Mutual Search*

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

Abstract

We introduce a search problem called "mutual search" where k agents , arbitrarily distributed over n sites, are required to locate one another by posing queries of the form "Anybody at site i?". We ask for the least number of queries that is necessary and sufficient. For the case of two agents using deterministic protocols we obtain the following worst-case results: In an oblivious setting (where all pre-planned queries are executed) there is no savings: $n-1$ queries are required and are sufficient. In a nonoblivious setting we can exploit the paradigm of "no news is also news" to obtain significant savings: in the synchronous case $0.586n$ queries suffice and $0.536n$ queries are required; in the asynchronous case $0.896n$ queries suffice and a fortiori 0.536 queries are required; for $o(\sqrt{n})$ agents using a synchronous deterministic protocol less than n queries suffice; there is a simple randomized protocol for two agents with worst-case expected $0.5n$ queries and all randomized protocols require at least $0.125n$ worst-case expected queries. The graph-theoretic framework we formulate for expressing and analyzing algorithms for this problem may be of independent interest.

1 Introduction

Search problems come in many forms [11]. Perhaps the following one is new: Suppose you and a friend check separately into the same hotel in Las Vegas in different rooms. For reasons we don't go

*A preliminary version of this work was presented at the *9th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, January 1998

†Partially supported by NWO through NFI Project ALADDIN under contract number NF 62-376 and by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556. Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. `buhrman@cwi.nl`.

‡Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA. `franklin@parc.xerox.com`.

§Bell Labs - Lucent Technologies, 600 Mountain Ave, Murray Hill, NJ 07974, USA. `garay@research.bell-labs.com`. Partially supported by NWO through NFI Project ALADDIN under contract number NF 62-376 and by a SION grant and by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556 while the author was visiting CWI.

¶University Twente, Dept. of Computer Science, Box 217, 7500 AE Enschede, The Netherlands. `jhh@xs4all.nl`.

‖Partially supported by NWO through NFI Project ALADDIN under contract number NF 62-376 and by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556. Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. `tromp@cwi.nl`.

**Partially supported by NWO through NFI Project ALADDIN under contract number NF 62-376 and by the European Union through NeuroCOLT ESPRIT Working Group Nr. 8556. Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. `paulv@ cwi.nl`.

into here you both don't want to draw any attention to your relation. You are supposed to phone one another at noon, but unfortunately you don't know each others' room number. What to do? Every room contains a room phone and room number. You can phone all other rooms in the hotel to find your friend and she can do the same (if the wrong person picks up the phone you simply hang up and nobody is the wiser). This will cost a lot of time: there are 1000 rooms. In the worst case you use almost 2000 room calls together. Luckily you and your friend know the protocol in this paper: you locate one another using only 586 room calls together in the worst case. There are more serious problems of the same nature that are listed in the Appendix A.

In general we can think of $k \geq 2$ agents having to find each others' locations in a uniform unstructured search space consisting of n sites ($n \geq k$). The sites have distinct identities, say $0, \ldots, n-1$ ($k \leq n$), every site can contain zero or one agent , and the agents execute identical protocols based on the values n, k with their site identity as input. The agents can execute queries of the form "Anybody at site i" and every such query results in an answer "yes" or "no." We say that two agents know each others' location as soon as one agent queries the location of the other agent or the other way around. Before that happens they don't know each other's location. The relation "know location" is transitive and the problem is solved if all k agents know one another's location. This type of search can be called *Mutual Search* (*MS*). We analyze the cost in number of queries for the case $k = 2$ under various timing assumptions for deterministic and randomized algorithms. We also give a result for the general case of $k = o(\sqrt{n})$ agents .

Our Results: We first look at *deterministic protocols* for two agents . If the protocol is *oblivious*, so that the cost for each agent is a fixed number of queries, then there are no significant savings possible: two agents need to place at least $n-1$ queries in total in the worst case. [1] Namely, given a protocol, construct the directed graph on $\{0, \ldots, n-1\}$ with an arc from i to j if an agent at i queries node j. For every pair there must be at least one arc. Hence there are at least $\binom{n}{2}$ arcs in total, and the average number of outgoing arcs per node is at least $\frac{n-1}{2}$. It follows that some pair of nodes must together have twice this number, or $n-1$, of outgoing edges (this can be refined to $2\lceil\frac{n-1}{2}\rceil$ for all $n > 2$). The tightness of this bound is witnessed by an algorithm called HalfInTurn, to be discussed in Section 2.2:

Oblivious case ($k = 2$): Both upper bound and lower bound are $2\lceil\frac{n-1}{2}\rceil$ queries in the deterministic worst-case.

