Information Processing Letters 45 (1993) 261–267
Elsevier

2 April 1993

The all-pairs quickest path problem

D.T. Lee * and E. Papadopoulou *

Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA

Communicated by F. Dehne
Received 3 November 1991
Revised 9 October 1992

Keywords: Design of algorithms; analysis of algorithms; data structures; graph algorithms; shortest paths

1. Introduction

The *quickest path problem*, proposed by Chen and Chin [2], is defined as follows. We have a communication network $N = (V, A, c, l)$, where $G = (V, A)$ is a directed graph without multiple arcs and self-loops, $c(u, v) \geq 0$ is the capacity of an arc $(u, v) \in A$, and $l(u, v) \geq 0$ is the lead time of an arc $(u, v) \in A$. For an arc (u, v), the capacity $c(u, v)$ denotes the maximum number of data units which can be transmitted from node u to node v per unit time, and $l(u, v)$ denotes the lead time required to send data from node u to node v. If $\sigma \geq 0$ units of data are transmitted from node u through arc (u, v), then the required transmission time is

$$l(u, v) + (\sigma / c(u, v)).$$

Let $p = (u_1, u_2, \ldots, u_k)$ be a path from u_1 to u_k. Then the lead time $l(p)$ along the path p is

$$l(p) = \sum_{i=1}^{k-1} l(u_i, u_{i+1})$$

and the capacity $c(p)$ of path p is

$$c(p) = \min_{1 \leq i \leq k-1} c(u_i, u_{i+1}).$$

To send σ units of data from u_1 to u_k through path p, the total transmission time required is

$$T_p(\sigma) = l(p) + \sigma / c(p).$$

Given nodes s and t, the quickest path problem is to find a path p from s to t, such that the total transmission time to send $\sigma \geq 0$ units of data from s to t is minimum among all possible paths from s to t in the network N. This problem is a variant of the shortest path problem. But the *optimality* property of shortest paths no longer holds. That is, if p is a shortest path from u to v, then any subpath of p must itself be shortest. For examples and more details see [2]. In [2], Chen and Chin developed an $O(m^2 + nm \log m)$ algorithm, where $m = |A|$, and $n = |V|$, for the *single-pair* quickest path problem. In fact their algorithm is $O(rm + rn \log rn)$, where r is the number of distinct capacity values of N. They also studied the single-pair quickest path problem assuming that the value σ is variable and developed an algorithm of the same complexity to find the single-pair quickest path as a function of σ. In [6], Rosen, Sun, and Xue proposed an alterna-

Correspondence to: Professor D.T. Lee, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208, USA.

* Supported in part by the National Science Foundation under Grant CCR 89-01815.

tive algorithm of the same time complexity, $O(rm + rn \log n)$, but less additional space requirement, for the single-pair problem for a given amount of data σ. A straightforward extension of this algorithm also solves the *single source*, i.e., one-to-all, quickest path problem. They also developed a polynomial algorithm to enumerate the first k quickest paths to send σ units of data from the source of a network N to the sink.

It is obvious that the *all-pairs* quickest path problem for a particular σ can be solved in $O(nrm + n^2 r \log n)$ time by applying the *single source* quickest path algorithm n times. As r can be as large as m, this approach results in $O(nm^2 + n^2 m \log m)$ time complexity in the worst case.

In this paper the *all-pairs* quickest path problem as a function of σ is discussed. That is, we preprocess the network in such a way that given any pair of nodes u, v and any amount of data $\sigma \geq 0$, report the quickest path to transmit the data from u to v. Throughout this paper, n will denote the number of nodes $|V|$, m will denote the number of arcs $|A|$, and r will denote the number of distinct capacities. The preprocessing time is $O(mn^2)$ or $O(rmn + rn^2 \log n)$ if r is small, space is $O(rn^2)$, and query time for a pair of nodes is $O(\log r)$ plus the number of arcs of the actual path if it is to be reported.

