

Transmissions in a Network with Capacities and Delays

Dimitrios Kagaris,¹ Grammati E. Pantziou,² Spyros Tragoudas,³ Christos D. Zaroliagis⁴

¹ Electrical Engineering Department, Southern Illinois University, Carbondale, Illinois 62901

² Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece and Technological and Educational Institute of Athens, GR 12210 Aigaleo, Greece

³ Electrical and Computer Engineering Department, The University of Arizona, Tucson, Arizona 85740

⁴ Department of Computer Science, King's College, University of London, Strand, London, WC2R 2LS United Kingdom

Received 22 January 1995; accepted 22 August 1996

Abstract: We examine the problem of transmitting in minimum time a given amount of data between a source and a destination in a network with finite channel capacities and nonzero propagation delays. In the absence of delays, the problem has been shown to be solvable in polynomial time. In this paper, we show that the general problem is NP-complete. In addition, we examine transmissions along a single path, called the quickest path, and present algorithms for general and special classes of networks that improve upon previous approaches. The first dynamic algorithm for the quickest path problem is also given. © 1999 John Wiley & Sons, Inc. *Networks* 33: 167–174, 1999

Keywords: data transmission; sparse network; transmission time; quickest path; dynamic algorithm

1. INTRODUCTION

Consider an n -node, m -edge network $N = (V, E, c, l)$, where $G = (V, E)$ is a directed graph, $c : E \rightarrow \mathbb{R}^+$ is the capacity function and $l : E \rightarrow \mathbb{R}^*$ is the delay function. The

nodes represent transmitters/receivers without data memories and the edges represent communication channels. The capacity $c(e)$ of an edge $e \in E$ represents the amount of data that can be transmitted in a time unit through e . The delay $l(e)$ of an edge $e \in E$ represents the time required for the data to traverse edge e . If σ units of data are to be transmitted from a node u to a node v through edge $e = (u, v)$, then the required transmission time is $l(e) + \lceil \sigma/c(e) \rceil$. Due to the pipelined nature of the transmission, the delay time $l(e)$ is also characterized as the *lead* time of e .

Let $p = (v_1, v_2, \dots, v_k)$ be a path from node v_1 to node v_k . The *capacity of p* is defined as $c(p) = \min_{1 \leq i \leq k-1}$

Correspondence to: G. E. Pantziou; pantziou@cti.gr

Contract grant sponsor: EU ESPRIT Basic Research Actions; contract grant numbers: 7141 (ALCOM II); 9072 (GEPPCOM)

Contract grant sponsor: EU ESPRIT LTR; contract grant number: 20244 (ALCOM-IT)

Contract grant sponsor: NSF; contract grant number: MIP-940990

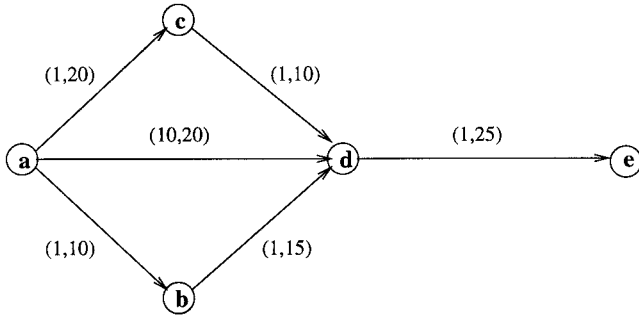


Fig. 1. The minimum transmission time to send $\sigma = 100$ units of data from a to e is 8 through paths $p_1 = (a, b, d, e)$ and $p_2 = (a, c, d, e)$ with $\sigma_1 = 50$ and $\sigma_2 = 50$. [Edge label (l, c) denotes lead time l and capacity c .]

$c(u_i, u_{i+1})$. The definition of the path capacity is motivated by the fact that, since the nodes have no data memories (with the exception of the source node), all data that are received by a node in a time unit must be pumped out of the node in the next time unit. The *lead time* of p is defined as $l(p) = \sum_{i=1}^{k-1} l(u_i, u_{i+1})$. The *transmission time* to send σ units of data from v_1 to v_k along p is $T(\sigma, p) = l(p) + \lceil \sigma/c(p) \rceil$.

Suppose that we want to transmit σ units of data from a designated source $s \in V$ to a designated destination $t \in V$. We wish to compute a partition $\sigma_1, \dots, \sigma_j$ of the σ units of data and a set of paths p_1, \dots, p_j , from s to t (not necessarily edge or node disjoint), in order to transmit the data so that the following conditions are satisfied:

- (i) Each σ_i , $1 \leq i \leq j$, is a positive integer.
- (ii) $\sum_{i=1}^j \sigma_i = \sigma$, where σ_i is the amount of data transmitted along path p_i .
- (iii) The total amount of data that passes through each $e \in E$ in a time unit does not exceed $c(e)$.
- (iv) The transmission time $\max_{1 \leq i \leq j} T(\sigma_i, p_i)$ is minimized over all possible data partitions and groups of transmission paths.

We call this problem the *Minimum Transmission Time (MTT) problem*. An example is given in Figure 1. Note that since we have no memory at the nodes of the network MTT needs, in fact, to secure each of the paths p_i from s to t beforehand and then transmit the data.