In the remainder of the paper we analyze the nonoblivious case. We obtain savings by exploiting the information inherent in timing ("no news is also news") and a prescribed order of events.

Synchronous case ($k = 2$): in Section 2.5 we present 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 2.4.

Asynchronous case ($k = 2$): In Section 3 we show that there is a mutual search algorithm that uses asymptotically $0.896n$ queries. The best lower bound we know of is the $0.536n$ lower bound in Section 2.4.

Randomized case ($k = 2$): We consider *randomized algorithms* for the problem in Section 4. A synchronous randomized algorithm is shown to have a worst-case (over agent location) expected (over random coin flips) cost of about $\frac{n+1}{2}$, thus beating the deterministic lower bound. We show

[1] "Oblivious" means that the queries scheduled at certain time slots take place independent of the replies received. The same lower bound holds if there is no FIFO discipline: the answer to a query can arrive only after the following queries are executed. This is the case when the sites are nodes in a computer network, the agents are processes at those nodes that query by sending messages over communication links with unknown bounded (or possibly unbounded as in the FLP model [8]) communication delay without waiting for the answers to earlier messages. Then, a process may have to send all its messages before an affirmative reply to one of the early messages is received.

a lower bound on the worst-case expected number of queries of $\frac{n-1}{8}$.
Synchronous multi-agent case ($k = o(\sqrt{n})$): In Section 5 we present $\mathrm{RS}_{n,k}$, a synchronous deterministic algorithm for $k \geq 2$ agents with a cost well below n for all $k = o(\sqrt{n})$.

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 Synchronous Case For Two Agents

In Section 2.1 we formulate the model for the synchronous case with $k = 2$ agents located at n sites and give a framework for expressing and analyzing the structure of algorithms for this problem. We analyze this case fairly completely, but in later sections we also present results for other instances of the MS problem.

2.1 Model and Definitions

Consider n sites numbered $0, \ldots, n-1$ with $k \leq n$ agents distributed over the n sites with zero or one agent per site. Time is discrete, with time slots numbered $0, 1, \ldots$. An agent at site i can perform *queries* of the form $q = q(i,j)$ with the following semantics: if site j contains an agent then the associated answer from site j to the agent at i is 1 (yes) otherwise 0 (no), $0 \leq i \neq j \leq n-1$. For definitional reasons we also require an *empty* query $\perp$ (skipped query) with an empty associated answer (skipped answer). The query and answer take place in the same time slot. Given the number n of sites and the number k of agents , a *mutual search protocol* A consists of a (possibly randomized) algorithm to produce the sequence of queries an agent at site i $(0 \leq i \leq n-1)$ executes together with the time instants it executes them: $A(i) = q_1, \ldots, q_{m_i}$ where q_t is the query executed at the tth time slot, $t := 0, \ldots, m_i$. If $q_t = \perp$ then at the tth time slot an agent at site i skips a query. A mutual search *execution* of k agents located at sites $i_1, \ldots, i_k$ consists of the list $\mathcal{A} = A(i_1), \ldots, A(i_k)$. We require that in every time slot t there are zero or one queries from this list that are $\neq \perp$. Hence we can view $\mathcal{A}$ as a total order on all queries by the k agents and $A(i)$ as a restriction to the entries performed by an agent at site i $(i := i_1, \ldots, i_k)$. The *cost* of an execution is the number of queries $q_t \neq \perp$ with $t \leq t_0$ and t_0 is the least index such that the answer to query q_{t_0} equals 1. That is, we are interested in the number of queries until first contact is made. [2] The (*worst-case*) *cost* of a *mutual search protocol* is the maximum cost of an execution of the protocol. The *worst-case cost* of *mutual search* is the minimum (worst-case) cost taken over all mutual search protocols.

