RAIRO-Oper. Res. 51 (2017) 1317–1330
DOI: 10.1051/ro/2016073

RAIRO Operations Research
www.rairo-ro.org

ALGORITHMS FOR THE QUICKEST TIME DISTRIBUTION OF DYNAMIC STOCHASTIC-FLOW NETWORKS

Chin-Chia Jane[1] and Yih-Wenn Laih[2]

Abstract. *Quickest time*, which is the least possible time necessary to transmit a fixed amount of data from a source node to a destination node, has been widely explored in the past few years. A *quickest path* transmits data *via* a single path, whereas a *quickest flow* transmits data *via* all possible paths. In a dynamic *stochastic-flow network* in which each arc capacity is a discrete random variable having several possible integer values, the quickest time is not a fixed value. Existing literature computes the reliability that the specified amount of flow can be sent simultaneously from the source to the destination through multiple k disjoint minimal paths within a given time horizon. This article presents a decomposition algorithm to compute the probability distribution of the quickest time of a dynamic stochastic-flow network from the viewpoint of flow (all disjoint and non-disjoint minimal paths simultaneously) rather than of k disjoint minimal paths only. The distribution of quickest time is important for the design and analysis of evacuation systems, as they are generally analyzed and optimized *via* the quickest flow models. As a result, the expected quickest time and the probability that the quickest time is no larger than a specified time threshold can be determined directly. The proposed algorithm can be easily modified to approximate the probability distribution by trading off accuracy for execution time when the network system is large. Computational experiments are conducted to illustrate the proposed algorithms.

Mathematics Subject Classification. 05C21, 90B15, 90B25.

Received March 20, 2016. Accepted November 1, 2016.

1. Introduction

In the fields of operations research, transportation, logistics, and applied mathematics, static network flow models have been widely employed to conduct performance analyses of real-world systems, such as shortest path problems, maximum flow problems, min-cost max-flow problems, and transportation problems [2, 10], in the past five decades. While static network flow models are useful to analyze a variety of optimization problems, they fail to capture two crucial elements of many practical problems. One is the evolution of systems over time, and the other is the transmission probabilities of flows through arcs.

To treat the evolution of systems over time, Ford and Fulkerson [10] introduced the dynamic networks in which flows take a discrete time to pass arcs of the network. The time is named *transit time*, and the flow is called *dynamic flow* or *flow over time* Let s and t be the source and sink nodes respectively. Various dynamic

Keywords. Dynamic stochastic-flow network, quickest time distribution, decomposition algorithm, reliability.

[1] Department of Business Administration, Ling Tung University 1, Lingtung road, Taichung 40852, Taiwan.
mrjane@teamail.ltu.edu.tw

[2] Department of Finance, Ling Tung University 1, Lingtung road, Taichung 40852, Taiwan. rubylai@teamail.ltu.edu.tw.

network flow problems exist. The *maximum dynamic flow problem* is the most basic; it searches a dynamic flow that maximizes the flow s to t within time horizon T. A maximum dynamic flow by every intermediate time step up to and including T is named the *earliest arrival flow* or *universal maximum flow* [3,26]. Another variant of the maximum dynamic flow problem is the *minimum cost dynamic flow problem* [9], which seeks a dynamic flow that minimizes the total shipment cost of a commodity to satisfy demands at certain nodes within time horizon T. Burkard *et al.* [5] and Lin and Jaillet [18] studied the *quickest flow problem*, which finds a feasible dynamic flow that sends a given flow value from s to $t$within the least possible time. This problem is considered the inverse of the maximum dynamic flow problem. When multiple sources and multiple sinks are present, the corresponding quickest flow problem is named the *quickest transshipment problem* [11]. Thus far, only Hoppe and Tardos [11] have proposed polynomial-time algorithms for the general quickest transshipment problem, which transforms the quickest transshipment problem into a lexicographic max-flow problem. According to Hoppe and Tardos [11], converting a quickest transshipment problem into a quickest flow problem by adding a super source and a super sink and then connecting them to the source and sink nodes cannot simplify the problem. Meanwhile, the *quickest path problem* [7,21] is a special case of the quickest flow problem because it focuses on a single path to send the flow from s to t Different quickest path problems are explored, including *constrained quickest path problem* [6], *first k quickest paths problem* [8,23], and *all-pairs quickest path problem* [16].

Algorithms for evaluating the reliability of static flow networks have been explored extensively to deal with the transmission probabilities of flows through arcs [1,12,15,17,22,24]. Assuming each arc is a binary random variable that switches between function and failure with associated probability, the two-terminal capacity reliability (2TCR) computes the probability of successfully distributing a specified demand from s to t. The 2TCR is a combination of probability connection and demand fulfillment from s to t. Transit time and transmission probability have recently been considered simultaneously. In Xue [27], Bang *et al.* [4], and Ruzika and Thiemann [25], a *most reliable quickest path* is the quickest path among the most reliable ones, a *quickest most reliable path* is the most reliable among the quickest paths, and a *reliable quickest path* is the quickest path among all paths with at least a predefined path reliability, respectively. In practical network systems the capacity of each arc is multi-valued because of failure, partial failure, maintenance, and so forth Lin [19,20] enlarged the dynamic network into a *stochastic-flow network* in which each arc capacity is a discrete random variable having several possible integer values. He computed the reliability that a specified amount of flow can be simultaneously sent from s to t through multiple k disjoint minimal paths within a given time horizon. Lin proposed an algorithm based on pre-searched k disjoint minimal paths to search all minimal capacity vectors that fulfill the requirements of flow units and time horizon. The reliability can then be calculated in terms of these minimal capacity vectors. Recently, Yeh [28] presents an algorithm to speed up the search of all minimal capacity vectors based on depth-first search.

In real-world network systems, data are transmitted in terms of flows (*i.e.*, all disjoint and non-disjoint minimal paths simultaneously). For instance, in an emergency evacuation, evacuees must move to a safety area as quickly as possible. Restricting the movements of evacuees along specified k disjoint paths only is unrealistic. This article is focused on computing the probability distribution of the least possible time needed by the network system to successfully transmit a given amount of data from s to t. The main contribution of this work is it evaluates the distribution of the least possible time of a quickest flow from the viewpoint of flow, rather than of paths, contrary to what most of existing literature does. For brevity, we will simply refer to *quickest time* as the least possible time for a feasible dynamic flow to transmit a given flow value from s to t in the quickest flow problem.

Rather than examining each capacity vector one by one, a decomposition algorithm is presented to divide the state space of capacity vectors into disjoint partitions. Each partition is bounded by an upper vector and a lower vector so that all elements in the partition have an identical quickest time. As a result, the probability of each partition can be directly calculated by its upper and lower bounding vectors. In addition to the distribution of quickest time, performance indexes, such as expected quickest time and the probability/reliability that the quickest time is no larger than a specified time threshold, can be established efficiently. For a large network system, the proposed algorithm is modified to approximate the probability distribution of quickest time with an acceptable tolerance within a satisfactory execution time. Existing literature evaluates the reliability that the specified amount of flow can be sent from s to t *via* multiple pre-specified k disjoint minimal paths within a given time horizon. By contrast, the proposed algorithm is, to the best of our knowledge, a pioneering one that evaluates the reliability problem of a stochastic-flow network from the viewpoint of system flows. We note here that the quickest transshipment problem is much more difficult than the quickest flow problem [11]. Thus, the proposed algorithms are not suitable for the quickest transshipment problem.

The remainder of this paper is organized as follows. Section 2 introduces the network flow models. Section 3 presents algorithms to compute and approximate the probability distribution of quickest time. Section 4 conducts computational experiments, and Section 5 contains concluding remarks.

2. Network flow models

A network $G = (N, A)$ consists of a set N of n nodes and a set A of m directed arcs. Let the source s and the sink t be two distinguished nodes in N. Each arc e in A is weighted with a capacity c_e and a free-flow transit time τ_e. The capacity c_e is the maximum amount of data that can be transmitted on arc e per unit time. Assume arc e is directed from node u to node v. If no flow occurs on arc e and one unit of flow in node u is sent along arc e at time τ, then it arrives at node v at time $\tau + \tau_e$.

