Treasure Hunt with Barely Communicating Agents*

Stefan Dobrev[1], Rastislav Královič[2], and Dana Pardubská[3]

1 Slovak Academy of Sciences, Bratislava, Slovakia
2 Comenius University, Bratislava, Slovakia
3 Comenius University, Bratislava, Slovakia

Abstract

We consider the problem of fault-tolerant parallel exhaustive search, a.k.a. "Treasure Hunt", introduced by Fraigniaud, Korman and Rodeh in [13]: Imagine an infinite list of "boxes", one of which contains a "treasure". The ordering of the boxes reflects the importance of finding the treasure in a given box. There are k agents, whose goal is to locate the treasure in the least amount of time. The system is synchronous; at every step, an agent can "open" a box and see whether the treasure is there. The hunt finishes when the first agent locates the treasure.

The original paper [13] considers non-cooperating randomized agents, out of which at most f can fail, with the failure pattern determined by an adversary. In this paper, we consider deterministic agents and investigate two failure models: The failing-agents model from [13] and a "black hole" model: At most f boxes contain "black holes", placed by the adversary. When an agent opens a box containing a black hole, the agent disappears without an observable trace.

The crucial distinction, however, is that we consider "barely communicating" or "indirectly weakly communicating" agents: When an agent opens a box, it can tell whether the box has been previously opened. There are no other means of direct or indirect communication between the agents.

We show that adding even such weak means of communication has very strong impact on the solvability and complexity of the Treasure Hunt problem. In particular, in the failing agents model it allows the agents to be 1-competitive w.r.t. an optimal algorithm which does not know the location of the treasure, but is instantly notified of agent failures. In the black holes model (where there is no deterministic solution for non-communicating agents even in the presence of a single black hole) we show a lower bound of $2f+1$ and an upper bound of $4f+1$ for the number of agents needed to solve Treasure Hunt in presence of up to f black holes, as well as partial results about the hunt time in the presence of few black holes.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases parallel exhaustive search, treasure hunt, fault-tolerant search, weak coordination, black holes

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2017.14

1 Introduction

Searching is one of the most studied problems in computer science. As the search space is often huge, in order to speed up the search time a frequently used approach is to resort to parallel and/or distributed search. However, employing a large number of parallel searching agents gives rise to two important issues:

* This work has been supported by grant VEGA 2/0165/16.

- how to efficiently coordinate them, so that the agents complement each other's work instead of being an obstacle to each other
- how to deal with the inevitable failures, so that the search is successfully and efficiently completed and no area of search space is omitted.

On one hand, a tight coordination is desired, so that the agents can efficiently split their workload and react to behaviour/observations of other agents. On the other hand, the coordination itself incurs costs that might overwhelm any gains obtained by using more agents. This has motivated research into non-coordinated and weakly coordinated search agents, as lack of coordination brings inherent robustness and resilience to failures and tampering.

There are numerous variants of the search problem, differing in the structure of the search space, the power and the number of search agents, the communication mechanisms available to them and in the search space itself, the failure models, termination conditions as well as cost and payoff functions.

The particular variant of the search problem we are interested in has been introduced by Fraigniaud, Korman and Rodeh in [13]. It can be seen as an abstraction of a distributed exhaustive search as employed by BOINC [2] (for Berkeley Open Infrastructure for Network Computing), made famous for SETI@home. The goal is to exhaustively search the search space, until a desired item ("treasure", e.g. SETI signal, cryptographic key, . . .) has been found. The search space is linear in nature, in the sense that there is a total ordering of the candidate solutions: given that a solution has not been found among the smaller candidates, the next candidate contains the most preferred solution, is the most likely to contain a solution, or is the least costly to check.

In this setting, a central server controls and distributes the work to volunteers. While typically there is a strong control over work tasks (which can be made of desired granularity and uniform size), there is very little control over the volunteer agents executing these work tasks. Therefore, it is highly desirable (for practical, security and anonymity reasons) to have no direct communication channels between the worker agents. Furthermore, for legacy and efficient implementation reasons, it is unfeasible to add environmental communication channels (e.g. whiteboards at the nodes of the search space). Hence, non-coordinated protocols are a very good match. It has been shown in [13] that by employing randomized agents, the cost of non coordination can be as low as a factor of 4.

Still, in this setting, there is a basic coordination mechanism which is always present, by the very nature of the setting: For each work task, the server knows whether it has been already solved or not. The central idea of this paper is to investigate how much can this essentially free information help in efficiently solving the treasure hunt problem.

2 Model, Outline and Related Work

Consider a Treasure Hunt (parallel linear search) problem: Given is a (potentially infinite) sequence of boxes, one of which contains a treasure. There are k deterministic agents in the system, each with its own unique ID from $\{1, \ldots, k\}$ [1]. Apart from the IDs, the agents are

[1] In fact, it is sufficient that the IDs are unique, and from a range $\{1, \ldots, \text{MAXID}\}$, where MAXID is an a-priori known upper bound on a maximal ID. A preprocessing phase of cost $O(f \log k \log \text{MAXID})$ (where f is an upper bound on the number of failures/black holes) can compact the IDs into the desired range, using standard parallel processing techniques. In order not to detract from the main focus of our paper, we prefer to assume this has already been done prior to commencing the treasure hunt.

identical, and have no means of communication. The system is synchronous and in each step each agent may open one box to check whether it contains the treasure. We consider only non-conflicting schedules[2], i.e. schedules with the property that, during any time step, each box is being accessed by at most one agent. When the box containing the treasure is opened, the hunt is finished. The goal is to minimize the hunt time.