In this paper, we first show (Section 2) that the MTT problem is NP-complete, which, in general, precludes a polynomial solution for MTT (unless $P = NP$). Note that in the absence of delays [i.e., $l(e) = 0, \forall e \in E$], which is a special case of the problem, a polynomial time algorithm was already known [10].

An alternative approach to the MTT problem would be to partition the σ units of data as $\sigma_1, \sigma_2, \dots, \sigma_k$, for some positive integer k and then transmit (in a sequential manner)

σ_i units, $1 \leq i \leq k$, along a path which minimizes the transmission time. The proposed partitioning scheme would consist of k stages, and at each stage i , one has to find a single path which minimizes the time for transmitting the σ_i units of data. We also show in this paper (Section 2) that there is no partition of data which, when routed in this manner, yields an overall transmission time less than the time needed for transmitting all σ units of data along a single path from s to t which minimizes the transmission time, called the *quickest path*. For this reason, we also investigate in this paper the quickest path problem. Note that the problem is a restricted version of the MTT one, where no partition of data is required and the transmission is done along a single path. The relation between the quickest path and the MTT problem is new, although the former problem has already been investigated [2, 11]. In particular, the best previous algorithm for the single-pair quickest path problem runs in time $O(rm + rn \log n)$ [2, 11], where r is the number of distinct edge capacities. For sparse networks [i.e., $m = O(n)$], this complexity becomes $O(rn \log n)$. The all-pairs quickest path problem has been solved in $O(\min\{rnm + rn^2 \log n, mn^2\})$ time [9], which for sparse networks becomes $O(\min\{rn^2 \log n, n^3\})$.

In this paper, we give efficient algorithms for the quickest-path problem in general and sparse networks that improve upon previous approaches. More specifically, our algorithm for the single-pair case runs in $O(r^*m + r^*n \log n)$ time, where r^* is a parameter that never exceeds the number of distinct capacities greater than the capacity of the shortest with respect to (w.r.t.) lead time path in N (Subsection 3.1). The benefit of the proposed algorithm is that it takes advantage of the distribution of the capacity values on the network edges, which existing algorithms ignore. Depending on the capacity of the shortest w.r.t. lead time path on the original network and, moreover, on the capacity of the shortest w.r.t. lead time paths in appropriately defined subnetworks, parameter r^* can be as low as 1, or a very small integer, even if $r = m$. Note that in the worst case $r^* = r$, but the proposed algorithm is better in all cases where $r^* < r$.

In addition, we present improved algorithms for the single-pair quickest path problem on sparse networks (Section 3). The complexities of the algorithms are expressed in terms of an additional parameter $\tilde{\gamma}$ which provides a measure of the topological complexity of N and ranges from 1 up to $\Theta(n)$. If N is planar, we give an $O(n \log n + n \log^3 \tilde{\gamma} + r^* \tilde{\gamma})$ -time algorithm. For arbitrary non-planar sparse networks, we obtain an algorithm with time complexity varying from $O(n \log n)$ up to $O(r^*n \log n)$, depending on the particular value of $\tilde{\gamma}$. We also give efficient algorithms for the all-pairs quickest path problem. For planar networks, the problem is solved in $O(rn^2)$ time, while for arbitrary sparse networks, in $O(r\tilde{\gamma}^2 \log \tilde{\gamma} + rn^2)$ time. Our results match the best previous bounds only when *both* parameters

r^* and $\tilde{\gamma}$ reach their extreme values [of r and $\Theta(n)$, respectively]; in all other cases, our bounds are better.

All the above-mentioned results, however, relate to the static version of the problem, that is, the network, the lead times, and the capacities on its edges do not change over time. We also consider (Section 4) a dynamic environment, where edges can be deleted and their lead times and capacities can be modified. In such a case, a most efficient approach is to preprocess the input network N such that, subsequently, *queries*, asking for the quickest path to send σ units of data between any two given nodes, can be efficiently answered. Moreover, after a dynamic change in N , the data structures set up during preprocessing should be efficiently updated. We shall refer to this as the *dynamic quickest path problem*. To the best of our knowledge, there were no previous algorithms for this problem. We present the first dynamic algorithm for the case of planar and sparse networks (Section 4).

2. THE INTRACTABILITY OF THE MTT PROBLEM

We show that the MTT problem is NP-complete. In particular, we show that the decision version of MTT [referred to as the Bounded Transmission Time (BTT)] is NP-complete. A solution of MTT in NP can be obtained by binary search on instances of BTT. BTT is formally defined as follows:

Problem BTT *Input:* Network $N = (V, E, c, l)$ with real edge capacities and real edge delays, a specified pair of nodes s and t , an integer amount of data σ , and a real number τ .

Question: Is there an integer partition $\sigma_1, \dots, \sigma_j$ of $\sigma = \sum_{i=1}^j \sigma_i$ and a set of not necessarily disjoint paths p_1, \dots, p_j from s to t so that the total amount of data that pass through each edge $e \in E$ in a time unit does not exceed $c(e)$, and $\max_{1 \leq i \leq j} T(\sigma_i, p_i) \leq \tau$?

We reduce from the Maximum Length-Bounded Edge-Disjoint Paths (MLEP) problem [8]. An instance of the MLEP problem consists of a directed graph $G = (V, E)$, two specified nodes s and t , and two integers $J, K \leq |V|$. The question is whether G contains at least J mutually edge-disjoint paths from s to t , with every path containing at most K edges. MLEP has been shown to be NP-complete for $K \geq 5$.