In this paper we consider the case of $k = 2$ agents unless explicitly stated otherwise. The case $k > 2$ is open except for the result in Section 5. Informally, a mutual search protocol specifies, for every site that an agent can find itself in, what to do at every time slot: either stay idle or query another site as specified by the protocol and receive the reply. Every time slot harbours at most one query and its answer. [3]

[2] This is the cost of a *non-oblivious* execution. The cost of an *oblivious* execution is simply the number queries $\neq \perp$ occurring in the list $\mathcal{A}$. This case was already completely analyzed in the Introduction. Therefore, in the remainder of the paper we only consider non-oblivious executions without stating this every time.

[3] We can allow simultaneous queries. If there are k agents , then every time slot can have at most k queries $\neq \perp$.

For every pair of sites such a protocol determines which site will first query the other. After the first such query takes place, the execution terminates, so that the latter site need never execute the now redundant query of the former site. Every such algorithm implies a *tournament*: a directed graph having a single arc between every pair of nodes. An edge from node i to node j represents site i querying site j. The different times at which the $\binom{n}{2}$ queries/edges are scheduled induce a total timing order on the edges. Since the cost of running the algorithm depends only on which queries are made before a certain contacting query, this total order by itself captures the essence of the timing of queries.

An algorithm can thus be specified by a tournament (telling us who queries whom) plus a separate total timing order on its edges (telling us when). Note that the timing order is completely unrelated to the ordering of the arcs; the tournament may well be cyclic in the sense of containing cycles of arcs. [4] For us an *ordered tournament* is a (tournament, order) pair where the order is a separate total order on the arcs of the tournament.

Definition 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.*

This way E_i is the set of queries agent i can potentially make. Define the cost of an edge as the number of queries that will be made if the two agents happen to reside on its incident nodes.

Definition 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 $\{f \in F : f \prec e\}$.*

If the agents are located at nodes i and j and the edge e between them is directed from i to j, then at the time i queries j, agent i has made all queries in $E_i^{\prec e}$, while agent j has made all queries in $E_j^{\prec e}$. We have now all what is needed to present and analyze some basic algorithms for the problem which will form the basis of a better algorithm.

2.2 Some Simple Mutual Search Algorithms

The first algorithm, $\mathrm{AllInTurn}_n$, lets each site in turn query all the other sites. For instance, $\mathrm{AllInTurn}_4$ can be depicted as[5]

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

Here, the sites are shown as labeling the rows of a matrix, whose columns represent successive time instances. A number j appearing in row i and column t of the matrix represents the query from i to j scheduled at time t. As an example execution, suppose the agents are situated at sites 0 and 2. Then at time 0, (the agent at site) 0 queries 1 and receives reply "no": the second agent is not there. Next 0 queries 2 and contacts the second agent, finishing the execution at a cost of 2 queries.

The precise cost then depends only on how we account the at most k queries in the time slot containing the first query with answer 1. Under different conventions the results can only vary by $k-1$, that is, by only 1 unit for $k = 2$.

[4] For example, algorithm $\mathrm{HalfInTurn}_n$ below has cycles of arcs.

[5] Another way would be to draw the tournament on nodes $0, \ldots, 3$ and label every arc with a time. The matrix representation we use seems more convenient to obtain the results.

Lemma 1 *Algorithm AllInTurn$_n$ has cost $n-1$.*

Proof. It is in fact easy to see that $c(i,j) = j - i$. A agent at site i makes this many queries to contact the other agent 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

$$\begin{array}{llllllllllll}
0: & 1 & 2 & & & & & & & & & \\
1: & & & 2 & 3 & & & & & & & \\
2: & & & & & 3 & 4 & & & & & \\
3: & & & & & & & 4 & 0 & & & \\
4: & & & & & & & & & 0 & 1 &
\end{array}$$

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

Lemma 2 *Algorithm HalfInTurn$_n$ has cost $n-1$.*

Proof. Suppose $i < j$. If $j - i \leq \lfloor n/2 \rfloor$, 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 *An algorithm is called* saturated *if its cost equals its maximum row length.*

An example of a saturated algorithm is AllInTurn$_n$, whose cost of $n-1$ equals the length of row 0.