In a reliable system, k agents can locate the treasure in $\lceil x/k \rceil$ steps, where x is the (unknown) location of the treasure: in i-th step, the agent j simply opens the box $k(i-1)+j$. Let us define the *speedup* as $\liminf_{x\mapsto\infty} x/T(k,x)$ where $T(k,x)$ is the hunt time with k agents, if the treasure is in box x. Hence, in a reliable system, k agents achieve a speedup of k. However, the just mentioned simple algorithm is very sensitive to errors: a failure of a single agent may lead to the treasure not being found at all. In this paper we study faulty systems; in particular we investigate treasure hunt under two failure models:

- in the *failing agents* model, also used in [13], at most $f < k$ agents may fail at any time during the computation. The *failure pattern* describing at what times which agents fail is chosen by an adversary.
- in the *black holes* model, we assume that at most f of the boxes contain a black hole. Whenever an agent opens a box with a black hole, the agent disappears without any trace. Again, the locations of the black holes are chosen by the adversary.

Consider first the failing agents model. Denote $T_f(k,x)$ the hunt time with k agents, taken as the worst case over all possible failure patterns in which up to f agents fail. Clearly, $f = k-1$ failing agents may be tolerated by a trivial algorithm in which every agent performs a sequential scan, having $T_f(k,x) = x + k$ [3]. However, this algorithm achieves only speedup of 1. Moreover, for $f = k-1$ essentially nothing better can be done, since the best achievable speedup for f failing agents is $k/(f+1)$ (see Claim 3.3).

The situation is even more bleak in the *black hole* model. Here even a single black hole makes the problem unsolvable in the deterministic setting (Claim 3.5). The proof of this heavily relies on the fact that the number of boxes is unbounded. If that is not true, the problem becomes solvable – Corollary 4 shows that treasure hunt can be solved by $O(\sqrt{n}\ln n)$ agents, where n is the number of boxes. The situation in randomized setting remains open.

The main contribution of this paper is to show that even a very limited means of communication have great impact on the solvability of the problem. The sole means of communication provided by the system is the following: When an agent opens a box, it learns whether this box has been opened before.

Even though the agents have no way to directly communicate, they can deduce the presence of faults from the presence of closed boxes which should have been opened by the agents that died. However, such boxes can be identified only by opening them, thus destroying any evidence; this read-once property makes algorithms in this setting rather tricky.

As we shall see shortly in Claim 3.6, even this indirect communication channel increases the power of the system: e.g. three agents can find the treasure in the presence of one black hole. The aim of this paper is to investigate the power of this indirect communication. In particular, the two central questions we try to answer are *“How fast can barely communicating agents find the treasure in the failing agents model?”* and *“How many black holes can be tolerated in the black hole model?”*.

[2] This simplifies arguments and technical issues; as we will see, the cost is negligible.
[3] The additive term is present because in order to ensure a non-conflicting schedule, the agents start their scans one after another.

For the first question, note that no speedup is possible in any of these models: indeed, in both failure models it is possible that $k-1$ agents die right at the beginning, leaving a single deterministic agent to do essentially the whole search. That's why we turn to competitive analysis, which is a well-established tool for analyzing similar types of adversarial settings, for the measurement of time efficiency: consider a setting where the agents are immediately notified about a failure (or, in the black holes model, when an agent enters a black hole). In this setting, there is a simple optimal algorithm which in each step assigns all k' living agents to the first k' unopened boxes. We compare the hunt time of a distributed algorithm to the hunt time of this optimal algorithm. In particular, the *competitive ratio* of an algorithm with k agents is $r_k = \limsup_{x\mapsto\infty} \sup_{\mathcal{F}} T_{\mathcal{F}}(k,x)/OPT_{\mathcal{F}}(k,x)$ where $\mathcal{F}$ is the *failure pattern* describing at what times which agents fail, x is the location of the treasure, and $T_{\mathcal{F}}(k,x)$ (resp. $OPT_{\mathcal{F}}(k,x)$) is the hunt time of the algorithm (resp. of the optimal algorithm). This measure is somewhat similar to the *non-coordination ratio* of Fraigniaud, Korman, and Rodeh [13]; the difference is that the latter considers the ratio of expected speedups, and as we have seen the speedup is not useful in the deterministic case. Also note that if we fix a failure pattern in which $k-1$ agents die at the beginning, the measure becomes directly related to speedup.

2.1 Our Results

We show that two agents (with the possibility that one of them fails) can achieve hunt time $T_{\mathcal{F}}(2,x) \leq OPT_{\mathcal{F}}(2,x) + O(\sqrt{x})$ (Claim 3.4). Since $OPT_{\mathcal{F}}(2,x) \geq x/2$, the algorithm is 1-competitive. This result is generalized to k agents with the possibility that $k-1$ of them fail in Theorem 2, in which case a hunt time $T_{\mathcal{F}}(k,x) \leq OPT_{\mathcal{F}}(k,x) + O(k^3\sqrt{x})$ can be achieved, again yielding a 1-competitive algorithm.