Theorem 1. *The BTT problem is NP-complete.*

Proof. BTT is clearly in NP. We show a polynomial time reduction of the MLEP problem to the BTT. We transform graph $G = (V, E)$ of the given instance of MLEP to a network $N = (V, E, c, l)$ for the BTT problem by assigning a capacity $c(e) = 1$ and a lead time $l(e) = 1/K$, $\forall e \in E$. We set $\sigma = J$ for the amount of data to be

transmitted from s to t and require that the transmission be completed within time $\tau = 2$.

If there is a solution to the MLEP problem, then there is clearly a solution to the BTT problem by splitting σ into J parts and routing each part on a single disjoint path. Conversely, assume that there is a solution to the BTT problem on the above network and the specified parameters. First, we observe that each path p_j from s to t in the BTT solution must have been assigned $\sigma_j = 1$ units of data, that is, there must be J paths in the BTT solution; otherwise, if some path p_i had $\sigma_i > 1 \Rightarrow \sigma_i \geq 2$ units of data, then $T(\sigma_i, p_i) = (\sigma_i/1) + l(p_i) \geq 2 + (1/K) > \tau$, which is forbidden. Second, each path p_j must comprise at most K edges, since, otherwise, a path p_i with more than K edges would have $T(\sigma_i, p_i) \geq 1 + (K + 1) \cdot (1/K) > \tau$. Finally, all paths in the BTT solution must be mutually edge-disjoint. Assume that an edge (a, b) were used by two paths p_{j_1} and p_{j_2} in the BTT solution. Let l_1, l_2 be the lead times from s to a along p_{j_1} and p_{j_2} , respectively. Assume that $l_1 \leq l_2$. Edge (a, b) will be occupied with the transmission of data of path p_{j_1} during time interval $[l_1, l_1 + (\sigma_{j_1}/1)] = [l_1, l_1 + 1]$. Since the capacity of (a, b) is $c(a, b) = 1$, the first piece of data of path p_{j_2} must arrive at node a at a time $l_2 > l_1 + 1$. But in such a case, $T(\sigma_{j_2}, p_{j_2}) = (\sigma_{j_2}/1) + l(p_{j_2}) \geq 1 + (l_1 + 1) > \tau$, that is, the J paths in the BTT solution are actually disjoint, constituting a solution for MLEP. ■

In [10], a generalization of the transmission problem was introduced, where each edge e is also associated to a *cost value* $p(e)$. It costs $s \cdot p(e)$ to transmit s units on an edge of cost $p(e)$. The problem is to find the minimum-cost solution that corresponds to a schedule with maximum transmission delay no more than τ . In the decision version of the problem, we want the transmission delay to be at most τ , and the cost, at most P . Assuming the reduction of Theorem 1, we can show that the latter problem is NP-complete by restricting all $p(e)$ to 1, $\tau = 2$, and $P = J \cdot K$. We have

Corollary 2.1. *The min-cost version of the MTT problem is NP-complete.*

We have shown that it is intractable to partition the data and transmit it simultaneously from s to t so that the transmission time is minimized. The difficulty is due to the interactions among the different paths selected for transmitting the data partitions. An alternative approach would be to partition the σ units of data into $\sigma_1, \sigma_2, \dots, \sigma_k$, for some positive integer k , and then transmit (in a sequential manner) each σ_i , $1 \leq i \leq k$, along a single path which minimizes the transmission time. The proposed partitioning scheme would consist of k stages, and at each stage i , we would have to find a single path which minimizes the time for transmitting the σ_i units of data. Lemma 2.1 below shows that there is no partition of data which, when routed

in this manner, yields an overall transmission time less than the time needed for transmitting all σ units of data along a single path from s to t which minimizes the transmission time. The latter path is called the *quickest path*. Lemma 2.1 together with Theorem 1 justify the importance of developing fast algorithms for the quickest path problem. The latter is the subject of Section 3 in this paper.

Lemma 2.1. *Transmitting σ units of data along a single quickest path is at least as fast as partitioning the data and transmitting sequentially each part along a single path.*

Proof. Let the data σ be partitioned into $k > 1$ parts $\sigma_1, \sigma_2, \dots, \sigma_k$ with $\sigma_i > 0, 1 \leq i \leq k$, and $\sum_{i=1}^k \sigma_i = \sigma$. Let q with capacity c and lead time l be the quickest path for transmitting the σ units of data, and let p_i with capacity c_i and lead time l_i be any path to transmit part $\sigma_i, 1 \leq i \leq k$. Let $T(q, \sigma) = (\sigma/c) + l$ be the transmission time along q and $T(p_i, \sigma_i) = (\sigma_i/c_i) + l_i$ be the transmission time along p_i . Since q is the quickest path, we have that $T(q, \sigma) \leq T(p_i, \sigma), \forall_i, 1 \leq i \leq k$. The total time for sequentially transmitting the parts σ_i along the corresponding paths is $\sum_{i=1}^k T(p_i, \sigma_i)$. Assume that $\sum_{i=1}^k T(p_i, \sigma_i) < T(q, \sigma)$. Then, for each $j, 1 \leq j \leq k$, we must have that $\sum_{i=1}^k T(p_i, \sigma_i) < T(q, \sigma) \leq T(p_j, \sigma) \Rightarrow \sum_{i=1}^k ((\sigma_i/c_i) + l_i) < (\sigma/c_j) + l_j \Rightarrow \sum_{i=1, i \neq j}^k l_i + \sum_{i=1, i \neq j}^k (\sigma_i/c_i) < (1/c_j) \sum_{i=1, i \neq j}^k \sigma_i \Rightarrow \sum_{i=1, i \neq j}^k \sigma_i/c_i < (1/c_j) \sum_{i=1, i \neq j}^k \sigma_i \Rightarrow \sum_{i=1, i \neq j}^k \sigma_i (1/c_i - 1/c_j) < 0$. However, for each of the latter sums (for each value of $j, 1 \leq j \leq k$) to be negative, at least one term must be negative, that is, for each $j, 1 \leq j \leq k$, there must be at least one $c_i, 1 \leq i \leq k$, such that $(1/c_i - 1/c_j) < 0 \Leftrightarrow c_j < c_i$. By the pigeonhole principle, this cannot happen. ■

