On the Computation of Fast Data Transmissions in Networks with Capacities and Delays*

(to appear in WADS'95)

Dimitrios Kagaris[1], Grammati E. Pantziou[2],
Spyros Tragoudas[3] and Christos D. Zaroliagis[4]

[1] Electrical Eng. Dept, Southern Illinois University, Carbondale IL 62901, USA
[2] Computer Science Dept, University of Central Florida, Orlando FL 32816, USA
[3] Computer Science Dept, Southern Illinois University, Carbondale IL 62901, USA
[4] Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany

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 non–zero 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–hard. In addition, we examine transmissions along a single path, called the quickest path, and present algorithms for general and sparse networks that outperform previous approaches. The first dynamic algorithm for the quickest path problem is also given.

1 Introduction

Consider an n-node, m-edge network $N = (V, E, c, l)$, where $G = (V, E)$ is a directed graph, $c : E \to \mathbb{N}$ is a capacity function and $l : E \to \mathbb{R}^*$ is a 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 through an edge $e = (u, v)$, then the required transmission time is $l(u, v) + \frac{\sigma}{c(u,v)}$. Due to the pipelined nature of the transmission, the delay time $l(e)$ is also characterized as the *lead* time of e. Let $p = (v_1, v_2, ..., v_k)$ be a path from node v_1 to node v_k. The *capacity of* p is defined as $c(p) = \min_{1 \le i \le k-1} c(u_i, u_{i+1})$. This definition is motivated by the fact that, since the nodes have no data memories, all data that are received by a node in a time unit must be pumped out of that 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 $T(\sigma, p) = l(p) + \frac{\sigma}{c(p)}$.

* This work was partially supported by the EU ESPRIT BRA No. 7141 (ALCOM II) and the NSF grant MIP-940990. E-mail: kagaris@zeus.c-engr2.siu.edu, pantziou@cs.ucf.edu, spyros@cs.siu.edu, zaro@mpi-sb.mpg.de.

Suppose 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 $\sigma_1, \cdots, \sigma_j$ of the σ units of data and a set of paths $p_1, \cdots, p_j$, 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 edge $e \in E$ in a time unit does not exceed $c(e)$; and (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 Fig. 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 [9].

An alternative approach to the MTT problem would be to partition the σ units of data as $\sigma_1, \sigma_2, ..., \sigma_k$, 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. Such a path is called the *quickest path*. The proposed partitioning scheme would consist of k stages, and at each stage i one has to find the quickest path for transmitting the σ_i units of data. We also show here (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 quickest path from s to t. For this reason, we also investigate in this paper the (single) 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 [1, 10]. In particular, the best previous algorithm for the single-pair quickest path problem runs in time $O(rm + rn \log n)$ [1, 10], where r is the number of distinct edge capacities. The all-pairs quickest path problem has been solved in $O(\min\{rnm + rn^2 \log n,\ mn^2\})$ time [8].

In this paper, we give efficient algorithms for the quickest path problem in general and sparse (i.e., $m = O(n)$) networks that outperform the 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 (wrt) lead time path in N (Section 3). 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 wrt lead time path on the original network and moreover, on the capacity of the shortest wrt 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 offers a substantial improvement

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 for the single-pair and the all-pairs quickest path problems compare favorably over the best previous ones and match the best previous bounds only when *both* parameters r^* and $\tilde{\gamma}$ reach their extreme values (of r and $\Theta(n)$, respectively).

All the above mentioned results, however, relate to the static version of the problem. We also consider here (Section 4) a dynamic environment, where edges can be deleted and their lead times and capacities can be modified, and also the amount of data to be transmitted can be changed. In such a case, a more 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 also present here 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–hard. In particular, we show that the decision version of MTT (referred to as Bounded Transmission Time (BTT)) is NP–complete. BTT is formally defined as follows:

Problem BTT. *Input:* Network $N = (V, E, c, l)$ with integer capacities and real positive 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 $\sigma_1, \cdots, \sigma_j$ of $\sigma = \sum_{i=1}^{j} \sigma_i$ and a set of not necessarily disjoint paths $p_1, \cdots, p_j$ 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 [7]. 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 $l(e) = \frac{1}{K}$, $\forall e \in E$. 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 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 $T(\sigma_i, p_i) = \frac{\sigma_i}{1} + l(p_i) \geq 2 + \frac{1}{K} > \tau$, which is forbidden. Secondly, each path p_j must comprise at most K edges, since otherwise a path p_i with more than K edges would have $T(\sigma_i, p_i) \geq 1 + (K + 1) \cdot \frac{1}{K} > \tau$. Finally, all paths in the BTT solution must be mutually edge–disjoint. Assume that an edge (a, b) was 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 $l_1 \leq l_2$. Edge (a, b) will be occupied with the transmission of data of path p_{j_1} during time interval $[l_1, l_1 + \frac{\sigma_i}{1}] = [l_1, l_1 + 1]$. Since the capacity of (a, b) is 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, $T(\sigma_{j_2}, p_{j_2}) = \frac{\sigma_{j_2}}{1} + l(p_{j_2}) \geq 1 + (l_1 + 1) > \tau$. That is the J paths in the BTT solution are actually disjoint, constituting a solution for MLEP. □

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. An alternative approach would be to partition the σ units of data into $\sigma_1, \sigma_2, ..., \sigma_k$, for some positive integer k, and then transmit (in a sequential manner) each σ_i, $1 \leq i \leq k$, along a single path, called the *quickest 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 the quickest path for the σ_i units of data. Lemma 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 quickest path from s to t. Lemma 1 together with Theorem 1, justify the importance of developing fast algorithms for the quickest path problem.

Lemma 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. We first consider partitioning the data σ into two parts $\sigma_1 > 0$ and $\sigma_2 > 0$, $\sigma_1 + \sigma_2 = \sigma$. Let q with capacity c and lead time l be the quickest path for transmitting the σ units of data, and let p_1 and p_2 with capacities c_1 and c_2 and lead times l_1 and l_2 be any paths to transmit σ_1 and σ_2, respectively. Let $T(q, \sigma) = \frac{\sigma}{c} + l$ be the transmission time along q and $T(p_1, \sigma_1) = \frac{\sigma_1}{c_1} + l_1$, $T(p_2, \sigma_2) = \frac{\sigma_2}{c_2} + l_2$ be the transmissions times along p_1 and p_2. Since q is the quickest path, we have that $T(q, \sigma) \leq T(p_1, \sigma)$ as well as $T(q, \sigma) \leq T(p_1, \sigma)$. The total time for sequentially transmitting σ_1 and σ_2 along p_1 and p_2 respectively is $T(p_1, \sigma_1) + T(p_2, \sigma_2)$. Assume that $T(p_1, \sigma_1) + T(p_2, \sigma_2) < T(q, \sigma)$.

Then we must have that $T(p_1, \sigma_1) + T(p_2, \sigma_2) < T(q, \sigma) \leq T(p_1, \sigma)$ which (after some calculations) gives $\frac{\sigma_2}{c_2} + l_2 < \frac{\sigma_2}{c_1}$, or $l_2 < \sigma_2(\frac{1}{c_1} - \frac{1}{c_2}) \Rightarrow \frac{1}{c_1} - \frac{1}{c_2} > 0$.

However, we must also have that $T(p_1, \sigma_1) + T(p_2, \sigma_2) < T(q, \sigma) \leq T(p_2, \sigma)$ which gives $\frac{\sigma_1}{c_1} + l_1 < \frac{\sigma_1}{c_2}$, or $l_1 < \sigma_1(\frac{1}{c_2} - \frac{1}{c_1}) \Rightarrow \frac{1}{c_2} - \frac{1}{c_1} > 0$, which is incompatible. Now it is a matter of a simple induction to finish the proof of the lemma. □

3 Restricting MTT: the Quickest Path problem

The quickest path problem [1] 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 is required). This requirement makes the problem solvable in polynomial time [1, 8, 10]. 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, if the amount of data is quite small, then the quickest path is the shortest path wrt 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 Fig. 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 a Dijkstra-like labeling algorithm could not be used to solve the quickest path problem and makes interesting the study of different approaches towards 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 also $C_1 < C_2 < ... < C_r$ 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 has been made in [10].