2.1. Static flows

For a node $u \in N$, let $A^O(u) \subseteq A$ denote the set of arcs in A that is directed out of u. Similarly, let $A^I(u) \subseteq A$ denote the set of arcs in A that is directed into u. A static flow f on G is a real-valued function f on arc set A that satisfies the following flow conservation constraints (2.1) and (2.2) and capacity constraints (2.3).

$$\sum_{e\in A^O(s)} f(e) - \sum_{e\in A^I(s)} f(e) = \sum_{e\in A^I(t)} f(e) - \sum_{e\in A^O(t)} f(e) \tag{2.1}$$

$$\sum_{e\in A^O(u)} f(e) - \sum_{e\in A^I(u)} f(e) = 0, \quad \text{for} \quad u \in N\backslash\{s,t\} \tag{2.2}$$

$$0 \le f(e) \le c_e, \quad \text{for} \quad e \in A. \tag{2.3}$$

The value $|f| = \sum_{e\in A^O(s)} f(e) - \sum_{e\in A^I(s)} f(e)$ of flow f is the net flow leaving source node s. A maximum flow is a flow that has the largest value among all flows. In case each arc e is weighted with w_e, which denotes a cost per unit of flow through e, the cost of a flow f is defined as $\sum_{e\in A} w_e f(e)$. A min-cost max-flow problem searches a maximum flow with the smallest possible cost.

2.2. Dynamic flows

A dynamic flow is defined on network $G = (N, A)$ with a finite time horizon T. The time itself can be either discrete or continuous. This paper focuses on the discrete time case. Time horizon T is the time until which the flow can travel in the network. It defines the set $\Gamma = \{0, 1, \ldots, T\}$ of the considered time moments. For $e \in A$ and $\tau \in \Gamma$, a *dynamic flow* is a function h on $A \times \Gamma$ that satisfies the following constraints:

$$\sum_{\tau=0}^{T} \sum_{e\in A^O(s)} h(e,\tau) - \sum_{\tau=0}^{T} \sum_{e\in A^I(s),\tau\geqslant\tau_e} h(e,\tau-\tau_e) = \sum_{\tau=0}^{T} \sum_{e\in A^I(t),\tau\geqslant\tau_e} h(e,\tau-\tau_e) - \sum_{\tau=0}^{T} \sum_{e\in A^O(t)} h(e,\tau) \tag{2.4}$$

$$\sum_{e\in A^O(u)} h(e,\tau) - \sum_{e\in A^I(u),\tau\geqslant\tau_e} h(e,\tau-\tau_e) = 0, \quad \text{for} \quad u \in N\backslash\{s,t\}, \tau \in \Gamma \tag{2.5}$$

$$0 \le h(e,\tau) \le c_e, \quad \text{for} \quad e \in A, \tau \in \Gamma. \tag{2.6}$$

Constraints (2.4) mean the net flow leaving source node s during the T periods equals the net flow arriving at sink node t within the time interval. Constraints (2.5) assert that for each intermediate node u and each time τ, the amount of flow that leaves u at time τ is equal to the amount that enters u at time τ. Constraints (2.6) restrict the flow along an arc that cannot exceed its capacity. Let $|h| = \sum_{\tau=0}^{T} \sum_{e \in A^O(s)} h(e,\tau) - \sum_{\tau=0}^{T} \sum_{e \in A^I(s), \tau \geqslant \tau_e} h(e, \tau\text{-}\tau_e)$ be the net flow value leaving s during periods T. The maximum dynamic flow is a dynamic flow that has the maximum flow value $|h|$.

Ford and Fulkerson [10] showed that if the capacities and transit times are discrete and constant, the maximum dynamic flow can be found by solving a min-cost flow problem on an enlarged network $G^+ = (N^+, A^+)$, where $N^+ = N \cup \{ t^+\}$ and $A^+ = A \cup \{ e^+\}$. To be more specific, G^+ comes from G by (1) adding a node t^+ and an arc e^+ directed from t to t^+ with dedicated capacity $c_{e+} = \infty$ and free-flow transit time $\tau_{e+} = -(T+1)$, and (2) setting the cost of each arc e to its transit time τ_e.

2.3. Quickest flows

A *quickest flow* is a feasible dynamic flow that sends σ units of flow from source s to sink t within the least possible time τ_σ. That is, a dynamic flow h during periods T is a *quickest flow* for sending σ units of from source s to sink t if it satisfies $|h| < \sigma$ when $T < \tau_\sigma$ and $|h| \geq \sigma$ when $T \geq \tau_\sigma$. Recall that the least possible time τ_σ is named as *quickest time*. The phrase "sends σ units of flow from source s to sink t" will also be omitted if its meaning is clear from the context. Burkard *et al.* [5] showed that the maximum amount of flow that can be sent through a network increases with time T. Thus, one can binary search over time T and solve a maximum dynamic flow problem at each iteration until the minimum time needed to send the given amount of flow is found. Burkard *et al.* also introduced a more complicated search algorithm with better worst-case bounds than the binary search algorithm. This strongly polynomial algorithm is constructed by embedding the strongly polynomial algorithm of Ford and Fulkerson [10] for the maximum dynamic flow problem into Megiddo's parametric search framework. Recently, Lin and Jaillet [18] design a new cost-scaling algorithm for the quickest flow problem that runs in the same time bound as Goldberg and Tarjan's cost-scaling algorithm. Their result shows for the first time that the quickest flow problem can be solved within the same time bound as the min-cost flow problem.

2.4. Stochastic-flow network

In a stochastic-flow network $G = (N, A)$, the capacity of arc e_i in A is a discrete random variable X_i that takes values on one of the n_i integers x_i^1, x_i^2, …, and x_i^{ni} with probabilities p_i^1, p_i^2, …, and p_i^{ni}, respectively. It satisfies $x_i^1 < x_i^2 <, \ldots, < x_i^{ni}$ and $\sum_{1 \leq j \leq n_i} p_i^j = 1$, for $1 \leq i \leq m$. Let $X = (X_1, X_2, \ldots, X_m)$ be a capacity vector with finite state space. An *element* of the state space of X is a vector $x^\gamma = (x_1^{\gamma 1}, x_2^{\gamma 2}, \ldots, x_m^{\gamma m})$, where $1 \leq \gamma_i \leq n_i$. For brevity, $\boldsymbol{\gamma}$ is used as a simplified representation for $\boldsymbol{x}^\gamma = (x_1^{\gamma^1}, x_2^{\gamma^2}, \ldots, x_m^{\gamma m})$. Obviously, $\boldsymbol{1} \leq \gamma \leq \boldsymbol{n}$, where $\boldsymbol{1}$ and $\boldsymbol{n}$ respectively represent $\boldsymbol{x}^1 = (x_1^1, x_2^1, \ldots, x_m^1)$ and $\boldsymbol{x}^n = (x_1^{n1}, x_2^{n2}, \ldots, x_m^{nm})$, which are the smallest and largest capacity vectors. As a result, in a stochastic-flow network, the quickest time to transmit a specified flow value from s to t is a random variable. Let $\Gamma_\sigma(\boldsymbol{\gamma})$ be the quickest time for sending σ units of flow from s to t under capacity vector $\boldsymbol{\gamma}$. This work explores the distribution of $\Gamma_\sigma(\boldsymbol{\gamma})$ for $\boldsymbol{1} \leq \boldsymbol{\gamma} \leq \boldsymbol{n}$.

3. Decomposition algorithm

To compute the probability distribution of $\Gamma_\sigma(\boldsymbol{\gamma})$ for $\boldsymbol{1} \leq \boldsymbol{\gamma} \leq \boldsymbol{n}$, we decompose $\boldsymbol{\Omega}$, the state space of X, into disjoint subsets. These disjoint subsets satisfy two properties. One is all elements in a subset have an identical quickest time. The other is each subset is bounded by an upper vector and a lower vector so that the probability of each subset can be calculated efficiently. Given that the changes of the quickest time are due to the changes of capacities on critical paths/flows, our decomposition algorithm focuses on searching such critical flows.

3.1. State space decomposition

The following important lemma is adapted from Burcard *et al.* [5].

Lemma 3.1. *$\Gamma_\sigma(\cdot)$ is a monotonic decreasing function. That is, for $\boldsymbol{\lambda}_i \geqslant \boldsymbol{\lambda}_j$, $\Gamma_\sigma(\boldsymbol{\lambda}_i) \leq \Gamma_\sigma(\boldsymbol{\lambda}_j)$.*

Proof. A quickest flow that sends σ units of flow from s to t under capacity vector $\boldsymbol{\lambda}_j$ is feasible under a non-smaller capacity vector $\boldsymbol{\lambda}_i$. Hence, the quickest time under $\boldsymbol{\lambda}_i$ is not larger than the quickest time under $\boldsymbol{\lambda}_j$. Lemma 3.1 is verified. □

Let $\underline{\tau}$ and $\bar{\tau}$ denote the smallest and largest quickest times among all capacity vectors respectively. The following Corollary 3.2 holds obviously.