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

2.3 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 *A* partial MS *algorithm is a partially ordered tournament $T = (V, E, \prec, R)$, where $R \subseteq E$ is the subset of* retired *edges, and $\prec$ is now a partial order, which:*

- *totally orders R,*
- *orders all of $E - R$ before all of R, and*
- *leaves $E - R$ (pairwise) 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 $\{(f, e) : f \in E - R'\}$. The edge e that is added to R was $\prec R$ and since $\prec'$ extends $\prec$, becomes the new earliest edge in R'.

Algorithm refinement proceeds backward in time—the queries to be made last are scheduled first. An example partial tournament, with 2 retired edges, is

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

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 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 5 *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 *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 the minimum retiring cost, and choosing e doesn't hamper us in any way. The following proof makes this notion of "non-hampering" precise.

Proof. Consider an optimal total refinement from T to some algorithm T'', in which, at some point, say after $e_1, e_2, \ldots, e_k$, 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 $e \prec'' e_k \prec'' \ldots \prec'' e_1$ whereas T' has $e_k \prec' \ldots \prec' e_1 \prec' e$. If we compare the costs c'' and c' for any edge in T'' and T' respectively, we see that for $1 \le i \le k$, $c'(e_i) \le c''(e_i)$, $c'(e) \ge c''(e)$, and all other edges cost the same. However, $c'(e) \le c(e_1)$ by assumption, and so T' must be optimal too. $\square$

Since optimal refinement is straightforward greedy procedure that can be performed automatically an optimal timed algorithm is uniquely determined by just its associated tournament. By graphically showing the tournament's adjacency matrix, one obtains a visually insightful representation; for instance, $\text{SaturatedHalfInTurn}_{13}$ 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:

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

2.4 Lower Bounds

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

Lemma 5 *Every* 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 6 *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 1 *Every* MS *algorithm T for n sites has cost at least $(4 - 2\sqrt{3})(n-1)$ $(\approx 0.536n)$.*

Proof. Since every row has length at most c, Lemma 6 implies:

$$\begin{aligned} |E| &= \frac{n(n-1)}{2} \leq nc - \sum_{k=0}^{c/2} c/2 - k \\ &= nc - \frac{(c/2)(c/2+1)}{2}, \end{aligned}$$

which in turn implies:

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

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

2.5 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 $U = \{0, \ldots, u-1\}$ consisting of u nodes and a lower group $L = \{u, \ldots, n-1\}$ consisting of $c = n-u$ nodes (which is the cost we are 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 $\mathrm{HalfInTurn}_u$, while the lower group engages in a slight variation on $\mathrm{AllInTurn}_c$ 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 $\mathrm{HalfInTurn}_u$, leaving up to $c - \lfloor \frac{u}{2} \rfloor$ slots per row to be filled with edges to L. The picture so far (with $u = 6, c = n-u = 8$) is

$$
\begin{array}{rcccccccc}
0: & 1 & 2 & 3 & * & * & * & * & * \\
1: & 2 & 3 & 4 & * & * & * & * & * \\
2: & 3 & 4 & 5 & * & * & * & * & * \\
3: & 4 & 5 & * & * & * & * & * & * \\
4: & 5 & 0 & * & * & * & * & * & * \\
5: & 0 & 1 & * & * & * & * & * & * \\
6: & 13 & 12 & 11 & 10 & 9 & 8 & 7 & \\
7: & * & 13 & 12 & 11 & 10 & 9 & 8 & \\
8: & * & 13 & 12 & 11 & 10 & 9 & & \\
9: & * & * & 13 & 12 & 11 & 10 & & \\
10: & * & * & 13 & 12 & 11 & & & \\
11: & * & * & * & 13 & 12 & & & \\
12: & * & * & * & 13 & & & & \\
13: & * & * & * & * & & & &
\end{array}
$$

