Probability in the Engineering and Informational Sciences, **33**, 2019, 50–63.
doi:10.1017/S026996481700047X

DECOMPOSITION ALGORITHMS TO COMPUTE THE QUICKEST TIME DISTRIBUTION IN DYNAMIC NETWORKS

Chin-Chia Jane
Department of Business Administration, Ling Tung University, Taichung, Taiwan
E-mail: jian2898@gmail.com

Yih-Wenn Laih
Department of Finance, Ling Tung University, Taichung, Taiwan

In dynamic networks, the quickest time is the least possible time required to transmit specified data from the source to the sink. When arcs in dynamic networks are independently subjected to failures, the quickest time is a random variable. Although previous studies have already explored the reliability of the quickest path, this work presents an algorithm that computes the probability distribution of the quickest time from the viewpoint of the quickest flow that contains all possible joint and disjoint paths. For moderate dynamic networks, the proposed algorithm can be easily modified to approximate the quickest time distribution with a trade-off between accuracy and running time. The performance and properties of the exact and modified algorithms are explored through computational experiments, which are conducted on 10 randomly generated networks. The exact algorithm is also compared with the exhaustive method which examines all state vectors.

Keywords: dynamic network, evacuation system, quickest time distribution, reliability

1. INTRODUCTION

To study the evolution of a system over time, Ford & Fulkerson [6] introduced the concept of *dynamic networks*, where flows take discrete time to pass through network arcs. The time is called *transit time*, and the flow is referred to as *dynamic flow* or *flow over time*. Various dynamic network flow problems have been studied [3,5,11,17]. The most basic problem is the dy*namic max-flow problem* [6] which identifies a dynamic flow that maximizes flow from source s to sink$t$ within a specified time horizon. The *quickest flow problem* aims to achieve a feasible dynamic flow that sends a given flow value from s to t within the least possible time [3,11]. For brevity, we define *quickest time* as the least possible time in the quickest flow problem. The *quickest path problem* [4], in which transmission occurs via a single path, is a special case of the quickest flow problem. Dynamic networks are widely utilized to model real-world evacuation problems [2,8,9,16]. According to Hamacher & Tjandra [9] and Tjandra [16], the analysis of the movement distribution of evacuees to determine evacuation time is necessary for evacuation planning. This need inspires the present study to determine the distribution of the quickest time for evacuation system planning.

Given that arcs are subject to failures, Bang, Choo & Mun [1], Ruzika & Thiemann [15], and Xue [18] studied the *quickest most reliable path* (the most reliable among the quickest paths), the *reliable quickest path* (the quickest path among all paths with at least a predefined path reliability), and the *most reliable quickest path* (the quickest path among the most reliable ones), respectively. In case of an emergency, the movement of evacuees should not be restricted to specific disjoint paths. Instead, evacuees should flee to safe areas via all accessible joint and disjoint paths as soon as possible. Contrary to references [1,15,18], the present study aims to compute the probability distribution of the quickest time of dynamic networks from the viewpoint of the quickest flow instead of the quickest path.

While existing references [1,15,18] considered the reliability issues of dynamic networks with binary-state arcs, Lin [12,13] and Yeh [19] enlarged the dynamic network into a dynamic *stochastic-flow network*, in which arc capacities are discrete random variables with multi-integer values. Lin [12,13] and Yeh [19] calculated the reliability that a specified amount of flow can be simultaneously sent from s to t through k disjoint minimal paths within a given time horizon (in [12], $k = 1$) for a dynamic stochastic-flow network. Lin [12,13] 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. Reliability was then calculated in terms of these minimal capacity vectors. Yeh [19] accelerated the search for all minimal capacity vectors on the basis of a depth-first search.

The present work is different from previous studies [12,13,19] in two aspects. First, the present work calculates the reliability issue from the viewpoint of a flow that consists of all joint and disjoint minimal paths instead of k disjoint minimal paths only. This feature is important in analyzing evacuation systems because evacuees escape via all possible paths regardless of whether these paths are disjoint or not. Second, Lin [12,13] and Yeh [19] discussed multi-state dynamic stochastic-flow networks, whereas the present work studies binary-state dynamic flow networks. The proposed algorithm is the first to compute the probability distribution of the quickest time in a dynamic network via the quickest flow rather than via k disjoint paths as references [12,13,19].

In a dynamic network where arc states are binary random variables that switch between success and failure, the quickest time from s to t is a random variable. To compute the probability distribution of the quickest time, a set of state vectors is denoted as an event. An event is classified as determined when all vectors in it have an identical quickest time. Otherwise, the event is undetermined. The undetermined universal event is first divided into disjoint sub-events. Each sub-event is recursively divided until all sub-events are determined. With respect to moderate networks, we set a critical level α and skip undetermined sub-events whose probabilities are smaller than α. Approximation of the probability distribution is achieved by recursively reducing α until the trade-off between accuracy and running time becomes acceptable.

The main contribution of this work is it presents a novel algorithm for the analysis of the probability distribution of the quickest time. After deriving the probability distribution, performance indices, such as the expected quickest time and the reliability (probability) that the quickest time is not longer than a specified time threshold, can be calculated directly. For moderate networks, the proposed algorithm can be easily modified to approximate the probability distribution with a trade-off between accuracy and running time.

The remainder of this paper is organized as follows. Section 2 introduces network models. Section 3 presents the exact algorithm and its modification for the approximation of the probability distribution of the quickest time. Section 4 illustrates the exact algorithm in terms of a benchmark network. Computational experiments are conducted on 10 randomly generated moderate networks to determine the performance and properties of the exact and approximation algorithms. Section 5 presents the concluding remarks and possible directions for future research.

