Randomised Mutual Search[⋆]

Jaap-Henk Hoepman

Department of Computer Science, University of Twente
P.O.Box 217, 7500 AE Enschede, the Netherlands
hoepman@cs.utwente.nl

Abstract We study the efficiency of randomised solutions to the mutual search problem of finding k agents distributed over n nodes. For a restricted class of so-called *linear* randomised mutual search algorithms we derive a lower bound of $\frac{k-1}{k+1}(n+1)$ expected calls in the worst case. A randomised algorithm in the shared-coins model matching this bound is also presented. Finally we show that in general more adaptive randomised mutual algorithms perform better (using $k-1+\left(\frac{k-1}{k+1}-\frac{k-2}{n}\right)(n-k)$ worst case expected calls in the shared coins model) than the lower bound for the restricted case, even when given only private coins. A lower bound of $k-1+\frac{n-k}{k+1}$ for this case is also derived.

1 Introduction

Buhrman *et al.* [BFG$^+$99] introduce the mutual search problem, where k agents distributed over a complete network of n distinct nodes are required to learn each other's location. Agents can do so by calling a single node at a time to determine whether it is occupied by another agent. The object is to find all other agents in as few calls as possible. Buhrman *et al.* study this problem extensively for synchronous and asynchronous networks, and both in the deterministic and randomised case, predominantly for $k = 2$ agents.

The prime motivation for studying this problem is the cost of conspiracy start-up in secure multi-party computations, or Byzantine agreement problems. Traditionally, this area of research assumes that all adversaries have complete knowledge of who and where they are, and that the adversaries can immediately collude to break the algorithm. The question is how hard it is to achieve this coordination, and how much information the good nodes in the system learn about the location of the adversaries during this coordination phase. For details and more examples we refer to Buhrman *et al.* [BFG$^+$99].

Lotker and Patt-Shamir [LPS99] focus on randomised solutions to the mutual search problem in the special case of $k = 2$ agents. They prove a lower bound of $\frac{n+1}{3}$ expected calls in the worst case for any synchronous randomised algorithm for mutual search, and present an algorithm achieving this bound in the shared coins model.

[⋆] Id: rnd-mutsearch.tex,v 1.12 2000/08/10 07:04:13 hoepman Exp

This paper is the first to consider the general case of $k < n$ agents for randomised solutions. We generalise the results of Lotker and Patt-Shamir to a lower bound of[1]

$$\frac{k-1}{k+1}(n+1) = k - 1 + \frac{k-1}{k+1}(n-k)$$

expected calls in the worst case for a restricted class of *linear* synchronous randomised algorithms (see Section 2 for an explanation of this term) that only depend on the calling history in a limited fashion. We also present an algorithm achieving this bound, in the shared coins model.

Compared to the upper bound of Buhrman *et al.* [BFG+99] of

$$\frac{k(k-1)}{k(k-1)+1}n$$

for the deterministic case, we see that our randomised algorithm can handle roughly the square of the number of agents at the same or less cost.

Moreover, we show that more adaptive randomised mutual search algorithms outperform linear algorithms, even when the nodes are given access to a private coin only. Using shared coins we obtain a randomised algorithm whose worst case expected cost equals

$$k - 1 + \left(\frac{k-1}{k+1} - \frac{k-2}{n}\right)(n-k) \ .$$

For the private coins model, and in the particular case where $k = n - 1$, we present a randomised algorithms whose worst case expected cost equals $k - 1 + \frac{1}{2}$.

Finally, we derive a lower bound of

$$k - 1 + \frac{n-k}{k+1}$$

expected calls in the worst case for any randomised mutual search algorithm.

For $k > 2$, there are basically two choices on how to proceed when two agents find each other. They either just record each other's location and proceed to find each other independently, or they 'merge' into a single node where a single master agent in the merger is responsible for making all calls, and a call reaching any agent in the merged set discovers all agents in the set. Our results hold in the latter, merging agents, model. Lower bounds in this model also hold for the independent agents model.