2. Preliminaries

Let $N = (V, A, c, l)$ be a network with V, A, c, l defined as above. An important factor to determine the quickest path from node u to node v is the amount of data σ to be transmitted. If σ is small enough then the quickest path is the shortest lead time path. If σ is rather large then the quickest path will be some path of maximum capacity.

Let (u, v) be a pair of nodes, $u, v \in V$, and p be a path from u to v. We call an arc of p of minimum capacity, a *critical arc* of path p. (A critical arc of p is not necessarily unique.) The capacity of p is the capacity of the critical arc.

For some capacity value α, let $N(\alpha)$ be the subnetwork of N induced by the set of arcs of capacity greater than or equal to α. That is, $N = (V, A_\alpha, c, l)$ where $A_\alpha = \{(u, v) \mid (u, v) \in A \text{ and } c(u, v) \geq \alpha\}$.

Property 1. *Let q be the quickest path to transmit σ units of data from node u to node v, and $c(q)$ be its capacity. Then q is the shortest path with respect to lead time from node u to node v in $N(c(q))$.*

Proof. Assume for a contradiction otherwise. Let p' be the shortest lead time path in $N(c(q))$. Thus $l(p') < l(q)$ and $c(p') \geq c(q)$. Then

$$\begin{aligned} T_{p'}(\sigma) &= l(p') + (\sigma / c(p')) \\ &\leq l(p') + (\sigma / c(q)) < l(q) + (\sigma / c(q)) \\ &= T_q(\sigma), \end{aligned}$$

which is a contradiction. □

This property is also mentioned in [6]. It suggests that if we know the capacity c_q of the quickest path to send σ units of data from node u to node v, then the quickest path from u to v is the shortest path with respect to lead time from u to v, considering only arcs of capacity greater than or equal to c_q.

Property 2. *Let p and p' be two different paths from node u to node v, with $c(p') < c(p)$. p' may be quicker than p if and only if $l(p') < l(p)$.*

Proof. Obvious. □

Consider a pair of nodes u, v. We call a path p of capacity $c(p)$ and lead time $l(p)$ *useless* if and only if there is a path q of capacity $c(q) \geq c(p)$ and lead time $l(q) < l(p)$. It is clear by Properties 1 and 2 that such a path p cannot be quickest for any value of σ, since q is always quicker than p, nor can it be a subpath of a quickest path. A path that is not useless is called *useful*.

Let $p_1, p_2, \ldots, p_k$ be the useful paths from u to v ordered in decreasing capacity order. Then

$$c(p_1) > c(p_2) > \cdots > c(p_k),$$
$$l(p_1) > l(p_2) > \cdots > l(p_k).$$

Since there are r distinct capacities, $k \leq r$.

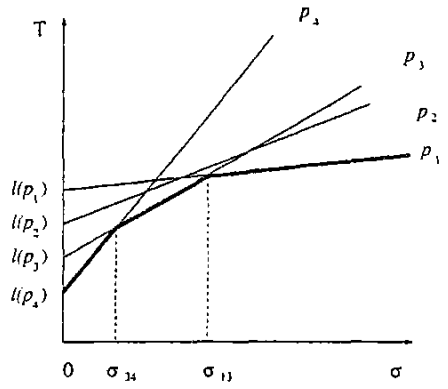

Fig. 1. Example to illustrate the partition of the range of σ for a pair of nodes u, v.

The transmission time $T_p(\sigma)$ of a path p from node u to node v is a linear function of σ with slope $(1/c(p))$. Let $T_{uv}(\sigma)$ denote the minimum transmission time function of σ from u to v.

$$T_{uv}(\sigma) = \min\{T_p(\sigma) \mid p \text{ is a path from } u \text{ to } v \text{ in } N\}.$$

Since T_p is linear for any p, T_{uv} is a piecewise linear function of decreasing slope, each piece corresponding to a different path. In other words we have the following lemma.