Asterisks indicate empty slots. By simple geometric properties of the picture we analyze the requirements. Define block B_U as the elements in the upper u rows determined by U and define block B_L as the elements in the lower $c = n-u$ rows determined by L. The block B_U has uc elements of which $\binom{u}{2}$ are used for the edges in $U \times U$. There are $uc - \binom{u}{2}$ slots in U that can be used for edges from U to L. In the lower block B_L the number of open slots that can be used for edges from L and U equals $(c-1)+(c-3)+\cdots+2 = (c^2-1)/4$ for odd c and $(c-1)+(c-3)+\cdots+1 = c^2/4$ for even c. That is $\lfloor c^2/4 \rfloor$ open slots. In order to fit all uc edges between U and L, the number of open slots must be sufficient:

$$uc - \binom{u}{2} + \lfloor c^2/4 \rfloor \geq uc.$$

As it happens, the number of elements in B_U, that is uc, equals the number of edges between U

and L. Therefore,

$$\lfloor \frac{c^2}{4} \rfloor \geq \binom{u}{2} \tag{1}$$

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

2.6 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 $\lfloor c/2 \rfloor$, to avoid filling a row twice with the same edge:

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

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 $u = 21, c = 29$.

2.7 Cost analysis

Theorem 2 *Partial algorithm SR_n has cost $c = \lceil (2-\sqrt{2})(n-1) \rceil$ ($\approx 0.586n$).*

Proof. To satisfy condition (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.

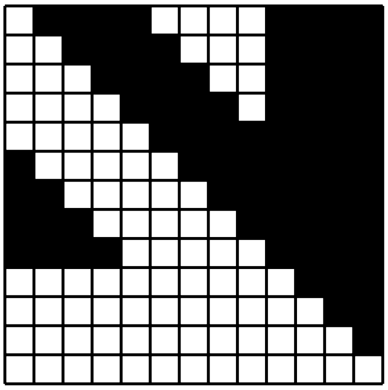

Figure 1: $\mathrm{HalfInTurn}_{13}$

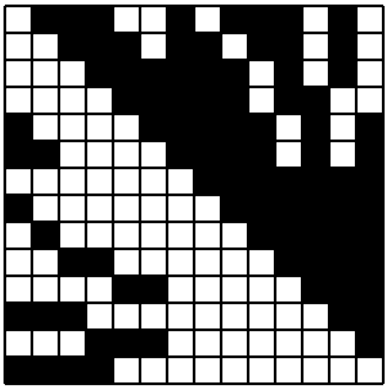

Figure 2: SR_{6+8}

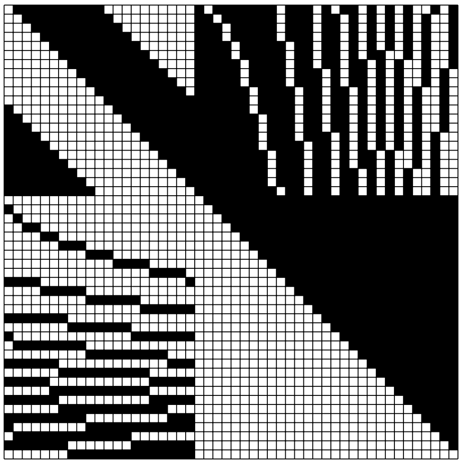

Figure 3: SR_{21+29}

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 $(j-1) + (j-3) + \cdots = \lfloor j^2/4 \rfloor$, while i appears once in every u consecutive slots. Condition (2) implies

$$\frac{j}{2u} \leq \frac{c-1}{2u} \leq 1,$$

and 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 (1), is bounded by c.

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

3 Asynchronous Case for Two Agents

In an asynchronous setting, one cannot rely on queries from different agents to be coordinated in time. In some cases the agents will have no access to a clock, in other cases the clocks may be subject to random fluctuations. In the asynchronous model, all an agent can control, is what other sites are queried, and in what order. We 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. [6] The cost of an edge is defined as its position in the row-ordering (querier cost) plus the length of the target row (queree cost), since it may happen that the queree has already made all of its queries.