Corollary 3.2. *For $\boldsymbol{1} \leq \boldsymbol{\gamma} \leq \boldsymbol{n}$, $\underline{\tau} \leq \boldsymbol{\Gamma}_\sigma(\boldsymbol{\gamma}) \leq \bar{\tau}$, where $\underline{\tau} = \Gamma_\sigma(n)$ and $\bar{\tau} = \sum_{e\in A} \tau_e + \sigma$ -1.*

Proof. According to Lemma 3.1, the smallest quickest time happens when the capacity vector is largest. Thus, $\underline{\tau} = \Gamma_\sigma(\boldsymbol{n})$. Whenever nodes s and t are connected, the very first unit of flow can be sent from s to t in $\sum_{e\in A} \tau_e$ time. All the σ units of flow can be sent in no more than $\bar{\tau} = \sum_{e\in A} \tau_e + \sigma$ -1 time. □

Definition 3.3. *Partition.* A subset $\boldsymbol{\omega}$ of $\boldsymbol{\Omega}$ is a (discrete) partition from $\boldsymbol{\Omega}$ with lower end vector $\boldsymbol{\alpha}$ and upper end vector $\boldsymbol{\beta}$ if it consists precisely of all vectors $\boldsymbol{\gamma}$ satisfying $\boldsymbol{\alpha} \leq \boldsymbol{\gamma} \leq \boldsymbol{\beta}$. This partition is denoted by $\boldsymbol{\omega} = [\boldsymbol{\alpha},\ \boldsymbol{\beta}]$. The probability of $\boldsymbol{\omega}$ is $\mathrm{pr}(\boldsymbol{\omega}) = \prod_{i=1}^{m} \sum_{\alpha_i \leq j \leq \beta_i} p_i^j$.

Note that the state space of X is denoted as $\Omega = [\boldsymbol{1}, \boldsymbol{n}]$. A partition $\boldsymbol{\omega} = [\boldsymbol{\alpha},\ \boldsymbol{\beta}]$ is further specified as determined if $\Gamma_\sigma(\boldsymbol{\lambda}_i) = \Gamma_\sigma(\boldsymbol{\lambda}_j)$ for any $\boldsymbol{\alpha} \leq \boldsymbol{\lambda}_i \neq \boldsymbol{\lambda}_j \leq \boldsymbol{\beta}$. Otherwise, $\boldsymbol{\omega}$ is undetermined. A determined partition $\boldsymbol{\omega}$ is distinguished as a *τ-partition* if $\Gamma_\sigma(\boldsymbol{\gamma}) = \tau$ for all $\boldsymbol{\gamma}$ satisfying $\boldsymbol{\alpha} \leq \boldsymbol{\gamma} \leq \boldsymbol{\beta}$. In the following, an efficient decomposition algorithm is presented to divide $\boldsymbol{\Omega}$ into determined and disjoint partitions so that the probability distribution of quickest time $\Gamma_\sigma(\boldsymbol{\gamma})$ can be easily computed.

For a partition $\boldsymbol{\omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}]$ (initially, $\boldsymbol{\omega} = \boldsymbol{\Omega}$), the simple binary search method of Burkard *et al.* [5] is first used to solve the quickest flow problem under upper vector $\boldsymbol{\beta}$. In practice, we binary search a min-cost flow on the enlarged network $G^+ = (N^+,\ A^+)$, where the capacity of arc e_i is set to $x_{\mathbf{i}}^{\beta\mathbf{i}}$. Let $F_\sigma(\boldsymbol{\beta}) = (f_\sigma(\boldsymbol{\beta})_1, f_\sigma(\boldsymbol{\beta})_2, \ldots, f_\sigma(\boldsymbol{\beta})_m)$ be the min-cost flow under $\boldsymbol{\beta}$, where $f_\sigma(\boldsymbol{\beta})_i$ is the flow passing arc e_i. Owing to capacity constraints, equation (2.3), $F_\sigma(\boldsymbol{\beta}) \leq \boldsymbol{x}^{\boldsymbol{\beta}} = (x_1^{\beta 1},\ x_2^{\beta 2}, \ldots, x_m^{\beta m})$.

Lemma 3.4. *For $\boldsymbol{\omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}]$, if $F_\sigma(\boldsymbol{\beta}) \leq \boldsymbol{x}^\alpha = (x_1^{\alpha 1}, x_2^{\alpha 2}, \ldots, x_m^{\alpha m})$, then $\boldsymbol{\omega}$ is a $\Gamma_\sigma(\boldsymbol{\beta})$-partition.*

Proof. For any $\boldsymbol{\gamma} \in \boldsymbol{\omega}$, $F_\sigma(\boldsymbol{\beta}) \leq \boldsymbol{x}^\alpha \leq \boldsymbol{x}^\gamma$ implies the min-cost flow $F_\sigma(\boldsymbol{\beta})$ is feasible under capacity vector $\boldsymbol{x}^\gamma$ and σ units of flow can be sent from $s$to t within $\Gamma_\sigma(\boldsymbol{\beta})$ time. According to Lemma 3.1, $\boldsymbol{x}^\beta \geqslant \boldsymbol{x}^\gamma$ implies $\Gamma_\sigma(\boldsymbol{\beta}) \leq \Gamma_\sigma(\boldsymbol{\gamma})$. Hence, $\Gamma_\sigma(\boldsymbol{\gamma}) = \Gamma_\sigma(\boldsymbol{\beta})$. Lemma 3.4 holds. □

Let $I = \{i\ | f_\sigma(\boldsymbol{\beta})_i > \boldsymbol{x}_i^{\alpha i}, \text{ for } 1 \leq i \leq m\} = \{k_1, k_2, \ldots, k_q\,|1 \leq k_1 < k_2 < \ldots < k_q \leq m\}$ be a set of arc index. According to Lemma 3.4, if I is empty, then partition $\boldsymbol{\omega}$ is a *$\Gamma_\sigma(\boldsymbol{\beta})$-partition.* The probability of $\boldsymbol{\omega}$, $\mathrm{pr}(\boldsymbol{\omega})$, contributes to the probability that the quickest time equals $\Gamma_\sigma(\boldsymbol{\beta})$. In case I is not empty, critical integers θ_i and δ_i for $i \in I$ are computed according to equations (3.1) and (3.2) respectively.

$$\delta_i = \max\{j|\boldsymbol{x}_i^{\alpha_i} \leq j < f_\sigma(\boldsymbol{\beta})_i\}, \quad \text{for } i \in I \tag{3.1}$$

$$\theta_i = \min\left\{j|f_\sigma(\boldsymbol{\beta})_i \leq j \leq \boldsymbol{x}_i^{\beta i}\right\}, \quad \text{for } i \in I. \tag{3.2}$$

Accordingly, δ_i is the largest capacity of e_i that is smaller than $f_\sigma(\boldsymbol{\beta})_i$, and θ_i is the smallest capacity of e_i that is not smaller than f_σ $(\boldsymbol{\beta})_i$. Namely, δ_i and θ_i are two successive capacities of arc e_i that satisfy

$$x_i^{\alpha i} \leq \delta_i < f_\sigma(\beta)_i \leq \theta_i \leq x_i^{\beta i} \tag{3.3}$$

We note here that capacities δ_i and θ_i of arc e_i play critical roles in our decomposition algorithm. To divide $\boldsymbol{\omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}]$ into disjoint sub-partitions, $\boldsymbol{Y}_i = \{\boldsymbol{x}_i^{\gamma i} | \boldsymbol{x}_i^{\alpha i} \leq \boldsymbol{x}_i^{\gamma i} \leq \boldsymbol{x}_i^{\beta i}\}$ is employed to denote the set of capacities of arc e_i, $1 \leq i \leq m$ and represent $\boldsymbol{\omega}$ by $\boldsymbol{Y}_1 \times \boldsymbol{Y}_2 \times \ldots \times \boldsymbol{Y}_m$. Let $\underline{\mathbf{Y}}_i = \{\boldsymbol{x}_i^{\gamma i} | \mathbf{x}_i^{\alpha i} \leq \mathbf{x}_i^{\gamma i} \leq \delta_i\}$ and $\overline{\mathbf{Y}}_i = \{\mathbf{x}_i^{\gamma i} | \theta_i \leq \mathbf{x}_i^{\gamma i} \leq \boldsymbol{x}_i^{\beta i}\}$. According to equation (3.3), $\boldsymbol{Y}_i = \underline{\mathbf{Y}}_i \cup \overline{\mathbf{Y}}_i$ and $\underline{\mathbf{Y}}_i \cap \overline{\mathbf{Y}}_i = \emptyset$. With respect to index set I, partition $\boldsymbol{\omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}] = \boldsymbol{Y}_1 \times \boldsymbol{Y}_2 \times \ldots \times \boldsymbol{Y}_m$ is divided into disjoint sub-partitions $\boldsymbol{\omega}^1, \boldsymbol{\omega}^2, \ldots, \boldsymbol{\omega}^q$, and $\boldsymbol{\omega}^{q+1}$ according to the following procedure.

