A CORRECTION ALGORITHM FOR TOKEN-PASSING SEQUENCES IN MOBILE COMMUNICATION NETWORKS

by

Y.I. Gold and S. Moran

Technical Report #428

August 1986
A CORRECTION ALGORITHM FOR TOKEN-PASSING SEQUENCES
IN MOBILE COMMUNICATION NETWORKS

by

Yaron I. Gold

and

Shlomo Moran

ABSTRACT

We present a distributed approximation algorithm for the Travelling Salesman Problem, that corrects an existing tour, rather than computing one "from scratch". Under certain circumstances the correction procedure is less costly than the "from scratch" procedure: the correction algorithm is suitable for dynamic graphs with slowly changing edge weights, and for which a Travelling Salesman tour (optimal or approximate) was previously computed and is "deteriorating" with time due to the weight changes. The algorithm can be used to "refresh" the tour whenever it deteriorates beyond a given level, and thus maintain a reasonable average tour length at relatively low computation and communication costs. For a Euclidean graph with \( n \) nodes laid out in a bounded area with diameter \( D \), the maximal length of the tour produced by the algorithm is proportional to \( D \sqrt{n} \), like the maximal length of an optimal tour in that graph (the two differ by a factor of 2 at the worst case).

The algorithm was designed for the practical application of maintaining a short token-passing path (which means low scheduling overhead) in certain communication networks with mobile nodes.
1. INTRODUCTION

Multi-access broadcast channels (e.g. radio) have become popular communication media for computer networks and other distributed systems. Networks using such a shared channel differ from networks that use point-to-point links (e.g. a telephone network) in several ways. Two major distinctive features of a multi-access broadcast channel are:

(a) Each single transmission is received by more than one node (and often by all nodes) and is hence considered a broadcast.

(b) All nodes can access the channel for transmission at any time, but a node can receive correctly only one transmission at a time. Simultaneous or overlapping transmissions cause loss of information.

Due to (b) the nodes must use some mechanism that controls the access to the common channel. Such a mechanism is called an "access protocol" and a variety of such protocols exists [15]. An access protocol that completely avoids overlapping transmissions is said to be "conflict-free". A family of distributed conflict-free access protocols, called "round-robin token-passing" protocols make very efficient use of the channel. These protocols schedule the nodes to access the channel in a cyclic sequence. A node that does not want to use its access rights (i.e., has nothing to transmit) "passes the token" to the next node in the sequence. The token may be an actual message or may be implemented as a distributed scheduling function using "time-out" mechanisms (a virtual token) [2,8,9,14]. In efficient token-passing protocols [e.g. 8,9] the time it takes to pass the token from some node, say, node-i, to the next node in the sequence, say, node-j, is equal to the propagation delay, $d(i,j)$, between these two nodes. It can be shown [9] that in networks where propagation delays are long with respect to message transmission time (e.g. high speed networks), the performance of such a protocol depends largely on the time it takes to pass
the token once around all the nodes in the network. An optimal token-passing sequence is one that minimizes this time. In Section 2 we demonstrate that selecting an optimal sequence among \( n \) nodes that are laid out randomly in an area with diameter (i.e., end-to-end propagation delay) \( D \), can reduce token-passing time from \( O(nD) \) to \( O(\sqrt{nD}) \). If the network is represented as a weighted graph with the nodes as vertices and the propagation delays as the costs of the edges (i.e., \( d(i,j) \) is the cost of edge \( (i,j) \)), then selecting an optimal sequence is equivalent to solving the (Euclidean) Travelling Salesman Problem (TSP) with respect to this graph, a problem that is NP-hard. In networks with constant topology an optimal sequence can be computed once before network initialization and used thereafter. However if the topology is dynamic (e.g. the network contains mobile nodes) it may be necessary to recompute the transmission sequence from time to time to maintain good performance.

Due to the difficulty of the TSP, finding an optimal sequence may be prohibitively costly. Several algorithms for constructing a suboptimal TSP tour exist [3,6,7,11,13]. In [7] we describe an algorithm that is specially adapted to multi-access broadcast networks.