Lemma 3. *For any pair of nodes u, v the range of σ can be partitioned into intervals and for each interval there is a path p which is quickest for any σ in that interval.*

In Fig. 1, the vertical axis denotes the transmission time and the horizontal axis denotes the amount of data σ to be transmitted. p_1, p_2, p_3, p_4 are the useful paths from node u to node v ordered in decreasing capacity order. $T_{uv}(\sigma)$ is shown in bold-face and it defines the following partition of the range of σ. For $\sigma \in [0, \sigma_{34}]$ the quickest path is p_4, for $\sigma \in [\sigma_{34}, \sigma_{13}]$ the quickest path is p_3, and for $\sigma \in [\sigma_{13}, \infty]$ the quickest path is p_1.

3. The algorithm and data structure

For each ordered pair of nodes u, v we want to identify $T_{uv}(\sigma)$, or equivalently the partition of the range of σ into intervals as well as the path that is quickest for each interval. For this purpose we construct an $n \times n$ matrix M whose entries $M[u, v]$ contain a list of interval-path nodes. Each interval-path node contains three fields of information:

- *sright*: contains the right end of a data interval,
- C, L: contain the capacity and lead time of the quickest path for that interval.

The list $M[u, v]$ represents $T_{uv}(\sigma)$ from right to left, i.e., it is ordered in decreasing capacity (increasing slope) order. Since the number of intervals cannot exceed the number of distinct capacities r, the space requirement for M is $O(rn^2)$.

In order to be able to retrieve the actual quickest path from a node to another we need to keep a matrix P of useful paths. An entry $P[u, v]$ contains a list of path nodes ordered by decreasing capacity. Each path node represents a useful path Q from u to v, and consists of the following fields:

- C: contains the capacity of the path,
- L: contains the lead time of the path,
- (I, J): contains the vertices of an arc (i, j) the path passes through,
- *leftpath*: contains the index of (or a pointer to) a path node in $P[u, I]$ which contains the shortest lead time path from u to I in $N(C)$,
- *rightpath*: contains the index of (or a pointer to) a path node in $P[J, v]$ which contains the shortest lead time path from J to v in $N(C)$.

Path Q is the concatenation of the path in $P_{uI}[leftpath]$, arc (I, J) if $I \neq J$, and the path in $P_{Jv}[rightpath]$. As the number of useful paths for any pair of nodes is at most r, the space requirement for P is $O(n^2r)$.

To fill in matrices M and P we can do the following. We sort the arcs in non-increasing order according to capacity. Starting from the network $N' = (V, \emptyset)$, we insert arcs in non-increasing capacity order. The insertion of an arc consists of updating three matrices D, M, P, where matrix D is the usual cost matrix of shortest paths, i.e., $D[u, v]$ holds the lead time of the shortest lead time path from u to v in N', and matrices M and P are defined on N'.

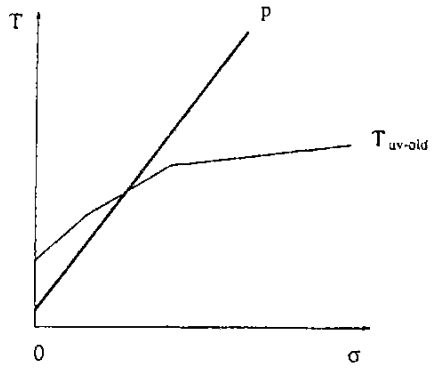

Fig. 2. $T_{uv\text{-}new} = \min\{T_{uv\text{-}old}, T_p\}$.

In the beginning $D[u, v] = \infty$, $M[u, v] = (\infty, 0, \infty)$, $\forall u,v \in V$, $u \neq v$, and $D[v, v] = 0$, $M[v, v] = (\infty, \infty, 0)$, $\forall v \in V$, and P is empty.