For the systems with black holes, we show that $2f+1$ agents are needed in order to guarantee a successful treasure hunt with f black holes (Theorem 6). We provide an upper bound in Theorem 8, showing that $4f+1$ agents are sufficient. The downside of the algorithm from Theorem 8 is that if the treasure is located at box x, boxes up to $O(xf^2)$ are used, making heavy use of the fact that the number of boxes is unlimited. At the expense of using $O\left((f\ln f)^2\right)$ agents, we develop in Theorem 9 an algorithm that uses only boxes up to $x + O\left((f\ln f)^2\right)$ in the case the treasure is in box x. This algorithm can be tweaked to work also for the setting where the number of boxes is finite (by having a set of agents dedicated to exploring the last $O\left((f\ln f)^2\right)$ boxes).

2.2 Related Work

The first work evaluating the search time as a function of the location of the searched item is [4, 5]. In this paper, which became a classic of online computing, the authors studied the *cow-path* problem with a single agent, and showed that locating the treasure (gate in their case) takes time $9d$, where d is the distance from the origin. As in our model, the search space is linear in nature and the search is for a single item. However, the principal difference is that unlike in our model in which the agents have direct access to boxes and pay unit price to access any box, in [5] the agent has to arrive to the box in order to search it, paying cost d to reach a box at distance d from its current location. In fact, in [5] authors consider several variants of the search problem, including searching in 2D and searching for large objects of known shape (lines, circles) but unknown location. [3] considers parallel search of large object by two robots, while the randomized version of the original cow-path problem was studied in [16]. The generalization of the cow path problem to many agents is studied in [7], where it is shown that independently of the number of agents, the group search cannot be performed faster than in time $9d - o(d)$.

A closely related research has been focused on ANTS problem, introduced in [12] and motivated by modeling ants foraging for food. The agents (ants) are randomized non-communicating Turing machines; the goal is to locate a treasure placed adversarially at some distance d from the origin. It is shown that n ants are able to achieve speedup of n, locating the treasure in time $O(d + d^2/n)$. Further work focused on limiting the power of the agents to finite machines [11] (but adding local communication), as well as considering agent failures [20]. In [21], the trade-off between the speedup and *selection complexity* of the ANTS problem is investigated, where the selection complexity is a measure of how likely the search strategy is to evolve in a natural environment. In [8], an infinite d-dimensional grid is explored by a group of probabilistic finite automata that can communicate only in a face-to-face manner. The aim is to find a hidden treasure, and the main question is how many agents are required to guarantee a finite expected hitting time. The destructive-read aspect of opening boxes in our model corresponds precisely to the ANTS model with a single pheromone. However, the crucial difference from our model is that ants have to pay the cost of travelling to the searched location, while in our model any box can be accessed directly at a unit cost.

Another closely related area of research deals with variants of the *do-all problem*: there are p processors that must collectively perform n tasks, such that each task is performed by at least one processor. The problem has been analyzed in a number of settings: synchronous processors with crashes (and possible restarts) under various modes of communication (full knowledge, message passing, etc.), asynchronous processors with adversarially limited rendezvous communication, etc. For a survey of results see [14].

Treasure hunt as a rendezvous between a searching agent and a static one has been studied in general graphs in [25, 26], in the context of universal traversal sequences. Finite search space, paying for edge traversal and a single searching agent make these results only tangentially relevant to our work. For a survey of results about the treasure hunt and rendezvous of two agents see also the book [1].

There is a long history of searching with failures in the classical Ulam-Renyi two players search game, where the failures represent incorrectness of the received answers. See [23] for a comprehensive survey of older results, and [15] for an example of more recent result in this area. Still, these works are only remotely related to our results.

In the black hole search problem introduced in [10, 9] (see [24] for a recent survey), some nodes of the network are black holes – nodes which destroy without observable trace any agent that enters them. The goal of the agents is to explore the network and locate all black holes. As in the ANTS problem, the agents pay for moving along the network, however, the primary focus is on establishing the number of agents necessary/sufficient to locate the black holes. Another difference from our setting is that the black holes search is considered solved only when all black holes have been identified, while in treasure hunt the hunt ends when the first agent locates the treasure.

The paper motivating our work and the one with the model closest to ours is [13], in which the authors show that k randomized non-cooperating agent achieve expected speedup of $k/4 + o(1)$ in solving the linear treasure hunt problem. In a recent follow-up [18], the authors investigate a variant of the problem, where the treasure is placed uniformly at random in one of a finite, large, number of boxes. They devise a non-coordinating algorithm that achieve a speed-up of $k/3 + o(1)$. In [19], an optimal algorithm for the case when the treasure is placed randomly with a known distribution is presented, and its running time is calculated for some specific distributions. The key differences are that we consider barely communicating deterministic agents and extend the results also to the black holes model of failures.

A somewhat related paper is [6], in which a similar problem is studied w.r.t. advice complexity: Given a number of n closed boxes, an adversary hides $k < \sqrt{n}$ items, each of unit value, within k consecutive boxes. The goal is to open exactly k boxes and gain as many items as possible.