3. RESTRICTING MTT: THE QUICKEST PATH PROBLEM

The quickest path problem [2] is a restricted version of MTT by requiring the transmission of data to be done along a single path from the source to the destination (i.e., no partition of data σ is required). This requirement makes the problem solvable in polynomial time [2, 9, 11].

The computation of a quickest path differs in many aspects from the computation of the related shortest path. First, the latter problem is defined on a network where each edge (u, v) owns only one attribute, namely, the distance from u to v . However, such a kind of network is not applicable in many practical situations, as in the case of a communication network, where the transmission time between two nodes depends not only on the distance but also on the capacity of the edges in the network. Moreover, in the quickest-path problem, the selection of the path depends also on the size of the data to be transmitted. In one extreme case, if the amount of data is huge, then the quickest path should be the path with largest capacity. On the other hand,

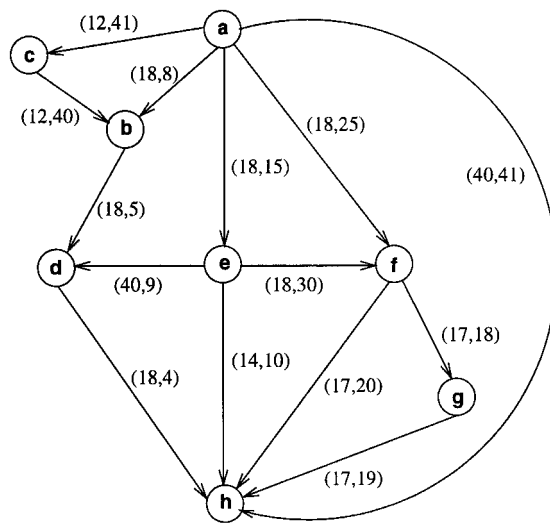


Fig. 2. An example network. Edge label (l, c) denotes edge lead time l and edge capacity c .

if the amount of data is quite small, then the quickest path is the shortest path w.r.t. lead time. An additional singularity of the quickest path problem is that a subpath of a quickest path is not necessarily a quickest path itself. For example, in Figure 2, the quickest path to transmit $\sigma = 100$ units of data from a to d is (a, b, d) with transmission time $36 + \frac{100}{5} = 56$. However, subpath (a, b) is not a quickest path to transmit $\sigma = 100$ units of data from a to b , since path (a, c, b) has smaller transmission time. This fact suggests that the principle of local optimality does not hold and, therefore, algorithms that rely on local optimality (e.g., Dijkstra-like labeling algorithms) could not be used to solve the quickest path problem. This makes interesting the study of different approaches toward the design of efficient algorithms.

3.1. Single-pair Quickest Paths in General Networks

Let $N(V, E, c, l)$ be a network as defined in the Introduction, and let $|V| = n$ and $|E| = m$. Let $C_1 < C_2 < \dots < C_r$ be the r distinct capacity values of the edges of $N, r \leq m$. We define $N^w = (V, E^w, c, l)$ to be a subnetwork of $N(V, E, c, l)$ such that $E^w = \{e : e \in E \wedge c(e) \geq w\}$. In the following, with *shortest lead time path* from u to v , we will refer to the shortest path from u to v with respect to the lead time. The following observation was made in [11]:

Fact 3.1 [11]. *If q is a quickest path, then q is a shortest lead time path in $N^{c(q)}$.*

The algorithm of [11] for computing the quickest path from s to t in N computes the shortest lead time path p_i in each network $N^{C_i}, 1 \leq i \leq r$, and outputs as the quickest

path the one that minimizes the quantity $l(p_i) + \sigma/c(p_i)$, $1 \leq i \leq r$. Using the algorithm of [7] that computes a shortest path in time $O(m + n \log n)$, the overall time complexity of the algorithm is $O(rm + rn \log n)$. For sparse networks, the corresponding complexity becomes $O(rn \log n)$.

The algorithm in [11] can be viewed as seeding serially for the capacity of the quickest path. If a hypothetical oracle would give us the capacity w_o of the quickest path, then the actual path could be found just in time $O(m + n \log n)$ by applying the shortest path algorithm of [7] on N^{w_o} . Below, we show that the seed for the capacity of the quickest path does not have to be serial. Let s^w denote the shortest lead time path in N^w and q the quickest path in N .