2. NETWORK MODELS

A network $G = [N, A]$ consists of a set N of n nodes and a set A of m directed arcs. Let source s and sink t be two distinct nodes in N. Each arc e in A is weighted with a capacity c_e and a transit time τ_e. Capacity c_e is the maximum amount of data that can be transmitted through arc e per unit time. Suppose that arc e is directed from node u to node v. If 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 node $u \in N$, $A^O(u)$ and $A^I(u)$ are subsets of arcs in A that are directed out of and into u, respectively. A static flow from s to t in G is a function f from A to a non-negative real that satisfies flow conservation constraints (1) and (2) and capacity constraints (3).

$$\sum_{e\in A^O(s)} f(e) - \sum_{e\in A^I(s)} f(e) = \sum_{e\in A^I(t)} f(e) - \sum_{e\in A^O(t)} f(e), \tag{1}$$

$$\sum_{e\in A^O(u)} f(e) - \sum_{e\in A^I(u)} f(e) = 0, \quad \text{for } u \in N\setminus\{s,t\}, \tag{2}$$

$$0 \le f(e) \le c_e, \quad \text{for } e \in A. \tag{3}$$

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 that leaves source node s. A *maximum flow* is the flow that has the maximum value of all flows. If each arc e is associated with w_e, which denotes a cost per unit of flow through e, then the cost of a flow f is $\sum_{e\in A} w_e f(e)$. A *min-cost max-flow problem* involves searching for a maximum flow with the lowest possible cost.

2.2. Dynamic flows

Dynamic flow is defined in network $G = [N, A]$ with a finite time horizon T. The time can either be discrete or continuous. We focus on the discrete time case. Time horizon T is the time until which the flow can travel in the network. It defines set $\mathit{\Gamma} = \{0, 1, \ldots, T\}$ of the analyzed time moments. For $e \in A$ and $\tau \in \mathit{\Gamma}$, a *dynamic flow* is a function h on $A \times \mathit{\Gamma}$ that satisfies the following constraints.

$$\begin{aligned}\sum_{\tau=0}^{T}\sum_{e\in A^O(s)} h(e,\tau) - \sum_{\tau=0}^{T}\sum_{e\in A^I(s),\tau\ge\tau_e} h(e,\tau-\tau_e) &= \sum_{\tau=0}^{T}\sum_{e\in A^I(t),\tau\ge\tau_e} h(e,\tau-\tau_e) \\ &\quad - \sum_{\tau=0}^{T}\sum_{e\in A^O(t)} h(e,\tau),\end{aligned} \tag{4}$$

$$\sum_{e \in A^O(u)} h(e,\tau) - \sum_{e \in A^I(u), \tau \geq \tau_e} h(e, \tau - \tau_e) = 0, \quad \text{for } u \in N \backslash \{s,t\}, \ \tau \in \Gamma, \tag{5}$$

$$0 \leq h(e,\tau) \leq c_e, \quad \text{for } e \in A, \ \tau \in \Gamma. \tag{6}$$

Constraint (4) states that the net flow that leaves source s during T periods equals the net flow that arrives at sink t within the time interval. Constraint (5) states that for each intermediate node u and each time τ, the quantity of flow that leaves u at time τ is equal to the quantity of flow that enters u at time τ. Constraint (6) requires that the flow along an arc should not exceed its capacity. Let $|h| = \sum_{\tau=0}^{T} \sum_{e \in A^O(s)} h(e,\tau) - \sum_{\tau=0}^{T} \sum_{e \in A^I(s), \tau \geq \tau_e} h(e,\tau - \tau_e)$ be the net flow value that leaves s during period T. A *dynamic max-flow* refers to a dynamic flow that has the maximum flow value.

Ford and Fulkerson [6] demonstrated that if the capacities and transit times are discrete and constant, then the dynamic max-flow can be determined by solving a min-cost max-flow problem in an enlarged network $G^+ = [N^+, A^+]$, where $N^+ = N \cup \{t^+\}$ and $A^+ = A \cup \{e^+\}$. Specifically, G^+ is derived from G by (1) adding a node t^+ and an arc e^+ from t to t^+ with capacity $c_e^+ = \infty$ and transit time $\tau_e^+ = -(T+1)$ and (2) setting the cost of each arc e to its transit time τ_e. Let $\boldsymbol{f} = (f(e_1), f(e_2), \ldots, f(e_m), f(e^+))$ be the min-cost max-flow from s to t^+. According to Ford and Fulkerson [6], the value $|h|$ of the dynamic max-flow is $-\sum_{e \in A^+} \tau_e f(e)$.

2.3. Quickest flows

The *quickest flow* is a dynamic flow that sends specified σ units of flow from s to 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 flow from s to t if it satisfies $|h| < \sigma$ when $T < \tau_\sigma$ and $|h| \geq \sigma$ when $T \geq \tau_\sigma$. This study describes the least possible time as the quickest time. Thus, whenever the quickest time is referred to, we omit the phrase "sending σ units of flow from s to t" if this expression is clear from the context. If s and t are not linked, then quickest time T is set as infinite. Otherwise, the quickest time is a finite integer. Burkard *et al.* [3] demonstrated that the maximum amount of flow that can be sent through a network increases with time T. Thus, we can conduct a binary search over time T and solve the dynamic max-flow problem per iteration until the minimum time required to send the given amount of flow is determined. Burkard *et al.* [3] also constructed a strongly polynomial algorithm that is a more complex search algorithm with superior worst-case bounds on time complexity than the binary search algorithm. Lin & Jaillet [11] recently designed a new cost-scaling algorithm for the quickest flow problem that runs in the same time bound as the cost-scaling algorithm of Goldberg and Tarjan. Their result revealed that the quickest flow problem can be solved within the same time bound as the min-cost max-flow problem. Given that the present study aims to discover the probability distribution of the quickest time, the simple binary search method of Burkard *et al.* [3], which is easy to implement, is employed in the proposed algorithm. In practice, we conduct a binary search of a min-cost max-flow in an enlarged network $[N^+, A^+]$ until the minimum time required to send the given amount of flow is reached. The min-cost max-flow that corresponds to the quickest time is represented as $\boldsymbol{f} = (f(e_1), f(e_2), \ldots, f(e_m), f(e^+))$.