3 Results

3.1 Building Blocks

As a point of interest, observe that given a long enough sequence of boxes guaranteed to not have been opened yet, the agents may implement any fault-tolerant message passing protocol; we present a general approach here. However, directly applying this approach on standard fault-tolerant message-passing protocols would generally result in too high search complexity. In the rest of the paper we use more efficient solutions specifically tailored to the problem at hand.

Let us consider the usual synchronous setting with k processors that start with some input values, and perform an r-round protocol, where in every round each processor receives messages from all other processors, performs some local computation, and sends messages to other processors. After r rounds, each processor has some output value. Moreover, a number of processors may crash during the protocol; in particular, a processor may also crash in the middle of the sending phase, in which case the messages to an arbitrary set of processors are delivered. Suppose that we have k agents that are assigned the same input values as the processors. We have the following observation:

▶ **Claim 3.1.** *Consider a message passing protocol with k processors and r rounds, that exchanges m-bit long messages, and tolerates $2f$ processor-crashes. This protocol can be implemented by k agents in $2k(r-1)(m+1)$ steps in the presence of $2f$ agent failures (or f black holes), provided that at the beginning the agents agree on a set of $(r-1)k^2(m+1)$ unopened boxes.*

Proof. The unopened boxes will be viewed as forming an array $M[t,i,j]$, where $t = 1,\dots,r-1$ denotes the round, and $i,j = 1,\dots,k$ are a pair of agents. The entry $M[t,i,j]$ consists of $m+1$ boxes that represent a message sent from processor i to processor j in t-th round. Each round t of the protocol is simulated as follows: at the beginning each agent j checks all entries $M[t-1,i,j]$, $i = 1,\dots,k$ (if $t > 1$). If the last (i.e. $m+1$st) box from an entry is opened, it signals that the writing of $M[t-1,i,j]$ was successful, and in that case j considers the opened/unopened status of the first m boxes of $M[t-1,i,j]$ as bits of the message from i. Then agent j performs the local computation of processor j, and then writes values $M[t,j,i]$, $i = 1,\dots,k$ based on the messages sent by the protocol. The last entry of $M[t,j,i]$ is opened to signal successful write. Both reading and writing of a round require $k(m+1)$ time steps, with the exceptions that there is no read phase in round 1, and no write phase in round r. So overall, the simulation takes $2k(r-1)(m+1)$ steps. Since no messages are sent in the last round, the overall number of boxes used by the communication is $(r-1)k^2(m+1)$.

Clearly, if an agent fails, it can be interpreted as a failure of a processor in the message-passing protocol. The last bit of $M[t,i,j]$ ensures that if an agent fails in the middle of writing an entry, incorrect message will not be considered.

On the other hand, note that every box is opened by at most two agents. Hence, a black hole can kill at most two agents, which leads to a crash of at most two processors. ◀

▶ Remark. Note that in the end, some boxes of $M[t,i,j]$ may remain unopened and hence need to be explicitly "cleaned" later on.

Observe that if in each round each processor communicates with at most one counterpart, the time complexity is $2r(m+1)$. Now we sketch how the simulation of message passing mechanism from the previous protocol can be used to compact the agents' IDs from a set $S = \{1\ldots, \mathrm{MAXID}\}$ to $\{1, \ldots k\}$.

- Let T be a complete binary tree with leaves corresponding to the set S. The leaves representing the ID of some agent are *assigned* to that agent, other leaves are *free*.
- Each internal vertex is assigned to the agent with the smallest ID in its subtree, or is free, if there is no such agent. Note that this assignment can be constructed level-by-level: for each internal vertex v there are at most two candidate agents. They don't know each other IDs, but may use a pre-determined place in the scratchpad (depending only on v) to exchange messages and find out the IDs. Hence, from that point on each agent knows which vertices are assigned to it.
- Use a message passing converge-cast in the tree (each agent simulates all vertices assigned to it) to collect the number of agents to the root. Along the way, collect also the number of active agents in each left and right subtree.
- Send this information back towards the leaves, where each agent computes its rank.
- Verify that each agent has computed its rank: Agent with rank i writes in row i of a $k \times k$ scratchpad and afterwards reads column i of the scratchpad. The number of opened boxes must equal the computed number of agents.
- If the verification fails (there must have been failure during the computation), repeat the whole process again.

As the depth of the tree is $\log \mathrm{MAXID}$ and $f+1$ repetitions are sufficient to have a failure-free run, the computed values are at most k, and in each round each node communicates only with its children and parent, we obtain:

▶ **Claim 3.2.** *The processor labels can be compacted from $\{1\ldots, MAXID\}$ to $\{1, \ldots k\}$ in $O(f \log k \log MAXID)$ time steps.*

Of course, compacting the IDs opens the boxes used in the scratchpads and can mess-up a subsequent treasure hunt algorithm. In some algorithms (e.g. for Theorem 8), it is sufficient to remember where the scratchpads used in the preprocessing end, so the new scratchpads needed use unopened boxes. Otherwise, a subroutine using non-communicating agents can be used to ensure opening of all boxes located in the pre-processing's scratchpads.

