May 1985 Report No. STAN-CS-85-1050

Fast Sequential Algorithms to Find Shuffle-Minimizing and Shortest Paths in a Shuffle-Exchange Network

by

John Hershberger
Ernst Mayr

Department of Computer Science

Stanford University
Stanford, CA 94305

FAST SEQUENTIAL ALGORITHMS TO FIND SHUFFLE-MINIMIZING AND SHORTEST PATHS IN A SHUFFLE-EXCHANGE NETWORK

John Hershberger†
Ernst Mayr†
Department of Computer Science
Stanford University
Stanford, California 94305

Abstract

This paper analyzes the problem of finding shortest paths and shuffle-minimizing paths in an n-node shuffle-exchange network, where $n = 2^m$. Such paths have the properties needed by the Valiant-Brebner permutation routing algorithm, unlike the trivial $(m-1)$-shuffle paths usually used for shuffle-exchange routing. The Valiant-Brebner algorithm requires n simultaneous route computations, one for each packet to be routed, which can be done in parallel. We give fast sequential algorithms for both problems we consider. Restricting the shortest path problem to allow only paths that use fewer than m shuffles provides intuition applicable to the general problem. Linear-time pattern matching techniques solve part of the restricted problem; as a consequence, a path using fewest shuffles can be found in $O(m)$ time, which is optimal up to a constant factor. The shortest path problem is equivalent to the problem of finding the Hamming distances between a bitstring and all shifted instances of another. An application of the fast Fourier transform solves this problem and the shortest path problem in $O(m \log m)$ time.

† This work was supported in part by an IBM Faculty Development Award, a National Science Foundation Fellowship, Army Research Office Fellowship USARO DAAG29-83-G0020, National Science Foundation Grants MCS 83-00984 and DCR-8351757, and a grant from AT&T Bell Laboratories.

I. Introduction

Algorithms for parallel computation invariably achieve their speedup by partitioning the original problem, solving the resultant subproblems concurrently, and combining the results to construct a complete solution. In many cases, almost no computation can bc done on an isolated datum (as in the case of sorting), so communication between processing elements dominates the algorithm. It follows that parallel algorithms depend strongly on the model of computation and communication assumed by the algorithm designer. Many different designs have been proposed to allow multiple processors with fixed interconnections to cooperate efficiently. One especially popular theme involves connecting processors to memory banks or to each other through a butterfly-style network [10] [14] [16]. The variant that is perhaps least complicated for its power is the shuffle-exchange network [18].

In the shuffle-exchange design, n = 2" processing elements are connected in a simple but powerful network. Each processing element or node has two data paths or edges along which it can send data packets to other nodes. Let the nodes be numbered 0, 1,. . . ,$n - 1$, and let the binary representation of an m-bit integer i be $i_{m-1}i_{m-2}\cdots i_0 = \sum_{0\leq j<m} i_j 2^j$. Then the *shuffle* edge out of node i sends packets to the node whose address in binary is $i_{m-2}i_{m-3}\ldots i_1 i_0 i_{m-1}$. This is node $2i$ mod $(n - 1)$ when i is not all ones $(i \neq n - 1)$ and $i = n - 1$ when it is. The exchange edge sends packets from node i to node i $1 = $i_{m-1}\ldots i_1\bar{i}_0$. (Here $\oplus$ is the binary bitwise exclusive-OR operator and $\bar{b}$ is the bitwise complement of b.) Interesting generalizations of the shuffle-exchange network include the delta network [14] and the Generalized Shuffle Network [3].

It is possible to route any permutation on the shuffle-exchange network in O(m) time [16], though the routing may take $O(m^4)$ time to determine explicitly [13]. Known permutation routing algorithms require $\Theta(m)$ expected routing time for a randomizing algorithm [19] and $\Theta(m^2)$ time for a deterministic algorithm [18]. Ajtai, Komlós, and Szemerédi have recently developed a sorting network of size $O(nm)$ and depth O(m) [1]. Leighton has extended their ideas to produce an $O(n)$-size network that sorts n numbers in O(m) time [1 I]. Both networks can be used to route permutations in O(m) time. Unfortunately, the constant factors involved in the construction of these networks are such that the networks arc impractical for realistic n.

In the shuffle-exchange network it is possible to route a packet from any node in the network to any other node in fewer than *2m* routing steps, since $m - 1$ shuffles separated by exchange operations when necessary will send a packet from node i to any address. Only $O($ 1) preliminary computation is needed to prepare a packet for shipping along such a path, but the routing may not be good enough for some algorithmic applications. For example, the analyses of the $O(m)$ expected time permutation and sorting algorithms of Valiant and Brebner [15] and Rcif and Valiant [19] depend on packets being sent via routes that 1) have no cycles and 2) have the *non-repeating* property: edges common to any pair of routes arc contiguous. The easily-computed route may not satisfy either of these requirements. For example, the following 3-shuffle path from 0101 to 1000 has a cycle:

$$0101 \rightarrow_s 1010 \rightarrow_s 0101 \rightarrow_e 0100 \rightarrow_s 1000.$$

The following pair of 4-shuffle paths does not satisfy the second property:

$$01000 \rightarrow_s 10000 \rightarrow_e 10001 \rightarrow_s 00011 \rightarrow_s 00110 \rightarrow_s 01100$$

$$01000 \rightarrow_s 10000 \rightarrow_s 00001 \rightarrow_s 00010 \rightarrow_e 00011 \rightarrow_s 00110.$$

Both shortest paths and shuffle-minimizing ones satisfy the Valiant-Brebner constraints. A shortest path clearly cannot have a cycle. Furthermore, if the route is chosen uniquely from among the shortest paths, for example by choosing the shortest path that uses fewest shuffle steps and performs exchanges at the first opportunity, the second condition will also be satisfied. The route that uses fewest shuffles and no redundant exchanges (at most one between any two shuffles) cannot be longer than twice the length of the shortest path. It also has the two required properties. No cycles are possible in such a path, since the only cycle that would not increase the shuffle count would be one involving only exchanges, and such exchange-only cycles are excluded. In any path that uses $k < m$ shuffles, the placement of the exchanges is completely determined. Thus for any p and q, the path between them that uses fewest shuffles is unique. Any sub-path also uses as few shuffles as possible and hence is also unique. It follows that two paths that converge, run together, and diverge cannot later reconverge.