2.4. Binary state arcs

In this study, each arc $e_i, 1 \leq i \leq m$, is a binary random variable $\boldsymbol{X}_i$ that assumes the values of good ($\boldsymbol{X}_i = 1$) and failure ($\boldsymbol{X}_i = 0$). Variables $\boldsymbol{X}_i$ are assumed to be statistically independent. A state vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_m)$ with $x_i = 1$ or $x_i = 0$ denotes the states of all arcs. Let $B_1(\boldsymbol{x})$=$\{e_i| \quad x_i = 1, 1 \leq i \leq m\}$ and $B_0(\boldsymbol{x})$= $\{e_i| \quad x_i = 0, 1 \leq i \leq m\}$ be sets of good and failed arcs with respect to vector $\boldsymbol{x}$, respectively. For $B \subseteq A$, let $T(B)$ be the quickest time when only arcs in B are used to send the flow. Under state vector $\boldsymbol{x}$, arcs in $B_0(\boldsymbol{x})$ fail. The *quickest time under* $\boldsymbol{x}$ is $T(B_1(\boldsymbol{x}))$. Given that 2^m state vectors exist, $T(B_1(\boldsymbol{x}))$ becomes a random variable. This study investigates the probability distribution of the quickest time $T(B_1(\boldsymbol{x}))$ under all 2^m possible state vectors.

3. ALGORITHMS

The exhaustive method, which completely enumerates all 2^m state vectors, is the only available method thus far to compute the probability distribution of the quickest time. However, such brute force is inefficient and cumbersome. Thus, we divide the set of 2^m state vectors into disjoint subsets, such that the quickest time under each vector in the same subset is identical. We introduce the concept of an event and its representation. An event, which is a set of state vectors, is then classified as determined or undetermined. A decomposition algorithm is proposed to accurately compute the probability distribution of the quickest time. By trading off accuracy for running time, the exact algorithm can be easily modified to approximate the probability distribution.

3.1. Representation of events

Let ε_i denote arc e_i as good $(x_i = 1)$ and $\bar{\varepsilon}_i$ denote e_i as failed $(x_i = 0)$. An *event*, which is a set of state vectors, is represented by a multiplication of symbols. For example, event $E = \bar{\varepsilon}_i\varepsilon_j\bar{\varepsilon}_k$ is the set of state vectors that satisfies the condition where arc e_j is good, arcs e_i and e_k are failed, and other arcs are stochastic (i.e., they can be either good or failed); that is, $E = \bar{\varepsilon}_i\varepsilon_j\bar{\varepsilon}_k = \{\boldsymbol{x}| \quad x_j = 1,\ x_i = x_k = 0,$ and $x_l = 1$ or 0 for $1 \leq l \leq m$, $l \neq i,\ j,\ k\}$. Event E splits arc set A into three disjoint subsets $A_g(E)$, $A_f(E)$, and $A_s(E)$ by setting $e_x \in A_g(E)$, $e_y \in A_f(E)$, and $e_z \in A_s(E)$, respectively, if and only if e_x is good, e_y is a failed arc, and e_z does not appear in E. For $E = \bar{\varepsilon}_i\varepsilon_j\bar{\varepsilon}_k$, $A_g(E) = \{e_j\}$, $A_f(E) = \{e_i, e_k\}$, and $A_s(E) = A\backslash\{e_i,\ e_j,\ e_k\}$. When arc e is good with probability p_e and failed with probability 1-p_e, the probability of event E is $\Pr(E) = \prod_{e\in A_g(E)} p_e \prod_{e\in A_f(E)} (1 - p_e)$. The universal event Ω is the initial event where all arcs are stochastic (i.e., $A_g(\Omega)$= $A_f(\Omega)$= $\emptyset$ and $A_s(\Omega) = A$). Events D and E are disjoint if and only if an arc e exists, such that $e \in A_g(D) \cap A_f(E)$ or $e \in A_f(D) \cap A_g(E)$.

3.2. Division of events

Recall that $T(B)$ is the quickest time when only arcs in B are used to send the flow. The following lemma was provided by Burcard *et al.* [3].

Lemma 1: *$T(\cdot)$ is a monotonic decreasing function, i.e., for $A_1 \subseteq A_2 \subseteq A$, $T(A_2) \leq T(A_1)$.*

Proof: Given that $A_1 \subseteq A_2$, the quickest time computed by employing a large arc set A_2 is not larger than the quickest time computed by employing a small arc set A_1. Consequently, $T(A_2) \leq T(A_1)$. ∎

Following Corollary proceeds directly from Lemma 1.

Corollary 1: *For event E, $T(A_g(E) \cup A_s(E)) \leq T(A_g(E))$.*

With respect to event E, in which all arcs in $A_f(E)$ are failed arcs, Corollary 1 implies that $T(A_g(E))$ and $T(A_g(E) \cup A_s(E))$ are the largest and smallest quickest times among all state vectors in E, respectively. If $T(A_g(E) \cup A_s(E)) = T(A_g(E))$ then the quickest times under all vectors in E are identical and event E is designated as τ_E-*determined* where $\tau_E = T(A_g(E))$. Otherwise, event E is *undetermined*.

For an undetermined event E (initially, $E = \Omega$), we compute the smallest quickest time $T(A_g(E) \cup A_s(E))$, and let $\boldsymbol{f} = (f(e_1),\ f(e_2),\ \ldots,\ f(e_m),\ f(e^+))$ be the min-cost max-flow that corresponds to the smallest quickest time. We then search *critical arc set* $A_s^*(E)$ by setting $A_s^*(E) = \{e_i | e_i \in A_s(E)$ and $f(e_i) > 0, 1 \le i \le m\} = \{e_1,\ e_2,\ \ldots,\ e_{mE}\}$. Set $A_s^*(E)$, which is a subset of $A_s(E)$, is further utilized to divide event E into disjoint sub-events and to compute the probability distribution of the quickest time. Theorem 1 is fundamental to our decomposition algorithm.

THEOREM 1: *Let $A_s^*(E) = \{e_1,\ e_2,\ \ldots,\ e_{mE}\}$ be a critical arc set with respect to event E.*

(1) If $A_s^*(E) = \emptyset$, then event E is τ_E-determined.
(2) If $A_s^*(E) \neq \emptyset$, then event $E^* = E\varepsilon_1\varepsilon_2 \ldots \varepsilon_{mE}$ is τ_E^*-determined, where $\tau_E^* = T(A_g(E^*))$.