The following properties hold:

$$D[v, v] = 0 \quad \forall v \in V,$$

$$D[u, v] = \min_{x \in V} \{D[u, x] + D[x, v]\} \quad \forall u,v \in V.$$

Let (i, j) be the arc to be inserted. Let $d_{old}(u, v)$ denote the lead time (cost) of the shortest lead time path from u to v before the insertion of (i, j), and $d_{new}(u, v)$ denote the lead time of the shortest lead time path after the insertion. It is obvious that if the insertion of the arc (i, j) to N' introduces a shorter lead time path for a pair of nodes u, v, this path will consist of the shortest lead time path from u to i, the arc (i, j), and the shortest lead time path from j to v. Hence

$$d_{new}(u, v) = \min\{d_{old}(u, v), d_{old}(u, i) + l(i, j) + d_{old}(j, v)\} \quad \forall u,v \in V.$$

Consider a pair of nodes u, v. If the insertion of arc (i, j) does not introduce a shorter lead time path from u to v, it cannot introduce a useful path from u to v either because of the non-increasing capacity order of the insertion, hence matrices M, P need not be updated. For the following discussion, suppose that the insertion of arc (i, j) results in a shorter lead time path p from u to v. Since arcs are inserted in non-increasing capacity order, arc (i, j) is critical for p, and p is a useful path from u to v in N'.

Let $T_{uv\text{-}old}(\sigma)$ and $T_{uv\text{-}new}(\sigma)$ denote the minimum transmission time function from u to v before and after the insertion of arc (i, j) respectively. Then

$$T_{uv\text{-}new}(\sigma) = \min\{T_{uv\text{-}old}(\sigma), T_p(\sigma)\}.$$

The transmission time function T_p of the new useful path p has the smallest intercept so far, and so it will contribute the leftmost piece to $T_{uv\text{-}new}(\sigma)$. Unless $T_{uv\text{-}old}$ is a line of the same slope as T_p, in which case $T_{uv\text{-}new} = T_p$, T_p must cross $T_{uv\text{-}old}$ at some point, because it has at least as large a slope as $T_{uv\text{-}old}$. Thus, to find $T_{uv\text{-}new}$ in the general case (when $T_{uv\text{-}new} \neq T_p$), we need to identify the point where T_p crosses $T_{uv\text{-}old}$ (see

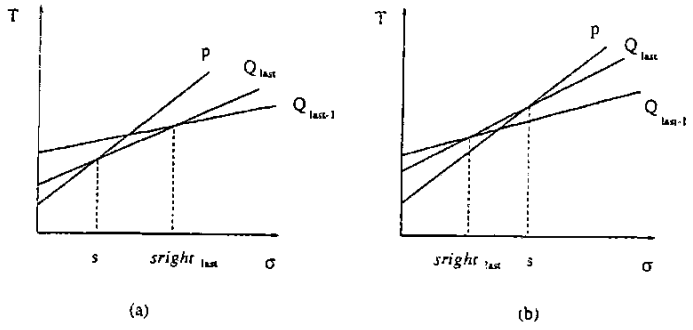

Fig. 3. Example to illustrate the update of matrix M.

Fig. 2). To achieve this, we can scan the pieces of $T_{uv\text{-}old}$ from left to right until the piece intersecting p is found. For this purpose, let *last* denote the last interval-path node in $M[u, v]$ and Q_{last} denote the corresponding path from u to v. If $c(i, j) < C_{last}$, let s denote the intersection on the axis of σ of T_p and $T_{Q_{last}}$. There are two cases (see Fig. 3).

Case 1: $s < sright_{last}$. Then $(s, T_{Q_{last}}(s))$ is the intersection point of T_p and $T_{uv\text{-}old}$. This means that Q_{last} remains the quickest path in $[s, sright_{last}]$ and p becomes the quickest path in $[0, s]$ and thus, s, p should be added to $M[u, v]$.