In the Valiant-Brebner permutation algorithm, n packets are sent through the network, and each must have a route computed for it. Since the n routes are independent, it is most efficient to compute one route at each of the processing nodes of the network. In particular, it. is impractical to find shortest paths by using breadth-first search in the network for each of the n routes. Therefore we consider sequential algorithms to compute paths in a shuffle-exchange network.

In Section III we present an O(m) algorithm for finding shuffle-minimizing paths, which are adequate for permutation routing. Nonetheless, the shortest path problem is still interesting in its own right. The most obvious algorithm for finding such paths takes $\Omega(m^2)$ time; it has the advantage of requiring only $O(m)$ space. We prove in Section V that the shortest path problem is closely related to the problem of computing the Hamming distance between one string and all shifted instances of another. In Section VI we use this fact to compute shortest paths in $O(m \log m)$ time and space.

II. Definition of the Problem

Before formalizing the shortest path problem, let us define some notation and three useful functions. First, let $H(a, b)$ be the Hamming distance between bitstrings a and b. That is, if $\nu(x)$ is the number of ones in the binary representation of x, then

$$H(a, b) = \nu(a \oplus b).$$

Let us define substrings of a bitstring as follows: if $q = q_{m-1}q_{m-2} \dots q_0$, then $q_{i,j}$ is the string $q_i q_{i-1} \dots q_j$ if i $\geq$ j and the empty string otherwise. We indicate concatenation of bitstrings by abutment, that is, $a_{i,j}$ is the same as $a_{i,k}a_{k+1,j}$ for any k such that i $\geq k >$ j. These conventions make it easy to define the function $s(a)$, which cyclically shifts a bitstring a = $a_{m-1,0}$ to the left by one place:

$$s(a_{m-1,0}) = a_{m-2,0}a_{m-1}.$$

This is exactly the mapping from the current address of a packet to its new address as it follows a shuffle edge. The address of the node reached by a packet starting from a and following shuffle edges i times is $s^i(a)$. That is,

$$s^i(a_{m-1,0}) = a_{m-i-1,0}a_{m-1,m-i}.$$

The function $e(u)$ corresponds to sending a packet along an exchange edge. It toggles the bottom bit of its argument:

$$e(a_{m-1,1}a_0) = a_{m-1,1}\bar{a}_0.$$

Note that $e^i(a) = e^{i \bmod 2}(a)$.

Now the Shuffle-Exchange Shortest Path problem, or SESP, can be formulated as follows:

Let the starting address of a packet be $p = p_{m-1,0}$ (in binary), and let the destination address bc $q = q_{m-1,0}$. For $0 \leq k < m$, define

$$d(k) = \begin{cases} 0, & \text{if } q_{m-1,k+1} = p_{m-k-1,1}, \\ m, & \text{otherwise.} \end{cases}$$

Thus $k + cl(k)$ is the number of shuffles required in a path from p to q if we only consider paths that use $k + rm$ shuffles. If only k shuffles arc used, the $m - k - 1$ bits in $p_{m-k-1,1}$ cannot be altered by an exchange step. The function $d(k)$ is zero when these bits are already in agreement with the corresponding bits of q. If some of them must be modified, m extra shuffles are needed and $d(k) = m$. The SESP problem is that of finding a k such that the quantity

$$k + d(k) + H(q, s^k(p))$$

is minimized. If k' is the minimizing k, a shortest path has $k' + d(k')$ shuffles and as many exchanges as are necessary to transform p_i into $q_{(i+k') \bmod m}$ for all $0 \leq i < m$. One can think of the path as a stepwise transformation of p into q: it cyclically shifts p to the left by $k' + d(k')$ places, changing each bit that must be changed when it first arrives in the low position of the shifting register.

Note that once a shortest path has been computed, the routing information that must be sent with a packet is minimal. All that a packet needs to ensure correct routing is a ticket (following the terminology of Valiant and Brebner [19]) carrying a $(\lceil \lg m \rceil + 1)$-bit integer t and the destination address q. The integer t initially holds the number of shuffles in a shortest path. Packet-handling during routing requires almost no computation: when a packet arrives at a node i with ticket (t, q), it leaves via the exchange edge if the bottom bit of the current address needs to be toggled; otherwise, if $t \neq 0$, it leaves via the shuffle edge with ticket $(t - 1, q)$. If $t = 0$ and $q_0 = i_0$, then $q = i$ and the routing is complete. This packet-handling algorithm is formalized in Figure 1.

```
co Packet arrives at node $i$ carrying destination
$q$ and current shuffle-distance $t$. oc

if $i_0 \neq q_{t \bmod m}$ then
   Send packet along exchange edge
else if $t \neq 0$ then
   begin
      $t := t - 1$;
      Send packet along shuffle edge;
   end
else
   Routing is complete.
```

Figure 1. Packet-handling Algorithm

Surprisingly enough, shortest paths can have more than $m - 1$ shuffles. (This contradicts an implicit assumption of [19].) For example, the shortest path connecting the eight-bit source-destination pair

$$p = 00101100$$

$$q = 11100001$$

has eleven shuffles. Small cases suggest the conjecture that every source-destination pair has a shortest path that uses fewer than $3m/2$ shuffles, but we have not yet been able to prove this.