The paper is organised as follows. In Section 2 we extend the notion of a mutual search algorithm to $k > 2$ nodes. We then present our results on linear algorithms in Section 3. This includes the upper bound in Section 3.1, a discussion on linear algorithms in Section 3.2, and the lower bound in Section 3.3.

[1] The second formulation of the bound visually separates the minimal number of successful calls $k-1$ from the fraction of unsuccessful calls among the $n-k$ unoccupied nodes.

Next, we present the upper bounds for unrestricted mutual search algorithms in Section 4, both in the shared coins model (Section 4.1) and the private coins model (Section 4.2). Section 4.3 contains the lower bound for the unrestricted case. We conclude with some directions for further research.

2 Preliminaries

Let V be a set of n labelled nodes, some of which are occupied by an agent. The set of nodes occupied by an agent is called an *instance* of the mutual search problem, and is represented by a subset I of the nodes V. We use $k = |I|$ for the number of agents. Agents are anonymous, and are only identifiable by the node on which they reside.

The nodes V lie on a completely connected graph, such that each agent can call every other node to determine whether it is occupied by another agent. If so, such a call is *successful*, otherwise it is *unsuccessful*. Nodes not occupied by agents are *passive*: they cannot make calls, and cannot store information to pass from one calling agent to the next.

Initially each agent only knows the size and labelling of the graph, the total number of agents on the graph, and the label of the node it occupies. The mutual search problem requires all agents on the graph to learn each other's location, using as few calls as possible (counting both successful and unsuccessful calls). A mutual search algorithm is procedure that tells each agent which nodes to call, such that all agents are guaranteed to find each other. We only consider synchronous algorithms where in each time slot t a single agent makes a call. For each agent, the decision whether to make a call, and if so, which node, depends on the time slot t, the label of the node on which it resides, and on the result of the previous calls made and received.

With each successful call, agents are assumed to exchange everything they know so far about the distribution of agents over the graph. We distinguish two models. In the *independent agents* model, each agent is responsible to learn the location of all other agents on its own (of course using all information exchanged with a successful call). In the *merging agents* model, a successful call transfers the responsibility to find the remaining agents to the agent called (or the agent responsible for making calls on this agent's behalf). In this model the mutual search algorithm conceptually starts with k *clans* (or equivalence classes [BFG$^+$99]) each containing a single agent. Each successful call merges two clans into a single clan with a single *leader*, until a single clan with all k agents remains. Information flow within a clan is for free: all agents in a clan implicitly learn the location of newfound agents. As the latter model is potentially more efficient, we only consider the merging agents model. Note that even if $k = n - 1$ all agents need to be contacted to pass knowledge of the one non-agent, and so the cost is larger than or equal to $n - 2$. For $n = k$ the cost drops to 0, because agents know n and k.

For $k = 2$, a deterministic synchronous mutual search algorithm can be represented by a list of directed edges E. Edge $\overrightarrow{vw}$ is at position t in the list if an agent on node v needs to query node w at time slot t. The order in the list is represented by $\prec$. We will call this list the *call list*. The actual calls made will depend on the distribution of the agents over the nodes. In this setting, $A = (V, E, \prec)$ completely describes a synchronous deterministic mutual search algorithm for $k = 2$ agents. In fact, because the algorithm needs to make sure that any pair of agents can find each other, the graph (V, E) is a tournament.

For $k > 2$ the picture is much less clear. As an agent learns the location of some of the other agents, it may adapt itself by changing the order of future calls, or dropping certain calls altogether. One way of describing such a mutual search algorithm would be through a *call tree* instead of a call list, where each vertex represents a potential call. At the root is the first potential call to be made (provided the calling node is occupied by an agent, of course). Each vertex has two children: one for the next potential call if this call was unsuccessful or if this node was not occupied by an agent, and one for the next potential call if this call was successful.