Fact 1 *[10] If q is a quickest path, then q is a shortest lead time path in $N^{c(q)}$.*

The algorithm of [10], 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 result of [6], which computes a shortest path in time $O(m+n\log n)$, the overall time complexity of the algorithm is $O(rm+rn\log n)$.

The algorithm in [10] 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 [6] 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 2. *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, $l(q) + \sigma/c(q) \leq l(s^{C_{i+1}}) + \frac{\sigma}{c(s^{C_{i+1}})}$. 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 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 $c(q) \geq C_i$ and $c(q) \neq C_r$. Since $c(q) > C_{i-1}$, then by Lemma 2, $c(q) \geq c(s^{C_{i+1}}) \Rightarrow c(q) \geq C_r \Rightarrow c(q) = C_r$, a contradiction. □

We also observe that some subnetworks N^w may not be connected graphs. 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. *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 2, 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 ith such successive query, it happens that $c(s^{w_i}) = C_r$ or $c(s^{w_i}) = \infty$, then, by Proposition 1 or Lemma 3 respectively, we are done since only $r^* = i$ queries are required to find the quickest path. 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 we need to find the quickest path based on the knowledge provided by Lemmata 2, 3 and Proposition 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 General_Single_Pair_Quickest_Path(N)
1. $w = C_1$; $r^* = 0$;
2. **while** $w < C_r$ **do**
3. $\quad r^* = r^* + 1$;
4. $\quad$ Find a shortest lead time path p_{r^*} in N^w;
5. $\quad$ **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 $l(p_i) + \sigma/c(p_i)$, $1 \leq i \leq r^*$.

Lemma 4. *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. The **while** loop in Step 2 terminates whenever $w = C_r$ or $w = \infty$. This is justified by Proposition 1 and Lemma 3. The candidate set of shortest paths from which the quickest path is chosen in Step 7, is justified by Lemma 2. Since there are r^* applications of the shortest path algorithm in Step 5, we obtain, by using the algorithm of [6], an overall time complexity of $O(r^*m + r^*n \log n)$. □

An example where r^* is significantly smaller than r is given in Fig. 2. Algorithm General_Single_Pair_Quickest_Path is applied on the network N of Fig. 2 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 no path). It thus terminates in $r^* = 3$ iterations, yielding (a, f, h) as the quickest path. In contrast, the algorithm of [10] would require $r = |E| = 13$ iterations to find the quickest path.

The above algorithm 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 by modifying slightly the shortest path algorithm used.

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 [10] 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 $N^{C_{i+1}}$ may be useful in the computation of the shortest lead time path in N^{C_i}. 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* is a decomposition of an n-vertex graph G into certain outerplanar digraphs called *hammocks* [4]. Hammocks satisfy certain separator conditions and the decomposition has the following properties: (i) each hammock has at most *four* vertices in common with any other hammock (and therefore with the rest of the graph), 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 [4]. 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 [4], 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. $\tilde{\gamma}=o(n)$), and which supports edge cost modifications and/or edge deletions is given in [2]. More precisely, the following result (partially based on the hammock decomposition) is proved in [2].

Fact 2 *[2] 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 that supports edge cost modification and edge deletion 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 in the shortest path); (iv) update time (after an edge cost modification or edge deletion) $O(\log n+\log^3\tilde{\gamma})$.*

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 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 consists of the following steps: (1) Decompose N into hammocks. (2) Apply the outerplanar preprocessing algorithm of Fact 2 to each hammock. (3) Repeat steps (3a) through (3e) r times. Description of stage i, $1\leq i\leq r$: (3a) Using the outerplanar query algorithm of Fact 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. (3b) 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 [2].) Let $N_{\tilde{\gamma}}$ be the resulting network, which is of size $O(\tilde{\gamma})$. (3c) 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 [6]. (3d) Use the shortest lead time path information computed in steps (3a) and (3c) to find L_i. (3e) 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 2. (4) 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 [6] on N^{C_k}.