Division procedure:

$$\begin{aligned}
\boldsymbol{\omega} &= \boldsymbol{Y}_1 \times \ldots \times \boldsymbol{Y}_{k1} \times \ldots \times \boldsymbol{Y}_m \\
&= \boldsymbol{Y}_1 \times \ldots \times \underline{\mathbf{Y}}_{k1} \times \ldots \times \boldsymbol{Y}_n \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \boldsymbol{Y}_m \\
&= \boldsymbol{\omega}^1 \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \boldsymbol{Y}_m \\
&= \boldsymbol{\omega}^1 \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \underline{\mathbf{Y}}_{k2} \times \ldots \times \boldsymbol{Y}_m \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{k2} \times \ldots \times \boldsymbol{Y}_m \\
&= \boldsymbol{\omega}^1 \cup \boldsymbol{\omega}^2 \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{k2} \times \ldots \times \boldsymbol{Y}_m \\
&\quad \ldots \\
&= \boldsymbol{\omega}^1 \cup \boldsymbol{\omega}^2 \cup \ldots \cup \boldsymbol{\omega}^{q-1} \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{k2} \times \ldots \times \overline{\mathbf{Y}}_{kq-1} \times \ldots \times \boldsymbol{Y}_m \\
&= \boldsymbol{\omega}^1 \cup \boldsymbol{\omega}^2 \cup \ldots \cup \boldsymbol{\omega}^{q-1} \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{k2} \times \ldots \times \overline{\mathbf{Y}}_{kq-1} \times \ldots \times \underline{\mathbf{Y}}_{kq} \times \ldots \times \boldsymbol{Y}_m \\
&\quad \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{k2} \times \ldots \times \overline{\mathbf{Y}}_{kq-1} \times \ldots \times \overline{\mathbf{Y}}_{kq} \times \ldots \times \boldsymbol{Y}_m \\
&= \boldsymbol{\omega}^1 \cup \boldsymbol{\omega}^2 \cup \ldots \cup \boldsymbol{\omega}^{q-1} \cup \boldsymbol{\omega}^q \cup \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{k2} \times \ldots \times \overline{\mathbf{Y}}_{kq-1} \times \ldots \times \overline{\mathbf{Y}}_{kq} \times \ldots \times \boldsymbol{Y}_m \\
&= \boldsymbol{\omega}^1 \cup \boldsymbol{\omega}^2 \cup \ldots \cup \boldsymbol{\omega}^{q-1} \cup \boldsymbol{\omega}^q \cup \boldsymbol{\omega}^{q+1}.
\end{aligned}$$

Specifically, for $i = 1, \boldsymbol{\omega}^1 = [\boldsymbol{\alpha}^1, \boldsymbol{\beta}^1] = \boldsymbol{Y}_1 \times \ldots \times \underline{\mathbf{Y}}_{k1} \times \ldots \times \boldsymbol{Y}_m$, where $\boldsymbol{\alpha}^1$ equal to $\boldsymbol{\alpha}$ represents vector $(x_1^{\alpha 1}, \ldots, x_{k1}^{\alpha k1}, \ldots, x_{k2}^{\alpha k2}, \ldots, x_{kq}^{\alpha kq}, \ldots, x_m^{\alpha m})$ and $\boldsymbol{\beta}^1$ is vector $(x_1^{\beta 1}, \ldots, \delta_{k1}, \ldots, x_{k2}^{\beta k2}, \ldots, x_{kq}^{\beta kq}, \ldots, x_m^{\beta m})$; for $2 \leq i \leq q$, $\boldsymbol{\omega}^i = [\boldsymbol{\alpha}^i, \boldsymbol{\beta}^i] = \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{ki-1} \times \ldots \times \underline{\mathbf{Y}}_{ki} \times \ldots \times \boldsymbol{Y}_m$, where vector $\boldsymbol{\alpha}^i$ is $(x_1^{\alpha 1}, \ldots, \theta_{k1}, \ldots, \theta_{ki-1}, \ldots, x_{ki}^{\alpha ki}, \ldots, x_m^{\alpha m})$ and vector $\boldsymbol{\beta}^i$ is $(x_1^{\beta 1}, \ldots, x_{k1}^{\beta k1}, \ldots, x_{ki-1}^{\beta ki-1}, \ldots, \delta_{ki}, \ldots, x_m^{\beta m})$; for $i = q+1$, $\boldsymbol{\omega}^{q+1} = [\boldsymbol{\alpha}^{q+1}, \boldsymbol{\beta}^{q+1}] = \boldsymbol{Y}_1 \times \ldots \times \overline{\mathbf{Y}}_{k1} \times \ldots \times \overline{\mathbf{Y}}_{kq} \times \ldots \times \boldsymbol{Y}_m$, where vector $\boldsymbol{\alpha}^{q+1}$ is $(x_1^{\alpha 1}, \ldots, \theta_{k1}, \ldots, \theta_{kq}, \ldots, x_m^{\alpha m})$ and vector $\boldsymbol{\beta}^{q+1}$ equal to $\boldsymbol{\beta}$ is $(x_1^{\beta 1}, \ldots, x_{k1}^{\beta k1}, \ldots, x_{kq}^{\beta kq}, \ldots, x_m^{\beta m})$.

The simple stochastic-flow network $G = (N, A)$ in Figure 1 and Table 1 is utilized to illustrate the *Division procedure.* Consider $\sigma = 2$ units of flow are sent from source node 1 to sink node 4. Consider $\boldsymbol{\Omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}] = \Omega$, where $\boldsymbol{\alpha} = \boldsymbol{1}$ represents $\boldsymbol{x}^1 = (0,0,0,0,0)$ and $\boldsymbol{\beta} = \boldsymbol{n}$ represents $\boldsymbol{x}^{\mathbf{n}} = (3,4,4,4,3)$. The min-cost flow $F_2(\boldsymbol{\beta}) = (f_2(\beta)_1, f_2(\beta)_2,\ f_2(\boldsymbol{\beta})_3,\ f_2(\boldsymbol{\beta})_4, f_2(\boldsymbol{\beta})_5)$ is (0, 2, 0, 0, 2) with quickest time $\Gamma_2(\boldsymbol{\beta}) = 4$. Accordingly, arc index set $I = \{i | f_2(\boldsymbol{\beta})_i > \boldsymbol{x}_i^{\alpha i}, 1 \leq i \leq 5\} = \{2, 5\}$. For $i = 2$, $\delta_2 = \max\{j\ | \boldsymbol{x}_2^{\alpha 2} \leq j < f_2(\boldsymbol{\beta})_2\} = \max\{j | 0 \leq j < 2\} = 0$, $\theta_2 = \min\{j\ | f_2(\boldsymbol{\beta})_2 \leq j \leq \boldsymbol{x}_2^{\beta 2}\} = \min\{j\ | 2 \leq j \leq 4\} = 2$, $\underline{\mathbf{Y}}_2 = \{0\}$, and $\overline{\mathbf{Y}}_2 = \{2,4\}$. Similarly, for $i = 5$, $\delta_5 = 1$, $\theta_5 = 2$, $\underline{\mathbf{Y}}_5 = \{0,1\}$, and $\overline{\mathbf{Y}}_5 = \{2,3\}$. Sub-partitions $\boldsymbol{\Omega}^1 = [\boldsymbol{\alpha}^1, \boldsymbol{\beta}^1] = \boldsymbol{Y}_1 \times \underline{\mathbf{Y}}_2 \times \boldsymbol{Y}_3 \times \boldsymbol{Y}_4 \times \boldsymbol{Y}_5$, where $\boldsymbol{\alpha}^1 = (0,0,0,0,0)$ and $\boldsymbol{\beta}^1 = (3,0,4,4,3)$; $\boldsymbol{\Omega}^2 = [\boldsymbol{\alpha}^2, \boldsymbol{\beta}^2] = \boldsymbol{Y}_1 \times \overline{\mathbf{Y}}_2 \times \boldsymbol{Y}_3 \times \boldsymbol{Y}_4 \times \underline{\mathbf{Y}}_5$, where $\boldsymbol{\alpha}^2 = (0,2,0,0,0)$ and $\boldsymbol{\beta}^2 = (3,4,4,4,1)$; and $\boldsymbol{\omega}^3 = [\boldsymbol{\alpha}^3, \boldsymbol{\beta}^3] = \boldsymbol{Y}_1 \times \overline{\mathbf{Y}}_2 \times \boldsymbol{Y}_3 \times \boldsymbol{Y}_4 \times \overline{\mathbf{Y}}_5$, where $\boldsymbol{\alpha}^3 = (0,2,0,0,2)$ and $\boldsymbol{\beta}^3 = (3,4,4,4,3)$.