To maintain a reasonable level of performance, updates must be performed often enough so that token-passing time will not be allowed to deteriorate beyond a given level. Node-travel between consecutive updates will thus be rather limited, and in many cases the newly computed sequence will only slightly differ from the previous one. This reasoning suggests that work can be saved if the updating is made by correcting the previous sequence rather than recomputing a new one "from scratch". For example: in many practical applications nodes move in groups (e.g., planes in formation, cars along streets, etc.). Such groups will most likely constitute segments of consecutive nodes in most TSP sequences, and hence, when the sequence is updated after token passing time has deteriorated enough to justify an update, the new sequence will contain many of the old segments. The algorithm described in this paper is based on this observation.
For our discussion we assume a model with the following properties:

(a) A cyclic order is imposed on the nodes at any given time. Without loss of generality we can assume that node indexes are assigned according to this order (i.e., node-$i$ immediately precedes node-$(i+1)$, and node-$N$ precedes node-1. In the following, "+" or "-" in the context of node-indexes will always be modulo-$n$). A natural access protocol for such a network is a round-robin protocol that has this order as its transmission order.

(b) A cost $d(i,j) = d(j,i)$ is associated with each edge $(i,j)$. This cost is equal to the propagation delay between node-$i$ and node-$j$ ($G$ is thus a weighted, undirected graph). The information on edge-costs is distributed among the $n$ nodes such that each node-$i$ stores only the values $d(i,i+1)$ and $d(i,j)$ for all $j$.

(c) Network topology changes with time, and unless nodes perform a topology measurement they do not have knowledge on any of the edge-costs.

(d) Network topology changes very slowly relative to signal propagation speed. Topology measurement procedures are viable if their processing time is at the order of magnitude of a round-trip propagation delay. The protocol described in [7], whose processing time equals about two round trip propagation delays, can be used to obtain the required knowledge and store it as described in (b) above.

The rest of the paper is organized as follows: Section 2 contains preliminaries about shortest and expected lengths of Hamiltonian paths through points in Euclidian spaces. These preliminaries are used for the analysis of a sequence correction algorithm that is described and analyzed in Section 3. The algorithm of Section 3 is based on partitioning the existing sequence into "good" segments, which are then re-connected to each other in a new order. Section 4 describes various methods for performing this partition. A conclusion is given in Section 5.
2. PRELIMINARIES: SHORTEST AND EXPECTED PATH LENGTHS

In this section we describe some results concerning (shortest and expected) lengths of Hamiltonian paths through points in \( d \)-dimensional Euclidean spaces. As stated in the introduction, the considerable difference between shortest and expected path lengths indicated by these results supports the motivation for updating the token-passing sequence. But mainly, the results are used in the evaluation of the sequence correction algorithm in Section 3.

2.1 Bounds on the Length of Shortest Paths

Let \( S \) be a bounded region of the \( d \) dimensional Euclidean space, whose boundary is of measure zero. Let \( A \) be the (positive) measure of \( S \), and let \( V_n \) be a set of \( n \) points in \( S \). Then there are constants \( \alpha_d \) and \( \beta_d \) for which the following holds for \( l_{\text{min}}(V_n) \), the length of the shortest path through the points of \( V_n \):

\[
\limsup_{n \to \infty} n^{-(d-1)/d} l_{\text{min}}(V_n) \leq \alpha_d A^{1/d}
\]

and with probability one

\[
\lim_{n \to \infty} n^{-(d-1)/d} l_{\text{min}}(V_n) = \beta_d A^{1/d}
\]

The exact values of \( \alpha_d \) and \( \beta_d \) are not known. For \( d = 2,3 \), the following bounds are given in ([1], [5]):

\[
0.75984 \leq \alpha_d \leq 1
\]

\[
0.44194 \leq \beta_2 \leq 0.6508
\]

\[
0.81650 \leq \alpha_3 \leq 1.0911
\]

\[
0.34208 \leq \beta_3 \leq 0.55696
\]

According to [1], empirical results indicate that these bounds hold for all practical values of \( n \).