Upper bound: With relatively little control over the ordering of queries, it seems even less likely to find algorithms which improve on the intuitive bound of $n-1$ queries. For instance, Lemma 3 no longer holds in the asynchronous case. But, surprisingly, a variation of SR_n, called ASR_n, achieves about 1.5 times its cost. It is obtained by reversing within every row the order of edges pointing to nodes in the lower group L of section 2.6. The example there now becomes:

[6] There is a subtlety here. In the synchronous case, we allow only one of any two given sites to query the other (unidirectional), reasoning that if both try to query the other, then one of those queries will always be made first. In the asynchronous case however, there is no control over which query 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 query each other, we have been unable to find ways of exploiting this. We conjecture that 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 use the above unidirectional definition of AMS algorithm in the remainder of this section.

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. Using an analysis similar to that of Theorem 2, one arrives at:

Theorem 3 *Asynchronous algorithm ASR_n has cost at most* $((5-\sqrt{2})/4)n$ $(\approx 0.896n)$.

Proof. We check that every one of the four types of edges has an asynchronous cost of at most (note that $u = n - c$):

$$c + \frac{3}{4}u = \frac{3n + c}{4},$$

where $c = \lceil (2-\sqrt{2})(n-1) \rceil$ as in Theorem 2. For some edges we use the fact that $c \le \frac{3}{2}u$, and show that the cost is at most $c + \frac{1}{2}c$. Recall that row $u + j$ has length $c - 1 - \lfloor \frac{j}{2} \rfloor$.

Edge $(u+i, u+j) \in L \times L$ has asynchronous cost:

$$\begin{aligned}
&\lceil \frac{i}{2} \rceil + j - i - 1 + c - 1 - \lfloor \frac{j}{2} \rfloor \\
&= c - 2 + \lceil \frac{j}{2} \rceil - \lfloor \frac{i}{2} \rfloor \\
&\le c - 2 + \lceil \frac{c-1}{2} \rceil.
\end{aligned}$$

Edge $(i, u+j) \in U \times L$ has asynchronous cost:

$$\begin{aligned}
&\le \frac{u}{2} + j - |\{k < j : (u+k, i) \in E_{u+k}\}| + c - 1 - \lfloor \frac{j}{2} \rfloor \\
&\le c - 1 + \lceil \frac{j}{2} \rceil + \frac{u}{2} - \lfloor \frac{j^2}{4u} \rfloor \\
&\le c + \frac{j + u - j^2/2u}{2}.
\end{aligned}$$

Writing j as xu gives

$$j + u - j^2/2u = (x + 1 - x^2/2)u \leq \frac{3}{2}u,$$

since $x + 1 - x^2/2$ assumes its maximum at $x = 1$. Hence,

$$c + \frac{j + u - j^2/2u}{2} \leq c + \frac{3}{4}u.$$

Edges in HalfInTurn_u $(U \times U)$ have asynchronous cost at most:

$$\frac{u}{2} + c \leq \frac{3}{2}c.$$

Finally, edges in $L \times U$ have cost at most:

$$\lceil \frac{c-1}{2} \rceil + c \leq \frac{3}{2}c.$$

□

Lower bound: The $(4 - 2\sqrt{3})(n-1)$ lower bound on the synchronous case (Theorem 1 holds a fortiori for the asynchronous case.

4 Randomized Case for Two Agents

For a randomized MS protocol the *worst-case expected cost* is the worst case, over all agent locations, of the expected (over the random coin flips) number of queries. We can use randomization to obtain an algorithm for mutual search with expected complexity below the proven lower bound for deterministic algorithms, namely, a cost of $n/2$.

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

$$\begin{array}{lcccccccccc}
0: & 2 & & & & & 1 & & & & \\
1: & & 2 & & & & & 3 & & & \\
2: & & & 3 & & & & & 4 & & \\
3: & & & & 0 & & & & & 4 & \\
4: & & & & & 1 & & & & & 0
\end{array}$$

Theorem 4 *Algorithm $\text{RandomHalfInConcert}_n$ has a worst-case expected cost $\frac{n+1}{2}$ for n is odd and about $\lceil \frac{n+1}{2} \rceil$ for n is even.*

