Mutual Search

[Extended Abstract]

Harry Buhrman* Matthew Franklin† Juan A. Garay‡ Jaap-Henk Hoepman§
John Tromp¶ Paul Vitányi‖

Abstract

We define a new type of search problem called "mutual search", where k players arbitrarily spread over n nodes are required to locate each other by sending "Anybody at node i?" query messages (for example processes in a computer network). If the messages are not delivered in the order they were sent (for example when the communication delay time is arbitrary) then two players require at least $n-1$ messages. In an asynchronous network, where the messages are delivered in the order they were sent, $0.88n$ messages suffice. In a synchronous network $0.586n$ messages suffice and $0.536n$ messages are required in the worst case. We exhibit a simple randomized algorithm with expected worst-case cost of $0.5n$ messages, and a deterministic algorithm for $k \geq 2$ players with a cost well below n for all $k = o(\sqrt{n})$. The graph-theoretic framework we formulate for expressing and analyzing algorithms for this problem may be of independent interest.

1 Introduction

Mutual Search (*MS*) is the problem of two or more entities trying to learn each other's location in a uniform unstructured search space. The most obvious application is in distributed computing: k processes residing in a computer network of n computers try to locate each other in as few messages as possible. The computers have distinct identities, say $0, \ldots, n-1$ ($k \leq n$), and the processes execute identical protocols possibly using the identifier of the computer they reside on. Two processes know each others location if at least one process received a message from the other process. The relation "know location" is transitive and the problem is solved if all k processes know each others location. We analyze this problem for the case $k = 2$ under various assumptions on network synchronicity and communication delay and for deterministic and randomized algorithms. We give a result for the general case of $k \leq \sqrt{n}$ processes.

Our Results: We first look at *deterministic algorithms* for two processes. If the messages are *not delivered in the order they were sent (nonFIFO)*, for example in networks with unknown and arbitrary communication delays (as email has to some extent), then two processes need to send at least $n-1$ messages in total in the worst case. Namely, given any protocol construct the directed graph on $\{0, \ldots, n-1\}$ with an arc from i to j if i sends a message to j. For each pair there must be at least one arc. With at least $\binom{n}{2}$ arcs in total, the average number of outgoing arcs is at least $\frac{n-1}{2}$ and it follows easily that some two nodes must together have twice this number, or $n-1$, of outgoing edges (in fact, one could refine this to $2\lceil\frac{n-1}{2}\rceil$ for $n > 2$). An adversarial demon can always schedule the communication delays such that all these $n-1$ message are sent out before contact is made. The tightness of this bound is witnessed by an algorithm called HalfInTurn, to be discussed in Section 3.

In Section 2 we formulate a framework for expressing and analyzing the structure of algorithms for this problem. In Section 9 we show that in *asynchronous* networks, where the messages arrive in the order they were sent, there is a mutual serach algorithm using asymptotically $0.88n$ messages.

For *synchronous* networks we present in Section 6

*Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. E-mail: buhrman@cwi.nl.

†AT&T Research, 600 Mountain Avenue, Murray Hill, New Jersey 07974. franklin@research.att.com.

‡IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598. garay@watson.ibm.com. Work partly done while the author was visiting CWI.

§CWI. jhh@cwi.nl.

¶CWI. tromp@cwi.nl.

‖CWI. paulv@cwi.nl.

the protocol SR_n, an algorithm with a worst-case cost of only $(2-\sqrt{2})n \approx 0.586n$. We also show this algorithm to be close to optimal, by proving a $(4-2\sqrt{3})n \approx 0.536n$ lower bound on the number of queries required by any mutual search algorithm in Section 5.

We consider *randomized algorithms* for the problem in Section 7. A randomized algorithm is shown to have a worst-case (over player location) expected (over random coin flips) cost of $\lceil \frac{n}{2} \rceil$, thus beating the deterministic lower bound.

In Section 8 we present $\mathrm{RS}_{n,k}$, a deterministic algorithm for $k \geq 2$ players for synchronous networks with a cost well below n for all $k = o(\sqrt{n})$. A quick roadmap to the results for $k = 2$:

Synchronous FIFO:

Deterministic worst-case: upper bound $0.586n$ (Section 6) lower bound $0.536n$ (Section 5).

Randomized expected: upper bound $\lceil \frac{n}{2} \rceil$ (Section 7) lower bound $\frac{n-1}{4}$ (Section 10).

Asynchronous FIFO:

Deterministic worst-case: upper bound $0.88n$ (Section 9) lower bound $0.536n$ (Section 5).

Randomized expected: upper bound $\frac{3}{4}n$ (Section 9) lower bound $\frac{n-1}{4}$ (Section 10).

nonFIFO:

Both upper bound and lower bound are $2\lfloor \frac{n-1}{2} \rfloor$ messages in the deterministic worst-case (Section 1) and similarly in the randomized expected case.