Lemma 3.1. *If the capacity of the quickest path q is $c(q) > C_i$ for some $i < r$, then $c(q) \geq c(s^{C_{i+1}})$.*

Proof. Since $c(q) > C_i$ for some $i < r$, q is, in fact, a path in subnetwork $N^{C_{i+1}}$. Let $s^{C_{i+1}}$ be the shortest lead time path in $N^{C_{i+1}}$. Since q is the quickest path, $l(q) + \sigma/c(q) \leq l(s^{C_{i+1}}) + [\sigma/c(s^{C_{i+1}})]$. However, since $s^{C_{i+1}}$ is the shortest lead time path in $N^{C_{i+1}}$, $l(s^{C_{i+1}}) \leq l(q)$. Adding the two inequalities, we get $\sigma/c(q) \leq \sigma/c(s^{C_{i+1}}) \Leftrightarrow c(q) \geq c(s^{C_{i+1}})$. ■

Proposition 3.1. *If for some i , $1 \leq i \leq r$, the capacity of the shortest lead time path s^{C_i} is $c(s^{C_i}) = C_r$, then the capacity of the quickest path q is either $c(q) < C_i$ or $c(q) = C_r$.*

Proof. Assume that $c(q) \geq C_i$. Since $c(q) > C_{i-1}$, then by Lemma 3.1, $c(q) \geq c(s^{C_{i+1}}) \Rightarrow c(q) \geq C_r \Rightarrow c(q) = C_r$. ■

We also observe that some subnetworks N^w may not be connected. In the case that there is no path from s to t in network N^{C_i} for some $i < r$, then s and t will remain disconnected in any network N^{C_j} , $j \geq i$, since N^{C_j} is a subnetwork of N^{C_i} , that is:

Lemma 3.2. *If there is no path from s to t in N^{C_i} for some i , $1 \leq i \leq r$, then the capacity $c(q)$ of the quickest path q is $c(q) < C_i$ (if $i = 1$, no path exists).*

By convention, we assume that if there is no path from s to t in some subnetwork, then the shortest path algorithm returns a path of “infinite” length and “infinite” capacity. Initially, all we know for the capacity of the quickest path is that $c(q) \geq C_1$. According to Lemma 3.1, each application of the shortest path algorithm on network N^{w_i} , where $w_1 = C_1$ and $w_i = c(s^{w_{i-1}})$, $i > 1$, can be regarded as a query that provides us successively with more information for the range of $c(q)$ [namely, $c(q) \geq c(s^{w_i})$]. If for the i -th such successive query it happens that $c(s^{w_i}) = C_r$ or $c(s^{w_i}) = \infty$, then by Proposition 3.1 or Lemma 3.2, respectively, no more than $r^* = i$ queries are required. On the other hand, if $\forall i$, $1 \leq i < r$, $c(s^{w_i}) = w_i \Leftrightarrow c(s^{C_i}) = C_i$, then (and only then) we end up making $r^* = r$ queries.

Hence, r^* is a measure of the number of steps that we need to find the quickest path based on the knowledge provided by Lemmata 3.1 and 3.2 and Proposition 3.1. It is also clear that r^* is at most the number of distinct capacities which are greater than the capacity of the shortest lead time path in N . The above discussion leads to the following algorithm:

ALGORITHM General_Single_Pair_Quickest_Path(N)

1. $w = C_1$; $r^* = 0$;
2. **while** $w < C_r$ **do**
3. $r^* = r + 1$;
4. Find a shortest lead time path p_{r^*} in N^w ;
5. **if** $\exists i < r$ such that $c(p_{r^*}) = C_i$ **then** $w = C_{i+1}$ **else** $w = \infty$;
6. **od**
7. The quickest path is p_k , where index k minimizes $l(p_i) + \sigma/c(p_i)$, $1 \leq i \leq r^*$.

End General_Single_Pair_Quickest_Path

Theorem 2. *Algorithm General_Single_Pair_Quickest_Path correctly finds the quickest path between two given nodes of a general network in $O(r^*m + r^*n \log n)$ time.*

Proof. By Proposition 3.1 and Lemma 3.2, the **while** loop in Step 2 terminates whenever $w = C_r$ or $w = \infty$. The candidate set of shortest lead time paths, from which the quickest path is chosen in Step 7, is justified by Lemma 3.1. Since there are r^* applications of the shortest path algorithm in Step 4, we obtain, by using the algorithm of [7], an overall time complexity of $O(r^*m + r^*n \log n)$. ■

An example where r^* is significantly smaller than r is given in Figure 2. Algorithm General_Single_Pair_Quickest_Path is applied on a network N to transmit $\sigma = 100$ units of data from a to h . The algorithm is applied successively on subnetworks $N^{(4)} = N$ [finding shortest lead time path (a, e, h) with capacity 10], $N^{(15)}$ [finding shortest lead time path (a, f, h) with capacity 20], and $N^{(25)}$ [finding shortest lead time path (a, h) with capacity 41]. It thus terminates in $r^* = 3$ iterations, yielding (a, f, h) as the quickest path. In contrast, the algorithm in [11] would require $r = |E| = 13$ iterations to find the quickest path.