When restricting our discussion to a circular region with diameter \( D \), the above results imply
that for some constant $c_1$ we have:

$$I_{\text{max}}(V) \leq c_1 D \sqrt{n} \quad (2a)$$

and that with very high probability (approaching 1.0 for large $n$) we have, for some other constant $c_2$:

$$I_{\text{max}}(V) = c_2 D \sqrt{n} \quad (2b)$$

(Note that since $c_2$ is about 60% of $c_1$, the shortest path for most node layouts is about 60% of the one for the worst layout).

There are efficient approximation algorithms for the TSP [3,7,13] which guarantee an output tour that is not longer than a small constant times the length of the shortest tour. The length of a tour produced by such an approximation algorithm is, therefore, bounded by $c \sqrt{n}$, where the constant $c$ depends on the diameter $D$ of the circular region containing the $n$ points, and on the specific algorithm.

2.2 The Expected Length of a Random Path

Let $S$ be a region in the $d$-dimensional Euclidean space, and let $C_n$ be a random closed path through $n$ random points in $S$. The length of $C_n$ is the sum of the distances between $n$ pairs of random points in $S$, hence it is intuitively clear that the expected length of $C_n$ equals $n$ times the expected distance between two random points in $S$ (a formal proof of this fact is omitted).\footnote{Note that the expected distance between random points in $S$ depends on the specific geometry of $S$, while the expected length of the shortest circuit through $n$ random points in $S$ depends only on the area of $S$.}

For example, if $S$ is a circle of diameter $D$, then the expected distance between two random points in $S$ is $\frac{64D}{45\pi} = 0.452837 \cdot D$ (see [12]), and the expected length of a random closed path is thus $0.452837 \cdot D \cdot n$. 
3. A SEQUENCE CORRECTION ALGORITHM

In networks where message durations are small with respect to token propagation times (e.g., high speed networks), scheduling overhead associated with passing the token (or the virtual token) becomes significant. For efficient token passing protocols this overhead is equal to (or just somewhat larger than) the sum of propagation delays along the closed token-passing path [8,9]. This path constitutes a Hamiltonian circuit in the network. Thus, one of the conclusions that may be drawn from the results in the previous section is that, even for the worst node layout, a properly selected token passing sequence can considerably decrease the scheduling overhead. Therefore it pays to update the token-passing sequence whenever this overhead (the round-trip propagation delay, for this matter) exceeds a given threshold. An upper bound on the shortest path, like the one given in the previous section, can be used to compute this threshold.

To construct a reasonably good sequence one can, in general, use one of several existing distributed algorithms that provide a solution or an approximate solution to the Travelling Salesman Problem (referred to in this paper as TSP algorithms). One example is the $O(n \log n)$ algorithm described in [7], that guarantees a path not longer than twice the shortest path.

In general, when no assumptions can be made on the nature of node motion, a new sequence is computed "from scratch", i.e., independently of the current sequence. However, as mentioned in the introduction, circumstances often exist under which groups of nodes are restricted to move in such a way that the relative positions of nodes within groups are more or less preserved (the grouping may be different at different inter-update intervals). Examples for such circumstances are patrol cars that move along streets, airplanes that fly in formations, etc. Under such circumstances a newly computed sequence may contain segments (subsequences) that remained unchanged, i.e., were also subsequences of the previous sequence. The following is a method that takes advantage of this situation to reduce the amount of computation (and communication)
required for the update: the token passing sequence is corrected, rather than computed "from scratch".

The sequence correction algorithm is performed in three steps, as follows:

1) **Partition the present token-passing sequence to form a set of segments** such that the average distance between consecutive nodes within a segment does not exceed a given threshold value, \( \tau \).

2) **Select one representative node from each of the segments, and construct a shortest or near shortest tour through these nodes.** The TSP algorithm used for this purpose will be referred to as \( A_l \).

3) **Construct a new sequence by connecting the segments head to tail in the order of the corresponding representative nodes, as determined by the tour constructed in step (2).**

In the sequel we show that, if a proper value for \( \tau \) is used in the partition, the resulting final path is never longer than the worst case guaranteed by computing the sequence "from scratch" (i.e., by performing \( A_l \) on the individual nodes). This is true regardless of node configuration or type of motion. In the discussion we assume that the partition results in at least two segments. To support this assumption we show later that if the partition results in a single segment, an update was not necessary, since the current sequence is good enough.