The framework we develop for reasoning about the Mutual Search problem may be of independent interest. Mutual search can serve as a preliminary stage to sharing random resources in a distributed setting or forming coalitions for Byzantine attacks and various cryptographic settings.

Related Work: The authors believe that this is a novel type of search problem that has not been considered before. We do not know of any directly related previous research. Several topics that are more or less related can be found in the Appendix A.

2 Model and Definitions: Synchronous Case

Consider an anonymous 2-player synchronous model, in which the querying behaviour of a player depends only on her site. Time is discrete, with time instants numbered $0, 1, \ldots$. Informally, an algorithm for the *MS* problem specifies, for each site that a player can find herself at, what to do at each time instant: either stay idle or (atomically) query some specific other site. For simplicity, we allow at most one query to be scheduled at any single time instant. The lack of simultaneous queries makes the number of queries until (and including) contact a well-defined cost measure.

Note that for any pair of sites, such an algorithm determines which site will first query the other. At that point the algorithm terminates, and the latter site need never query the former. Any algorithm thus implies a *tournament*, which is a directed graph having one (directed) edge between every pair of nodes. An edge from node i to node j represents site i querying site j. An algorithm can then be specified in full by totally ordering all the queries in a tournament, indicating the querying order.

DEFINITION 2.1. *An algorithm for MS is an ordered tournament $T = (V, E, \prec)$, where the set of nodes (sites) is $V = \{0, 1, \ldots, n-1\}$, E is a set of $\binom{n}{2} = \frac{1}{2}n(n-1)$ edges (queries), and $\prec$ is a total order on E. For a node i, E_i is the set of outgoing edges from i, and is called* row i. *The number of queries $|E_i|$ is called the* length *of row i.*

DEFINITION 2.2. *The cost $c(T)$ of an algorithm T is the maximum over all edges $e = (i, j)$ of the edge cost $c(e) = |E_i^{\prec e}| + 1 + |E_j^{\prec e}|$, where for any $F \subseteq E$, $F^{\prec e}$ denotes $\{f \in F : f \prec e\}$.*

The edge cost counts the number of queries made by both players located on the respective incident nodes. Next, we present and analyze some basic algorithms for the problem which will form the basis of a better algorithm.

3 Some Simple Mutual Search Algorithms

The first algorithm, $\mathrm{AllInTurn}_n$, lets each site in turn queries all the other sites. For instance, $\mathrm{AllInTurn}_4$ can be depicted as

$$\begin{array}{lcccccc} 0: & 1 & 2 & 3 & & & \\ 1: & & & & 2 & 3 & \\ 2: & & & & & & 3 \\ 3: & & & & & & \end{array}$$

in which an edge (i, j) is shown on row i of the picture as number j.

LEMMA 3.1. *Algorithm $AllInTurn_n$ has cost $n-1$.*

Proof. It is in fact easy to see that $c(i, j) = j - i$. A player at site i makes this many queries to contact the other player at site j, and the latter never gets to make any queries. The maximum value of $j - i$ is $n-1$.

A somewhat more balanced algorithm is HalfInTurn$_n$, where each site in turn queries the next $\lfloor n/2 \rfloor$ sites (modulo n). HalfInTurn$_5$ looks like

0 :	1	2								
1 :			2	3						
2 :					3	4				
3 :							4	0		
4 :									0	1

For even n, sites $n/2 \dots n-1$ only get to make $\lfloor n/2 \rfloor - 1$ queries.

LEMMA 3.2. *Algorithm HalfInTurn$_n$ has cost $n-1$.*

Proof. Suppose $i < j$. If $j - i \le \lfloor n/2 \rfloor$, we find $c(i,j) = j - i$, otherwise $i - j \bmod n = n + i - j$ giving $c(j,i) = \lfloor n/2 \rfloor + n + i - j$. Taking $j = i + \lfloor n/2 \rfloor + 1$ achieves the maximum of $\lfloor n/2 \rfloor + n - (\lfloor n/2 \rfloor + 1) = n - 1$.

Our next result shows HalfInTurn$_n$ to be the basis of a much better algorithm.

DEFINITION 3.1. *An algorithm is called* saturated *if its cost equals its maximum row length.*

LEMMA 3.3. *An algorithm that is not saturated can be extended with another site without increasing its cost.*

Proof. Let T be an algorithm on n nodes whose cost exceeds all row lengths. Add a new node n, and an edge from every other node to this new node. Order the new edges after the old edges (and arbitrarily amongst each other). This does not affect the cost of the old edges, while the cost of edge (i,n) becomes one more than the length of row i, hence not exceeding the old algorithm cost.

As the proof shows, the maximum row length increases by exactly one, so we may add as many sites as the cost exceeds the former. HalfInTurn$_{2k+1}$ has cost $2k$ and uniform row length k so we may add k more sites to get a saturated algorithm SaturatedHalfInTurn$_{3k+1}$ of the same cost:

COROLLARY 3.1. *Algorithm SaturatedHalfInTurn$_n$ has cost $\lceil \frac{2}{3}(n-1) \rceil$.*

4 Algorithm Refinement

In order to get a better understanding of the structure of *MS* algorithms, we need to focus on their essential properties. In this section we consider algorithms with only a partial edge ordering. The question arises how such a partial ordering can be extended to a good total edge ordering. The following terminology helps us answer this question.