Case 2: $s \geqslant sright_{last}$. Then Q_{last} is no more a quickest path for any σ and so it should be deleted. In order to find the place where p crosses $T_{uv\text{-}old}$ this process must be repeated until case 1 occurs. If $c(i, j) = C_{last}$ then path Q_{last} becomes useless and the case is equivalent to case 2.

Since p is a useful path, it should be also added to $P[u, v]$. If the capacity of arc (i, j) equals the capacity of the last useful path Q_{last} in $P[u, v]$ before the insertion then Q_{last} becomes useless and it must be deleted before adding p. All other paths in $P[u, v]$ have larger capacity than $c(p)$ thus they remain useful.

Let R_{ij} denote the set of pairs of nodes whose lead time decreases with the insertion of arc (i, j). $R_{ij} = \{(u, v) \mid d_{new}(u, v) < d_{old}(u, v)\}$. Upon insertion of an arc (i, j), matrices D, M, and P should be updated for every $(u, v) \in R_{ij}$. This can easily be done in $O(n^2)$ time by checking every ordered pair of nodes (u, v). To avoid unnecessary comparisons, since in general $|R_{ij}| < n^2$, we can use Rohnert's cost decreasing algorithm [5] or the *ADD* procedure of Even and Gazit [3] that deal with dynamic arc insertions in graphs. The only difference here is that when $D[u, v]$, an entry of the cost matrix, is updated, $M[u, v]$ and $P[u, v]$ should also be updated as explained above.

For simplicity we demonstrate the algorithm without using any dynamic insertion data structure here. For the update of M, P two auxiliary $n \times n$ matrices *Mlast* and *Plast* are needed. $Mlast[u, v]$, $Plast[u, v]$ hold the index of the last entry in $M[u, v]$ and $P[u, v]$ respectively. In the beginning, $Mlast[u, v] = Plast[u, v] = 0$.

Algorithm All-Pairs-Quickest-Path
Input: The network $N = (V, A, c, l)$.
Output: Matrices M and P.

```
begin
  Sort the arcs in non-increasing capacity order
  into arclist;
  Initialize D, M, P, Mlast, Plast
  while arclist ≠ ∅ do begin
   (i, j) := pop(arclist);
   if D[i, j] ⩽ l(i, j) then skip (i, j);
   else
    for all pairs of nodes (u, v) do
     if D[u, i] + l(i, j) + D[j, v] < D[u, v] then
     begin
      D[u, v] := D[u, i] + l(i, j) + D[j, v];
      Update-M(u, v, i, j);
      Update-P(u, v, i, j);
     end;
  end;
end.
```

Procedure Update-M(u, v, i, j)
$[u, v]$ is the entry to be updated.
(i, j) is the inserted arc.

```
begin
  last := Mlast[u, v];
  newnode.L := D[u, v];
  newnode.C := c(i, j);
  if (last > 0) and (M_uv[last].C = c(i, j))
   then last := last − 1;
  while (last > 1) and (intersection(newnode,
   M_uv[last]) ⩾ M_uv[last].sright) do
   last := last − 1;
  if (last = 0) then newnode.sright := ∞
  else newnode.sright := intersection(newnode,
   M_uv[last]);
  M_uv[last + 1] := newnode;
  Mlast(u, v) := last + 1;
end;
```

Procedure Update-P(u, v, i, j)
$[u, v]$ is the entry to be updated.
(i, j) is the inserted arc.

```
begin
  last := Plast[u, v];
  newnode.L := D[u, v];
  newnode.C := c(i, j);
```

```
newnode.I := i;
newnode.J := j;
if u ≠ i then newnode.leftpath := Plast[u, i];
else newnode.leftpath := 0;
if v ≠ j then newnode.rightpath := Plast[j, v];
else newnode.rightpath := 0;
if (last > 0) and c(i, j) = P_uv[last].C
  then last := last − 1;
P_uv[last + 1] := newnode;
Plast[u, v] := last + 1;
end;
```

The function *intersection*(p_1, p_2) returns the value of σ such that $T_{p_1}(\sigma) = T_{p_2}(\sigma)$.