We also observe that Algorithm General_Single_Pair_Quickest_Path can be further enhanced in practice by making sure that among the potentially many paths in N^w with the same shortest lead time the shortest such path with the largest minimum capacity is always chosen. This can be easily done in the same time complexity by modifying slightly the shortest path algorithm used, that is, the comparison for each derived subpath is now based on a vector $(l', -c')$, where l' is the subpath's lead time, c' is the

minimum capacity of the subpath so far, and all comparisons are performed in lexicographic order.

3.2. Computing Quickest Paths in Sparse Networks

In the previous section, we saw that there is a kind of “information redundancy” in the approach of [11] in the sense that not all queries that are asked may, in fact, be necessary. However, another potential source of redundancy may be found in the repeated applications of the shortest path algorithm on network versions that, loosely speaking, do not differ too much, that is, subnetworks N^{C_i} and $N^{C_{i+1}}$ differ only in that $N^{C_{i+1}}$ has some fewer edges than N^{C_i} and, therefore, the information obtained by computing the shortest lead time path in N^{C_i} may be useful in the computation of the shortest lead time path in $N^{C_{i+1}}$. This suggests the use of a dynamic algorithm for computing shortest paths that allows edge-cost updates and/or deletions of edges. Below, we show how dynamic shortest path algorithms can be used advantageously in the quickest path context. Before proceeding to the description of our algorithms, we need the notion of a hammock decomposition.

A *hammock decomposition* [4, 5] is a decomposition of an n -vertex graph G into outerplanar digraphs, called *hammocks*, that satisfy certain separator conditions. The decomposition has the following properties: (i) Each hammock has at most *four* vertices in common with any other hammock, called the *attachment vertices*; (ii) each edge of the graph belongs to exactly one hammock; and (iii) the number $\tilde{\gamma}$ of hammocks is proportional to $g(G) + q$, where G is assumed to be embedded into its orientable surface of genus $g(G)$ so as to minimize the number q of faces that collectively cover all vertices [5]. If G is sparse, then $\tilde{\gamma}$ ranges from 1 up to $\Theta(n)$. [Note that if G is planar, then $g(G) = 0$ and $\tilde{\gamma} = O(q)$. Also, if G is outerplanar then $\tilde{\gamma} = 1$.] As it has been proved in [5], the hammock decomposition can be obtained in time linear to the size of G and an embedding of G into its orientable surface does not need to be provided with the input.

A dynamic shortest path algorithm that works efficiently for planar digraphs and digraphs with small genus [i.e., $g(G) = o(n)$] and that supports edge-cost modifications and/or edge deletions is given in [3]. More precisely, the following result (partially based on the hammock decomposition) has been proved in that paper:

Fact 3.2 [3]. *Given an n -vertex planar (or small genus) digraph G with real-valued edge costs but no negative cycles, there exists an algorithm for the dynamic shortest path problem on G with the following performance characteristics: (i) preprocessing time and space $O(n + \tilde{\gamma} \log \tilde{\gamma})$; (ii) single-pair distance query time $O(\tilde{\gamma} + \log n)$; (iii) single-pair shortest path query time $O(L + \tilde{\gamma} + \log n)$ (where L is the number of edges of the shortest path); and*

(iv) *update time, after an edge-cost modification or edge deletion, $O(\log n + \log^3 \tilde{\gamma})$.*

3.2.1. Single-pair Quickest Paths

We will first give an algorithm for the case of sparse networks which is based on the decomposition of the original network into hammocks and on the dynamic algorithm implied by Fact 3.2 for outerplanar digraphs ($\tilde{\gamma} = 1$). Then, we give a more efficient algorithm for the case of planar networks and networks with small genus. Let L_i be the length of the shortest lead time path from the source s to the destination t in subnetwork $N^{C_i} = (V, E^{C_i})$. Let also E_i be the set of edges with capacity C_i , $1 \leq i \leq r$. Note that $E^{C_{i+1}} = E^{C_i} - E_i$.

The algorithm for sparse networks is as follows:

ALGORITHM Sparse_Single_Pair_Quickest_Path(N)

1. Decompose N into hammocks.
2. Apply the outerplanar preprocessing algorithm of Fact 3.2 to each hammock.
3. **for** $i = 1$ to r **do**
4. Using the outerplanar query algorithm of Fact 3.2, compute in N^{C_i} the shortest lead time paths from s to each attachment node of H_s (where $s \in H_s$) and from each attachment node of H_t (where $t \in H_t$) to t .
5. Substitute each hammock H , in N^{C_i} , by a constant size outerplanar network, called its *sparse representative*, that keeps shortest lead time path information among the attachments nodes of H . (This can be done in time linear in the size of H [3].) Let $N_{\tilde{\gamma}}$ be the resulting network, which is of size $O(\tilde{\gamma})$.
6. In $N_{\tilde{\gamma}}$, compute the four shortest lead time path trees rooted at the four attachment nodes of H_s using the algorithm of [7].
7. Use the shortest lead time path information computed in Steps 4 and 6 to find L_i .
8. For each edge $e \in E_i$, delete e from N^{C_i} and update the hammock H^e containing e using the outerplanar update algorithm of Fact 3.2.
9. **od**
10. Find the index k that minimizes $L_i + \sigma/C_i$, $1 \leq i \leq r$, and obtain the quickest path by applying the shortest path algorithm of [7] on N^{C_k} .