PROOF: Following the definition of $A_s^*(E)$, $T(A_g(E) \cup A_s(E)) = T(A_g(E) \cup A_s^*(E))$.

(1) If $A_s^*(E) = \emptyset$, then $A_g(E) \cup A_s^*(E) = A_g(E)$. Thus, $T(A_g(E) \cup A_s(E)) = T(A_g(E))$, and event E is τ_E-determined.
(2) If $A_s^*(E) \neq \emptyset$, then event $E^* = E\varepsilon_1\varepsilon_2 \ldots \varepsilon_{mE}$, implying that $A_g(E^*) = A_g(E) \cup A_s^*(E)$, $A_f(E^*) = A_f(E)$, and $A_g(E^*) \cup A_s(E^*) = A_g(E) \cup A_s(E)$. Consequently, $\Gamma(A_g(E^*) \cup A_s(E^*)) = \Gamma(A_g(E) \cup A_s(E)) = \Gamma(A_g(E) \cup A_s^*(E)) = \Gamma(A_g(E^*))$. Then, event $E^* = E\varepsilon_1\varepsilon_2 \ldots \varepsilon_{mE}$ is τ_E^*-determined. ■

When $A_s^*(E) \neq \emptyset$, the complement of event $\varepsilon_1\varepsilon_2 \ldots \varepsilon_{mE}$, $(\varepsilon_1\varepsilon_2 \ldots \varepsilon_{mE})^C$ can be expressed as the sum of disjoint sub-events as follows [14]:

$$(\varepsilon_1\varepsilon_2 \ldots \varepsilon_{mE})^C = \bar{\varepsilon}_1 + \varepsilon_1\bar{\varepsilon}_2 + \ldots + \varepsilon_1\varepsilon_2 \ldots \varepsilon_{m_E-1}\bar{\varepsilon}_{m_E}. \tag{7}$$

Consequently, event E can be divided into the following disjoint sub-events.

$$\begin{aligned} E &= E(\varepsilon_1\varepsilon_2 \ldots \varepsilon_{m_E} + (\varepsilon_1\varepsilon_2 \ldots \varepsilon_{m_E})^C) = E\varepsilon_1\varepsilon_2 \ldots \varepsilon_{m_E} + E\bar{\varepsilon}_1 \\ &\quad + E\varepsilon_1\bar{\varepsilon}_2 + \ldots + E\varepsilon_1\varepsilon_2 \ldots \varepsilon_{m_E-1}\bar{\varepsilon}_{m_E}. \end{aligned} \tag{8}$$

THEOREM 1 states that $E^* = \mathrm{E}\varepsilon_1\varepsilon_2 \ldots \varepsilon_{mE}$ is τ_E^*-determined. However, $E\bar{\varepsilon}_1$, $E\varepsilon_1\bar{\varepsilon}_2$, …, and $E\varepsilon_1\varepsilon_2 \ldots \varepsilon_{m_E-1}\bar{\varepsilon}_{m_E}$ are undetermined. The probability of event E^*, $\Pr(E^*)$, contributes to the probability that the quickest time is equal to $\tau_E^* = T(A_g(E^*))$.

Disjoint sub-events $E^i = E\varepsilon_1\varepsilon_2 \ldots \varepsilon_{i-1}\bar{\varepsilon}_i 1 \le i \le m_E$ are contained in storage Z. To minimize the space of Z instead of $\{E^1,\ E^2,\ \ldots,\ E^{mE}\}$, a collection with respect to E is employed. A collection with respect to E consists of E, $A_s^*(E)$, m_E, and index i with an initial value of 0. A collection is a set of $\{E,\ A_s^*(E),\ m_E,\ i\}$. Index i denotes the event E^i that is currently considered for classification and further decomposition. In Z, all collections are stored in a stack (first-in last-out). The advantage of the first-in last-out policy is that it enables Z to hold a minimum number of collections. Whenever E^i is retrieved in accordance with the top collection in Z, i is updated by $i + 1$ on the condition that $i < m_E$. However, if $i = m_E$, then the probability distribution of the quickest time is updated first, and the topmost collection in Z is discarded.

3.3. Decomposition algorithm

Let R^{τ} be the probability that the quickest time (for sending σ units of flow from s to t in network G) is τ. The largest quickest time in G is $\tau^{max} = \tau^* + \sigma$, where τ^* is the largest transit time among all s–t paths. This condition occurs if the s–t path with the largest transit time is the only path that connects s and t, and the capacity of this s–t path is 1. The step-by-step procedure to compute the probability distribution of R^{τ} by the proposed algorithm is presented below.

Input: $G = (N, A)$, s, t, σ, c_i, τ_i, and p_i for each $e_i \in A$.
Output: R^{τ}.
Steps:

```
1. Set $R^{\tau} = 0,\ 1 \leq \tau \leq \tau^{max}$.
2. Set initial event $E$ as the universal event $\Omega$ (i.e., $E = \Omega$).
3. Compute the quickest time $T(A_g(E) \cup A_s(E))$ in accordance with [3].
4. Search critical arc set $A_s^*(E) = \{e_1, e_2, \ldots, e_{m_E}\}$ based on the min-cost max-flow $\boldsymbol{f}$.
5. $Z = \{\{E, A_s^*(E), m_E, i = 0\}\}$//initial collection w.r.t. $E = \Omega$ //
6. if $A_s^*(E) = \emptyset$, then $R^{\infty} = 1$, discard the top collection in $Z$ // where $s$ and $t$ are not
   connected//
7. while $Z \neq \emptyset$ do
8. {if $i = m_E$, then $\{R^{\tau} = R^{\tau} + \Pr(E^*)$, where $E^* = E\varepsilon_1\varepsilon_2\ldots\varepsilon_{m_E}$, $\tau = T(A_g(E^*))$.
9. Discard top collection in $Z$.}
10. else $\{i = i + 1$.
11. Derive an undetermined $E = E^i = E\varepsilon_1\varepsilon_2\ldots\varepsilon_{i-1}\bar{\varepsilon}_i, 1 \leq i \leq m_E$.
12. Compute quickest time $T(A_g(E) \cup A_s(E))$ in accordance with [3].
13. Search critical arc set $A_s^*(E)$ based on the min-cost max-flow $\boldsymbol{f}$.
14. A new collection $\{E, A_s^*(E), m_E, i = 0\}$ is stored on top of $Z$.}
15. }//End of while $Z \neq \emptyset$ do (step 7)//
```