Shortest paths are not necessarily unique. Banyan networks [7] like the Omega network [10] have exactly one shortest path between a source-destination pair. The m-cube network (see, for example, [17]) allows multiple shortest paths, but its nodes have unbounded degree. Even with a node out-degree of only two, the shuffle-exchange network allows multiple shortest paths: an example of a source-destination pair connected by three different shortest paths is

$$p = 11100010$$

$$q = 00001110.$$

Shortest paths from p to q require a total of thirteen routing steps and any of eight, ten, or twelve shuffle steps. Of course, uniqueness can be enforced by selecting the shortest path with fewest shuffles that performs exchanges at the first opportunity.

The problem of finding a shortest path among those that use fewer than m shuffles has more structure than the general problem. This restricted problem is called BSSP, an abbreviation for Bounded-Shuffle Shortest Path. The restriction to using fewer than m shuffle steps implies that only k's for which $d(k)$ is zero need be considered. It also reduces the number of Hamming distances that must be computed. The obvious algorithm to find an optimal k for either problem just computes $H(q, s^i(p))$ in $\Omega(m)$ time for each value of i; $d(k)$ is a by-product of the calculation. This algorithm runs in $O(m^2)$ time, but one can do much better, as we show in Section VI.

The restricted problem splits naturally into two parts. The first is determining a set S of k's for which $d(k) = 0$. The second is minimizing $k + H(q_{k,0}, p_0 p_{m-1,m-k})$ over S. The first part is nothing more than pattern matching: the set S contains the shifts that correspond to exact matches between a prefix of the destination q and a suffix of $p_{m-1,1}$, the left $m - 1$ bits of the source string. A modification of the Knuth-Morris-Pratt linear pattern matching algorithm [9] can be used to determine S in O(m) time. This is described in the following section.

If $|S| > \lg m$ (the base 2 logarithm of m), there are repeated patterns in the source and destination strings. In the BSSP case, these can be used to reduce the overall complexity of determining Hamming distances for bits in the repeats, but they are no help for bits in regions without repeating structure. Since the algorithm of Section VI achieves $O(m \log m)$ whether or not repeats are present, the analysis of repeats is merely an interesting exercise and has been omitted. (Readers interested in such analyses in a more useful setting are referred to the work of Galil and Seiferas on space-optimal pattern matching [6].)

The computation of Hamming distances is the hard part of finding shortest paths in a shuffle-exchange network. The reduction presented in Section V shows that solving BSSP is no easier than finding the minimum Hamming distance between a bitstring a and all cyclic shiftings of a bitstring b, where the length of the bitstrings is a fixed fraction of m. Section VI shows how to find the Hamming distances $H(a, s^i(b))$ for $0 \leq i < m$ with the fast Fourier transform. This gives an O(m log m) bound on the time needed for Hamming distance computation. Combining this with the next section's $O(m)$ method of computing the feasible set S, we get an $O(m \log m)$ algorithm for both SESP and BSSP.

III. Computation of the Feasible Set

Consider the problem of finding the set S of feasible k's. Given p and q, S is the set of k's such that the $m - k - 1$ bits of p that cannot be changed using k shuffles are already correct, that is, $S = \{k \mid q_{m-1,k+1} = p_{m-k-1,1}\}$. If $0 \in S$, the match depends on $\Omega(m)$ bits, and hence finding S requires $\Omega(m)$ time in the worst case. The set S can be found in $O(m)$ time using the modification of the Knuth-Morris-Pratt [9] linear pattern matching algorithm that appears below.

The KMP algorithm searches for a match between a pattern string and a substring of a text string. One can visualize its operation as sliding the pattern string along the text string looking for a match. The algorithm maintains pointers into both strings at the current point of comparison. It compares the pattern and the text string character by character,, moving from left to right. When it encounters a difference (the match fails), it slides the pattern to the right while keeping the text string pointer fixed. At the time of the mismatch, the pattern and text are identical to the left of the text pointer. The pattern slides right until this is true again. For cxample, in Figure 2, the pattern and text match in their first six characters, but disagree at the seventh. The pattern slides right until it matches left of the text pointer again. In this case it slides two places to the right, leaving the first four characters of the pattern matched against the text string. At each step the text left of the text pointer matches the pattern, and hence the KMP algorithm can precompute the shift function from the pattern string alone.

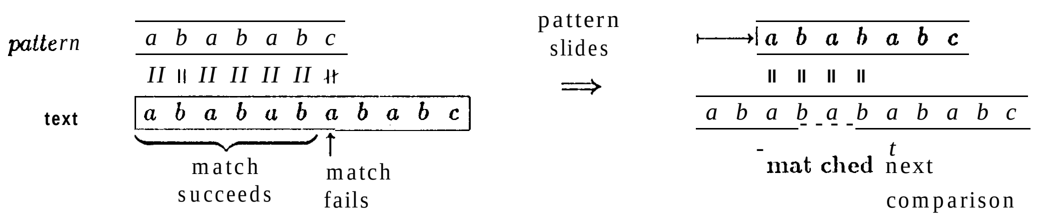


Figure 2. KMP Example

The time taken by the algorithm's matching phase depends only on the lengths of the strings involved. Each time two characters are compared either 1) the characters match and the current position pointers move right in both strings or 2) the match fails and the pattern slides right. Since the sliding pattern and the text string pointer can both move right only to the end of the text string, the pattern matching phase of the KMP algorithm takes time proportional to the length of the text string.