Another way of describing such a mutual search algorithm would be to divide each run of the algorithm on a particular instance I in rounds. Whenever the algorithm makes a successful call (finding another agent or clan and merging the clans), a new round starts. The algorithm starts in round 1 to find the first pair of agents, and stops after the $k - 1$-th round when all k agents are merged into one[2]. Round boundaries represent knowledge changes. At such a boundary, the algorithm has to decide on a new strategy to find the next agent, and has to stick to this strategy until it finds it (or gets found). This strategy change cannot be global: only merged nodes can change their strategy, because they are the only nodes that are aware of the merge. Also, this strategy change can only depend on the agents found so far.

Such a strategy is again a call list[3]. For an algorithm A running on instance I, we write $E_A^i(I)$ for the call list to be used in the i-th round. A call is again a directed edge $\overrightarrow{vw}$, where we require that only leaders of a clan make calls. This list in fact only depends on the first i successful calls made, such that for any two instances I and J where the same i successful calls are made, $E_A^i(I) = E_A^i(J)$. For each clan C we write $E_A^i(I, C)$ for the call-list of the leader of the clan at the start at round i. This is the list of calls this agent will make until it makes a successful call or any agent in its' clan gets called. For a call list E we write $W(E) = \{w \mid \overrightarrow{vw} \in E\}$ for the set of destinations called by E. For a call list E and call f we write $E \upharpoonleft f = \{e \in E \mid f \prec e\}$ for the set of edges in E following e in the total order of edges $\prec$ (defined by the time slots in which these (potential) calls have to be made).

[2] We assume that no algorithm makes calls between agents in the same clan (that already know each other's location). Therefore the graph of successful calls is acyclic, and spans all k agents when $k - 1$ successful calls have been made.

[3] We allow call lists where certain positions are empty, representing that in that time slot no call will be made. This may be necessary to satisfy the requirement that the algorithm as a whole never makes more than one call in a single time slot

The class of linear mutual search algorithms defined below includes algorithms which are only marginally adaptive, but also *forward calling* algorithms that merge agents one by one into one single growing clan.

Definition 2.1. *A mutual search algorithm A to find k agents is* linear*, if for all instances I* of size $k+1$

- *A makes at least $k-1$ successful calls*[4] *when started on instance I, and*
- *for the first $k-1$ successful calls $\overrightarrow{vw}$ made by A when started on instance I, A still makes a (now unsuccessful) call to w when started on instance $I - \{w\}$.*

This topic is discussed in Sect. 3.2. We note that any mutual search algorithm for $k = 2$ agents is always linear.

3 Bounds on linear algorithms

In this section we prove a lower bound of

$$\frac{k-1}{k+1}(n+1)$$

on the worst case expected number of calls made by any randomised mutual search algorithm to find k agents among $n > k$ nodes, and match this with a randomised algorithm (in the shared coins model) achieving this bound. We present the algorithm first, and then prove the lower bound.

3.1 Upper bound

The upper bound is proven using the following straightforward algorithm.

Algorithm 3.1 *First all nodes relabel the nodes according to a shared random permutation of $\{0, \ldots, n-1\}$. The algorithm then proceeds in synchronous pulses (starting with pulse 0).*

An agent on node i (after relabelling) is quiet upto pulse i. If it does not receive a call at pulse $i-1$ (or if $i = 0$), in pulses $i, i+1, \ldots$ it calls nodes $i+1, i+2, \ldots$ until it has contacted all $k-1$ other agents. If a node i receives a call at pulse $i-1$, it remains quiet for all future pulses.

Using this algorithm the agent on the first node (after relabelling) of the instance will call all other nodes. All other agents will remain silent. We note that the algorithm is linear.

In our proof of the upper bound, we will use the following properties of binomials.

[4] It is not immediately obvious that an algorithm to find k agents when started on an instance of $k' > k$ agents will always run until it has made $k-1$ successful calls.

Property 3.2. We have

$$c\binom{c-1}{k-2} = (k-1)\binom{c}{k-1} \ . \tag{1}$$

From Feller [Fel57], p. 64, we get

$$\sum_{v=0}^{r}\binom{v+k-1}{k-1} = \binom{r+k}{k} \ . \tag{2}$$

Using this equation we derive

$$\sum_{c=k-1}^{n-1}\binom{c}{k-1} = \binom{n}{k} \ . \tag{3}$$

Combining Equation (1) and (3) we conclude

$$\sum_{c=k-1}^{n-1} c\binom{c-1}{k-2} = (k-1)\binom{n}{k} \ . \tag{4}$$

We are now ready to prove the upper bound.