Proof. A worst case occurs when an agent located at node $n-1$ ends up querying the other agent at node 0 (with the latter already having made a query in that round). The expected number of queries is twice the number of queries the agent at $n-1$ makes in a uniformly random order of the sites $0, 1, \ldots, \lceil \frac{n-1}{2} \rceil$ ending in, and including, the final successful query to site 0. This is about $\lceil \frac{n+1}{2} \rceil$ for n is even and $\frac{n+1}{2}$ for n is odd. □

Asynchronous Randomized Case: Allowing randomness in the algorithm, a $\frac{3n}{4}$ upper bound is obtained by a variation on $\text{RandomHalfInConcert}_n$ in which each row is ordered randomly. This appears (but is not proven) to be the best one can do. The best lower bound we have is the synchronous randomized $\frac{n-1}{8}$ lower bound below.

Lower bound: We prove a lower bound for the synchronous case (and hence for the asynchronous case).

Theorem 5 *For every randomized MS algorithm for two agents on n sites using a bounded number of coin flips the worst-case expected cost is at least $\frac{n-1}{8}$.*

Proof. Fix a randomized MS algorithm for n sites with $k = 2$ agents. We assume that every agent uses at most a number of coin flips that is bounded by a total function of n. Assume that the maximum of the expected number of queries is c where the maximum is taken over all placements of two agents on n sites and the expectation is taken over the randomized coin flips. By Markov's inequality, for every placement of the two agents there is probability at least $\frac{1}{2}$ that there are at most $2c$ queries by both parties together up to contact. Hence, at least $\frac{1}{2}$ of all combinations of agents' positions and sequences of used coin flips have cost at most $2c$.

Now consider a matrix where the rows correspond to the agents' positions and the columns to the pairs of sequences of coin flips. It is important for the remainder of the proof that two agents at i, j can use sequences of coin flips α and β in two ways: the agent at i uses α and the agent at j uses β and vice versa. [7] Therefore, there are $2\binom{n}{2}$ rows and (by the boundedness of the length of the coin flip sequence) a bounded number of columns. By the pigeon hole principle at least one out of all coin flip sequence pairs (the columns) has at least $\frac{1}{2}$ of the $2\binom{n}{2}$ agents' positions (the row entries) incurring cost at most $2c$.

Fix any such column, say the one determined by coin flip sequences α and β. We define a "deterministic pseudo-MS" algorithm for two agents in $2n$ sites by splitting every original site i into two copies: a site i_α and a site i_β $(0 \leq i \leq n-1)$. Every new site i_γ $(\gamma \in \{\alpha, \beta\})$ executes a new deterministic algorithm based on the coin flip sequence γ. This new algorithm is completely specified by the following: If the original algorithm specifies that an agent at site i using coin flip sequence α queries site j in time slot t, then the new algorithm specifies [8] that an agent at site i_α queries site j_β in time slot t $(0 \leq i \neq j \leq n-1,\ t := 1, 2 \ldots)$. The analogous rule holds with α and β interchanged. [9]

Consider only the (at least) $\binom{n}{2}$ row entries (pairs of nodes) that have cost at most $2c$ and put an arc from node i_α to node j_β if i_α queries j_β in the new algorithm. This results in a directed bipartite graph on $2n$ nodes with at least $\binom{n}{2}$ arcs. The average number of outgoing arcs per node is at least $\frac{n-1}{4}$. Fix a node with at least $\frac{n-1}{4}$ outgoing arcs, say i_α, and consider the last node queried by i_α, say node j_β. Then i_α queries j_β but j_β doesn't query i_α. Hence for the pair (i_α, j_β) there are at least $\frac{n-1}{4}$ queries executed which, by assumption, is at most $2c$. Therefore the expectation $c \geq \frac{n-1}{8}$. □

5 Synchronous Case for Many Agents

In the case of $k > 2$ agents we define the mutual search as before, but now the two agents involved in a query with an affermative answer, as well as their nodes, "merge" into one, sharing all the

[7]Otherwise it can happen that in the same column in agent positions (i, j) the agent at i uses sequence α and in the agent positions (i, k) with $k \neq j$ the agent at i uses sequence β $(\beta \neq \alpha)$. This would contradict our intended reduction to a deterministic algorithm.

[8]In the actual random execution the pair of coin flip sequences in use in this fixed column is α and β. Therefore, if there are agents at site i and site j and the agent at site i uses coin flip sequence α then the agent at site j uses coin flip sequence β.