Let us define the shift function $sh(j)$ to be the smallest positive s for which the first $j - s$ bits of q are the same as the $j - s$ bits that begin at the $(s + 1)^{th}$ position (counting from the left) of q:

$$sh(j) = \min\{s > 0 \mid q_{m-1-s,m-j} = q_{m-1,m-j+s}\}.$$

If $s \geq j$, then $q_{m-1-s,m-j}$ and $q_{m-1,m-j+s}$ are both empty and therefore equal. Thus $0 < sh(j) \leq j$. When we are searching for a match between text and pattern, $sh(j)$ is the amount we shift the pattern string if the text and pattern agree in the first j positions but disagree at the (j + 1)th. As implemented in Figure 3, the KMP algorithm will find the longest match between an initial substring of the pattern q and a suffix of $p_{m-1,1}$. Since q is longer than $p_{m-1,1}$, no complete match is possible, but the algorithm will find the longest prefix/suffix match at the end of its main loop.

The shift sh is produced by matching the pattern string q against itself. A matching loop much like the pattern/text comparison loop of the main algorithm generates the values of $sh(j)$ in ascending order. For each j from 1 to $m - 1$ it uses earlier values of sh to find the longest non-identity match between a prefix of q and a suffix of $q_{m-1,m-j}$. The difference between j and the length of this match is $sh(j)$.

The function sh is no larger than $next$ of the KMP algorithm [9], which is defined minimizing over a smaller set:

$$next(j) = \min \{s > 0 \mid q_{m-1-s,m-j} = q_{m-1,m-j+s} \text{ and } q_{m-j-1} \neq q_{m-j+s-1}\} .$$

The extra information contained in $next$ speeds up ordinary pattern matching, but is not appropriate for finding the set S of feasible k's. After the main KMP loop finds the longest prefix of q that is a suffix of $p_{m-1,1}$, repeated shifting with sh finds all prefix/suffix matches of q and $p_{m-1,1}$. The condition on the $next$ function that the characters to the right of the matching regions differ is irrelevant for prefix/suffix matching and could cause some matches to be missed.

The algorithm presented in Figure 3 generates S in three phases, namely **1)** producing sh, 2) finding the smallest element of S, and 3) producing the rest of S. Each phase uses O(m) time. The first loop is executed $m - 1$ times, once for each value of j between 1 and $m - 1$. Its inner loop increases s each time it is executed. Since s never decreases and is bounded from above by $m - 2$, the inner loop cannot require more than $O(m)$ time altogether. The second loop is executed once for each value of i between 0 and $m - 2$. Because j is bounded from below by 0, is increased by 1 each trip through the outer loop, and is decreased by at least 1 each pass through the inner loop, that inner loop can be executed no more than $m - 1$ times. At the end of the second loop, j is no larger than i, which is equal to $m - 1$. Since each trip through the third loop begins with j non-negative and decreases it by at least 1, the loop is executed at most m times.

As we have noted above, the smallest k in S specifies a path from p to q that satisfies the requirements of the Valiant-Brebner permutation algorithm. If such a path is all that we require, we can compute it in $O(m)$ time using just the first two phases of the algorithm in Figure 3.

```
co Produce sh, working from small j's to large. This is a minor modification of
the program used in the KMP algorithm to generate next. (Note that and_cond is
"conditional and;" it only evaluates the second expression if the first one evaluates
true.) oc

$sh(0) := 1$;     $s := 1$;     $j := 1$;
while $j < m$ do
  begin
    while $s < j$ and$_{cond}$ $q_{m-j} \neq q_{m-j+s}$ do
      $s := s + sh(j - s - 1)$;
    $sh(j) := s$;
    $j := j + 1$;
  end;

co Find the longest match between p and q. This is the basic KMP algorithm. In
the following loop, i is the text pointer and j is the pattern pointer. oc

$j := 0$;     $i := 0$;
while $i < m - 1$ do
  begin
    while $j \geq 0$ and$_{cond}$ $p_{m-i-1} \neq q_{m-j-1}$ do
      $j := j - sh(j)$;
    $i := i + 1$;     $j := j + 1$;
  end;
co Now $p_{j,1} = q_{m-1,m-j}$. oc

co Produce S, the set of all distances we can shift q right and have it match up
with the right end of $p_{m-1,1}$. oc

$s := 0$;
while $j \geq 0$ do
  begin
    $S := S \cup \{m - 1 - j\}$;
    $j := j - sh(j)$;
    co If we were to use next instead of sh, we
    might miss some feasible k. oc
  end;
```

Figure 3. Algorithm to find S

IV. Hamming Distance Computation

This section defines and discusses several different Hamming distance problems. Let us define the All Hamming Distances problem AHD as follows:

> Given two m-bit binary numbers a and 6, find $H(a, s^i(b))$ for all i such that $0 \leq i < m$.