In the sequel we shall use the following *coordination problem* (actually a binary consensus on whether the agent 1 is alive) as a building block in more advanced algorithms: An unknown subset of the agents is alive. There is a distinguished leader agent all alive agents agree on (w.l.o.g.[4] agent 1); however, the leader might not be alive. All agents have to agree on a common boolean value, such that if 1 was dead at the beginning of the protocol, all agents agree on `false`, and if 1 was alive and there were no faults during the protocol, all agents agree on `true`. In view of Claim 3.1, we could use a well known simple consensus protocol (see e.g. [22]) to solve the problem in $O(k^2)$ steps using $O(k^3)$ boxes. However, there is a more efficient solution:

▶ **Lemma 1.** *The coordination problem can be solved in $2k-2$ steps, provided that the agents agree on a set of $k(k-1)/2$ unopened boxes.*

[4] If the agreed-upon leader is i, the agents cyclically assign themselves virtual IDs such that i gets 1: agent $(i+j) \bmod k$ gets virtual ID of $1+j \bmod k$.

Proof. The boxes will be viewed as an upper triangular matrix $M[i, j]$, $i = 1, \dots, k; j = i + 1 \dots, k$, such that open box $M[i, j]$ is a signal from agent i to agent j that agent 1 was alive at the beginning of the protocol.

Each agent has a boolean value that is initialized to `true` for agent 1, and `false` for other agents. The algorithm for each agent consists of two phases. In the *look* phase, agent $i > 1$ in step $i + r - 1$ opens box $M[r, i]$ for $r = 1, \dots, i - 1$. If it finds the box already opened, it sets its value to `true`. Afterwards, in the *shout* phase, an agent with value `true` opens box $M[i, r]$ in steps $i - 2 + r$ for $r = i + 1, \dots, k$. Note that the agent 1 (if alive) skips the look phase and proceeds directly to the shout phase.

If agent 1 is not alive at the beginning of the protocol, by a simple minimization argument, no agent can acquire value `true`: First, note that for every box $M[i, j]$, if the agent i has value `true`, it will open $M[i, j]$ before agent j opens it. Let t be the first time when some agent i obtains value `true`. To do so, it must have opened an already opened box $M[t - i, i]$, i.e. the agent $t - i$ already had a value `true` – a contradiction.

Further, if agent 1 is alive, and there are no faults, it successfully opens all boxes $M[1, i]$, and subsequently every agent j finds an open box at $M[1, j]$, acquiring value `true`.

Finally, we argue the agreement, i.e. if there is some surviving agent that finishes with value `true`, then all surviving agents finish with value `true`. Let i be the smallest[5] surviving agent that finished with value `true`. Then i opened boxes $M[i, j]$ for $j > i$ in its shout phase, and all larger surviving agents opened those boxes in their look phases and hence gained value `true`. ◀

3.2 Hunt Time in the Failing Agents Model

First let us observe that in systems without communication, the best possible speedup is $k/(f + 1)$:

▶ **Claim 3.3.** *Consider a system with k non-communicating agents, in which at most $f < k$ agents may fail. There is an algorithm that achieves hunt time $T_f(k, x) \le x(f + 1)/k + \binom{k-1}{f}$. Moreover, for any algorithm, $T_f(k, x) \ge x(f + 1)/k$.*

Proof. For the upper bound, let $m = \binom{k}{f+1}$. We construct a schedule for m boxes and k agents, such that after $t = \binom{k-1}{f} = m(f + 1)/k$ steps, every box is opened by $f + 1$ distinct agents. The algorithm repeats this schedule in sequence for first, second, etc. m-tuple of boxes. Hence, if the treasure is somewhere in the i-th tuple (i.e. $\lceil x/m \rceil = i$), at least one agent opens the box with the treasure by the time

$$ti \le t\left(\frac{x}{m} + 1\right) = x\frac{f + 1}{k} + \binom{k - 1}{f}.$$

What remains to be shown is how to construct the schedule. Consider a bipartite graph with partitions A, B, where A contains k vertices, and B contains $\binom{k}{f+1}$ vertices, corresponding to all $f + 1$ element subsets of A. An edge connects every vertex from $v \in A$ with all vertices from B, such that v is in the corresponding set. Obviously, this graph has maximum degree $t = \binom{k-1}{f}$, and following König [17], it can be colored by t colors. This coloring is naturally interpreted as a schedule: an agent corresponding to a vertex from A opens a box corresponding to a vertex from B in time that is represented by the color of the edge.

[5] w.r.t. the virtual IDs

For the lower bound, the same argument works: let $x = \binom{k}{f+1} + 1$, and consider the first x boxes. Let $t = \binom{k-1}{f}$, and consider the actions of the k agents in a fault-free execution over the first t steps. Construct a t-colored bipartite graph of agents and boxes by connecting an agent i with box j by an edge with color c iff the agent opened the box in time step c. The graph has at most kt edges, so there must be a box $x' \leq x$ that is visited at most $\lfloor kt/x \rfloor \leq f$ times during the first t steps. Hence, if f agents may die, the adversary may delay the opening of x' to some time $t' > t$. Putting it together we have

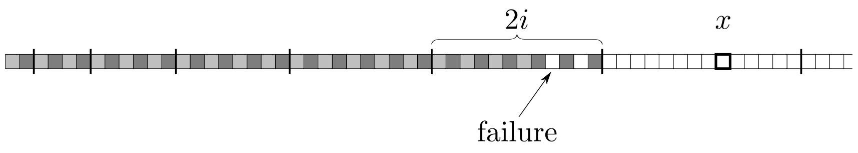
$$t' > t = \frac{(x-1)(f+1)}{k} \geq \frac{x'(f+1)}{k} - \frac{f+1}{k}$$

and the result follows, since $(f+1)/k$ is at most 1. ◀

However, even barely communicating agents can achieve competitive ratio of 1:

▶ **Claim 3.4.** *Two agents (with the possibility that one of them fails) can achieve hunt time $T_{\mathcal{F}}(2,x) \leq OPT_{\mathcal{F}}(2,x) + O(\sqrt{\min\{t_1, x\}})$, where t_1 is the time of the failure.*

Proof. The algorithm proceeds in phases, the aim of phase i is to open the next $2i$ boxes. The agent A_0 consecutively opens the even boxes, and the agent A_1 opens the odd boxes. At the end of the phase, each agent opens the last box that should have been opened by the other agent. If agent A_j finds an unopened box, it means that A_{1-j} died during the phase. A_j then goes back over the boxes that should have been opened by A_{1-j}, until it finds an opened box. A_j then proceeds sequentially from the beginning of the next phase.



Obviously, each box is eventually opened. What must be argued is how long the hunt may take compared to the optimal algorithm OPT. Note that in OPT, each agent that is alive opens a new box, and the opened boxes always form a consecutive interval. In order to obtain the result, we count the number of steps the algorithm behaves differently. First, before the failure occurs in phase i, there is one time step of delay per phase, in which both agents check each other. This accounts for at most $i = O(\sqrt{\min\{t_1, x\}})$ time steps. Finally, after the failure of one agent, the remaining agent may proceed to the end of the phase, and then return to find the treasure, accounting for another at most $2i = O(\sqrt{\min\{t_1, x\}})$ time steps. If the treasure is located beyond the phase in which the failure occurred, no additional delay is incurred. ◀

Now let us generalize the result to k agents. The main idea it the same as in the previous claim: proceed in phases of linearly increasing length, divide each phase among the surviving agents, then check at the end if some agents died during the phase, and if so, finish their part. However, care must be taken, because now also an agent finishing the part of an already failed agent, may fail. Also, in order to divide the work among themselves, the agents need to know the number of surviving agents, which is not always possible. We can formulate the following theorem:

▶ **Theorem 2.** *k agents (with the possibility that $k-1$ of them fails) can achieve hunt time $T_{\mathcal{F}}(k,x) \leq OPT_{\mathcal{F}}(k,x) + O(k^3\sqrt{x})$.*

Algorithm 1 Algorithm for one phase.

```
1: procedure PHASE(i)
2:     if $i > 1 \wedge S_i \neq S_{i-1}$ then
3:         all agents check all $k_{i-1}d_{i-1}$ boxes from phase $i-1$
4:     end if
5:     divide the $k_i d_i$ boxes among the $k_i$ agents
6:     each agent checks the assigned $d_i$ boxes
7:     compute set $S_{i+1}$ using $k^3$ boxes from phase $i+1$
8: end procedure
```

Proof. The algorithm proceeds in phases. Each phase i starts with a set $S_i \subseteq \{1, \dots, k\}$, such that all agents agree on S_i, and S_i contains all agents that are alive at the beginning of phase i. Initially $S_1 = \{1, \dots, k\}$. Let $k_i = |S_i|$, and $d_i = k^3 i$. The aim of phase i is to check $k_i d_i$ boxes.

The $k_i d_i$ boxes are divided among the k_i agents such that each agent is responsible for one class of boxes modulo k_i. Agents spend d_i time steps to check the assigned class. However, some agent may have died during the phase, and some of them might have already been dead at the beginning. At the end of the phase, the agents agree on the new set S_{i+1} as follows. Each agent $j \in S_i$ in turn uses the consensus algorithm from Lemma 1, using a fresh set of k^2 boxes from the next phase. If the result of the consensus algorithm is `true`, j is included in the set S_{i+1}. If, at the beginning of phase $i+1$, $S_i = S_{i+1}$, it means that all agents from S_i were alive when the consensus computation started, and so all boxes from phase i have been checked. If, on the other hand $S_{i+1} \neq S_i$, some agents from S_i died. It might have been during the consensus computation, but in any case, the boxes of phase i must be rechecked. In order to avoid the necessity to argue about further failures, all agents from S_{i+1} recheck all boxes from phase i. Since at least one agent is guaranteed to survive, all boxes from phase i will be opened.

To argue about the speedup, we again count the number of time steps in which the algorithm behaves differently from OPT. First, there are $O(k^2)$ time steps to compute the set S_{i+1} in each phase. Since there are $O(\sqrt{x})$ phases, this account for $O(k^2\sqrt{x})$ steps. Also, there are at most $O(k)$ failures, and each failure can trigger at most one recheck of a phase.