DEFINITION 4.1. *A* partial MS *algorithm is a partially ordered tournament $T = (V, E, \prec, R)$, where $R \subseteq E$ is the subset of* retired *edges, such that $\prec$*

- *totally orders R,*
- *orders all of $E - R$ before all of R, and*
- *leaves $E - R$ unordered.*

An edge $e = (i,j)$ in row prefix *$E_i - R$ has* retiring *cost $c(e) = |E_i - R| + |E_j - R|$. Retiring an edge e results in a more* refined *partial algorithm $T = (V, E, \prec', R')$, where $R' = R \cup \{e\}$ and $\prec' = \prec \cup (E - R', e)$.*

Note that relation $\prec$ is viewed as a set of pairs; $(E - R', e)$ denotes the set $\{(f, e) : f \in E - R'\}$. An example partial tournament, with 2 retired edges, is

$$\{(0,3), (0,1), (1,2), (2,0)\} \prec (2,3) \prec (3,1)$$

Note that any sequence of $|E - R|$ refinements yields a (totally ordered) algorithm, which we call a total refinement of T. A mere tournament corresponds to a partial algorithm with no retired edges.

Observe that the cost of e in a total refinement depends only on its ordering with respect to the edges in rows i and j, which is determined as soon as it retires. This shows the following

FACT 4.1. *If T' results from T by retiring edge $e = (i,j)$, then the retiring cost of e equals the cost of that edge in any total refinement of T'.*

DEFINITION 4.2. *The cost $c(T)$ of a partial algorithm T is the minimal cost among all its total refinements. A total refinement achieving minimum cost is called optimal.*

LEMMA 4.1. *The cost of a partial algorithm T equals the cost of the partial tournament that results from retiring the edge e of minimum retiring cost.*

Informally, any refinement from T will have cost at least equal to the minimum retiring cost, and choosing e doesn't hamper us in any way. The following proof makes precise this notion of non-hampering.

Proof. Consider an optimal total refinement from T to some algorithm T'', in which, at some point, say after $e_1, e_2, \dots, e_k$, edge e is retired. Let algorithm T' be the result of retiring e first, and then continuing the same total refinement with e skipped. Then T'' will have $e \prec'' e_k \prec'' \dots \prec'' e_1$ whereas T' has $e_k \prec' \dots \prec' e_1 \prec' e$. If we compare the costs c''

and c' for any edge in T'' and T' respectively, we see that for $1 \leq i \leq k$, $c'(e_i) \leq c''(e_i)$, $c'(e) \geq c''(e)$, and all other edges cost the same. However, $c'(e) \leq c(e_1)$ by assumption, and so T' must be optimal too.

Seeing that optimal refinement is a straightforward task, we can present algorithms as plain tournaments. By graphically showing the tournament's adjacency matrix, one obtains a visually insightful representation; for instance, SaturatedHalfInTurn$_{13}$ is shown in Figure 1.

Our algorithm HalfInTurn$_n$ now betrays a bad ordering for even n. It retires $(n-1, 0)$ first, at a cost of $n-1$, whereas an optimal refinement can keep the cost down to $n-2$. It takes advantage of the bottom rows being shorter, and first retires an edge between nodes in this bottom half. For example, the following reordering of HalfInTurn$_4$ has cost 2:

$$(0,1) \prec (3,0) \prec (0,2) \prec (1,3) \prec (1,2) \prec (2,3)$$

5 Lower Bounds

Given that the maximum row length is a lower bound on algorithm cost, the following result is easily obtained.

LEMMA 5.1. *Any* MS *algorithm T for n sites has cost at least* $\lceil \frac{n}{2} \rceil$.

Proof. The average outdegree of a node in T is $\binom{n}{2}/n = \frac{n-1}{2}$, so some row has length at least $\lceil \frac{n-1}{2} \rceil = \lfloor \frac{n}{2} \rfloor$. It remains to show that for odd n, an algorithm of cost $\frac{n-1}{2}$ is not possible. This is because for any collection of n rows each of length $\frac{n-1}{2}$, the last edge on every row has retiring cost $\frac{n-1}{2} + \frac{n-1}{2} = n-1$.

The last argument used in the proof shows that the sum length of the shortest two rows is a lower bound on an algorithm's cost. An algorithm of cost c thus necessarily has a row of length at most $c/2$. Careful analysis allows us to prove the following generalization:

LEMMA 5.2. *Let T be an* MS *algorithm for n sites with cost c. Then the $(k+1)$st shortest row of T has length at most $c/2 + k$.*

Proof. Let $e = (i, j)$ be the last edge for which i and j are not among the shortest k rows. Consider the moment of e's retirement in the refinement from the unordered tournament in T to T. Since R includes at most k edges from each of the rows i and j, the retiring cost of e equals $c(e) = |E_i - R| + |E_j - R| \geq |E_i| - k + |E_j| - k \geq 2(\min(|E_i|, |E_j|) - k)$. Furthermore, $c(e) \leq c$, since the cost of T is the maximum of all retirement costs. It follows that the smallest of rows i and j has length at most $c/2 + k$.