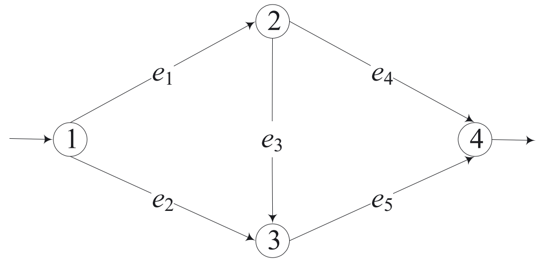

FIGURE 1. The illustration network.

TABLE 1. Capacities and transit times of arcs in Figure 1.

Arc	Capacity				τ_{ei}
e_1	0	1	3		2
e_2	0	2	4		2
e_3	0	1	2	4	3
e_4	0	2	4		3
e_5	0	1	2	3	4

According to the *Division procedure*, partitions $\boldsymbol{\omega}^1$, $\boldsymbol{\omega}^2, \ldots, \boldsymbol{\omega}^q$, and $\boldsymbol{\omega}^{q+1}$ satisfy the following Lemmas 3.5 and 3.6.

Lemma 3.5. *Partitions $\boldsymbol{\omega}^1$, $\boldsymbol{\omega}^2, \ldots, \boldsymbol{\omega}^q$, and $\boldsymbol{\omega}^{q+1}$ are disjoint,* i.e., *$\boldsymbol{\omega}^i \cap \boldsymbol{\omega}^j = \emptyset$, for $1 \le i \ne j \le q+1$.*

Proof. Given that $\boldsymbol{\omega}^1$ comes from $\boldsymbol{\omega}$ by restricting the capacities of e_{k1} in $\underline{\mathbf{Y}}_{k1}$, $\boldsymbol{\omega}^2$ comes from $\boldsymbol{\omega} \backslash \boldsymbol{\omega}^1$ (*i.e.*, capacities of e_{k1} in $\overline{\mathbf{Y}}_{k1}$) by restricting the capacities of e_{k2} in $\underline{\mathbf{Y}}_{k2}, \ldots, \boldsymbol{\omega}^q$ comes from $\boldsymbol{\omega} \backslash \bigcup\limits_{1 \le i \le q-1} \omega^i$ by restricting the capacities of e_{kq} in $\underline{\mathbf{Y}}_{kq}$. Finally, $\boldsymbol{\omega}^{q+1}$ comes from $\boldsymbol{\omega} \backslash \bigcup\limits_{1 \le i \le q-1} \omega^i$ by restricting the capacities of e_{kq} in $\overline{\mathbf{Y}}_{kq}$. Thus, Lemma 3.5 holds. $\square$

Lemma 3.6. *Partition $\boldsymbol{\omega}^{q+1}$ is a $\Gamma_\sigma(\boldsymbol{\beta})-$partition.*

Proof. Let $\boldsymbol{\gamma}$ be an element of $\boldsymbol{\omega}^{q+1}$. According to the *Division procedure*, for $i \in I$, $\theta_i \le \boldsymbol{x}_i^{\gamma i} \le \boldsymbol{x}_i^{\beta i}$; for $i \notin I$, $\boldsymbol{x}_i^{\alpha i} \le \boldsymbol{x}_i^{\gamma i} \le \boldsymbol{x}_i^{\beta i}$. Due to equation (3.2) and definition of index set I, $f_\sigma(\boldsymbol{\beta})_i \le \theta_i$ and $f_\sigma(\boldsymbol{\beta})_i \le \boldsymbol{x}_i^{\alpha i}$ respectively. As a result, $F_\sigma(\boldsymbol{\beta}) \le \boldsymbol{x}^\gamma \le \boldsymbol{x}^\beta$. Lemma 3.6 comes from Lemma 3.4. $\square$

Lemma 3.6 shows that a $\boldsymbol{\Gamma}_\sigma(\boldsymbol{\beta})$-partition $\boldsymbol{\omega}^{q+1}$ can be derived from $\boldsymbol{\omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}]$. The probability of $\boldsymbol{\omega}^{q+1}$, $\mathrm{pr}(\boldsymbol{\omega}^{q+1})$, contributes to the probability that the quickest time equals $\boldsymbol{\Gamma}_\sigma(\boldsymbol{\beta})$. However, $\boldsymbol{\omega}^1$, $\boldsymbol{\omega}^2, \ldots$, and $\boldsymbol{\omega}^q$ are not guaranteed to be *determined*. The proposed algorithm recursively divides each $\boldsymbol{\omega}^i (1 \le i \le q)$ according to *Division procedure* until all partitions are determined.

3.2. Distribution of the quickest time

Let R_σ^T be the probability that the quickest time to send σ flow from s to t *via* the stochastic-flow network is T. To compute R_σ^T, $\underline{\tau} \le T \le \bar{\tau}$, the proposed algorithm starts at the state space $\boldsymbol{\Omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}] = [\mathbf{1}, \boldsymbol{n}]$. It first evaluates the quickest time $\boldsymbol{\Gamma}_\sigma(\boldsymbol{\beta})$, searches min-cost flow $F_\sigma(\boldsymbol{\beta})$, and lists the set of arc index I. Then it verifies if I is empty or not. If I is empty, Ω itself is an *$\Gamma_\sigma(\boldsymbol{\beta})$-partition*. Thus, $\Gamma_\sigma(\boldsymbol{\beta}) = \Gamma_\sigma(\boldsymbol{n}) = \underline{\tau}$, and $R_\sigma \underline{\tau} = 1$. If not, integers δ_i and θ_i, for $i \in I$, are computed according to equations (3.1) and (3.2). Finally, it divides $\boldsymbol{\omega} = \Omega$ into disjoint sub-partitions $\boldsymbol{\omega}^i = [\boldsymbol{\alpha}^i, \boldsymbol{\beta}^i], 1 \le i \le q+1$. Given that $\boldsymbol{\omega}^{q+1} = [\boldsymbol{\alpha}^{q+1}, \boldsymbol{\beta}^{q+1}]$ is a *τ-partition*, where $\tau = \Gamma_\sigma(\boldsymbol{\beta}^{q+1})$, R_σ^τ is updated by $R_\sigma^\tau +\ \mathrm{pr}(\boldsymbol{\omega}^{q+1})$. Each undetermined $\boldsymbol{\omega}^i$, $1 \le i \le q$, is recursively divided by the above steps until the partitions are determined.

Disjoint sub-partitions $\boldsymbol{\omega}^i$, $1 \le i \le q{+}1$, are stored in a storage Z. According to the *Division procedure*, each $\boldsymbol{\omega}^i = [\boldsymbol{\alpha}^i, \boldsymbol{\beta}^i]$, $1 \le i \le q{+}1$ is derived from $\boldsymbol{\alpha}, \boldsymbol{\beta}, \delta_i$, and θ_i for $i \in I$, whereas δ_i and θ_i are derived from $F_\sigma(\boldsymbol{\beta})$ and I. To minimize the space of storage Z, a collection with respect to $\boldsymbol{\omega}$ is stored instead of $\boldsymbol{\omega}^1$, $\boldsymbol{\omega}^2, \ldots, \boldsymbol{\omega}^q$, and $\boldsymbol{\omega}^{q+1}$. The collection with respect to $\boldsymbol{\omega}$ is a set $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, F_\sigma(\boldsymbol{\beta}), I, q, i\}$. The index i points to the first $\boldsymbol{\omega}^i$ to be retrieved for further division. When $\boldsymbol{\omega}^i$ is retrieved, i is updated by $i{+}1$ as long as $i \le q$. When $i{>}q$, R_σ^T is updated by $R_\sigma^T{+}\ \mathrm{pr}(\boldsymbol{\omega}^{q+1})$, where $T = \mathit{\Gamma}_\sigma(\boldsymbol{\beta}^{q+1})$, and the top collection on Z is discarded. The decomposition algorithm for computing R_σ^T is summarized as the following pseudo code.