Since a phase contains at most $O(k^2\sqrt{x})$ boxes, this accounts for another $O(k^3\sqrt{x})$ steps. ◀

3.3 Number of Agents in Black Holes Model

While in the failing agents model even a failure of $k-1$ agents can be tolerated without communication, in the black holes mode a single black hole can prevent any deterministic non-communicating algorithm from finding the treasure:

▶ **Claim 3.5.** *Consider a system with k non-communicating agents with a single black hole. There is no algorithm that always successfully finds the treasure.*

Proof. For a box given x, let the *signature* of x, $\tau_x = [t_{x,1}, \dots, t_{x,k}]$ be the k-tuple such that $t_{x,j}$ is the first time that agent j opens x in a fault-free execution; if j never opens x, then $t_{x,j} = \infty$. If there are two boxes x, x' such that $\tau_x \le \tau_{x'}$, i.e. for all j $t_{x,j} \le t_{x',j}$, then placing a black hole in x and treasure in x' results in the algorithm never finding the treasure. To conclude, it is sufficient to find two such boxes. Let us call a *pattern* of a signature τ_x the set of positions j for which $\tau_{x,j} = \infty$. Since there is a finite number of patterns (namely, less than 2^k), and the number of boxes in unlimited, there is a pattern p that is shared by infinitely many boxes. Let us fix a box b with pattern p, and count the maximum number of boxes x (with pattern p) for which $\tau_x \not\geq \tau_b$. For any such box x there must be some index j such that $t_{x,j} < t_{b,j}$, and $t_{b,j}$ is finite. However, for any (finite) integer d there is at most one box x such that $t_{x,j} = d$. Hence, there are at most finitely many boxes (with pattern p) such that $\tau_x \not\geq \tau_b$. Since the number of boxes with pattern p is unlimited, there is a box x for which $\tau_x \geq \tau_b$. ◀

At the core of the proof of the previous claim is the fact that the sequence of boxes is potentially unlimited. If there is an upper bound on the number of boxes, the situation is much different.

▶ **Lemma 3.** *For large enough z, let $x = z^4$. Boxes $[x] = \{1, 2, \ldots, x\}$ can be cleared in z^3 steps by $g(x)$ agents without communication in the presence of at most z black holes, where $g(x) = O(z^2 \ln z)$.*

▶ **Corollary 4.** *For $n \geq f^4$, $O(\sqrt{n} \ln n)$ non-communicating agents can perform treasure hunt in the presence of f black holes.*

▶ **Corollary 5.** *In the algorithm from Lemma 3, every box e is scheduled to be opened by at least $z + 1$ agents.*

Let us focus now on the barely communicating model. We start by observing that enhancing the model with communication indeed makes it stronger, as it can now cope with a single black hole:

▶ **Claim 3.6.** *Three agents can find the treasure in the presence of one black hole.*

Proof. The first agent, A, starts by sequentially opening the boxes, i.e. A opens box i at time i, $i = 1, 2, \ldots$. It is followed by two agents, B_1, B_2 opening even and odd boxes, respectively, i.e. B_j opens box $2i + 1 - j$ at time $2i + 1$, $i = 1, 2, \ldots$. If no box contains a black hole, B_1, B_2 are always behind A, and see their boxes had already been opened. So when there is a black hole, A finds it first, and dies there, followed by one of the agents B_1, B_2. The other B agent survives, and finds an unopened box, z, in the next step. Then it knows that the black hole is located in $z - 1$, so it can finish the search sequentially. ◀

Let us note that each black hole can kill at least two agents, giving us a lower bound on the number of agents:

▶ **Theorem 6.** *$2f + 1$ agents are needed to perform treasure hunt in the presence of f black holes.*

Proof. For the sake of contradiction consider a treasure hunt algorithm with at most $2f$ agents. We construct a configuration of f black holes such that in finite time all $2f$ agents will be killed. Since the number of boxes is unlimited, if the treasure is located far enough, it will not be found.

The configuration of black holes will be constructed in phases. First, consider the computation C_0 of the algorithm in a configuration without black holes. There must be a box that is checked by at least two agents: if every box is checked by at most one agent, then there must be an agent that is the only one to check at least two boxes. By placing a black hole in the first one, and the treasure in the second one, we have a configuration in which the algorithm fails to find the treasure. Consider the first time t_1 when an agent checks an already opened box x_1. Construct a computation C_1 by placing a black hole in box x_1. We claim that up to time t_1, the computations C_0, and C_1 are identical: indeed the agents are deterministic, and up to time t_1 each of them opens a yet unopened box. In C_1, the black hole at x_1 killed two agents, the second one at time t_1. Let S_1 be the set of unopened boxes at t_1, and repeat the same argument. There must be a box from S_1 that is opened by two agents in C_1 (after time t_1). Construct a computation C_2 by placing a black hole in the first box x_2 from S_1 that is visited by a second agent at time t_2. Up to t_2, C_1 and C_2 are identical: let t_2', $t_1 < t_2' < t_2$, be the time x_2 is first opened by an agent A. Clearly, C_1 and C_2 are identical up to t_2'. After t_2', agent A in C_1 either opened boxes outside S_1 that were already opened anyway, or another boxes from S_1 that were never opened by another agent (up to t_2). In any case, the disappearance of A in C_2 has no effect on the set of opened boxes in C_2. Hence, the black hole kills another two agents. Continuing this way, we find a set of f black holes that kill all $2f$ agents. ◀