This shows that the best possible distribution of row lengths looks like, where the $\binom{n}{2}$ entries are divided over $n - c/2$ rows of maximum length c, followed by $c/2$ increasingly shorter rows, producing a triangular "wasted" space of size about $(c/2)^2/2$.

THEOREM 5.1. *Any* MS *algorithm T for n sites has cost at least* $(4 - 2\sqrt{3})(n-1)$.

Proof. Since each row has length at most c, Lemma 5.2 implies $|E| =$

$$\frac{n(n-1)}{2} \leq nc - \sum_{k=0}^{c/2} c/2 - k = nc - \frac{(c/2)(c/2+1)}{2}$$

$$\Rightarrow (c/2)^2 - 2(n-1)c + (n-1)^2 \leq 1 - 1.5c - n \leq 0.$$

Solving for c, we find $c \geq (4 - 2\sqrt{3})(n-1)$.

6 Algorithm "Smooth Retiring"

In this section we present our best algorithm, building on the insights gained in the prevous sections.

Algorithm SR$_n$ is not quite as easy to describe as our earlier algorithms. It is best described as a partial algorithm with ordered rows, an optimal refinement of which will be presented in its cost analysis.

SR$_n$ divides the nodes into two groups: an upper group $U = \{0, \ldots, u-1\}$ of size u and a lower group $L = \{u, \ldots, n-1\}$ of size $c = n - u$ (which is the cost we're aiming for). As can be expected, construction of SR$_n$ presumes certain conditions on the relative sizes of u and c, which will be derived shortly. The value of c will then be chosen as the smallest which satisfies the conditions.

The upper group engages in HalfInTurn$_u$, while the lower group engages in a slight variation on AllInTurn$_c$ in which each row is reversed.

Row $u+i$ will have length $c - 1 - \lfloor \frac{i}{2} \rfloor$, of which $(u+i, n-1) \ldots (u+i, u+i+1)$ are the last $n - 1 - (u+i) = c - 1 - i$ edges. That leaves $c - 1 - \lfloor \frac{i}{2} \rfloor - (c - 1 - i) = \lceil \frac{i}{2} \rceil$ 'slots' available at the front of row $u+i$, to be filled with edges to U.

Row $i < u$ starts with the $\lceil \frac{u}{2} \rceil$ or $\lfloor \frac{u}{2} \rfloor$ edges in HalfInTurn$_u$, leaving up to $c - \lfloor \frac{u}{2} \rfloor$ slots to be filled with edges to L. The picture so far (with $u = 6, c = 8$) is

0 :	1	2	3	*	*	*	*	*
1 :	2	3	4	*	*	*	*	*
2 :	3	4	5	*	*	*	*	*
3 :	4	5	*	*	*	*	*	*
4 :	5	0	*	*	*	*	*	*
5 :	0	1	*	*	*	*	*	*
6 :	13	12	11	10	9	8	7	
7 :	*	13	12	11	10	9	8	
8 :	*	13	12	11	10	9		
9 :	*	*	13	12	11	10		
10 :	*	*	13	12	11			
11 :	*	*	*	13	12			
12 :	*	*	*	13				
13 :	*	*	*	*				

Asterisks indicate empty slots. Note that the number of edges in the upper rows, uc, equals the number of edges between U and L. The number of lower slots equals $(c-1)+(c-3)+\cdots+2 = \frac{c^2-1}{4}$ for odd c and $(c-1)+(c-3)+\cdots+1 = \frac{c^4}{4}$ for even c. In order to fit all uc edges between U and L, we thus require

$$\lfloor \frac{c^2}{4} \rfloor \geq \binom{u}{2} \tag{6.1}$$

In the example, the 16 lower slots make up for the 15 which HalfInTurn$_6$ takes out of the top section of size $6 \cdot 8 = 48$.

6.1 Filling in the slots The bottom slots are filled in from top to bottom, left to right, modulo u, starting with $(u+1, 0)$. The top slots are then filled with the remaining edges, in reverse order:

0 :	1	2	3	12	10	9	8	6
1 :	2	3	4	12	10	9	7	6
2 :	3	4	5	12	10	8	7	6
3 :	4	5	*	11	10	8	7	6
4 :	5	0	13	11	9	8	7	6
5 :	0	1	13	11	9	8	7	6
6 :	13	12	11	10	9	8	7	
7 :	0	13	12	11	10	9	8	
8 :	1	13	12	11	10	9		
9 :	2	3	13	12	11	10		
10 :	4	5	13	12	11			
11 :	0	1	2	13	12			
12 :	3	4	5	13				
13 :	0	1	2	3				

We assume that u is at least the maximum number of slots per row $\lfloor \frac{c}{2} \rfloor$, to avoid filling a row twice with the same edge:

$$\lfloor \frac{c}{2} \rfloor \leq u \tag{6.2}$$

This condition also finds use in the next subsection to show optimality of a certain refinement.