**Lemma 1:** let \( l_{\text{good}}(V_n) \) denote the aggregate length of the segments formed by the partition of step (1), and let \( s \) denote the number of these segments (\( 2s \leq n \)). We then have:

\[
l_{\text{good}}(V_n) \leq (n-s)\tau.
\]  

**Proof:** out of a total of \( n \) token-hops (edges) in the closed path, \( s \) hops are between segments and the average length of the remaining \( n-s \) hops is, by definition, no
longer than \( t \). Since the sum of these \( n-s \) hops is equal to \( l_{\text{good}}(V_s) \), the Lemma is proven. \( \square \)

Let \( AI \) denote the algorithm used for constructing the tour in Step 2 of the sequence-correction algorithm, and assume (as justified in Section 2) that the length \( l_{\text{min}}(V_s) \), of a tour computed by \( AI \) through the \( s \) representative nodes satisfies:

\[
l_{\text{min}}(V_s) \leq c \sqrt{n}
\]  

Then we have:

**Lemma 2**: Let \( t_{\text{min}} = \frac{c}{4 \sqrt{n}} \). Then when executing Step 1 with threshold \( t = t_{\text{min}} \), the length \( l_o(V_n) \) of the tour produced by Step 3 is bounded by \( c \sqrt{n} \).

**Proof**: Let the segments formed by the partition be numbered consecutively from 1 to \( s \), according to the order established in step (2) above. Let \( l_p(i) \) be the distance between the representative nodes of segments \( i \) and \( i+1 \) (segment \( s+1 \) refers cyclically to segment 1). Let \( l_{\text{conn}}(i) \) be the distance between the last node in segment \( i \) and the first node in segment \( i+1 \) (the length of the new connecting hop), and let \( l_{\text{good}}(i) \) be the length of the \( i^{th} \) segment. By the Triangle Inequality and summing over \( i \) from 1 to \( s \) we can show that:

\[
\sum_{i=1}^{s} l_{\text{conn}}(i) \leq \sum_{i=1}^{s} l_{\text{good}}(i) + \sum_{i=1}^{s} l_p(i) := l_{\text{good}}(V_n) + l_{\text{min}}(V_s). \tag{5}
\]

The length of the new token passing path (result of step 3) is thus given by:

\[
l_o(V_n) = l_{\text{good}}(V_n) + \sum_{i=1}^{s} l_{\text{conn}}(i) \leq 2l_{\text{good}}(V_n) + l_{\text{min}}(V_s). \tag{6}
\]

and by substituting from Lemma 1 and Equation (4) we get:

\[
l_o(V_n) \leq 2(n-s)t + c \sqrt{n}. \tag{7}
\]

The right hand side of (7) provides an upper bound on \( l_o(V_n) \), expressed as a function...
of \( s \) and \( t \). By taking the derivative with respect to \( s \) and equating to zero we get the value of \( s \) for which this bound is maximized, at a given value of \( t \):

\[
s_{\text{max}} = \frac{c^2}{16t^2}
\]  

(8)

By substituting \( s_{\text{max}} \) for \( s \) in (7) we get \( l_s(V_n)_{\text{max}} \), an upper bound on \( l_s(V_n) \), as a function of \( t \):

\[
l_s(V_n)_{\text{max}} = 2(n - \frac{c^2}{16t^2}) + \frac{c^2}{4t} = 2nt + \frac{c^2}{8t}
\]  

(9)

Taking the derivative with respect to \( t \) of the right hand side of (9), and equating to zero we get the value of \( t \) for which \( l_s(V_n)_{\text{max}} \) is minimized:

\[
t_{\text{min}} = \frac{c}{4n^{\frac{1}{3}}},
\]  

(10)

By selecting the above value for \( t \) we can guarantee a worst case path length of:

\[
l_s(V_n)_{\text{max}} = c\sqrt{n}
\]  

(11)

**Theorem 1**: if the threshold value \( t \) is selected to be \( t_{\text{max}} \) as determined by Equation 10, then the length of the path produced by the correction procedure never exceeds the worst-case output of the "from scratch" procedure.