Theorem 3.3. *Algorithm 3.1 has worst case expected cost*

$$\frac{k-1}{k+1}(n+1) \ .$$

Proof. Let I be a distribution of k agents over the n nodes. By performing the relabelling, the algorithm R in effect performs input-randomisation and selects a random instance I' after which it continues deterministically performing $c(I')$ calls. Note that in pulse i, R calls $i+1$ if and only if there is an agent at some $j < i+1$ and not all agents have been found yet. We conclude that $c(I')$ equals the distance between the first and last node in I'.

The expected cost of any instance I is given by

$$\sum_{|I'|=k} c(I')\Pr[\mathcal{I} = I'] \ .$$

Splitting into instances with equal cost, this is equal to

$$\binom{n}{k}^{-1} \sum_{c} c \times \text{the number of instances } I' \text{ with } |I'| = k \text{ and cost } c \ . \tag{5}$$

The minimal cost is $k-1$, the maximal cost is $n-1$. For a given cost c, the range $0, \ldots, n-c-1$ of $n-c$ nodes is a viable location f for the first agent in the instance. The last agent then resides at $f+c$ (or else the cost would not equal c).

The remaining $k-2$ agents can be distributed over the range $f+1,\dots,f+c-1$ of $c-1$ nodes. This shows that Equation (5) equals

$$
\begin{aligned}
&\binom{n}{k}^{-1} \sum_{c=k-1}^{n-1} c(n-c)\binom{c-1}{k-2} \\
&= \binom{n}{k}^{-1} \left(\sum_{c=k-1}^{n-1} cn\binom{c-1}{k-2} - \sum_{c=k-1}^{n-1} c^2\binom{c-1}{k-2} \right) \\
&\{\text{Using Equation (4) and (1). }\} \\
&= \binom{n}{k}^{-1} \left(n(k-1)\binom{n}{k} - (k-1)\sum_{c=k-1}^{n-1} c\binom{c}{k-1} \right) \\
&= \binom{n}{k}^{-1} \left(n(k-1)\binom{n}{k} \right. \\
&\quad \left. - (k-1)\sum_{c=k-1}^{n-1} (c+1)\binom{c}{k-1} + (k-1)\sum_{c=k-1}^{n-1} \binom{c}{k-1} \right) \\
&\{\text{Using Equation (1).}\} \\
&= \binom{n}{k}^{-1} \left(n(k-1)\binom{n}{k} \right. \\
&\quad \left. -(k-1)\sum_{c=k-1}^{n-1} k\binom{c+1}{k} + (k-1)\sum_{c=k-1}^{n-1} \binom{c}{k-1} \right) \\
&\{\text{Using Equation (3).}\} \\
&= \binom{n}{k}^{-1} \left(n(k-1)\binom{n}{k} - (k-1)k\binom{n+1}{k+1} + (k-1)\binom{n}{k} \right) \\
&= n(k-1) - \frac{k(k-1)(n+1)}{k+1} + (k-1) \\
&= \frac{k-1}{k+1}(n+1)
\end{aligned}
$$

This completes the proof. □

3.2 Linear Algorithms

In this section we give a characterisation of the class of linear mutual search algorithms as defined by Definition 2.1. This includes two important classes of mutual search algorithms defined next.

Definition 3.4. *A mutual search algorithm A is* non-adaptive, *if for all instances I and for each successful call f between clan C and C' at the end of round i the following holds.*

- *The call list for round $i+1$ of the merged clan $C \cup C'$ equals the union of the call lists for round i of the clans C and C' restricted to the calls ordered after the successful call f*

- *In the resulting call list, for each call the caller is replaced with the new leader of the clan $C \cup C'$.*
- *From this list all calls to nodes already called by one of the clans are removed, as well as duplicate calls to the same destination, in which case the first call (according to $\prec$) is retained.*

Definition 3.5. *A mutual search algorithm A is* forward-calling*, if for all instances I and for all successful calls $\overrightarrow{vw}, \overrightarrow{xy}$ made by A on instance I*

$$w \neq y \quad \text{and} \quad w = x \quad \text{implies} \quad \overrightarrow{vw} \prec \overrightarrow{xy} \; .$$

We have the following characterisation of linear mutual search algorithms.