***Exact algorithm*:**

***Input*:** $G = (N, A), s, t, \sigma, \mathit{\Omega} = [\boldsymbol{1}, \boldsymbol{n}]$.

***Output*:** $R_\sigma^T, \underline{\tau} \le \mathit{\Gamma}_\sigma(\boldsymbol{\gamma}) \le \bar{\tau}$.

Steps:

1 Compute $\underline{\tau} = \mathit{\Gamma}_\sigma(\boldsymbol{n})$ and $\bar{\tau} = \sum_{e \in A} \tau_e + \sigma{-}1$, search $F_\sigma(\boldsymbol{n})$ and arc index set$I$
2 Set $R_\sigma^T = 0$, for $\underline{\tau} \le T \le \bar{\tau}$
3 $Z = \{\boldsymbol{\alpha} = \boldsymbol{1}, \boldsymbol{\beta} = \boldsymbol{n}, F_\sigma(\boldsymbol{\beta}) = F_\sigma(\boldsymbol{n}), I, q, i = 1\}$//initial collection w.r.t. $\mathit{\Omega}$ //
4 ***if*** $I = \emptyset$ ***then*** $R_\sigma \underline{\tau} = 1$, and top collection on Z is discarded
5 ***while***$Z \ne \emptyset$ ***do***
6   {Compute $\boldsymbol{\alpha}^i$ and $\boldsymbol{\beta}^i$ according to the index i in top collection on Z
7   Set $\boldsymbol{\alpha} = \boldsymbol{\alpha}^i, \boldsymbol{\beta} = \boldsymbol{\beta}^i$, and $\boldsymbol{\omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}]$
8   ***if*** $i \le q$ ***then*** {Set $i = i{+}1$ //update top collection on Z//
9   Compute $\mathit{\Gamma}_\sigma(\boldsymbol{\beta})$, search $F_\sigma(\boldsymbol{\beta})$ and I
10   ***if*** $I = \emptyset$ ***then*** update R_σ^T by $R_\sigma^T{+}\ \mathrm{pr}(\boldsymbol{\omega})$ where $T = \mathit{\Gamma}_\sigma(\boldsymbol{\beta})$
11   ***else*** Put $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, F_\sigma(\boldsymbol{\beta}), I, q, i = 1\}$on top of Z
12   ***else*** {Update R_σ^T by$R_\sigma^T{+}\ \mathrm{pr}(\boldsymbol{\omega})$ where $T = \mathit{\Gamma}_\sigma(\boldsymbol{\beta})$;
13   Pop out top collection on Z }
14   }//end of step 5 ***while do*** //.

Lemma 3.7. *When dividing state space* $\boldsymbol{\mathit{\Omega}}$ *according to the exact algorithm until all partitions are determined, the number of disjoint sub-partitions is upper bounded by* $\prod_{i=1}^{m} n_i$.

Proof. The worst case happens if the lower vector and upper vector of each determined sub-partition are identical. In such a case, each sub-partition contains one capacity vector only. There are $\prod_{i=1}^{m} n_i$ determined sub-partitions derived from $\boldsymbol{\mathit{\Omega}}$. Lemma 3.7 holds. □

Remark 3.8. In the worst case, the exact algorithm acts like a complete enumeration for all capacity vectors. However, in the computational experiments, the running time of the exact algorithm is much shorter than the time of the complete enumeration. This result implies that the worst case rarely happens.

Lemma 3.9. *Running time of the exact algorithm is* $O(\max[\min(\log\sigma, f_{\max})\ m\ \log\ n(m + n\log n), \sum_{1\le i\le m} n_i] \prod_{i=1}^{m} n_i)$, *where* $f_{\max}$, n, *and* m *are the value of the maximum static flow and the numbers of nodes and arcs, respectively.*

Proof. To search the quickest flow using the binary search method of Burkard *et al.* [5], Steps 1 and 9 take $O(\min(\log\ \sigma,\ f_{\max}) m \log n(m + n\log\ n))$ time. According to Corollary 3.2, Step 2 needs $O(\sum_{e\in A} \tau_e{+}\ \sigma$ -1) time. Steps 3, 4, 11, and 13 need $O(m)$ time to discard/put a collection from/into storage Z. Steps 5 and 8 use $O(1)$

time. Steps 6 and 7 take $O(m)$ time to scan each arc once. The probability $\mathrm{pr}(\boldsymbol{\omega}) = \prod_{i=1}^{m} \sum_{\alpha_i \le j \le \beta_i} p_i^j$ is the multiplication of the summation of probabilities between $p_i^{\alpha_i}$ and $p_i^{\beta_i}$ for e_i, Steps 10 and 12 take$O(\sum_{1 \le i \le m} n_i)$ time. Owing to Lemma 3.7, the ***while* … *do*** loop (Steps 5–14) repeats $O(\prod_{i=1}^{m} n_i)$ times. As a result, the exact algorithm takes$O(\max[\min(\log \sigma,\ f_{\max})m \log n(m + n\log n), \sum_{1 \le i \le m} n_i] \prod_{i=1}^{m} n_i)$ time. $\square$

In the forthcoming computational experiments section, the exact algorithm is found to spend considerable time for large network systems. To improve its performance, the exact algorithm is modified so that accuracy is traded-off for execution time. The modification is made by ignoring insignificant partitions. Specifically, partitions whose probabilities are less than a pre-specified threshold are discarded and excluded from further division. Let *tolerance* be the summation of probabilities of discarded partitions. The modified method reaches an acceptable tolerance with satisfactory execution time by decreasing the pre-specified thresholds one by one. The modified algorithm that approximates R_σ^T is listed as the following pseudo code.

***Approximation algorithm*:**

***Input*:** $G = (N,A)$, s, t, σ, $\Omega = [\mathbf{1},\ \boldsymbol{n}]$, threshold ρ.
***Output*:** R_σ^T, $\underline{\tau} \le T \le \bar{\tau}$.
Steps:

1 Compute $\underline{\tau} = \Gamma_\sigma(\boldsymbol{n})$ and $\bar{\tau} = \sum_{e \in A} \tau_e + \sigma$ -1, search $F_\sigma(\boldsymbol{n})$ and arc index set I
2 Set $R_\sigma^T = 0$, for $\underline{\tau} \le T \le \bar{\tau}$
3 $Z = \{\boldsymbol{\alpha} = \mathbf{1}, \boldsymbol{\beta} = \boldsymbol{n},\ F_\sigma(\boldsymbol{\beta}) = F_\sigma(\boldsymbol{n}),\ I,\ q,\ i = 1\}$//initial collection w.r.t. Ω //
4 ***if*** $I = \emptyset$ ***then*** $R_\sigma \underline{\tau} = 1$, and top collection on Z is discarded
5 ***while*** $Z \ne \emptyset$ ***do***
6 {Compute $\boldsymbol{\alpha}^i$ and $\boldsymbol{\beta}^i$ according to the index i in top collection on Z
7 Set $\boldsymbol{\alpha} = \boldsymbol{\alpha}^i$, $\boldsymbol{\beta} = \boldsymbol{\beta}^i$, and $\boldsymbol{\omega} = [\boldsymbol{\alpha}, \boldsymbol{\beta}]$
8 ***if*** $\mathrm{pr}(\boldsymbol{\omega}) \geqslant \rho$ ***then***
9 ***if*** $i \le q$ ***then*** {Set $i = i + 1$ //update top collection on Z//
10 Compute $\Gamma_\sigma(\boldsymbol{\beta})$, search $F_\sigma(\boldsymbol{\beta})$ and I
11 *if* $I = \emptyset$ ***then*** update R_σ^T by$R_\sigma^T + \mathrm{pr}(\boldsymbol{\omega})$ where $T = \Gamma_\sigma(\boldsymbol{\beta})$
12 ***else*** Put $\{\boldsymbol{\alpha}, \boldsymbol{\beta},\ F_\sigma(\boldsymbol{\beta}),\ I,\ q,\ i = 1\}$on top of Z
13 ***else*** {Update R_σ^T by$R_\sigma^T + \mathrm{pr}(\boldsymbol{\omega})$ where $T = \Gamma_\sigma(\boldsymbol{\beta})$;
14 Pop out top collection on Z }
15 ***else*** Pop out top collection on Z//in case $\mathrm{pr}(\boldsymbol{\omega}) < \rho$ //
16 }//end of step 5 ***while* … *do*** //.