**Proof**: the upper bound on the length of the path resulting from executing \( AI \) on the individual nodes (the "from scratch" procedure) is equal to \( c\sqrt{n} \). When the appropriate threshold value is selected then, by Lemma 2, \( c\sqrt{n} \) is also equal to the upper bound on \( l_s(V_n) \). □

In fact, if we select \( t = t_{\text{min}} \) then, by Equations (8) and (10), the right hand side of Equation (7)
is maximized (equal to $l_s(V_s)_{\text{max}}$) when $s=n$, i.e., when step (1) partitions the sequence into individual nodes. If a partition results in $s<n$ segments then a smaller or equal bound is guaranteed. This does not, however, mean that $l_s(V_s)$ will then be shorter than the result of performing $At$ on the individual nodes: configurations that tend to produce a small number of segments will also tend to have a short TSP tour.

A partition is likely to produce fewer segments when node motion belongs to the class described earlier. In practice one may select a threshold value that differs from the one determined by Equation 10: If a larger value is selected then $s$ will tend to be smaller, resulting in reduced update cost, possibly at the expense of a larger $l_s(V_s)$. For a given network, experiments that determine the average token passing time and the average update cost should be conducted to select the optimum value for $t$.

To prevent the system from continuously updating the token passing sequence, the correction procedure should only be executed when an improvement is guaranteed, namely, when the network's current token-passing pathlength, $l(V_s)$, exceeds the right hand side of Equation (9), namely when:

$$l(V_s) > 2nt + \frac{e^2}{8t} \quad (12)$$

For example, when $t_{\text{min}}$ is selected as the value for $t$ then the update will be performed only when

$$l(V_s) > 4\cdot n \cdot t_{\text{min}}.$$  

**Theorem 2:** if an update is performed only when the current pathlength exceeds the right hand side of Equation (12), then the partition (step 1) will result in at least two segments.

**Proof:** if, as a result of step 1, $s=1$ then by Lemma 1 we have that $l_{\text{short}}(V_s) \leq (n-1)t$. 

By the Triangle Inequality we have that the length of the remaining edge in the closed path is also less than or equal to \((n-1)\mu\). Thus \(l(V_n) \leq 2(n-1)\mu < 2n\mu\), which contradicts the condition of Equation (12), i.e., an update could not have been initiated. \(\square\)

4: PARTITIONING THE SEQUENCE

Partitioning the current token passing sequence into "good" segments, i.e., step (1) of the sequence correction algorithm of Section 3, is based on the average distance (or, equivalently, average token passing time) between consecutive nodes in a segment.

Consider a token passing sub-sequence that starts with node\(i\) and ends with node\(m\). Recall that for convenience and without loss of generality we assume that node indexes are assigned according to the token-passing sequence, and that index \(n+1\) is really index 1. Let \(R(i,m)\) be the sum of internode distances (or token passing times) along consecutive nodes in the above sub-sequence:

\[
R(i,m) = \sum_{j=i}^{m-1} d(j, j+1),
\]

and let \(\bar{R}(i,m)\) be the average distance between consecutive nodes in that sub-sequence:

\[
\bar{R}(i,m) = \frac{R(i,m)}{(m-i) \mod n}.
\]

Node\(1\) is the start node of the first segment. The end node of this segment is the one with the largest index, \(k\), for which \(\bar{R}(1,k)\) is less than or equal to some threshold, \(t\). \(k+1\) is the index of the next segment's start node. In general, if node\(i\) is the start node of some segment, then node\(m\) is the end node of that segment if \(m\) is the largest index for which \(\bar{R}(i,m) \leq t\). To find \(m\) the
partition algorithm checks node indexes from $n$ down to $i$.

As stated in the introduction, it is assumed that each node has explicit knowledge of all distances of type $d(j,j+1)$. Therefore all $\tilde{R}(i,m)$ can be computed independently by each and every node. It is also assumed that all nodes know the size of the region in which they are allowed to move. Since this is a fixed (though rather loose) bound on $D$, all nodes can compute the same threshold $t$ independently. Should a tighter bound on $D$ be used, a round of messages must precede step (1) of the correction procedure: in this round each node broadcasts the maximal distance from itself to any other node. The largest of these maxima is equal to $D_0$, the network diameter at measuring time. A rather tight bound on $D$ at updating time can be computed from $D_0$ and the bound on topological change due to motion that is given in [10]. All nodes can then simultaneously execute the same partition algorithm, given in Figure 1.