Lemma 3.6. *Both non-adaptive and forward calling mutual search algorithms are linear.*

Proof. Observe that for a forward calling algorithm started on an instance I, each successful call of a clan reaches a clan containing a single member agent. Therefore, a forward calling algorithm grows a single clan that contains i agents during round i. If I contains $k + 1$ agents, A stops at round k when the clan contains k agents. So A makes $k - 1$ successful calls.

If A calls w in the course of the algorithm, w is the single member of its own clan. Hence w made no successful calls and was not contacted by another agent up till now. Hence removing the agent from w does not affect the strategy of A up to this point, and hence A will call w.

Observe that for a non adaptive algorithm, a clan C calls a node w iff there is a node v in the clan containing an agent for which $\overrightarrow{vw}$ is on its initial call list (at the very start of the algorithm). If we remove the agent from v, all other agents in C will call only a subset of the nodes called by C originally, and will do so no sooner than C would have done.

Hence if for an instance I of size $k + 1$ two clans C, C' merge at some time t by the call $\overrightarrow{vw}$, removing the agent from node w in C' will not result in any calls to C from the remaining agents in C' before time t. Hence C cannot distinguish these different instances, and will make the same calls (including the call to w).

Moreover, any clan formed during a run on an instance I of size $k + 1$ is a possible clan during a run over an instance of size k, unless that clan contains $k + 1$ agents. Hence A will stop if it finds at least k agents in a single clan, in which case it has made at least $k - 1$ calls. □

3.3 Lower bound

We now proceed to prove a lower bound on linear mutual search algorithms. Fix n and $k < n$. Let A be a linear deterministic mutual search algorithm with $|V| = n$ to locate k agents. This fixes the call lists $E^i_A(I)$ for all instances I and rounds $i < k$.

Definition 3.7. *Let I be an instance, and let $v, w \in V$. Define*

$$\Phi_A(\overrightarrow{vw}, I) = 1$$

if and only if $\overrightarrow{vw}$ is an unsuccessful call (i.e. $v \in I, w \notin I$) made by algorithm A to locate all agents in the instance I.

Then for the total cost $C_A(I)$ of algorithm A on instance I we have

$$C_A(I) = k - 1 + \sum_{\substack{v \in I \\ w \in V - I}} \Phi_A(\overrightarrow{vw}, I) \ , \tag{6}$$

where the $k - 1$ term is contributed by all successful calls made.

Lemma 3.8. *Let J be an arbitrary set of $k + 1 \leq n$ nodes. Let A be linear. Then*

$$\sum_{\substack{v,w \in J \\ v \neq w}} \Phi_A(\overrightarrow{vw}, J - \{w\}) \geq k - 1 \ . \tag{7}$$

Proof. Run A on instance J. By Definition 2.1, A will have made at least $k - 1$ successful calls. Take the first $k - 1$ successful calls. By Def. 2.1, for each of the first $k - 1$ successful calls $\overrightarrow{vw}$, removing w does not affect the fact that v calls it. Hence $\Phi_A(\overrightarrow{vw}, J - \{w\}) = 1$. Hence the sum is at least $k - 1$. □

This lemma allows us to give a bound on the expected cost of a random instance for a given algorithm A.