An example of AHD appears in Figure 4. We define the minimum AHD problem MAHD to be that of finding a shift index i that minimizes $H(a, s^i(b))$. This may be easier than AHD because it is not necessary to produce all the Hamming distances in AHD's solution. It is certainly asymptotically no harder than AHD, since one can find the solution to MAHD from that of AHD with $O(m)$ more work. For m-bit numbers a and 6, $H(a, 6)$ clearly takes $\Omega(m)$ sequential time to compute, since it depends on every bit position in both a and

6. However, it is by no means obvious that solving AI-ID will take $\Omega(m^2)$ time. (In fact, Section VI provides an O(m logm) upper bound.) If we allowed ourselves to use table lookup with a table $m2^{2m}$ long and lg m bits wide, we could solve AIID in O(m) time by looking at $2m$ bits to index into the table, then reading off the m Hamming distance values from the table. This approach is clearly impractical, but it demonstrates that the best decision tree lower bound wc can get is only n(m).

a	i	$s^i(b)$	$H(a, s^i(b))$
11100001	0	11100010	2
	1	11000101	2
	2	10001011	4
	3	00010111	6
	4	00101110	6
	5	01011100	6
	6	10111000	4
	7	01110001	2

Figure 4. AHD Example

Taking another tack, note that if each of the $H(a, s^i(b))$ were independent of all the others, there would be about $m^m = 2^{m \lg m} = n^{\lg \lg n}$ possible solutions to AHD. But there arc only $2^{2m} = n^2$ possible inputs to AHD, so not all the $H(a, s^i(b))$ are independent. In fact, independence is even further restricted: many different inputs may map to a single m-tuple solution to AHD. For example, complementing both inputs leaves the result unchanged, i.e., $H(a, s^i(b)) = H(\bar{a}, s^i(\bar{b}))$. A more extreme case occurs when all m components of AHD's solution have the same value t. Then if $a = 0^m$, 6 may be chosen from the $\binom{m}{t}$ bitstrings with t one-bits. Including complements and oxchanges, at least $4\binom{m}{t}$ inputs map to the same m-tiiplc. Applying Stirling's formula when $t = m/2$ shows that there is a set of inputs to AHD of size $\Omega(\binom{m}{m/2}) = \Omega(2^m/\sqrt{m}) = \Omega(n/\sqrt{m})$ that have the same solution. The result of AHD says very little about the values of a and 6.

Though the solution of AHD has m lg m bits, fewer than m^2 solution vectors are possible. This suggests that knowing some elements of the m-tuplc solution to AHD, possibly $O(m/$ lg $m)$ of them, might be enough to completely specify the rest. While such relations may be hard to find, there arc functions that use a few solution elements to partially characterize the rest. Perhaps the simplest such function is parity: for a given a and 6, all the answers to AHD will be either even or odd, i.e., $H(a, s^i(b)) \equiv H(a,b) \pmod 2$ for any i. (This is not hard to see: the parity function is $\nu(a \oplus 6) \bmod 2$, and $\nu(a \oplus 6) \equiv \nu(a) + \nu(b) \pmod 2$, since each l-bit of a $\oplus$ 6 corresponds to a single l-bit in a or 6, and each O-bit of a $\oplus$ 6 corresponds to zero or two l-bits in a and 6. Thus $H(a, 6) = \nu(a \$6) \equiv \nu(a) + v(6) = \nu(a) + \nu(s^i(b)) \equiv H(a, s^i(b)) \pmod 2$.)

V. Relation of AHD to Shortest Path Problems

This section presents a simple linear reduction from MAHD to the BSSP problem, demonstrating that BSSP is asymptotically no easier than MAHD. Let a and b be the two m-bit inputs to MAHD. Consider the following source and destination strings of a $6m$-bit instance of BSSP:

$$p = 0^{m-1}b0^m1^m0^m1^m0$$
$$q = 1^m aa1^m0^m1^m.$$

The only prefix/suffix match lengths are 0 through m, i.e., $S = \{5m - 1 + i \mid 0 \leq i \leq m\}$. If we shift p left by $5m - 1$, we get the string $\hat{p} = 1^m0^mb0^m1^m0^m$. The strings p, q, and $\hat{p}$ appear schematically in Figures 5(a) and (b). The Hamming distance $H(q, s^{5m-1}(p))$ is

$$H(1^m, 1^m) + H(a, 0^m) + H(a, b) + H(1^m, 0^m) + H(0^m, 1^m) + H(1^m, 0^m) \qquad (1)$$
$$= u(a) + H(a, b) + 3m,$$

where $u(a)$ is the number of 1's in the binary representation of a. (Figure 5(b) shows the match-up between the two strings q and $\hat{p}$.) Each time $\hat{p}$ shifts to the left, the left term of (1) decreases by one and the right three terms increase by one. In the second and third terms, there will always be a copy of a matched against m zeros and a copy matched against a shifted version of b. Figure 5(c) depicts q matched against a shifted version of $\hat{p}$.

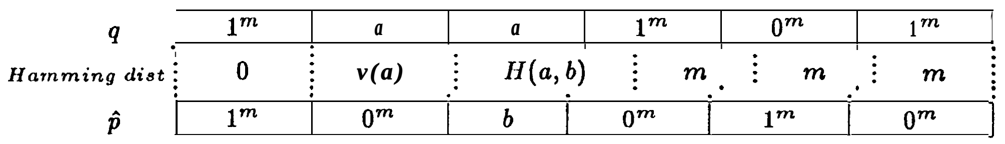
If q remains fixed and $\hat{p}$ shifts to the left i places, the Hamming distance will be

$$\begin{aligned} H(q, s^{5m-1+i}(p)) &= H(q, s^i(\hat{p})) \\ &= H(1^m, 1^{m-i}0^i) + H(aa, 0^{m-i}b0^i) \\ &\qquad + H(1^m, 0^{m-i}1^i) + H(0^m, 1^{m-i}0^i) + H(1^m, 0^{m-i}1^i) \\ &= i + H(a, 0^m) + H(s^{m-i}(a), b) + 3(m - i) \\ &= u(a) + H(a, s^i(b)) + 3m - 2i. \end{aligned}$$