The tournament underlying this partial algorithm is shown in Figure 2. Figure 3 makes the pattern clearer with the bigger instance $u = 21, c = 29$.

6.2 Cost analysis

THEOREM 6.1. *Partial algorithm SR_n has cost $c \leq \lceil (2-\sqrt{2})(n-1) \rceil$.*

Proof. To satisfy condition (6.1), it suffices to have $\frac{c^2}{4} \geq \frac{(n-1-c)^2}{2}$, or equivalently, $c^2 - 4(n-1) + 2(n-1)^2 \leq 0$, which, solving for c, translates to $c \geq (2-\sqrt{2})(n-1)$. It remains to show that SR$_n$ actually has cost c. This we do by presenting a total refinement sequence and verifying all retiring costs.

First, all edges in $L \times L$ are retired, bottom-up and right to left. Upon retirement of edge $(u+i, u+j)$, $|E_{u+i} - R|$ equals $|E_{u+i} \cap L \times U| + n - (u+j)$, while $|E_{u+j} - R|$ equals $|E_{u+j} \cap L \times U|$, giving a retiring cost of

$$\lceil \frac{i}{2} \rceil + c - j + \lceil \frac{j}{2} \rceil = c + \lceil \frac{i}{2} \rceil - \lfloor \frac{j}{2} \rfloor \leq c,$$

since $i < j$.

Next, all edges $(i, u+j) \in U \times L$ are retired, in increasing order of j. Upon retirement of edge $(i, u+j)$,

$$|E_i - R| = c - |\{k < j : (u+k, i) \notin E_{u+k}\}|$$

$$= c - (j - |\{k < j : (u+k, i) \in E_{u+k}\}|) \leq c - (j - \lceil \frac{j^2}{4u} \rceil),$$

since the number of slots in the first j bottom rows equals $(j-1)+(j-3)+\cdots = \lfloor \frac{j^2}{4} \rfloor$, while i appears once in every u consecutive slots. Condition (6.2) implies $\frac{c-1}{2u} \leq 1$ hence

$$|E_i - R| \leq c - (j - \lfloor \frac{j}{2} \rfloor \cdot \lceil \frac{j}{2u} \rceil) \leq c - (j - \lfloor \frac{j}{2} \rfloor) \leq c - \lceil \frac{j}{2} \rceil.$$

Combined with $|E_{u+j} - R| \leq \lceil \frac{j}{2} \rceil$ we conclude $c(i, u+j) = |E_i - R| + |E_{u+j} - R| \leq c$.

Next, all edges in HalfInTurn$_u$ are retired in their usual order at maximum cost $u-1$, which, by condition (6.1), is bounded by c.

Finally, all edges in $L \times U$ are retired in arbitrary order, at costs no more than $\lfloor \frac{c}{2} \rfloor$.

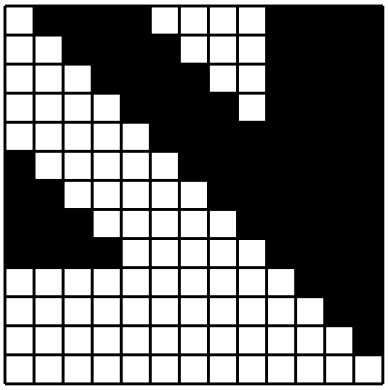

Figure 1: HalfInTurn_{13}

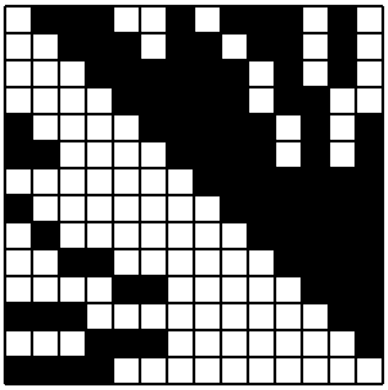

Figure 2: SR_{6+8}

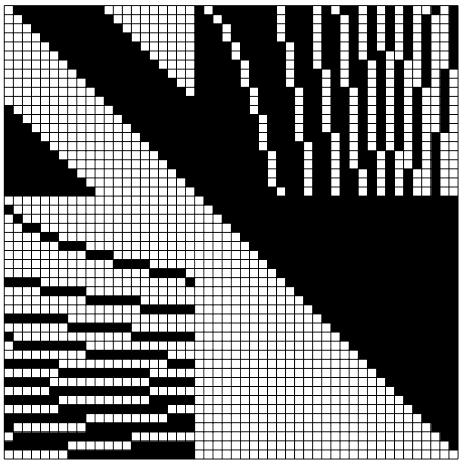

Figure 3: SR_{21+29}

7 Randomized Solutions

We can use randomization to obtain an algorithm for mutual search with expected complexity below the proven lower bound for deterministic algorithms, namely, its cost is $n/2$. The cost is the worst case, over all player locations, of the expected (over the random choices) number of queries.