Lemma 3.9. *Let A be a linear deterministic mutual search algorithm for k agents over $n > k$ nodes. Then the expected cost* $\mathrm{E}[C_A(\mathcal{I})]$ *of a random instance $\mathcal{I} = I$ of k agents is bounded by*

$$\mathrm{E}[C_A(\mathcal{I})] \geq \frac{k-1}{k+1}(n+1) \ . \tag{8}$$

Proof.

$$\begin{aligned}
\mathrm{E}[C_A(\mathcal{I})] &= \sum_{\substack{I \subset V \\ |I| = k}} C_A(I) \, \Pr[\mathcal{I} = I] \\
&= \sum_{\substack{I \subset V \\ |I| = k}} C_A(I) \binom{n}{k}^{-1} \\
&\quad \text{\{By Equation (6).\}} \\
&= \binom{n}{k}^{-1} \sum_{\substack{I \subset V \\ |I| = k}} \left(k - 1 + \sum_{\substack{v \in I \\ w \in V - I}} \Phi_A(\overrightarrow{vw}, I) \right) \\
&= k - 1 + \binom{n}{k}^{-1} \sum_{\substack{I \subset V \\ |I| = k}} \sum_{\substack{v \in I \\ w \in V - I}} \Phi_A(\overrightarrow{vw}, I)
\end{aligned}$$

$$
\begin{aligned}
&\{\text{Rearranging sums, setting } J = I + \{w\}.\} \\
&= k - 1 + \binom{n}{k}^{-1} \sum_{\substack{J \subset V \\ |J| = k+1}} \ \sum_{\substack{v,w \in J \\ v \neq w}} \Phi_A(\overrightarrow{vw}, J - \{w\}) \\
&\{\text{By Lemma 3.8.}\} \\
&\geq k - 1 + \binom{n}{k}^{-1} \binom{n}{k+1} (k-1) \\
&= \frac{k-1}{k+1}(n+1)
\end{aligned}
$$

□

Following Yao [Yao86], by the von Neumann's minimax principle, we have the following theorem.

Theorem 3.10. *The maximal expected cost of any randomised linear mutual search algorithm over n nodes on some instance of size k is at least*

$$\frac{k-1}{k+1}(n+1) \ .$$

Proof. We first compute the expected cost of a randomised linear algorithm R on a random instance of size k of the mutual search problem over n nodes. Because the random coin flips used by R are independent of the choice of the instance we can first condition on the contents of the random tape. This fixes R, in effect making it a deterministic linear algorithm A. The expected cost over a random instance is now given by Lemma 3.9 to be at least

$$\mathrm{E}[C_A(\mathcal{I})] \geq \frac{k-1}{k+1}(n+1) \ .$$

As this is the expected cost of R on a random instance, there must be an instance for which the expected cost is at least this high. □

4 Unrestricted algorithms

In this section we investigate the power of more adaptive algorithms whose call patterns depend heavily on the presence or absence of earlier successful calls. We show that these algorithms perform better than the lower bound for linear algorithms, given either a shared coin or a private coin.

4.1 Shared coins

First we restrict our attention to algorithms deploying a shared random coin. This shared coin can be used to perform a global relabelling of nodes, as explained in section 3.1.

The following algorithm uses

$$k - 1 + \left(\frac{k-1}{k+1} - \frac{k-2}{n}\right)(n - k)$$

expected calls in the worst case. In this algorithm, all agents call node 0, unless node 0 does not contain an agent, in which case the first agent that discovers this calls all other nodes until all remaining agents are found (similar to Algorithm 3.1).

Algorithm 4.1 *Globally relabel the nodes using the shared random coin. Agents call other nodes depending on the pulse number (starting with pulse* 0*) as follows.*

An agent on node 0 *(after relabelling) does not make any calls.*

An agent on node $i > 0$ (after relabelling) is quiet upto pulse $2i - 1$. If it did not receive any calls at earlier pulses, on pulse $2i - 1$ it calls node 0*. If this call is successful, it makes no more calls. Otherwise, at pulse $2i, 2i + 1, \ldots$ it calls nodes $i + 1, i + 2, \ldots$ until it has contacted all remaining $k - 1$ agents. If an agent on node $i > 0$ receives a call before pulse $2i - 1$, it remains quiet for all future pulses.*

Theorem 4.2. *Algorithm 4.1 is a mutual search algorithm for k agents on n nodes, making at most $k - 1 + \left(\frac{k-1}{k+1} - \frac{k-2}{n}\right)(n - k)$ expected calls in the worst case.*