4. Computational experiments

To evaluate the reliability problem of a dynamic stochastic-flow network, existing algorithms [19,20,28] restrict the transmission of demands on pre-specified k disjoint minimal paths. These algorithms are different from our probability distribution problem in which demands are transmitted simultaneously *via* flows, that is, by way of all possible minimal paths (including disjoint and non-disjoint ones). Experiences in solving multi-state network reliability problems [13, 14] suggest that exhaustive and indirect methods are the two possible alternatives. An exhaustive method completely enumerates all $\prod_{i=1}^{m} n_i$ state vectors. An indirect method consists of three steps: Step 1 searches all minimal paths; Step 2, based on all minimal paths, searches all minimal capacity vectors that fulfill the requirements of demand and time horizon; and Step 3 uses the inclusion–exclusion principle to evaluate R_σ^T based on all minimal capacity vectors. With restrictions on k disjoint minimal paths,

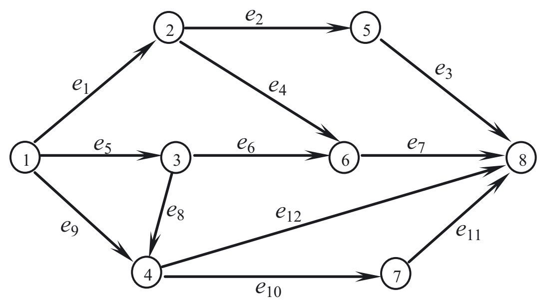

FIGURE 2. The testing network.

TABLE 2. Capacities, operation probabilities, and transit times of arcs in Figure 2.

Arc	Capacity						Operation probability						τ_{ei}
e_1	0	1	3	5			0.05	0.05	0.05	0.85			2
e_2	0	1	3	5			0.05	0.05	0.10	0.80			2
e_3	0	1	2	4			0.05	0.05	0.05	0.85			3
e_4	0	1	3				0.05	0.05	0.90				3
e_5	0	1	3	5			0.05	0.05	0.05	0.85			2
e_6	0	1	2	4	6		0.05	0.05	0.05	0.05	0.80		3
e_7	0	1	2	4			0.05	0.05	0.05	0.85			3
e_8	0	2					0.05	0.95					1
e_9	0	1	2	4	6		0.05	0.05	0.05	0.05	0.80		2
e_{10}	0	1	2	4	5	7	0.05	0.05	0.05	0.10	0.10	0.65	2
e_{11}	0	1	2	4	6		0.05	0.05	0.10	0.10	0.70		4
e_{12}	0	2					0.05	0.95					4

the methods of Lin [19,20] and Yeh [28] are available to search minimal capacity vectors in Step 2 of the indirect method. However, to the best of our knowledge, when common arcs are present among minimal paths (*i.e.*, minimal paths are not disjoint), there is no method developed to search the minimal capacity vectors. In the following, we conduct computational experiments and compare the proposed algorithm with the exhaustive method. The first testing network, which has 8 nodes and 12 arcs (Fig. 2), is cited from Lin [20]. Table 2 lists the capacities, operation probabilities, and transit times of arcs. Computation results are listed in Table 3. The following observations are made:

(1) The quickest time is set to infinite ∞ to denote that the source and sink nodes are not connected. In the testing, the probability that source and sink are not connected is $R_\sigma^\infty = 0.00029$. The R_σ^∞ is independent of σ. Note that the infinite quickest time is excluded from the computation of the expected quickest time.
(2) A large demand σ results in a large range of quickest time value. The expected quickest times for $\sigma = 10, 15,$ and 20 are 8.064776, 8.349875, and 8.864810, respectively. The CPU times for σ = 10, 15, and 20 are 3375, 3563, and 3828 units, respectively.
(3) The reliability/probability that the quickest time is no larger than a specified T^* can be evaluated by $\sum_{T \le T^*} R_\sigma^T$. For instance, in case $\sigma = 15$, the probability that the quickest time is no larger than 10 is $R_{15}^8 + R_{15}^9 + R_{15}^{10} = 0.98763$.
(4) In the experiments, the presented algorithm computes probability distribution in less than 4 seconds (4000 time units) for all three cases. By contrast, the execution time of the exhaustive method ranges from 345 to 402 seconds. The proposed algorithm is faster than the exhaustive method.

The benchmark network of Ford and Fulkerson [10] in Figure 3, which has 11 nodes and 21 arcs, is tested to explore the performance of the proposed method. For simplicity, each arc has three states, which results

TABLE 3. Computational results of Figure 2.

Probability R_σ^T		Demand σ 10	15	20
	∞	0.000229	0.000229	0.000229
Quickest time T	8	0.947118	0.712549	0.330472
	9	0.045229	0.249909	0.545061
	10	0.004956	0.025172	0.088903
	11	0.001254	0.007306	0.022122
	12	0.000491	0.001559	0.005369
	13	0.000022	0.001543	0.004009
	14	0	0.000790	0.000580
	15	0.000162	0.000226	0.001194
	16	0.000349	0.000017	0.000847
	17	0.000186	0	0.000491
	18	0.000003	0	0.000022
	20		0.000162	0
	21		0.000349	0
	22		0.000186	0
	23		0.000003	0
	25			0.000162
	26			0.000349
	27			0.000186
	28			0.000003
Expected quickest time*		8.064776	8.349875	8.864810
CPU time†	Proposed algorithm	3375	3563	3828
	Exhaustive method‡	345127	379681	401233

* The infinite quickest time is not included in the computation of expected quickest time; † The unit of CPU time is 0.001 second; ‡ There are 9216000 capacity vectors.

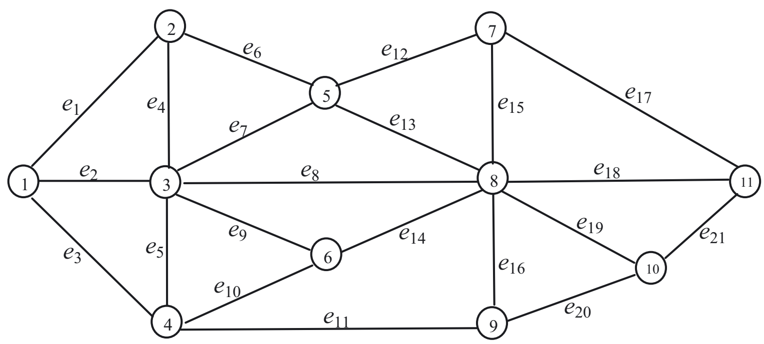

FIGURE 3. A large network.

in a total of $3^{21} \approx 10^{10}$ capacity vectors. Table 4 lists the capacities, operation probabilities, and transit times of arcs in Figure 3. Both the proposed and exhaustive methods cannot compute the probability distribution within three hours. The R_σ^T is estimated by a modified approximation algorithm. In the experiments, demand is $\sigma = 50$, and the threshold ρ decreases from 10^{-10} to 10^{-12}, 10^{-14}, until 10^{-16}. Results are listed in Table 5. Observations from Table 5 are drawn as follows:

According to Table 5, a small threshold ρ results in a small tolerance and a large CPU time. In contrast to the exact and exhaustive methods that cannot compute the probability distribution within three hours

TABLE 4. Capacities, operation probabilities, and transit times of arcs in Figure 3.

Arc	Capacity			Operation probability			τ_{ei}
e_1	0	20	50	0.01	0.05	0.94	3
e_2	0	10	30	0.05	0.15	0.80	6
e_3	0	10	15	0.01	0.09	0.90	8
e_4	0	20	50	0.05	0.10	0.85	2
e_5	0	10	15	0.01	0.05	0.94	2
e_6	0	10	25	0.05	0.15	0.80	2
e_7	0	15	45	0.01	0.05	0.94	1
e_8	0	10	15	0.05	0.10	0.85	8
e_9	0	5	10	0.01	0.05	0.94	3
e_{10}	0	5	10	0.01	0.09	0.90	1
e_{11}	0	10	20	0.01	0.05	0.94	3
e_{12}	0	30	90	0.01	0.05	0.94	9
e_{13}	0	5	10	0.05	0.15	0.80	8
e_{14}	0	20	60	0.01	0.09	0.90	5
e_{15}	0	5	10	0.05	0.05	0.90	1
e_{16}	0	15	20	0.01	0.05	0.94	2
e_{17}	0	5	10	0.05	0.15	0.80	2
e_{18}	0	40	80	0.01	0.05	0.94	4
e_{19}	0	5	10	0.01	0.09	0.90	1
e_{20}	0	5	10	0.05	0.15	0.80	3
e_{21}	0	5	10	0.05	0.10	0.85	3

TABLE 5. Computational results of Figure 3.