Lemma 4. *Throughout the algorithm we maintain the following invariants. $D[u, v]$ gives the lead time of the shortest lead time path from u to v in N'. $M[u, v]$ gives a partition of the range of σ into intervals and for each interval the capacity and lead time of the corresponding quickest path in N' and $P[u, v]$ gives a list of all useful paths from u to v in N'.*

Proof. We prove by induction on the number of insertions. At the beginning N' contains no arcs and D, M, P are initialized appropriately. Thus the basis of the induction is true.

Suppose that the lemma is true before the insertion of arc (i, j). Let $u, v \in V$. $D[u, v]$, $M[u, v]$, $P[u, v]$ are updated if and only if $D[u, i] + l(i, j) + D[j, v] < D[u, v]$, i.e., if and only if the insertion of arc (i, j) results in a new useful path from u to v. From the preceding discussion it is clear that $D[u, v]$ is updated to $d_{new}(u, v)$, $M[u, v]$ to $T_{uv\text{-}new}(\sigma)$, and $P[u, v]$ to the new list of useful paths, i.e., the lemma remains true after the insertion. Note that $D[v, v]$, $M[v, v]$, $P[v, v]$ are never updated for any $v \in V$, and so they always hold their initial values which are appropriate. □

Lemma 5. *For any pair of nodes $u, v \in V$, and for any $\sigma > 0$ there is a quickest path to send σ units of data from u to v in N, which uses only arcs inserted in N'.*

Proof. Assume for a contradiction that there is a pair of nodes $u, v \in V$ and a value $\sigma_0 > 0$ such that any quickest path q to send σ_0 units of data from u to v uses arc (i, j) which was skipped by the algorithm. Let $D_{old}[i, j]$ denote the value of $D[i, j]$ before arc (i, j) was considered. Then $D_{old}[i, j] \leqslant l(i, j)$. This means that there is a path p from i to j with $l(p) \leqslant l(i, j)$ and because arcs are inserted in non-increasing capacity order, $c(p) \geqslant c(i, j)$. Let q' denote the part of q except arc (i, j), i.e., $q = q' \cup (i, j)$. Let $p' = q' \cup p$; p' is a path from u to v in N. Then $T_q(\sigma_0) = l(q) + \sigma_0/c(q) = l(q') + l(i, j) + \sigma_0/\min(c(q'), c(i, j))$. But $\min(c(q'), c(i, j)) \leqslant \min(c(q'), c(p))$. Thus

$$T_q(\sigma_0) \geqslant l(q') + l(p) + \sigma_0/\min(c(q'), c(p)) = T_{p'}(\sigma_0)$$

which is a contradiction. □

By Lemmas 4 and 5, at the end of the algorithm matrices M and P are correctly constructed for network N.

The time required to update $D[u, v]$ and $P[u, v]$ after an arc insertion is constant. Updating $M[u, v]$, a number of deletions may take place in Case 2. But the paths deleted are all distinct useful paths in N and there are at most r distinct useful paths from u to v in N. Thus, throughout the algorithm updating $M[u, v]$ cannot require more than r deletions.

The time required for the insertion of arc (i, j), without counting the time required for possible deletions while updating M, is $O(n^2)$. There are m arcs to insert, thus the total time requirement to construct matrices M and P is $O(mn^2)$ plus $O(m^2)$ for the deletions while updating M. Since $r \leqslant m$, the time required is $O(mn^2)$. Using the dynamic insertion data structures will not change the worst case time bound of the algorithm, but will improve its average performance.