To reduce the amount of internal computation, at the expense of additional communication, we can have a single node execute the algorithm and then broadcast the result to the rest of the network. Internal computation is reduced by the factor of $n$, but additional communication in the form of a message that contains the identities of all $s$ segments (i.e., $s$ indexes of the corresponding start or end nodes) is required.

Further, the computation load can be distributed among the $s$ new start nodes: node 1 will identify the start node of the second segment, and broadcast its index. The latter node will take over computation and identify the next start node, broadcast its index, and so on.

---

2This information is obtained as a by-product (and thus with no additional cost) of the topology measurement algorithm described in [7]. Recall that performing such a measurement is a prerequisite for updating the token passing sequence. The protocol in [7] requires $2n$ short messages.
procedure partition (t: real; {threshold}
  n: integer; {number of nodes}
  d: array[1..n] of real; {d(i,j+1)}
var s: integer; {resulting number of segments}
var head: array[1..n] of integer; {segment heads}
);

var
  last : integer; {candidate for last-node in current segment}
  sn : integer; {segment number}
  ncum : integer; {cumulative # of nodes in current segment}
  rcum : real; {cumulative length of current segment}
  rprv : real; {cumulative length of previous segs. + connect}
  rest : real; {length of the rest of tour}
  ravg : real; {average edge length in current segment}

begin {partition tour into good segments}
  head[1] := 1;
  rprv := 0.0;
  sn := 0;
  s := 0;
  repeat
    sn := sn + 1;
    last := n;
    rrest := rtotal - rprv;
    rcum := rrest + d[n];
    ncum := n - head[sn];
    ravg := rcum / ncum;
    while (ravg > t) and (ncum > 1) do
      begin
        last := last - 1;
        rcum := rcum - d[last];
        ncum := ncum - 1;
        ravg := rcum / ncum;
      end;
    if ravg > t then
      begin
        head[sn+1] := head[sn] + 1;
        rprv := rprv + d[last-1]
      end
    else
      begin
        head[sn+1] := last + 1;
        rprv := rprv + rcum + d[last];
      end;
    s := s + 1;
    until (head[sn+1] > n) or (sn = n-1);
  if head[sn+1] = n then s := s + 1;
end; {partition}

Figure 1 - The Partitioning Algorithm
5. CONCLUSION

We described and analyzed a distributed approximation algorithm for the Travelling Salesman Problem, that corrects an existing tour, rather than computing one "from scratch". Thus the correction algorithm is suitable for dynamic graphs with slowly changing edge weights, and for which a Travelling Salesman tour (optimal or approximate) was previously computed and is slowly "deteriorating" due to the changing edge weights. The algorithm can be used to "refresh" the tour whenever it deteriorates beyond a given level, and thus maintain a reasonable average tour length at relatively low computation and communication costs.

The algorithm is adaptive in the sense that it shifts gradually between performing a slight correction ($s \ll n$) and recomputing the tour "from scratch" ($s = n$), depending on the type of changes that have occurred in the graph since the previous tour was computed. The smaller the correction, the lower the cost.

The algorithm was designed for the practical application of maintaining a short token-passing path, which means low scheduling overhead - an important factor in the performance of high-speed or long distance packet radio networks.

For a Euclidean graph with $n$ nodes laid out in a bounded area with diameter $D$, the maximal length of the tour produced by the algorithm is proportional to $D \sqrt{n}$, like the maximal length of an optimal tour in the same graph (the two differ by a factor of 2 at the worst case). As a result, one can guarantee a reasonable level of performance in the network at a cost that is as low as, and under many practical circumstances lower than that of other TSP approximation algorithms. Moreover, since the expected length of an optimal tour is not much smaller than its worst case, further drastic improvement (i.e., by an order of magnitude) in average performance cannot be achieved.
REFERENCES