THEOREM 2: *The decomposition algorithm correctly computes the probability distribution of the quickest time.*

PROOF: Step 1 involves initializing all the probabilities of R^{τ} to zero. In steps 2–5, the algorithm starts at the universal event $E = \Omega$ where all arcs are stochastic. The smallest quickest time $T(A_g(E) \cup A_s(E)) = T(A)$ is computed with respect to E, the critical arc set $A_s^*(E) = \{e_1, e_2, \ldots, e_{m_E}\}$ is searched, and $E\varepsilon_1\varepsilon_2\ldots\varepsilon_{m_E}$, $E\bar{\varepsilon}_1$, $E\varepsilon_1\bar{\varepsilon}_2$, $\ldots$, and $E\varepsilon_1\varepsilon_2\ldots\varepsilon_{m_E-1}\bar{\varepsilon}_{m_E}$ are stored in the form of a collection. In step 6, an empty $A_s^*(E)$ implies that s and t are not connected in $G = [N, A]$. Thus, $R^{\infty} = 1$. The quickest time $T(A)$ set to ∞ indicates that the flow never arrives at sink t. If $A_s^*(E)$ is not empty, E is divided into disjoint $E\varepsilon_1\varepsilon_2\ldots\varepsilon_{m_E}$, $E^{\bar{\varepsilon}_1}$, and $E^i = E\varepsilon_1\varepsilon_2\ldots\varepsilon_{i-1}\bar{\varepsilon}_i, 2 \leq i \leq m_E$ in the while loop (steps 7–15). The τ_E^*-determined $E^* = E\varepsilon_1\varepsilon_2\ldots\varepsilon_{m_E}$ is considered in step 8, where $\tau = T(A_g(E^*))$. Accordingly, R^{τ} is updated by R^{τ}+$\Pr(E^*)$. Step 9 involves outputting the top collection in Z that implies that all sub-events $E^i 1 \leq i \leq m_E$ have been examined in steps 10–14, which recursively divide the undetermined sub-event $E\varepsilon_1\varepsilon_2\ldots\varepsilon_{i-1}\bar{\varepsilon}_i, 1 \leq i \leq m_E$. The while loop terminates when no undetermined sub-events remain. ■

THEOREM 3: *Storage Z requires $O(m^2)$ memory space.*

PROOF: Storage Z stores collections with respect to events. A collection w.r.t. event E is a set $\{E, A_s^*(E), m_E, i\}$. Given that an event consists of a maximum of m good/failed arcs and $A_s^*(E) \subseteq A$, a collection requires $O(m)$ space. Given that sub-events $E^i, 1 \leq i \leq m_E \leq m$, are derived from E, a maximum of m collections are stored in Z in a first-in last-out manner. Theorem is thus proven. ■

THEOREM 4: *The decomposition algorithm requires a running time of* $O(min(log\sigma, g_{max}) \cdot m \cdot logn \cdot (m+n \cdot logn) \cdot \#E)$, *where* g_{max}, n, m, *and* $\#E$ *are the value of the maximum static flow and the numbers of nodes, arcs, and sub-events generated, respectively.*

PROOF: In step 1, the largest quickest time $\tau^{max} = \tau^* + \sigma$ can be searched in $O(m + n \cdot \log n)$ time [7]. Steps 2, 7, and 10 require a constant time (i.e., $O(1)$). According to Burkard *et al.* [3], steps 3 and 12 require $O(\min(\log\sigma, g_{max}) \cdot m \cdot \log n \cdot (m+n \cdot \log n))$ time to compute the quickest time. To search the critical arc set $A_s^*(E)$, steps 4 and 13 require $O(m)$ time to examine each arc once. Steps 5 and 14 assign a collection with respect to E into Z, and steps 6 and 9 discard the top collection in Z. All these steps require $O(m)$ time. $\Pr(E^*)$ is computed by multiplying the probabilities of the good arcs in $A_g(E^*)$ and the failed arcs in $A_f(E^*)$. Step 8 requires $O(m)$ time. Step 11 derives event E^i by designating arcs in $A_s^*(E)$ as good or failed and requires $O(m)$ time. Given that the while ... do loop (steps 7–15) is executed $\#E$ times, it requires $O(\min(\log\sigma, g_{max}) \cdot m \cdot \log n \cdot (m+n \cdot \log n) \cdot \#E)$. As a result, $O(\min(\log\sigma, g_{max}) \cdot m \cdot \log n \cdot (m+n \cdot \log n) \cdot \#E)$ is the running time of the proposed algorithm. ■

3.4. Approximation algorithm

Given that 2^m state vectors exist, the number of sub-events $\#E$ grows exponentially in the worst case. As a result, the proposed algorithm requires exponential time to run in the worst case. The direct approach to obtain an acceptable running time for a large network is to limit the growth of $\#E$. In the following, an undetermined sub-event with a probability that is less than a pre-specified critical level α is further designated as an *irrelevant* sub-event. Skipping irrelevant sub-events, we modify the proposed algorithm to approximate the probability distribution R^τ to trade-off accuracy for running time. We empirically verify that the modified algorithm obtains an acceptable accuracy with satisfactory execution time by reducing the critical level α recursively. If $\alpha = 0$, then the modified algorithm and the proposed accuracy algorithm will be identical. The modified algorithm that approximates R^τ is provided as the following pseudo code.