End Sparse_Single_Pair_Quickest_Path

Theorem 3. *The quickest path between a given pair of nodes in an n -node sparse network can be computed in $O(n \cdot \log n + r \cdot (n + \tilde{\gamma} \cdot \log \tilde{\gamma}))$ time.*

Proof. Let us first discuss the correctness of the algorithm Sparse_Single_Pair_Quickest_Path. From Fact 3.1, the quickest path q is a shortest lead time path in $N^{C(q)}$.

Hence, it is enough to compute the length L_i of the shortest lead time path in N^{C_i} , $1 \leq i \leq r$, and then compute the minimum of $L_i + \sigma/C_i$, $1 \leq i \leq r$. Since network $N^{C_{i+1}} = (V, E^{C_{i+1}})$ differs from $N^{C_i} = (V, E^{C_i})$ in that $E^{C_{i+1}} = E^{C_i} - E_i$, and since we have already computed L_i in N^{C_i} , it is clear that L_{i+1} can be computed by deleting the edges in E_i from N^{C_i} . Since each edge of the network belongs to exactly one hammock, once an edge e is deleted from the current network, exactly one hammock H^e needs to be modified, and the shortest lead time path information among its attachment nodes, to be updated. The updated hammock H^e is then substituted by its sparse representative that keeps the new shortest lead time path information among its four attachment nodes, and in this way, the network $N_{\tilde{\gamma}}$ is updated. At the query time, that is, when the distance from s to t is computed in the current network, the single-source algorithm of [7] finds the updated shortest lead time paths from the attachment nodes of H_s to the attachment nodes of H_t . Regarding the time complexity, Steps 1, 2, 4, and 5 take $O(n)$ time by Fact 3.2 and the results in [3, 5]. Step 6 takes $O(\tilde{\gamma} \cdot \log \tilde{\gamma})$ time [7] and Step 7 takes $O(1)$ time. The for-loop of Steps 3–9 consists of r iterations which can be reduced to r^* , as was discussed in the previous subsection. Hence, the overall time complexity of Steps 3–9, excluding Step 8, is $O(r^* \cdot (n + \tilde{\gamma} \cdot \log \tilde{\gamma}))$. Step 8 needs, by Fact 3.2, $O(\log n)$ time per each deleted edge. During the execution of the algorithm, this step is called at most $m = O(n)$ times, contributing a time complexity of $O(n \log n)$ to the total execution time of the algorithm. Finally, Step 10 takes $O(n \log n)$ time [7]. The bound follows. ■

We give now a more efficient algorithm for finding the quickest path between two nodes s and t , in the case where the input network is planar or has small genus. The algorithm is as follows:

ALGORITHM Planar_Single_Pair_Quickest_Path(N)

1. Run the preprocessing algorithm implied by Fact 3.2, to build the appropriate data structures on N for answering shortest lead time path queries.
2. **for** $i = 1$ to r **do**
3. Run the query algorithm of Fact 3.2, to find the length L_i of the shortest lead time path from s to t in N^{C_i} .
4. **for** each edge e in E_i **do**
5. Run the update algorithm of Fact 3.2, to delete e from N^{C_i} .
6. **od**
7. **od**
8. Find the index k that minimizes $L_i + \sigma/C_i$, $1 \leq i \leq r$.
9. Obtain the quickest path by applying the (shortest path) query algorithm of Fact 3.2 on N^{C_k} .

End Planar_Single_Pair_Quickest_Path

Using similar arguments with those in the proof of Theorem 3, we can prove

Theorem 4. *The quickest path between two given nodes of an n -node planar (or small genus) network can be computed in $O(n \cdot \log n + n \cdot \log^3 \tilde{\gamma} + r^* \cdot \tilde{\gamma})$ time.*

3.2.2. All-pairs Quickest Paths

A straightforward algorithm (based on Fact 3.1) for computing all-pairs quickest paths in planar (respectively, sparse) networks is to apply the all-pairs shortest paths algorithm of [4] (respectively [5]) for planar (respectively, sparse) digraphs, on each one of the subnetworks N^{C_i} , $1 \leq i \leq r$. This, in combination with the results in [6], gives an algorithm for the all-pairs quickest path problem which does an $O(r(n + \tilde{\gamma}^2))$ [respectively, $O(r(n + \tilde{\gamma}^2 \log \tilde{\gamma}))$] preprocessing of the subnetworks, such that a shortest lead time path between any two nodes is answered in $O(L + \log n)$ time in each N^{C_i} and, hence, the quickest path can be found in $O(r(L + \log n))$ time. If we want an $O(rL)$ query time algorithm, or, alternatively, to store the quickest path information in the standard form [of rn shortest lead time path trees which requires $\Omega(rn^2)$ preprocessing], the algorithms in [4] (respectively, [5]) can be easily modified to compute all-pairs quickest paths in $O(rn^2)$ [respectively, $O(r(n^2 + \tilde{\gamma}^2 \log \tilde{\gamma}))$] time. In the next section, we will give a more efficient approach to the all-pairs quickest path problem.

4. DYNAMIC QUICKEST PATHS

In this section, we present our solution to the dynamic quickest path problem in the case where the input network is planar or has small genus. Before describing the dynamic algorithm, we define a variation of the input network as follows: Let $N_\infty^w = (V, E, c, l^w)$ be a variation of N such that for each $e \in E$, $l^w(e) = l(e)$ if $c(e) \geq w$ and $l^w(e) = \infty$ otherwise. Our dynamic algorithm consists of three procedures, namely, preprocessing, query, and update ones.