Since the number of shuffles in the path being tested is $5m - 1 + i$, the total path length is $H(a, s^i(b)) + v(a) + 8m - 1 - i$. This is not quite what we want, since the non-Hamming distance part of the expression is not constant. Therefore consider the BSSP problem with $12m$-bit p' and q' obtained from p and q by inserting zeros between their bits in the following asymmetrical way: let $p' = 0p_{6m-1}0p_{6m-2} \ldots 0p_10p_0$, and $q' = q_{6m-1}0q_{6m-2}0 \ldots q_10q_00$. This means that the only prefix/suffix match lengths are the even integers from 0 to 2m: $S' = \{10m - 1 + 2i \mid 0 \leq i \leq m\}$. Since in every match the interleaved zeros are matched against each other, they contribute nothing to the Hamming distances. The Hamming distances for the new problem are the same as for the old one:, $H(q', s^{10m-1+2i}(p')) = H(q, s^{5m-1+i}(p))$. Only the shift count changes in figuring the path length, so the total length is

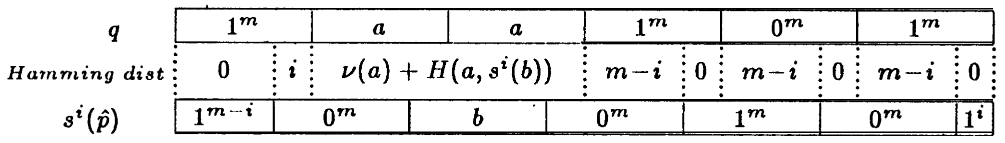
$$10m - 1 + 2i + H(q', s^{10m-1+2i}(p')) = H(a, s^i(b)) + u(a) + 13m - 1.$$

Everything except the Hamming distance is constant, so the shortest BSSP path also gives a solution for MAHD. Hence we can say that BSSP is asymptotically no easier than



$$H(q,\hat{p}) = \nu(a) + 3m + H(a,b)$$

(b)



$$\begin{aligned} H(q,s^i(\hat{p})) &= v(a) + 3(m-i) + i + H(a,s^i(b)) \\ &= v(a) + 3m - 2i + H(a,s^i(b)) \end{aligned}$$

(c)

Figure 5. Reduction of MAHD to BSSP

MAT-ID; similarly, a variation of the bounded-shuffle shortest path problem in which path lengths are produced for every k in S is no easier than AHD.

Conversely, the path-producing BSSP problem is no harder than Al-ID. The set S, generated in $O(m)$ time, can be used in combination with the results of AHD (applied to p and q) to produce a solution to BSSP. It takes only $O(m)$ additional time to minimize $k + H(q, s^k(p))$ over all $k \in$ S. Along the same lines, we can solve the unrestricted SESP problem as quickly as we can solve AI-ID. We simply minimize $k + d(k) + H(q, s^k(p))$ over all $0 \le k < m$, where $d(k) = 0$ if $k \in$ S and m otherwise.

VI. AHD and the Fast Fourier Transform

The restriction of the shortest path problem to shuffle lengths less than m gives an interesting sub-problem to which one can apply the algorithmic techniques of linear pattern matching. The machinery needed to solve the restricted problem also helps solve the more general SESP problem. However, the restriction does not affect the hard part of the problem, which is computing Hamming distances quickly. Thus while pattern matching may improve by a constant factor the running time of an algorithm that solves SESP, it cannot improve its asymptotic performance. The reductions presented above show that we must be able to solve AHD quickly if we are to find fast solutions to the shortest path problem. The fast Fourier transform [5][2] provides an $O(m \log m)$ solution when m is a power of 2 (or a product of small primes).

Before plunging into the application of the Fourier transform to AHD, let us define the operation of convolution, which will be needed later. The convolution of two m-vectors u and v is written $u \otimes v$. The two vectors have subscripts ranging from 0 to $m-1$, and the result of convolving them is a similar m-vector. The i^{th} component of $u \otimes v$ is

$$\sum u_j v_{(i-j) \bmod m}.$$

Convolution is fundamentally related to the multiplication of polynomials.

The All Hamming Distances problem requires finding the values of $H(q, s^i(p))$ for $0 \le i < m$ given bitstrings p and q. The i^{th} such value is

$$H(q, s^i(p)) = \nu(q \oplus s^i(p)) = \sum_{j=0}^{m-1} q_j \oplus p_{(j-i) \bmod m}.$$

This can be expressed using the convolution operator.

There is a close analogy between the exclusive-OR, of bits and multiplication of $+1$ and -1. Define the function f of a bit a as follows:

$$\begin{aligned} f(a) &= \begin{cases} 1 & \text{if } a = 1 \\ -1 & \text{if } a = 0 \end{cases} \\ &= 2a - 1. \end{aligned}$$

Given two bits a and 6, $f(a \oplus 6) = -f(a) \times J(6)$. The inverse function of f is

$$f^{-1}(x) = \frac{x+1}{2}.$$

It should be noted that although replacing 0 bits by -1 and exclusive-OR by multiplication is convenient for the convolution operations that follow, it is not the only possible approach. Multiplication can bc used on the bits and their complements directly without the introduction of -1. That is, the bit identity

$$a \oplus b = (a \times \bar{b}) + (\bar{a} \times b)$$

allows the exclusive-OR operation to be decomposed into two multiplication operations and an addition. One can perform convolutions similar to those described below on each multiplication subproblem and add the results together at the end.