Input: $G = (N, A)$, s, t, σ, α, c_i, τ_i, and p_i for each $e_i \in A$.
Output: R^τ.
Steps:

```
1. Set $R^\tau = 0, 1 \leq \tau \leq \tau^{max}$.
2. Set initial event $E$ as the universal event $\Omega$ (i.e., $E = \Omega$).
3. Compute quickest time $T(A_g(E) \cup A_s(E))$ in accordance with [3].
4. Search critical arc set $A_s^*(E) = \{e_1, e_2, \ldots, e_{m_E}\}$ based on the min-cost max-flow $\boldsymbol{f}$.
5. $Z = \{\{E, A_s^*(E), m_E, i = 0\}\}$//initial collection w.r.t. $E = \Omega$//
6. if $A_s^*(E) = \emptyset$, then $R^\infty = 1$, discard the top collection in $Z$ // where $s$ and $t$ are not
   connected//
7. while $Z \neq \emptyset$ do
8. {if $i = m_E$, then $\{R^\tau = R^\tau + \Pr(E^*)$ where $E^* = E\varepsilon_1\varepsilon_2\ldots\varepsilon_{m_E}$, $\tau = T(A_g(E^*))$.
9. Discard top collection in $Z$.}
10. else {$i = i + 1$.
11. Derive an undetermined $E = E^i = E\varepsilon_1\varepsilon_2...\varepsilon_{i-1}\bar{\varepsilon}_i, 1 \leq i \leq m_E$.
12. If $\Pr(E) >= \alpha$ then // Check sub-event $E$ is *irrelevant* or not//
13. {Compute quickest time $T(A_g(E) \cup A_s(E))$ in accordance with [3].
14. Search critical arc set $A_s^*(E)$ based on the min-cost max-flow $\boldsymbol{f}$.
15. A new collection $\{E, A_s^*(E), m_E, i = 0\}$ is stored on top of $Z$.}
16. else $i = i + 1$}// *Irrelevant* $E$ is skipped//
17. } // End of while $Z \neq \emptyset$ do (step 7) //
```

4. COMPUTATIONAL EXPERIMENTS

In this section, we illustrate the initial implementation of the exact decomposition algorithm based on a benchmark network and compare it with the exhaustive method. Then, we conduct experiments on moderate networks to explore the performance and properties of the exact and approximation algorithms.

4.1. Illustration of the decomposition algorithm

The decomposition algorithm is illustrated in terms of a benchmark network, which is cited from Ford and Fulkerson [6]. It contains 11 nodes and 21 arcs, as shown in Figure 1. For simplicity, the operation and failure probabilities of all arcs are set to 0.9 and 0.1, respectively.

In Figure 1, the minimum capacity among all arcs is 10, and the value of the maximum static flow from s to t is 85. The three demand units, $\sigma = 10, 50$, and 100, are tested based on the aforementioned facts ($\sigma = 10$ implies that any single path can supply the demand, $\sigma = 100$ is a demand over the maximum static flow, and $\sigma = 50$ is a moderate demand). The computational results are listed in Table 1. The infinite quickest time T implies that s and t are not connected. The infinite quickest time is excluded from the computation of the expected quickest time.

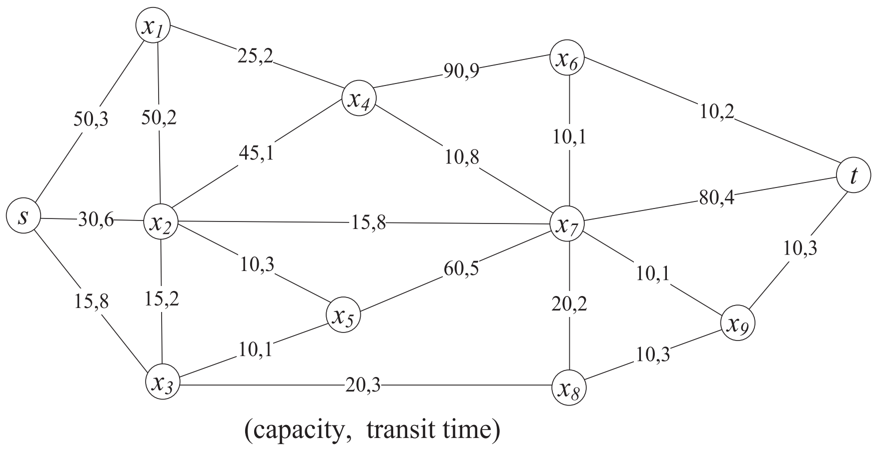

FIGURE 1. The test benchmark network.

TABLE 1. Quickest time distribution of the benchmark network

Probability R_σ^T		Demand σ		
		10	50	100
Quickest time T	∞	0.003855	0.003855	0.003855
	16	0.828794	–	–
	17	0.153349	0.364242	–
	18	0.013962	0.543247	0.593021
	19	0.000038	0.049668	0.250738
	20	0.000001	0.027842	0.040648
	21	0.000001	0.009043	0.066137
	22		0.002072	0.008103
	23		0.000029	0.008065
	24		0.000001	0.000156
	25		0.000001	0.018303
	26			0.008879
	27			0.002064
	28			0.000029
	29			0.000001
	30			0.000001
Expected	quickest time	16.119720	17.707295	18.752770

The following observations are drawn from Table 1.

1. The probability that s and t are not connected is $R_\sigma^\infty = 0.003855$. R_σ^∞ is independent of demand σ.
2. A large demand σ results in a large range of quickest time value T and a large expected quickest time.
3. Given that the minimum capacity among all arcs is 10, in the case of $\sigma \leq 10$, any single path that connects $s$and t can completely send demand σ. In such a case, the time of the quickest flow equals the time of the quickest path. As a result, probability distribution R_{10}^T can be treated as the probability distribution of time to send not larger than 10 units of flow via the quickest path.
4. For each demand σ, the smallest T that has a non-zero probability R_σ^T is the quickest time when all arcs are perfect. In the test example, if all arcs are not subject to failure, the quickest time for demands 10, 50, and 100 are 16, 17, and 18, respectively.
5. The reliability (probability) that the quickest time is not larger than a specified T^* can be easily evaluated by formula $\sum_{T \leq T^*} R_\sigma^T$. For instance, in the case where the demand is 100 units, the reliability that the quickest time is not larger than 20 is $R_{100}^{18}+R_{100}^{19} + R_{100}^{20} = 0.593021 + 0.250738 + 0.040648 = 0.884407$.

The exhaustive method, which completely enumerates all $2^{21} = 2{,}097{,}152$ state vectors in Figure 1, is the only alternative for computing the probability distribution of the quickest time. The execution times of both algorithms in Figure 1 are listed in Table 2.