Algorithm $\text{RandomHalfInConcert}_n$ uses the same tournament as HalfInTurn_n, but each player randomizes the order of its queries, and the querying proceeds "in concert," i.e. in rounds that give each row one turn for their next query. An example where the random choices have already been made can be depicted as

$$
\begin{array}{l|cccccccccc}
0: & 2 & & & & & 1 & & & & \\
1: & & 2 & & & & & 3 & & & \\
2: & & & 3 & & & & & 4 & & \\
3: & & & & 0 & & & & & 1 & \\
4: & & & & & 1 & & & & & 0
\end{array}
$$

LEMMA 7.1. *Algorithm $\text{RandomHalfInConcert}_n$ has a worst-case expected cost* $\lceil \frac{n}{2} \rceil$

Proof. In the worst case, a player located at node $n-1$ ends up querying the other player at node 0 (with the latter already having made a query in that round). The expected cost is then twice the number of queries player $n-1$ randomly orders before $(n-1, 0)$, the latter being halfway the interval $[0, \ldots, \lceil \frac{n}{2} \rceil]$

8 More than 2 Players

There is no natural semantics of the Mutual Search problem for more than two players. The view we take is that in case of a postive query, the two players involved, as well as their nodes, "merge" into one, sharing all the knowledge they acquired. A query of some node then becomes a query of the equivalance class of that node. In this view the goal of the problem is to merge all players into one.

In the two player case, a player has no identity other than the node she's located at. In the new setting, a player is an equivalence class, whose identity comprises the complete joining history of its constituent players, i.e., which joins took place when. Consequently, algorithms in the new setting have a vast scope for letting the querying behaviour depend on all those details. To avoid overly complicated definitions, we refrain from formalizing the notion of a multi-player *MS* algorithm in this extended abstract. Note that limiting the number of players in the new setting to two reduces exactly to our old model.

We now describe algorithm $\mathrm{RS}_{n,k}$ (for "RingSegments") for k players. The algorithm has a cost below n for all $k = o(\sqrt{n})$. Algorithm $\mathrm{RS}_{n,k}$ splits the n-node search space into a "ring" R of $k(k-1)m$ nodes and a "left-over" group L of m nodes. For simplicity of description we assume that n is of the form $(k(k-1)+1)m$.

The algorithm consists of two phases. During the first phase, players residing on the ring engage in a sort of HalfInTurn making $(k-1)m$ queries ahead in the ring. During the second phase, if not all the players completely joined yet, players query all the leftover nodes. If, in the first phase, one player positively queries another, then the merged player continues where the front left off, adding up the number of remaining ring queries of both. The latter ensures that a collection of k' players on the ring ends up querying $k'(k-1)m$ of ring nodes, with no node queried twice.

LEMMA 8.1. *Algorithm* $\mathrm{RS}_{n,k}$ *has cost* $k(k-1)m$.

Proof. Let k' be the number of actual players residing on the ring. Consider first the case $k' < k$. Then

$$c(\mathrm{RS}_{n,k}) = \underbrace{k'(k-1)m}_{\text{ring queries}} + \underbrace{k'm}_{\text{left-over queries}}$$

$$\leq (k-1)\ [(k-1)m + m] = (k-1)km\ .$$

Otherwise ($k' = k$), the players find each other around the ring, making $(k-1)m$ queries each in the worst case.

9 Asynchronous Mutual Search

In an asynchronous setting, one cannot rely on calls from different players to be coordinated in time. In some cases the players will have no access to a clock, in other cases the clocks may be subject to random fluctuations. In the asynchronous model, all a player can control, is what other sites are called, and in what order. We could thus formalize an asynchronous mutual search (AMS) algorithm as a partially ordered tournament in which the rows are totally ordered and edges from different rows are unordered.

But there is another subtlety. In the synchronous case, we allow only one of any two given sites to call the other (unidirectional), reasoning that if both try to call the other, then one of those calls will always be made first. In the asynchronous case however, there is no control over which call occurs first, and thus we need to allow for more general, *bidirectional* algorithms (which we refrain from defining formally here).

Although there may be possible benefits to having two sites call each other, we have been unable to find ways of exploiting this, tempting us to

CONJECTURE 9.1. *For any bidirectional algorithm, there exists a unidirectional algorithm of the same or less cost.*

Since bidirectional algorithms don't fit too well in the existing model, and since we lack nontrivial results regarding them, we choose to use the above unidirectional definition of AMS algorithm in the remainder of this section. The cost of an edge can then be defined as its position in the row-ordering (caller cost) plus the length of the target row (callee cost), since it may happen that the callee has already made all of its calls.