If v is a bit vector, let f(v) be the vector obtained by replacing every bit v_j by $f(v_j)$. Hamming distances are easy to express in terms of the function f. If u and v are bit vectors of length m, then

$$H(u,v) = \nu(u \oplus v) = \sum_j u_j \oplus v_j.$$

If each term $u_j \oplus v_j$ of the summation is replaced by $f(u_j \oplus v_j)$, every 0 term will be replaced by **-1;** the result of the addition will range from -m to m by steps of **2** rather than from *0* to *m* by steps of **1.** A more formal approach to this transformation proceeds by introducing f and its inverse into the summation:

$$\begin{aligned} H(u,v) &= \sum_j (u_j \oplus v_j) \\ &= \sum_j f^{-1}(f(u_j \oplus v_j)), \end{aligned}$$

which by definition is

$$\begin{aligned} &\sum_j \frac{f(u_j \oplus v_j) + 1}{2} \\ &= \frac{1}{2}(m - \sum_j f(u_j) f(v_j)) \\ &= \frac{1}{2}(m - f(u) \cdot f(v)), \end{aligned}$$

where $f(u) \cdot f(v)$ is the vector inner product of *f(u)* and *f(v).*

Now consider the vector p^R obtained from p by reversing the order of p's bits. That is, $p_j^R = p_{m-1-j}$. Let $\tilde{p}$ be the vector produced by applying f to the bits of p^R: $\tilde{p} = f(p^R)$. The convolution of *f(q)* and $\tilde{p}$ is closely related to the AHD problem.

The vector $f(q) \otimes \tilde{p}$ has as its i^{th} term the inner product

$$\begin{aligned} \sum_j f(q_j)(\tilde{p})_{i-j \bmod m} &= \sum_j f(q_j) f(p_{(m-1-i+j) \bmod m}) \\ &= \sum_j f(q_j) f(p_{(j-i-1) \bmod m}) \end{aligned}$$

by the definitions of convolution and $\tilde{p}$. Applying the identity $f(a)\, f(b) = -f(a \oplus b)$ gives

$$-\sum_j f(q_j \oplus p_{(j-i-1) \bmod m}).$$

The subscript of p is offset from j by i + 1. Writing this in terms of the s function leads to an expression involving the Hamming distance:

$$-\sum_j f(q_j \oplus (s^{i+1}(p))_j) = \sum_j \left(1 - 2(q_j \oplus (s^{i+1}(p))_j)\right) \qquad (2)$$
$$= m - 2H(q, s^{i+1}(p)).$$

Thus the convolution of f(q) and $\tilde{p}$ gives a solution to AHD for p and q.

It is well-known (see, for example, [2]) that in any ring that has both a principal mth root of unity and a multiplicative inverse for m, convolution may be performed using the following

Convolution Theorem.

(This theorem can be found in [2], among other places.)

Let $\mathcal{F}(u)$ be the discrete Fourier transform of the m-vector u and let $\mathcal{F}^{-1}(u)$ be the inverse Fourier transform of u. That is, $\mathcal{F}(u)$ is a vector whose ith term is $\sum_j u_j\omega^{ij}$, where w is a principal mth root of unity; the ith term of $\mathcal{F}^{-1}(u)$ is $\sum_j u_j\omega^{-ij}$. These definitions imply that $\mathcal{F}^{-1}(\mathcal{F}(u)) = mu$. Then the term-by-term product of two transformed vectors is the transform of their convolution:

$$\mathcal{F}(u \otimes v) = \mathcal{F}(u) \times \mathcal{F}(v),$$

where the terms of the transformed vectors are multiplied pairwise.

An immediate consequence of this theorem is the fact that

$$u \otimes v = \frac{1}{m}\mathcal{F}^{-1}(\mathcal{F}(u) \times \mathcal{F}(v)).$$

The discrete Fourier transform and its inverse can be performed in $O(m$ log m) time using the fast Fourier transform algorithm of Cooley and Tukey [5] [2], and hence convolution also takes only 0(m log m) time. Since the Hamming distances can take on the $m + 1$ values between 0 and m, the FFT must be performed over a ring with at least those elements. Furthermore, the element 2 must have a multiplicative inverse, since extracting the Hamming distances from f(q) $\otimes$ $\tilde{p}$ requires a division by two. If α is a prime of the form $\alpha = rm + 1$, the required ring may be chosen to be the field Z_α, the integers modulo α. The multiplicative subgroup of this Ecld has size rm and is a cyclic Abelian group. Since it has an element g of order rm, the element g^r is a principal mth root of unity. This element may be used as w to perform the Fourier transform. Because Z_α is a field, every non-zero element has a multiplicative inverse, and in particular, 2 and m have inverses. As shown in (2) above, $H(q, s^{i+1}(p))$ is the ith component of the vector

$$\frac{m - (f(q) \otimes \tilde{p})}{2}.$$

This vector can be expressed using the Fourier transform as

$$\frac{m - \frac{1}{m}\mathcal{F}^{-1}(\mathcal{F}(f(q)) \times \mathcal{F}(\tilde{p}))}{2},$$

where the divisions by 2 and m *are* to be understood as multiplications by the inverses of 2 and m over the field Z_α.

If we could not guarantee the existence of a small prime of the form a = rm + 1, the arithmetic operations in the FFT computation might require an impractical amount of time and space. Fortunately, a theorem due to Linnik [12] states that the first prime in an arithmetic progression $\{rm + a\}$, where a and m are fixed and $\gcd(a, m) = 1$, is less than m^c for sonic constant c. If one assumes the extended Riemann hypothesis, this constant is 2 + ϵ; without ERH, Linnik's "Large Sieve" techniques are necessary to prove the result with a larger value of c. For a proof of Linnik's result, sec [4]. Since gcd(m, 1) = 1, there is always a small prime (bounded by a polynomial in m) of the form α = rm + 1. (Note that α and g^r need be found only once in a preprocessing phase.) Hence the FFT can be done using arithmetic on O(logm)-bit integers. It is probably reasonable to assume that arithmetic operations on such small numbers have unit cost. With this assumption, AHD can be solved in $O(m$ logm) time using O(m $\log m$) random-access memory: If one uses a stricter model and assumes that multiplication of $O(\log$ m)-bit numbers requires $O(\log^2$ m) time, the AHD time bound increases to $O(m \log^3 m)$.