There may be cases where there is some capacity value c_α such that the number of arcs of capacity c_α is large. In such a case it may be preferable to use an all-pairs shortest path algorithm on $N(c_\alpha)$ rather than updating D after considering each arc of capacity c_α. Let D_α be

the all-pairs lead time (cost) matrix of $N(c_\alpha)$ and let P'_α be a matrix where $P'_\alpha[u, v]$ holds an intermediate vertex k that the shortest lead time path from u to v in $N(c_\alpha)$ passes through [1]. Any all-pairs shortest path algorithm gives those matrices or similar ones. Then we can update D, M, P as follows. For each pair u, v such that $D[u, v] \neq D_\alpha[u, v]$ ($D[u, v] > D_\alpha[u, v]$) update $D[u, v]$ to $D_\alpha[u, v]$, update $M[u, v]$ as before for path of capacity c_α and lead time $D_\alpha[u, v]$, and add to the end of the list in $P[u, v]$ the new useful path of capacity c_α and lead time $D_\alpha[u, v]$ with fields I, J set to $P'_\alpha[u, v]$ and fields *leftpath* and *rightpath* set to 0. At the end, for those entries $P[u, v]$ that were changed and (u, v) is not an edge of $N(c_\alpha)$, assign to fields *leftpath* and *rightpath* of the last entry of $P[u, v]$, the index of the last entry of $P[u, k]$ and $P[k, v]$ respectively, where k is the vertex in $P'_\alpha[u, v]$. It is easy to see that the update of matrices D, M, P is equivalent to the one we would get if arcs of capacity c_α were inserted one by one in N'.

Using Fibonacci Heaps of Fredman and Tarjan [4] the all-pairs shortest path problem in $N(c_\alpha)$ can be solved in $O(n^2 \log n + nm_\alpha)$ time, where $m_\alpha = |A_{c_\alpha}|$. Hence, the time required to construct matrices M and P is $O(\min\{mn^2, rn^2 \cdot \log n + rnm\})$. If r is constant the latter method will yield the same time complexity as the all-pairs shortest path problem.

4. Answering queries

Once matrices M and P are available it is easy to answer quickest path queries. Given a pair of nodes s, t and an amount of data σ_0 to transmit from s to t we need to do the following to identify the quickest path.

1. Do binary search in $M[s, t]$ to identify the node k such that $M_{st}[k-1].sright < \sigma_0 \leqslant M_{st}[k].sright$. Then the fields C and L of $M_{st}[k]$ give the capacity and lead time of the quickest path to send σ_0 units of data from s to t, i.e., the transmission time is $L + \sigma_0/C$. If the actual path is needed then
2. Do binary search in $P[s, t]$ to determine the path node of that capacity. (Path nodes are ordered in decreasing capacity order.) Since a quickest path is certainly useful, there is a path node in $P[s, t]$ with the same capacity.
3. By following fields *leftpath* and *rightpath* we can retrieve the actual path in time proportional to its number of arcs.

Thus, the query time is $O(\log r + k)$ where k is the number of arcs of the actual path. By Property 1, the useful path retrieved is indeed the quickest path from s to t for that amount of data.

5. Conclusion

In this paper we discussed the quickest path problem and proposed an $O(rn^2)$ space data structure which allows to answer quickest path queries in $O(\log r)$ time and whose construction requires $O(\min\{mn^2, rn^2 \log n + rnm\})$ time.

Acknowledgment

The authors would like to thank the anonymous referee for the suggestions that have helped improve the presentation of the paper.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms* (Addison-Wesley, Reading, MA, 1983).
[2] Y.L. Chen and Y.H. Chin, The quickest path problem, *Comput. Oper. Res.* 17 (1990) 153–161.
[3] S. Even and H. Gazit, Updating distances in dynamic graphs, *Methods Oper. Res.* 49 (1985) 371–387.
[4] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* 34 (1987) 596–615.
[5] H. Rohnert, A dynamization of the all pairs least cost path problem, in: *Proc 2nd Ann. Symp. on Theoretical Aspects of Computer Science* (1985) 279–286.
[6] J.B. Rosen, S.Z. Sun and G.L. Xue, The quickest path problem and the enumeration of quickest paths, Manuscript.