Lemma 5. *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. From Fact 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, i.e., when the distance from s to t is computed in the current network, the single source algorithm of [6] 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), (3a) and (3b) take $O(n)$ time by Fact 2 and the results in [2, 4]. Step (3c) takes $O(\tilde{\gamma} \cdot \log \tilde{\gamma})$ time [6] and Step (3d) takes $O(1)$ time. Step (3) consists of r iterations which can be reduced to r^*, as it was discussed in the previous subsection. Hence, the overall time complexity of Step (3), excluding Step (3e), is $O(r^* \cdot (n + \tilde{\gamma} \cdot \log \tilde{\gamma}))$. Step (3e) needs, by Fact 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 (4) takes $O(n \log n)$ time [6]. 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. We present the algorithm for the planar case. (The other case is similar.) The algorithm consists of the following steps: (1) Run the preprocessing algorithm for planar digraphs implied in Fact 2 to build the appropriate data structures on N for answering shortest lead time path queries. (2) Repeat steps (2a) and (2b) r times. Description of stage i, $1 \leq i \leq r$: (2a) Run the query algorithm for planar digraphs of Fact 2, to find the length L_i of the shortest lead time path from s to t in N^{C_i}. (2b) For each edge e in E_i run the update algorithm of Fact 2 for planar digraphs, to delete e from N^{C_i}. (3) Find the index k that minimizes $L_i + \sigma/C_i, 1 \leq i \leq r$. (4) Obtain the quickest path by applying the (shortest path) query algorithm of Fact 2 on N^{C_k}.

Using similar arguments with those in the proof of Lemma 5, we can prove:

Lemma 6. *The quickest path between two given nodes of an n-node planar network can be computed in* $O(n \cdot \log n + n \cdot \log^3 \tilde{\gamma} + r^* \cdot \tilde{\gamma})$ *time.*

All–pairs quickest paths. A straightforward algorithm (based on Fact 1) for computing all–pairs quickest paths in planar (resp. sparse) networks is to apply the all–pairs shortest paths algorithm of [3] (resp. [4]) for planar (resp. sparse) digraphs, on each one of the subnetworks N^{C_i}, $1 \leq i \leq r$. This, in combination with the results in [5], gives an algorithm for the all–pairs quickest paths problem which does an $O(r(n + \tilde{\gamma}^2))$ (resp. $O(r(n + \tilde{\gamma}^2 \log \tilde{\gamma}))$) 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 [3] (resp. [4]) can be easily modified to compute all–pairs quickest paths in $O(rn^2)$ (resp. $O(r(n^2 + \tilde{\gamma}^2 \log \tilde{\gamma}))$) time. In the next section we will give a more efficient approach to the all-pairs quickest paths 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. We will first give a dynamic algorithm for the case of edge deletions as well as edge lead time and edge capacity modifications. Then, we shall give our dynamic algorithm in the case where the amount of data, to be transmitted between two specific nodes, changes. We shall refer to this latter problem, as the *modified* version of the dynamic quickest path problem.

Before describing the dynamic algorithm, we define a variation of the input network as follows. Let $N = (V, E, c, l)$ be the input network and $C_1 < C_2 < ... < C_r$ be the r distinct capacity values of the edges of N. Let $N^w_\infty = (V, E, c, l^w)$ 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 $N^{C_i}_\infty$, $\forall 1 \leq i \leq r$. Call the preprocessing algorithm implied in Fact 2 in each $N^{C_i}_\infty$ 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 2, in each one of the networks $N^{C_i}_\infty$, $i = 1, \cdots, r$, and compute the length $L_i^{u,z}$ of the shortest lead time path from u to z in $N^{C_i}_\infty$. In each $N^{C_i}_\infty$, 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 $N^{C_k}_\infty$, using the shortest path query algorithm of Fact 2.

Update procedure: There are two update procedures. 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 the edge with an ∞ lead time.

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. Then, the lead time update algorithm uses the update algorithm implied in Fact 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. Then, the capacity update algorithm proceeds as follows: If $C_i \leq C_j$, it uses the update algorithm implied in Fact 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 ($C_j < C_i$), it uses the update algorithm implied by Fact 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 following lemma discusses the correctness and the complexity of the above algorithm.