Proof. We split the proof into two cases.

an agent at node 0**:** This case occurs with probability k/n. Here, all remaining $k - 1$ agents call node 0, so that $k - 1$ calls are made in total, and all agents are found.

no agent at node 0**:** This case occurs with probability $(n-k)/n$. The first agent at node $i > 0$ that discovers this case in phase $2i - 1$, will call all remaining agents at nodes $j > i$. In fact, in this case the k agents run algorithm 3.1 on $n - 1$ nodes. Hence the worst case expected cost is given by Theorem 3.3 as $\frac{k-1}{k+1}n$. We have to add 1 for the extra call to node 0 to arrive at the total cost in this case.

The worst case expected cost is therefore given by

$$\frac{k}{n}(k-1) + \frac{n-k}{n}\left(1 + \frac{k-1}{k+1}n\right) = k - 1 + \left(\frac{k-1}{k+1} - \frac{k-2}{n}\right)(n - k)$$

This completes the proof. □

Note that the bound of Sect. 3 can be rewritten to $k-1+\frac{k-1}{k+1}(n-k)$. Algorithm 4.1 improves this bound by $\frac{k-2}{n}(n - k)$.

4.2 Private coins

Using private coins the agents have only limited possibilities to counter bad node assignments made by the adversary. Global relabelling is not possible, for instance, because the agents cannot exchange the outcome of their private coin tosses.

We again consider the special case of $k = n - 1$, where all nodes except one are occupied by an agent. Here, the following algorithm uses

$$k - 1 + \frac{1}{2}$$

expected calls in the worst case.

Algorithm 4.3 *This algorithm runs as follows.*

round 1**:** *If there is an agent at node* 0*, it either calls node* 1 *or node* 2*, each with probability* 1/2*.*
round 2**:** *If the previous call was unsuccessful, the other node is called.*
round 3**:** *If there is an agent at node* 2*, which did not receive a call from node* 0*, it calls node* 1*.*
round 4**:** *If there is an agent at node* 1*, which did not receive any calls (either from node* 0 *or node* 2*), it calls node* 2*.*
round $2 + i$**,** $3 \leq i \leq n - 1$**:** *If there is an agent at node* 1*, and node* 1 *did not receive a call from node* 0 *or was called by node* 2 *it calls node* i *at round* i*.*
round $n - 1 + j$**,** $3 \leq j \leq n - 1$**:** *If there is an agent at node* j*, and this node did not receive a call from node* 1*, then it calls node* 0*.*

Theorem 4.4. *Algorithm 4.3 is a mutual search algorithm for* $k = n - 1$ *agents on* n *nodes, making* $k - 1 + \frac{1}{2}$ *expected calls in the worst case.*

Proof. By case analysis, using the fact that all but one node contain an agent.

no agent at node $i > 2$**:** The agent at node 0 either calls node 1 (and then node 2 calls node 1 at round 3) or node 2 (and then node 1 calls node 2 at round 4). No calls are made at round $2 + i$ for $3 \leq i \leq n - 1$, and all agents on nodes $3 \leq j \leq n - 1$ call node 0 at round $n - 1 + j$. One of these nodes is not occupied by an agent. In this case, $k - 1$ calls are made in total.
no agent at node 0**:** The agent at node 2 calls node 1 at round 3. Hence the agent at node 1 calls the agents at node i for $3 \leq i \leq n - 1$ at round $2 + i$. Again, $k - 1$ calls are made in total.
no agent at node 1**:** With probability 1/2, the agent at node 0 calls node 2. No other calls are made in round $2, \ldots, n+1$. All agents on nodes $3 \leq j \leq n-1$ call node 0 at round $n - 1 + j$. Again $k - 1$ calls are made in total.
With probability 1/2, the agent at node 0 calls node 1. Because this call is unsuccessful, it also calls node 2 at round 2. No other calls are made in round $3, \ldots, n + 1$. All agents on nodes $3 \leq j \leq n - 1$ call node 0 at round $n - 1 + j$. Now k calls are made in total.
no agent at node 2**:** Similar to the previous case.