Approximated R_{50}^T		Threshold ρ			
		10^{-10}	10^{-12}	10^{-14}	10^{-16}
Quickest time T	∞	$3.625*10^{-5}$	$3.701*10^{-5}$	$3.705*10^{-5}$	$3.705*10^{-5}$
	17	$3.939*10^{-1}$	$3.969*10^{-1}$	$3.969*10^{-1}$	$3.969*10^{-1}$
	18	$5.893*10^{-1}$	$5.866*10^{-1}$	$5.866*10^{-1}$	$5.867*10^{-1}$
	19	$3.767*10^{-3}$	$3.327*10^{-3}$	$3.072*10^{-3}$	$3.068*10^{-3}$
	20	$9.698*10^{-3}$	$9.838*10^{-3}$	$9.996*10^{-3}$	$9.997*10^{-3}$
	21	$5.720*10^{-4}$	$5.493*10^{-4}$	$5.464*10^{-4}$	$5.462*10^{-4}$
	22	$3.032*10^{-4}$	$3.017*10^{-4}$	$3.008*10^{-4}$	$3.008*10^{-4}$
	23	$7.242*10^{-8}$	$3.379*10^{-8}$	$1.714*10^{-8}$	$1.579*10^{-8}$
	24	$2.370*10^{-10}$	$1.188*10^{-9}$	$9.410*10^{-10}$	$1.982*10^{-9}$
	25	$2.059*10^{-3}$	$2.140*10^{-3}$	$2.145*10^{-3}$	$2.145*10^{-3}$
	26	$2.133*10^{-4}$	$2.576*10^{-4}$	$2.615*10^{-4}$	$2.616*10^{-4}$
	27	$4.865*10^{-5}$	$5.369*10^{-5}$	$5.418*10^{-5}$	$5.420*10^{-5}$
	28	$3.080*10^{-9}$	$6.163*10^{-8}$	$7.081*10^{-8}$	$7.181*10^{-8}$
	29	0	$2.000*10^{-11}$	$4.790*10^{-10}$	$5.850*10^{-10}$
	30	0	$3.000*10^{-12}$	$1.030*10^{-10}$	$1.470*10^{-10}$
Tolerance§		$5.329*10^{-5}$	$1.124*10^{-6}$	$1.586*10^{-8}$	$1.560*10^{-10}$
Expected quickest time*		17.6471	17.6458	17.6459	17.6459
CPU time†	Approximation algorithm	76738	252209	493711	653839
	Exhaustive method‡	$>1.08*10^7$	$>1.08*10^7$	$>1.08*10^7$	$>1.08*10^7$

§ Tolerance is the total probabilities of discarded partitions; * The infinite quickest time is not included in the computation of expected quickest time; † The unit of CPU time is 0.001 second; ‡ There are more than 10^{10} capacity vectors, $1.08*10^7$ CPU time is 3 hours.

(10 800 000 CPU time), the approximation algorithm, which attains 5.329×10^{-5}, 1.124×10^{-6}, 1.586×10^{-8}, and 1.560×10^{-10} tolerances (summation of probabilities of discarded partitions) in 76.738, 252.209, 493.711, and 653.839 seconds, respectively, is an acceptable alternative to estimate the probability distribution of the quickest time.

5. Conclusions

This paper considers a dynamic stochastic-flow network in which each arc is simultaneously weighted with capacity, transit time, and operation probability. It uses the network to present an exact algorithm that can solve the probability distribution problem of the quickest time in transmitting a fixed amount of data from a source node to a sink node. This work is the first one in which the transmission is *via* a quickest flow rather than *via* disjoint quickest paths as the majority of literature does. The proposed algorithm utilizes the simple binary search method for the quickest flow problem as suggested by Burkard *et al.* [5]. Efficient but sophisticated algorithms for the quickest flow problem are available from Burkard *et al.* [5] and Lin and Jaillet [18]. The expected quickest time and the probability that the quickest time is no larger than a specified time threshold can be determined directly from the distribution of the quickest time. When solving a large network system, the proposed algorithm can be easily modified to approximate the probability distribution of the quickest time. The modification is made by discarding partitions whose probabilities are smaller than a pre-specified threshold. Computational experiments verify that the approximation algorithm attains an acceptable tolerance by decreasing the threshold one by one until a satisfactory execution time is reached.

Acknowledgements. The authors gratefully acknowledge the editor-in-chief, Prof. Ridha Mahjoub, and Doctor Maokai Lin for their valuable comments and constructive suggestions, which have significantly improved the paper. This research is partially supported by the Ministry of Science and Technology, Taiwan, ROC under Grant MOST 104-2221-E-275-002-MY3.

References

[1] K.K. Aggarwal, A fast algorithm for the performance index of a telecommunication network. *IEEE Trans. Reliab.* **37** (1988) 65–69.

[2] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network Flows.* Prentice Hall, New Jersey (1988).

[3] J.E. Aronson, A survey of dynamic network flows. *Ann. Oper. Res.* **20** (1989) 1–66.

[4] Y.-C. Bang, H. Choo and Y. Mun, Reliability problem on all pairs quickest paths. In *ICCS 2003 – Computational Science.* Springer Berlin Heidelberg (2003) 518–523.

[5] R.E. Burkard, K. Dlaska and B. Klinz, The quickest flow problem. *Z. Oper. Res.* **37** (1993) 31–58.

[6] C. Gen-Huey and H. Yung-Chen, Algorithms for the constrained quickest path problem and the enumeration of quickest paths. *Comput. Oper. Res.* **21** (1994) 113–118.

[7] Y.L. Chen and Y.H. Chin, The quickest path problem. *Comput. Oper. Res.* **17** (1990) 153–161.

[8] Y.L. Chen, Finding the k quickest simple paths in a network. *Inf. Proc. Lett.* **50** (1994) 89–92.

[9] L. Fleischer and M. Skutella, The quickest multicommodity flow problem. In Integer Programming and Combinatorial Optimization. Springer, Berlin, Heidelberg (2002) 36–53.

[10] L.R. Ford and D.R. Fulkerson, Flows in networks. Princeton University Press: Princeton (1962).

[11] B. Hoppe and É. Tardos, The quickest transshipment problem. *Math. Oper. Res.* **25** (2000) 36–62.

[12] C.C. Jane and Y.-W. Laih, Algorithms to determine the threshold reliability of flow networks. *IIE Trans.* **36** (2004) 469–479.

[13] C.C. Jane and Y.-W. Laih, A practical algorithm for computing multi-state two-terminal reliability. *IEEE Trans. Reliab.* **57** (2008) 295–302.

[14] C.C. Jane and Y.-W. Laih, A dynamic bounding algorithm for approximating multi-state two-terminal reliability. *Eur. J. Oper. Res.* **205** (2010) 625–637.

[15] C.C. Jane and J. Yuan, A sum of disjoint products algorithm for reliability evaluation of flow networks. *Eur. J. Oper. Res.* **131** (2001) 664–675.

[16] D.T. Lee and E. Papadopoulou, The all-pairs quickest path problem. *Inf. Process. Lett.* **45** (1993) 261–267.

[17] S.H. Lee, Reliability evaluation of a flow network. *IEEE Trans. Reliab.* **29** (1980) 24–26.

[18] M. Lin and P. Jaillet, On the quickest flow problem in dynamic networks: a parametric min-cost flow approach. In *Proc. of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms.* SIAM (2015) 1343–1356.

[19] Y.K. Lin, Extend the quickest path problem to the system reliability evaluation for a stochastic-flow network. *Comput. Oper. Res.* **30** (2003) 567–575.

[20] Y.K. Lin, Transmission reliability of k minimal paths within time threshold. *Comput. Ind. Eng.* **61** (2011) 1160–1165.

[21] M.H. Moore, On the fastest route for convoy-type traffic in flow rate constrained networks. *Trans. Sci.* **10** (1976) 113–124.

[22] S. Rai and S. Soh, A computer approach for reliability evaluation of telecommunication networks with heterogeneous link-capacities. *IEEE Trans. Reliab.* **40** (1991) 441–451.

[23] J.B. Rosen, S.Z. Sun and G.L. Xue, Algorithms for the quickest path problem and the enumeration of quickest paths. *Comput. Oper. Res.* **18** (1991) 579–584.

[24] W.J. Rueger, Reliability analysis of networks with capacity-constraints and failures at branches & nodes. *IEEE Trans. Reliab.* **35** (1986) 523–528.

[25] S. Ruzika and M. Thiemann, Reliable and Restricted Quickest Path Problems. In *INOC 2011*. In vol. 6701 of *Lecture Notes Comput. Sci.* (2011) 309–314.

[26] W.L. Wilkinson, An algorithm for universal maximal dynamic flows in a network. *Oper. Res.* **19** (1971) 1602–1612.

[27] G. Xue, End-to-End data paths: Quickest or most reliable? *IEEE Trans. Commun. Lett.* **2** (1998) 156–158.

[28] W.C. Yeh, A Fast Algorithm for Quickest Path Reliability Evaluations in Multi-State Flow Networks. *IEEE Trans. Reliab.* **64** (2015) 1175–1184.