Preprocessing procedure: Construct the networks $N_\infty^{C_i}$, $\forall 1 \leq i \leq r$. Call the preprocessing algorithm implied in Fact 3.2 in each $N_\infty^{C_i}$ and create the appropriate data structures for answering shortest lead time path queries.

Query procedure: Assume that the two query nodes are u and z . Call the distance query algorithm implied in Fact 3.2, in each one of the networks $N_\infty^{C_i}$, $i = 1, \dots, r$, and compute the length $L_i^{u,z}$ of the shortest lead time path from u to z in $N_\infty^{C_i}$. In each $N_\infty^{C_i}$, compute the quantity $L_i^{u,z} + \sigma/C_i$. Let k be the index that minimizes $L_i^{u,z} + \sigma/C_i$, overall $1 \leq i \leq r$. Then, find the shortest lead time path from u to z in $N_\infty^{C_k}$, using the shortest path query algorithm of Fact 3.2.

The *update procedure* is split into two parts: The lead

time update and the capacity update procedure. The first one updates the data structure in the case of a modification to the lead time of an edge, while the second one in the case of a modification to the capacity of an edge. Note that deletion of an edge e corresponds to updating the lead time of e with ∞ .

Lead time update: Let e be the edge whose lead time l_1 is to be modified and let $c(e)$ be its capacity. Let also l_2 be the new lead time of e . Run the update algorithm implied in Fact 3.2 to change the lead time of e in each network N_∞^w , with $w \leq c(e)$, from l_1 to l_2 .

Capacity update: Let e be the edge whose capacity C_i is to be modified and let $l(e)$ be its lead time. Suppose also that C_j is the new capacity of e . If $C_i \leq C_j$, then run the update algorithm implied in Fact 3.2 to change the lead time of e in each network N_∞^w , with $C_i < w \leq C_j$, from ∞ to $l(e)$. Otherwise ($C_j < C_i$), run the update algorithm implied by Fact 3.2 to change the lead time of e in each network N_∞^w , with $C_j \leq w < C_i$, from $l(e)$ to ∞ .

The above discussion is summarized in the following theorem (whose correctness can be easily verified):

Theorem 5. *Given an n -node planar (or small genus) network N , there exists an algorithm for the dynamic quickest path problem on N that supports edge lead time modification, edge-capacity modification, and edge deletion, with the following characteristics: (i) preprocessing time $O(r(n + \tilde{\gamma} \log \tilde{\gamma}))$; (ii) single-pair quickest path query time $O(r(\tilde{\gamma} + \log n) + L)$, where L is the number of edges of the quickest path; and (iii) update time $O(r(\log n + \log^3 \tilde{\gamma}))$, after any modification and/or edge deletion.*

5. CONCLUSIONS

We investigated the problem (MTT) of transmitting in minimum time a given amount of data between a source and a destination in a network with finite channel capacities and nonzero propagation delays. We have shown that the problem is NP-complete. It is an open problem to examine the complexity of the MTT problem for special classes of networks. We also concentrated on a restricted version of the MTT problem, called the quickest path problem, for which we presented algorithms for general, sparse, and planar networks that improve upon previous approaches.

We have also given the first dynamic algorithms for computing quickest paths in planar networks and networks with small genus.

We are grateful to the anonymous referee for carefully reading the paper and making several valuable comments and suggestions. This work was done while the second author was at the University of Central Florida, the third author was at the Southern Illinois University, and the fourth author was at the Max-Planck-Institut für Informatik.

REFERENCES

- [1] J.A. Bondy and U.S. R. Murty, Graph Theory with Applications, North-Holland, New York, 1976.
- [2] Y.L. Chen and Y.H. Chin, The quickest path problem, *Comput Oper Res* 17 (1990), 153–161.
- [3] H.N. Djidjev, G.E. Pantziou, and C.D. Zaroliagis, On-line and dynamic algorithms for shortest path problems. [Proc. 12th Symp. on Theoretical Aspects of Computer Science (STACS'95)], LNCS 900, Springer-Verlag, 1995, pp. 193–204.
- [4] G.N. Frederickson, Planar graph decomposition and all pairs shortest paths, *J ACM* 38 (1991), 162–204.
- [5] G.N. Frederickson, Using cellular graph embeddings in solving all pairs shortest path problems, *J Alg* 19 (1995), 45–85.
- [6] G.N. Frederickson, Searching among intervals and compact routing tables [Proc. 20th International Coll. on Automata, Languages and Progr. (ICALP'93)], LNCS 700, Springer-Verlag, 1993, pp. 28–39.
- [7] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J ACM*, 34 (1987), 596–615.
- [8] M.R. Garey and D.S. Johnson, Computers and intractability. A guide to the theory of NP-completeness, W.H. Freeman, New York, 1979.
- [9] Y.-C. Hung and G.-H. Chen, On the quickest path problem (Proc. ICCI'91), LNCS 497, Springer-Verlag, 1991, pp. 44–46.
- [10] A. Itai and M. Rodeh, Scheduling transmissions in a network, *J Alg* 6 (1985), 409–429.
- [11] 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.