With relatively little control over the ordering of calls, it seems even less likely to find algorithms which improve on the intuitive bound of $n-1$ calls. For instance, Lemma 3.3 no longer holds in the asynchronous case.

But, surprisingly, a variation of SR_n, called ASR_n, achieves 1.5 times its cost. It is obtained by filling the slots of section 6.1 in reverse. The example there would thus become:

0 :	1	2	3	6	8	9	10	12
1 :	2	3	4	6	7	9	10	12
2 :	3	4	5	6	7	8	10	12
3 :	4	5	*	6	7	8	10	11
4 :	5	0	6	7	8	9	11	13
5 :	0	1	6	7	8	9	11	13
6 :	7	8	9	10	11	12	13	
7 :	0	8	9	10	11	12	13	
8 :	1	9	10	11	12	13		
9 :	2	3	10	11	12	13		
10 :	4	5	11	12	13			
11 :	0	1	2	12	13			
12 :	3	4	5	13				
13 :	0	1	2	3				

The key observation is that the shortest row has half the length of the maximum row and that edges to nodes with shorter rows appear in the later positions. In this way, the cost of any edge is at most the maximum row length plus the minimum row length. Using an analysis similar to that of Theorem 6.1, one arrives at

THEOREM 9.1. *Asynchronous algorithm* ASR_n *has cost* $c \leq \frac{3}{2}\lceil(2-\sqrt{2})(n-1)\rceil$.

Time and space constraints prevent us from including a proof here. Allowing randomness in the algorithm, a $\frac{3n}{4}$ upper bound is obtained by a variation on HalfInTurn$_n$ in which each row is ordered randomly. This appears to be the best one can do.

10 Directions for Further Research

Our lower and upper bounds for the 2-player case leave a small gap. We suspect Lemma 5.2 of being unnecessarily weak. It is tempting to try and prove a strengthened version claiming a length of no more than $(c+k)/2$ for the $(k+1)$th shortest row, which would immediately imply the optimality of SR$_n$. All algorithms we have looked at so far satisfy this condition. Unfortunately, there exist simple counterexamples, as witnessed by row distribution —where the upper half engages in a HalfInTurn algorithm before querying the lower half, which in turn engages in an AllInTurn algorithm (giving a saturated result). Such algorithms however have lots of relatively short rows, making them far from optimal. It seems reasonable to expect that an optimal algorithm has only a constant number of rows shorter than half the cost. In this light we pose the following conjecture as a lead on optimality of SR$_n$.

CONJECTURE 10.1. *Let T be an algorithm for n sites with cost c, such that no row is shorter than $\lfloor \frac{c}{2} \rfloor$. Then the $(k+1)$st shortest row of T has length at most $(c+k)/2$.*

The randomized model also leaves some open questions. We hoped to be able to proof the following result, which would imply optimality of RandomHalfInConcert:

CONJECTURE 10.2. *No randomized MS algorithm for n sites has expected cost less than $\frac{n-1}{2}$.*

It's not too hard to prove a lowerbound of $\frac{n-1}{4}$, by looking only at the number of queries one player makes, but that seems far from optimal.

The lower bounds we established for the synchronous case of course carry over to the asynchronous setting, but seem unnecessarily weak. In fact, algorithm ASR$_n$ appears to be very close to optimal. A lower bound of $\frac{3}{4}n$ might be provable, even with randomization, but we haven't had much time to ponder this issue. Finally, the conjecture concerning bidirectional asynchronous algorithms begs further investigation.

Acknowledgment

Matt Franklin thanks Sheryl Koenigsberg for helpful discussions.

References

[1] B. Awerbuch and D. Peleg, "On-line tracking of mobile users," Technical Memo TM-410, MIT, Lab. for Computer Science, 1989.

[2] B. Awerbuch and D. Peleg, "Sparse Partitions," *Proc. 31st IEEE Symp. on Foundations of Computer Science*, pp. 503-513, 1990.

[3] R. Axelrod and W. Hamilton, "The evolution of cooperation," *Science*, Vol. 211, pp. 1390-1396, March 1988.

[4] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation," *Proc. 20th Annual ACM Symp. on the Theory of Computing*, pp. 1-10, 1988.

[5] E. Billard and J. Pasquale, "Probabilistic Coalition Formation in Distributed Knowledge Environments", *IEEE Transactions on Systems, Man, and Cybernetics*, 25(2), pp. 277-286, February 1995.

[6] B. Chor, S. Goldwasser, S, Micali, and B. Awerbuch, "Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults," *Proc. 26th Annual IEEE Symposium on the Foundations of Computer Science*, pp. 383-395, 1985.

[7] R. Gallager, P. Humblet and P. Spira, "A distributed algorithm for minimum-weight spanning trees," *ACM Transactions on Programming Languages and Systems,* 5(1):66-77, January 1983.

[8] S.L.Hakimi, "Steiner's problem in graphs and its implications," *Networks*, 1 (1971) 113-133.