Table 2 shows that with respect to demands 10, 50, and 100, the CPU times of the exhaustive methods are 7.49 (=4,492/600), 12.21 (=12,942/1,060), and 12.72 (=14,250/1,120) times that of the proposed algorithm, respectively. The running time of the proposed algorithm is superior to that of the exhaustive method.

Table 2. CPU times (unit 0.001 second) of the proposed algorithm and the exhaustive method

	Demand σ		
	10	50	100
Proposed algorithm	600	1,060	1,120
Exhaustive method	4,492	12,942	14,250

4.2. Performance of the proposed algorithms

To investigate the performance and properties of the exact and approximation algorithms, 10 moderate networks that consist of 15 nodes and 35 arcs are generated by the network generator NETGEN [10]. The integer capacity and transit time of each arc are assigned uniformly from sets {10, 15, 40, 60, 90} and {1, 3, 5, 7}, respectively. The demand σ of each network is assigned to its maximum static flow from the source node to the sink node. In addition, to determine how the operation probabilities of arcs affect the performance of the proposed algorithms, operation probabilities of arcs are set to high, middle, and low as following three network families.

H-networks: All the operation probabilities of the arcs in the 10 networks are set to 0.9.

M-networks: All the operation probabilities of the arcs in the 10 networks are set to 0.6.

L-networks: All the operation probabilities of the arcs in the 10 networks are set to 0.3.

We examine six critical levels: 10^{-8}, 10^{-9}, 10^{-10}, 10^{-11}, 10^{-12}, and 0 (the exact algorithm case). To achieve reliable and robust comparisons, the average computational results over the 10 networks in the three network families are obtained, as shown in Table 3.

The following points are drawn from Table 3.

1. For every network family, the CPU time, number of determined sub-events, and expected quickest time increase as α decreases and are upper bounded when $\alpha = 0$. The total probabilities of irrelevant sub-events decrease as α decreases and are lower bounded by 0 (in case $\alpha = 0$).
2. For H-networks, to reach the expected quickest time of 14.144 under $\alpha = 10^{-10}$, which is accurate to the second decimal place, the CPU time of the approximation algorithm is 12.30% of that of the exact algorithm ($\alpha = 0$). Meanwhile, an identical expected quickest time of 14.145 can be obtained in 22.99% CPU time of that of the exact algorithm. The trade-off between accuracy and CPU time is satisfactory.
3. For M-networks, the approximation algorithm obtains expected quickest times accurate to the first and second decimal places under $\alpha = 10^{-10}$ and $\alpha = 10^{-11}$, respectively, whereas the CPU times of the approximation algorithm are 66.96% and 94.02% of that of the exact algorithm ($\alpha = 0$). In addition, the results of critical level $\alpha = 10^{-12}$ indicate that the probability of each sub-event is larger than 10^{-12}. As a result, the approximation algorithm runs similar to the exact algorithm when the critical level is not larger than 10^{-12}. The approximation algorithm is unsuitable for M-networks.
4. For L-networks, under $\alpha = 10^{-9}$, the approximation algorithm consumes 10.77% CPU time of that of the exact algorithm and has an expected quickest time of 4.111, which is precise to the first decimal. To obtain a second decimal precision of the expected quickest time, the approximation algorithm requires 57.38% CPU

TABLE 3. Average computational results of the proposed algorithms w.r.t. three network families

		Critical level α					
		10^{-8}	10^{-9}	10^{-10}	10^{-11}	10^{-12}	0
H-networks	CPU time (unit 0.001 second)	57,871.5 (2.32%)	143,276.5 (5.74%)	306,985.5 (12.30%)	573,683.7 (22.99%)	938,307.3 (37.59%)	2,495,872.5 (100%)
	Number of determined sub-events	324,396.0 (1.22%)	898,743.5 (3.37%)	2,126,438.3 (7.98%)	4,322,007.7 (16.21%)	7,605,355.1 (28.53%)	26,659,939.6 (100%)
	Expected quickest time	14.116	14.139	14.144	14.145	14.145	14.145
	Total probabilities of irrelevant sub-events	1.648×10^{-3}	3.306×10^{-4}	5.537×10^{-5}	7.793×10^{-6}	9.386×10^{-7}	0
M-networks	CPU time (unit 0.001 second)	155,750.2 (6.43%)	616,125.9 (25.46%)	1,620,708.4 (66.96%)	2,275,701.2 (94.02%)	2,419,592.6 (99.97%)	2,420,383.9 (100.00%)
	Number of determined sub-events	875,545.7 (3.28%)	486,6087 (18.25%)	15,609,970.8 (58.55%)	24,570,934.1 (92.16%)	26,659,939.6 (100.00%)	26,659,939.6 (100.00%)
	Expected quickest time	12.799	14.753	15.408	15.476	15.478	15.478
	Total probabilities of irrelevant sub-events	1.903×10^{-1}	4.134×10^{-2}	3.931×10^{-3}	1.160×10^{-4}	0	0
L-networks	CPU time (unit 0.001 second)	79,195.5 (3.29%)	259,013.5 (10.77%)	674,414.2 (28.03%)	1,380,315 (57.38%)	2,014,008.8 (83.72%)	2,405,734 (100%)
	Number of determined sub-events	766,758.6 (2.88%)	2,766,003.2 (10.38%)	7,414,154.8 (27.81%)	1,548,9476.8 (58.10%)	22,577,274.3 (84.69%)	26,659,939.6 (100%)
	Expected quickest time	4.017	4.111	4.138	4.143	4.144	4.144
	Total probabilities of irrelevant sub-events	8.207×10^{-3}	1.977×10^{-3}	3.449×10^{-4}	3.100×10^{-5}	1.684×10^{-6}	0

Figures in the parentheses are the percentages of the approximation algorithm with critical level α relative to the exact algorithm ($\alpha = 0$).

time of that of the exact algorithm. The approximation algorithm is acceptable for L-networks.