Lemma 7. *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.*

Proof. From the definition of the network N_∞^w, each path in N_∞^w that has lead time not equal to ∞, has capacity greater than or equal to w. It is not difficult to see that if q is a quickest path in N, then q is a shortest lead time path in $N_\infty^{c(q)}$. Thus, the query procedure correctly computes the quickest path from a node u to a node z. Suppose now that the lead time of an edge e is changed. Then, we have to update all the networks where e belongs to, namely, all the networks N_∞^w with $w \leq c(e)$. If the capacity of an edge e is changed, then we have to change the lead time of e in each network N_∞^w with w between the old and the new capacity of e. The time complexity of the algorithm comes easily by Fact 2. □

Consider now the modified version of the dynamic quickest path problem: The source and destination nodes of the network remain unchanged, and the amount of data to be transmitted from the source to the destination changes more frequently than the network itself changes. Then, we can modify our dynamic algorithms, given above, in such a way that the computation of the shortest lead time distance is incorporated in the preprocessing and the update procedures instead of the query one. This improves considerably the query time without increasing the preprocessing bound, but with degrading the update time by an additive factor of $r\tilde{\gamma}$. More precisely we have the following lemma.

Lemma 8. *Given an n-node planar (or small genus) network N with a source s and a destination t, there exists an algorithm for the modified 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) $s-t$ quickest path query time $O(r+L)$, where L is the number of edges in the quickest path; and (iii) update time $O(r(\tilde{\gamma}+\log n+\log^3\tilde{\gamma}))$, after any modification and/or edge deletion.

References

1. Y. L. Chen and Y. H. Chin, "The quickest path problem", *Computers and Operations Research*, 17, pp. 153–161, 1990.
2. H. N. Djidjev, G. E. Pantziou and C. D. Zaroliagis, "On-line and Dynamic Algorithms for Shortest Path Problems," *Proc. 12th Symp. on Theor. Aspects of Computer Science (STACS'95)*, LNCS 900, pp.193-204, Springer-Verlag, 1995.
3. G. N. Frederickson, "Planar Graph Decomposition and All Pairs Shortest Paths," *J. ACM*, Vol.38, No. 1, pp.162–204, 1991.
4. G. N. Frederickson, "Using Cellular Graph Embeddings in Solving All Pairs Shortest Path Problems", *Proc. 30th Annual IEEE Symp. on FOCS*, 1989, pp.448-453.
5. G.N. Frederickson, "Searching among Intervals and Compact Routing Tables", *Proc. 20th ICALP*, 1993, LNCS 700, pp.28-39, Springer-Verlag.
6. M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, Vol. 34, pp. 596–615, 1987.
7. M.R. Garey, and D.S. Johnson, "Computers and Intractability. A Guide to the Theory of NP-Completeness", W.H. Freeman and Company, New York, NY, 1979.
8. Y.-C. Hung and G.-H. Chen, "On the quickest path problem," *Proc. ICCI'91*, LNCS 497, Springer-Verlag, pp. 44–46, 1991.
9. A. Itai, and M. Rodeh, "Scheduling Transmissions in a Network", *Journal of Algorithms*, 6, pp. 409–429, 1985.
10. J.B. Rosen, S.Z. Sun and G.L. Xue, "Algorithms for the quickest path problem and the enumeration of quickest paths," *Comp. and O.R.*, 18, pp.579-584, 1991.

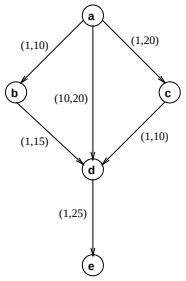

Fig.1: Edge label (l, c) denotes lead time l and capacity c. The minimum transmission time to send $\sigma = 100$ units of data, from a to e, is 8 through paths $p_1 = (a, b, d, e)$ and $p_2 = (a, c, d, e)$, with $\sigma_1 = 50$ and $\sigma_2 = 50$ respectively.

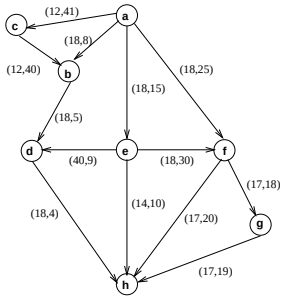

Fig.2: An example network. (Edge label (l, c) denotes edge lead time l and edge capacity c.)