VII. Conclusion

This paper has presented analysis of and solutions to the problems of finding shuffle-minimizing and shortest paths in a shuffle-exchange network. Restricting the latter problem to allow only paths with fewer than m shuffles led to the idea of using pattern matching techniques, in particular a modification of the linear-time Knuth-Morris-Pratt algorithm. This approach solved the first problem, but not the second; it did not address the problem of finding Hamming distances. This subproblem is the hard part of finding shortest paths,. as Section V showed with its demonstration of the rough equivalence of SESP and AHD. Finally, the fast Fourier transform was applied to solve AHD-and hence SESP as well-in $O(m \log m)$ time and space.

The most obvious extension of these results is to the the **d-way shuffle. This is the** base d counterpart of the binary shuffle-exchange network. In such a network, addresses are written in base d, shuffles cycle addresses left (as in the binary case), and exchange operations can change the bottom digit of the address arbitrarily. This paper's ideas apply directly to the d-way shuffle; in such a setting, as well as in the ordinary shuffle-exchange case, the $O(m)$ shuffle-minimizing and $O(m \log m)$ shortest path algorithms presented here provide suitable routes for the sorting and permutation algorithms of [15] and [19].

We have shown that AHD and a path-length producing version of SESP have the same asymptotic time complexity. We believe that AHD requires $O(m \log m)$ time, since it is a weak form of convolution, but our evidence is not a proof. We would like to see either a proof of our conjecture or an $o(m \log m)$ algorithm for AHD.

A second open question involves the definition of optimal routes. This paper has focused on minimizing total path length; in some cases other criteria might be important. For example, in some VLSI layouts of the shuffle-exchange network, shuffle edges are relatively short in comparison with exchange edges [8]. In such cases one might want to find routes that use as few exchanges as possible, even at the expense of greater total path length. Generalizing this criterion, one might look for paths that minimize some linear combination of the number of shuffles and the number of exchanges they contain. The problem of selecting a particular weighting of shuffles and exchanges on the basis of design constraints remains unexplored; however, once such a linear objective function is chosen, the optimal routes it defines may be found quickly using the algorithms of this paper.

Acknowledgements

Thanks to Dave Foulser for suggesting the FFT solution to AHD and to Peter Sarnak and Tom Spencer for their pointers to the work of Linnik and Bombicri. Special thanks to Jane Hershberger for her contributions to this work's readability.

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi, "An O(n logn) sorting network," *Proc. 15th ACM* Symp. on *Theory* of *Computing*, pp. 1-9, 1983.

[2] A. Aho, J. Hopcroft, and J. Ullman, *The* Design *and Analysis* of *Computer* Algorithms, Addison- Wesley, Reading, Mass., 1974, Chap. 7.

[3] L. N. Bhuyan and D. P. Agrawal, "Design and performance of generalized interconnection networks," *IEEE* ***Trans. Comput.,*** vol. C-32, pp. 1081-1090, 1983.

[4] E. Bombieri, "le Grand Crible," *Asterisque*, vol. 18, 1974.

[5] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, April 1965.

[6] Z. Galil and J. Seiferas, "Time-Space-Optimal String Matching," *Proc.* ***13th ACM Symp.*** on *Theory* of Computing, pp. 106–113, 1981.

[7] G. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. 1st Annu.* Symp. *Comput.* Arch., pp. 21–28, 1973.

[8] D. Kleitman, F. T. Leighton, M. Lepley, and G. L. Miller, "An asymptotically optimal layout for the shuffle-exchange graph," MIT, Cambridge, Mass., LCS TM-231, Oct. **1982.**

[9] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast Pattern Matching in Strings," ***SIAM*** *J. Comput.*, 6, pp. 323-350, 1977.

[10] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans.* Comput., vol. C-24, pp. 1145–1155, **1975.**

[11] F. T. Leighton, "Tight bounds on the complexity of parallel sorting," MIT, Cambridge, Mass., LCS preprint, 1984.

[12] Y. V. Linnik, "The large sieve," *Dokl. Akad. Nauk* SSSR, vol. 30, pp. 292-294, 1941.

[13] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Benes permutation network," *IEEE Trans.* ***Comput.,*** vol. C-31, pp. 148–154, Feb. 1982.

[14] J. A. Patel, "Processor-memory interconnections for multiprocessors," in *Proc. 6th Annu.* Symp. Comput. ***Arch.,*** pp. 168–177, April 1979.

[15] J. Reif and L. Valiant, "A logarithmic time sort for linear size networks," *Proc. 15th* ***ACM*** *Symp. on Theory* ***of*** Computing, pp. 10–16, 1983.

[16] J. T. Schwartz, "Ultracomputers," ***ACM*** *TOPLAS*, pp. 484–521, 1980.

[17] H. J. Siegel, "A model of SIMD machines and a comparison of various interconnection networks," *IEEE Trans. Comput.*, vol. C-28, pp. 907–917, 1979.

[18] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE* ***Trans.*** *Comput.*, vol. ***c-20,*** pp. 153-161, Feb. **1971.**

[19] L. Valiant and G. Brebner, "Universal schemes for parallel communication," *Proc. 13th* ***ACM*** *Symp. on Theory* of Computing, pp. 263-277, 1981.