In summary, according to the experimental results on moderate networks, the performance of the approximation algorithm depends heavily on the operation probabilities of the arcs and the critical level. The approximation algorithm is satisfactory for systems that consist of high operation probability arcs, is unsuitable for systems that consist of middle operation probability arcs, and is acceptable for systems that consist of low operation probability arcs. The approximation algorithm starts at a large critical level α, then it recursively reduces α until the trade-off between accuracy and running time becomes acceptable. In addition, we performed a comparison of the CPU time of the exact algorithm and that of the exhaustive method. The exhaustive method examines all $2^{35} = 34{,}359{,}738{,}368$ state vectors for more than 24 h, whereas the exact algorithm ($\alpha = 0$) divides all state vectors into an average of 26,659,939.6 determined sub-events within 2,496 s. The exhaustive enumeration of all state vectors is infeasible in practice for moderate networks.

5. CONCLUDING REMARKS AND FURTHER RESEARCH

This study considers dynamic flow networks in which arcs are subject to failures. The quickest time (i.e., the least possible time that a dynamic flow requires to send a given amount of flow from the source to the sink) is a random variable. To evaluate the probability distribution of the quickest time, we utilize the simple binary search method for the quickest flow problem [3]. When the source and sink are not connected, the quickest time is specified as infinite. Otherwise, the quickest time is a finite integer. Consequently, the probability of the infinite quickest time is the probability that the source and sink are not connected. When the source and sink are linked, the expected quickest time and the reliability/probability that the quickest time is not larger than a specified time threshold can be directly obtained from the probability distribution of the quickest time. By skipping irrelevant sub-events whose probabilities are smaller than a pre-specified critical level, the proposed algorithm trades off accuracy for running time. The results of the computational experiments conducted on moderate networks suggest that the efficiency of the approximation algorithm depends on the critical level and the operation probabilities of arcs. It is satisfactory, unsuitable, and acceptable for moderate networks whose arc operation probabilities are high, moderate, and low, respectively.

Contrary to previous studies on the *quickest most reliable path* [1], the *reliable quickest path* [15], the *most reliable quickest path* [18], and the reliability evaluation of dynamic *stochastic-flow networks* in terms of k disjoint minimal paths [12,13,19], the present work is the first to investigate the probability distribution and reliability of the quickest time in dynamic flow networks. The simple binary search method of Burkard *et al.* [3] for computing the quickest time is embedded in the proposed algorithm. Embedding the strongly polynomial algorithms of Burkard *et al.* [3] and Lin & Jaillet [11] into the proposed algorithm improves the running time for large networks.

This article presents the computation of the probability distribution of the quickest time, where the following interesting question arises: "Can we find the quickest time possible to send σ units of data from the source to the sink with a reliability of at least γ without looking at the entire probability distribution?" This question remains a topic for future research. Furthermore, an efficient simulation method that estimates the probability distribution is also worthy of further research.

Acknowledgements

The authors are grateful to Editor, Professor Jamol Pender, and anonymous referees 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. Bang, Y.-C., Choo, H., & Mun, Y. (2003). Reliability problem on all pairs quickest paths. In P.M.A. Sloot, D. Abramson, A.V. Bogdanov, Y.E. Gorbachev, J.J. Dongarra & A.Y. Zomaya (eds.), *Computational Science—ICCS 2003*. Berlin, Heidelberg: Springer, pp. 518–523.
2. Bretschneider, S. & Kimms, A. (2011). A basic mathematical model for evacuation problems in urban areas. *Transp. Res. Part A Policy Pract.* 45(6): 523–539.
3. Burkard, R.E., Dlaska, K., & Klinz, B. (1993). The quickest flow problem. *Z. Oper. Res.* 37(1): 31–58.
4. Chen, Y.L., & Chin, Y.H. (1990). The quickest path problem. *Comput. Oper. Res.* 17(2): 153–161.
5. Fleischer, L. & Skutella, M. (2002). The quickest multicommodity flow problem. In W.J. Cook & A.S. Schulz *Integer Programming and Combinatorial Optimization*. Berlin, Heidelberg: Springer, pp. 36–53.
6. Ford, L.R. & Fulkerson, D.R. (1962). *Flows in networks*. Princeton: Princeton University Press.
7. Fredman, M.L. & Tarjan, R.E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* 34(3): 596–615.
8. Göttlich, S., Kühn, S., Ohst, J.P., & Ruzika, S. (2016). Evacuation modeling: a case study on linear and nonlinear network flow models. *EURO J. Comput. Optim.* 4(3–4): 219–239.
9. Hamacher, H.W. & Tjandra, S.A. (2001). *Mathematical modelling of evacuation problems: a state of art*. Berlin: Springer.
10. Klingman, D., Napier, A., & Stutz, J. (1974). NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow network problems. *Manage. Sci.* 20(5): 814–821.
11. Lin, M. & Jaillet, P. (2015). On the quickest flow problem in dynamic networks: a parametric min-cost flow approach. *In Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*. 1343-1356. SIAM.
12. Lin, Y.K. (2003). Extend the quickest path problem to the system reliability evaluation for a stochastic-flow network. *Comput. Oper. Res.* 30(4): 567–575.
13. Lin, Y.K. (2011). Transmission reliability of k minimal paths within time threshold. *Comput. Ind. Eng.* 61(4): 1160–1165.
14. Locks, M.O. (1982). Recursive disjoint products: a review of three algorithms. *IEEE Trans. Reliab.* 31(1): 33–35.
15. Ruzika, S. & Thiemann, M. (2011). Reliable and restricted quickest path problems. *Lect. Notes Comput. Sci.* 6701: 309–314.
16. Tjandra, S. A. (2003). *Dynamic network optimization with application to the evacuation problem*, PhD thesis, Universität Kaiserslautern, Shaker Verlag, Aachen.
17. Wilkinson, W.L. (1971). An algorithm for universal maximal dynamic flows in a network. *Oper. Res.* 19(7): 1602–1612.
18. Xue, G. (1998). End-to-end data paths: quickest or most reliable? *IEEE Trans. Commun. Lett.* 2(6): 156–158.
19. Yeh, W.C. (2015). A fast algorithm for quickest path reliability evaluations in multi-state flow networks. *IEEE Trans. Reliab.* 64(4): 1175–1184.