The worst case occurs if there is no agent at node 1 or node 2. In either case the expected number of calls made equals $\frac{1}{2}k + \frac{1}{2}(k - 1) = k - 1 + \frac{1}{2}$. This completes the proof. □

4.3 Lower bound

Using the results of Section 3.3 we now derive a lower bound of

$$k - 1 + \frac{n-k}{k+1}$$

expected calls in the worst case for arbitrary randomised mutual search algorithms. We use the same notational conventions and definitions as used in that section.

Lemma 4.5. *Let J be an arbitrary set of $k+1 \leq n$ nodes. Let A be an arbitrary deterministic mutual search algorithm Then*

$$\sum_{\substack{v,w \in J \\ v \neq w}} \Phi_A(\overrightarrow{vw}, J - \{w\}) \geq 1 \ . \tag{9}$$

Proof. Run A on instance J. Because A is a mutual search algorithm, it will make at least one successful call. Let $\overrightarrow{vw}$ be the first successful call. Now remove w from J, and run A on $J - \{w\}$. As discussed in Section 2, the call pattern of a node depends on its initial state and all calls made so far. For $u \neq w$, the initial state does not change. Moreover, any calls before the call from v to w must, by assumption, be unsuccessful. Removing w does not influence the result of any of these unsuccessful call (they do not involve w). Removing w only removes any unsuccessful calls made by w from the call pattern. They also have no effect on the call patterns of the other nodes. Hence all $u \neq w$ make the same calls as before upto the call to w. Hence v calls w, which is now unsuccessful. □

Theorem 4.6. *The maximal expected cost of any randomised mutual search algorithm over n nodes on some instance of size k is at least*

$$k - 1 + \frac{n-k}{k+1}$$

Proof. Using the same steps of the proof of Lemma 3.9 and using Lemma 4.5 we get

$$\begin{aligned} \mathrm{E}[C_A(\mathcal{I})] &= k - 1 + \binom{n}{k}^{-1} \sum_{\substack{J \subset V \\ |J| = k+1}} \sum_{\substack{v,w \in J \\ v \neq w}} \Phi_A(\overrightarrow{vw}, J - \{w\}) \\ &\quad \{\text{By Lemma 4.5.}\} \\ &\geq k - 1 + \binom{n}{k}^{-1} \binom{n}{k+1} \\ &= k - 1 + \frac{n-k}{k+1} \ . \end{aligned}$$

The theorem follows from this fact by the same reasoning as used in the proof of Theorem 3.10. □

5 Further research

We have investigated the mutual search problem in the randomised case with $k > 2$ agents. The main questions remaining are the following.

First of all, we would like to derive efficient adaptive algorithms for an arbitrary number of agents in the private coins model. Moreover, we would like to close the gap between the lower bound and the upper bound in the shared coins model for arbitrary, non-linear, mutual search algorithms.

Secondly, the question is how changes to the model (different graphs, non-anonymous agents, independent agents, etc.) affect the cost of mutual search.

Finally, time constraints may have an adverse effect on the cost of a mutual search algorithm. If a synchronous mutual search algorithm is required to terminate within time t, with full knowledge of the distribution of the agents, the algorithm may have to make more than one call per time slot, and may not be able to fully exploit the “silence is information” paradigm.

References

[BFG+99] Buhrman, H., Franklin, M., Garay, J., Hoepman, J.-H., Tromp, J., and Vitányi, P. Mutual search. *J. ACM* **46**, 4 (1999), 517–536.

[Fel57] Feller, W. *An Introduction to Probability Theory and Its Applications*, 2nd ed. Wiley & Sons, New York, 1957.

[LPS99] Lotker, Z., and Patt-Shamir, B. A note on randomized mutual search. *IPL* **71**, 5–6 (1999), 187–192.

[Yao86] Yao, A. Probabilistic computations: toward a unified measure of complexity. In *18th STOC* (Berkeley, CA, USA, 1986), ACM Press, pp. 222–227.