosh. Evaluating downside risks in reliable networks. Technical Report IIMA Working Paper No. 2009-09-02, Indian Institute of Management Ahmedabad, India, 2009.

[11] M. Sharma and D. Ghosh. Speeding up the estimation of expected maximum flows through reliable networks. Technical Report IIMA Working Paper No. 2009-04-05, Indian Institute of Management Ahmedabad, India, 2009.