[9] B. Huberman and T. Hogg, "The behavior of computational ecologies," in *The Ecology of Computation* (B. Huberman, ed.), North Holland, Elsevier Science Publishers, 1988.

[10] B.O. Koopman, "The Theory of Search, Parts I–III," *Operations Research* Vol. 4, pp. 324-346 (1956), Vol. 4, pp. 503-531 (1956), and Vol. 5, pp. 613-626 (1957).

[11] E. Kranakis and P.M.B. Vitanyi, "A note on weighted distributed Match-Making," *Mathematical Systems Theory*, 25(1992), pp. 123-140.

[12] L. Lamport, R.E. Shostak and M. Pease, "The Byzantine generals problem," *ACM Trans. Prog. Lang. and Systems*, 4:3 (1982), pp. 382–401.

[13] M. Maekawa, "A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems," *ACM Transactions on Computer Systems*, 3(1985), pp. 145-159.

[14] J. Maynard-Smith, *Evolution and the Theory of Games*, Cambridge University Press, 1982.

[15] S.J. Mullender and P.M.B. Vitanyi, "Distributed match-making," *Algorithmica*, 3 (1988), pp. 367-391.

[16] Q. Zhu and J. Oommen, "Optimal Search with Unknown Target Distributions," to appear in *Proc. XVII International Conference of the Chilean Computer Science Society*, Valparaíso, Chile, November 1997.

A Related Research

DISTRIBUTED MATCH-MAKING. In "distributed match-making" [15] the set-up is similar to mutual search except that if a player at node i queries a node k previously queried by a player at node j, then the query to node k returns j [15, 11]. In general it is assumed that the search is in a structured database in the sense that there have been an initial set of queries from players at all nodes to leave traces of their whereabouts at other nodes. This problem is basic to distributed mutual exclusion [13] and distributed name server [15]. The difference is that distributed match-making operates in a cooperative structured environment while mutual search operates in a noncooperative unstructured environment. Some of our protocol representation ideas were inspired by this seminal paper.

TRACKING OF MOBILE USERS. Another related search problem is the (on-line) tracking of a mobile user defined by Awerbuch and Peleg [1, 2], where the goal is to access an object which can change location in the network. The mobile user moves among the nodes of the network. From time to time two types of requests are invoked at the nodes: $move(i, j)$ (move the user from node i to node j) and $find(i)$ (send a message from node i to the current location of the user). The overall goal is to minimize the communication cost. In contrast, our search problem is symmetric, and the players are static.

DISTRIBUTED TREE CONSTRUCTION. The goal of *MS* can be thought of as forming a clique among the nodes of the players. In this sense the problem is related to tree construction problems, such as the (distributed) minimum-weight spanning tree (MST) [7] and Steiner tree (e.g., [8]). Besides other differences (e.g., in those problems the nodes are given, and it is the (weight of the) edges which are (globally) unknown; in the version of *MS* we consider it is required that the players be directly connected, not just in the same connected component; etc.), *MS* is concerned with optimizing the process, and not the outcome of the construction.

CONSPIRACY START-UP. Another possible application of *MS* is to secure multi-party computation. The fields of fault-tolerant distributed computing and secure multi-party computation are concerned with n players, a fraction (t) of which may be arbitrarily faulty. It is traditionally assumed (e.g., [4, 12]) that the faulty players have complete knowledge of who they are, and that they can collude and act in concert. We would like to weaken this assumption and investigate the complexity and cost of achieving such a perfect coordination. We consider this paper as a first step towards the study of such *spontaneous* adversaries and coalition forming. In fact, many testbed problems (e.g., Byzantine agreement [12]) and secure multi-party primitives (e.g., verifiable secret sharing [6]) are bound to have interesting characterizations and efficient solutions under this new adversary.

PROBABILISTIC COALITION FORMATION. Billard and Pasquale [5] study the effect of communication environments on the level of knowledge concerning group, or coalition, formation in a distributed system. The motivation is the potential for improved performance of a group of agents depending on their ability to utilize shared resources. In this particular model the agents make randomized decisions regarding with whom to coordinate, and the payoffs are evaluated in different basic structures and amounts of communication (e.g., broadcast, master-slave, etc.). Their work has in turn been influenced by work on computational ecologies [9] and game theory studies [14]. In constrast, ours is a search problem with the goal of minimizing the communication cost of achieving a perfect coalition.

SEARCH THEORY. Finally, *MS* is also somewhat related to Search Theory and Optimal Search (e.g., [10]). Search Theory is generally concerned with locating an object in a set of n locations, given a "target distribution," which describes the probability of the object being at the different locations. In turn, Optimal Search involves computing how resources (e.g., search time) can be allocated so as to maximize the probability of detection. Typically, it is assumed that the target distribution is known, although more recently this assumption has been relaxed [16]. Besides the multiple agent aspect, the setting of *MS* is more adversarial, as we measure worst-case cost.