[9]Note that α-sites only query β-sites and the other way around.

knowledge they acquired previously. A query of some node then becomes a query to the equivalance class of that node. In this view the goal of the problem is to merge all agents into one. [10]

In the two agent case, an agent has no input or “knowledge” other than the index of the site he is located at. In the multi-agent case we assume a “full information protocol” where every agent is an equivalence class whose knowledge comprises the complete timed querying and answering history of its constituent agents . Consequently, algorithms in the new setting have a vast scope for letting the querying behavior depend on all those details in case $k > 2$. Limiting the number of agents in the new setting to two reduces exactly to our old model. [11]

We now describe algorithm $\mathrm{RS}_{n,k}$ (for “RingSegments”) for k agents . 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, agents 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 agents are completely joined yet, agents query all the leftover nodes. If, in the first phase, one agent queries a node affirmatively, then the agents merge and the merged agent continues where the front agent left off, adding up the number of remaining ring queries of both. The latter ensures that a collection of k' agents on the ring ends up querying $k'(k-1)m$ of ring nodes, with no node queried twice.

Theorem 6 *Algorithm* $\mathrm{RS}_{n,k}$ *has cost* $k(k-1)m$.

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

$$\begin{aligned} c(\mathrm{RS}_{n,k}) &= \underbrace{k'(k-1)m}_{\text{ring queries}} + \underbrace{k'm}_{\text{left-over queries}} \\ &\leq (k-1)\ [(k-1)m + m] = (k-1)km\ . \end{aligned}$$

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

6 Conclusion

The lower and upper bounds for the synchronous deterministic two agent case leave a small gap. We suspect Lemma 6 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: “Let T be an algorithm for

[10] Of course, there are other possibilities to generalize the Mutual Search problem $k > 2$ agents , in terms of how agents that have contacted one another coordinate the remainder of their mutual search.

[11] We refrain from giving complicated formal definitions of a multi-player *MS* protocol and cost measure which are not needed for the simple upper bound derived here.

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 and asynchronous two-agent cases leave large gaps between lower bound and upper bound. The multi-agent case is almost completely unexplored for all models. The same holds for bidirectional asynchronous algorithms as in footnote 6.

Acknowledgment

[HB] dedicates this paper to Kees Buhrman.

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] M. Fischer, N. Lynch, and M.S. Paterson, Impossibility of Distributed Consensus with One Faulty Processor. *J. Assoc. Comput. Mach.* 32:2(1985), 374–382.

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

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

[11] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, 1998.

[12] 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).

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

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

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

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

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

[18] 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 Work

DISTRIBUTED MATCH-MAKING. In "distributed match-making" [17] the set-up is similar to mutual search except that if an agent at node i queries a node k about an agent residing at node j and the latter agent has posted its whereabout at node k, then the query to node k returns j [17, 13]. 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 agents at all nodes to leave traces of their whereabouts at other nodes. This problem is basic to distributed mutual exclusion [15] and distributed name server [17]. 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 $find(i)$ (do a query 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 agents are static.

DISTRIBUTED TREE CONSTRUCTION. The goal of *MS* can be thought of as forming a clique among the nodes at which the agents are located. In this sense the problem is related to tree construction problems, such as the (distributed) minimum-weight spanning tree (MST) [7] and Steiner tree [9]. Besides other differences *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. Fault-tolerant distributed computing and secure multi-party computation are concerned with n agents , a fraction (t) of which may be faulty. It is traditionally assumed [4, 14] that every faulty agent has complete knowledge of who and where all faulty agents 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 test-bed problems (Byzantine agreement [14]) and secure multi-party primitives (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 (broadcast, master-slave, etc.). Their work has in turn been influenced by work on computational ecologies [10] and game theory studies [16]. In contrast, ours is a search problem with the goal of minimizing the communication cost of achieving a perfect coalition.

Search Theory. Finally, *MS* is also related to search theory and optimal search [12]. 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 (like 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 [18]. Besides the multiple agent aspect, the setting of *MS* is more adversarial, as we measure worst-case cost.