We now proceed to showing an asymptotically matching upper bound of $4f + 1$. The idea is to sequentially scan the boxes in phases, where i-th phase ensures that the i-th box is opened. In order to do so, the agents elect a leader that probes the box, and let the others know if it survived, using a fresh set of boxes. However, if the agents agree that the leader is not alive, they still can not be sure whether the i-th box contains a black hole, since the leader election algorithm itself might have failed. Hence, the i-th box must be checked once more, maybe requiring several iterations in a single phase. However, care must be taken that if the i-th box is indeed a black hole, not too many agents die there. First, let us formulate the *leader election* algorithm that will be employed:

▶ **Lemma 7.** *Using a set of k fresh boxes, the agents may perform an algorithm, such that*
1. *At most one agent declares itself a leader.*
2. *If the k boxes do not contain a black hole, exactly one leader is elected.*

Proof. In the first step, agent i opens the i-th box. Then it scans the subsequent boxes, until it either finds an opened box, or reaches the last box. In the latter case, the agent is declared a leader. ◀

▶ **Theorem 8.** $4f + 1$ *agents are sufficient to find the treasure in the presence of f black holes.*

Proof. The algorithm works in phases, such that at the end of phase i, boxes $\{1, \ldots, i\}$ (and possibly some others) are opened. A phase to check box x consists of several iterations. Each iteration considers a fresh set of $k^2 + k + 1$ boxes, called a *scratchpad*, that are used for communication, and have the following structure: there are two sets of k boxes to perform a leader election algorithm from Lemma 7, two sets of $k(k-1)/2$ boxes to perform a coordination algorithm from Lemma 1, and one box called *survival flag*. The iteration starts by the agents performing a leader election. Subsequently, all agents except the leader perform another leader election algorithm to elect a *deputy*. Next, the leader opens the survival flag box, and immediately proceeds to check x, after which it performs the coordination algorithm from Lemma 1 to signal the other agents. If the result of the coordination is `true`, both the iteration, and the phase are ended, and a new phase starts to check the next box. If the result of the coordination is `false`, the deputy checks the survival flag. If it was opened, the deputy uses the second coordination algorithm to inform the other agents. If the result of the second coordination is `true`, again, both the iteration, and the phase are finished. If the result is `false`, new iteration of the same phase starts.

First, we show that at the end of a phase, box x was opened. The phase ends only when at least one of the coordination algorithms results in `true`. The first one may result in `true` only when the leader survives the visit of box x. The second one only if the deputy finds the survival flag open and survives to tell that to others. Hence, the leader also survives opening the survival flag and as it immediately proceeds to open box x, x was definitively opened.

Second, let us note that if the scratchpad does not contain a black hole, the corresponding iteration is the last iteration of a phase. Indeed, if there is no black hole in the scratchpad, both leader, and deputy are successfully elected. The leader then successfully opens the survival flag, and proceeds to open x. If it survives, it again successfully performs the coordination, and phase ends. If the leader dies in x, the deputy reads the survival flag, and successfully signals the others.

Finally, let us count how many agents may die. In any given iteration, agents may die either in the scratchpad or in the box being checked, x. The scratchpads use fresh boxes in each iteration, and each box in the scratchpad is accessed by at most two different agents. Hence, a single black hole may kill at most two agents in a scratchpad. Note that the particular box with the black hole will be checked at some later phase (playing the role of x) where it may kill additional agents; these will be accounted for separately. Overall during the whole computation, at most $2f$ agents die in scratchpads.

Now let us count the agents that die while entering the checked box x. Obviously, there must be a black hole located in x, so there are at most f such boxes, each of them killing several agents over a number of iterations. However, as argued above, an iteration without a black hole in the scratchpad ends the phase. As there are, overall, at most f iterations with a black hole in the scratchpad, at most $2f$ agents die when entering the checked box.

Altogether, at most $4f$ agents may die. ◀

While the previous theorem showed that $4f + 1$ agents are sufficient, it relied heavily on the fact that the number of boxes is unlimited. Indeed, in order to locate the treasure in box x, boxes up to $O(xf^2)$ were used. A natural scenario, however, is that the number of boxes is finite. The algorithm from the previous theorem can not be used in this case, since it would not be able to find a treasure located in the last part of the boxes. In order to remedy this situation, we propose another algorithm that uses more agents (in particular, $O\left((f \ln f)^2\right)$ of them), but uses only boxes up to $x + O\left((f \ln f)^2\right)$ to locate the treasure in box x. Although omitted in this paper, it is possible to adjust the algorithm to the setting with finite number of boxes (by using a few additional agents).

The first idea is to use the algorithm from Lemma 3 to clean the scratchpad after each iteration. However, it can not be used in a straightforward manner, since the size of the scratchpad is quadratic in the number of agents, and Lemma 3 needs at least $\Omega(\sqrt{n} \ln n)$ agents to clean it. Therefore we shall implement a similar idea more carefully.

▶ **Theorem 9.** *$O(f^2 \ln^2 f)$ agents can locate the treasure in the presence of f black holes, even if the number of boxes is limited, provided that it is at least $x + O(f^2 \ln^2 f)$ where x is the location of the treasure.*

The algorithms from Theorems 8 and 9 are very slow, in fact much slower than a single searching agent in absence of black holes. Hence, the question is: What kind of speedup can be achieved in the presence of black holes, given sufficiently many agents?

As we see in the following theorem, a single black hole does not pose much of a problem:

▶ **Theorem 10.** *In the presence of one black hole, $k > 2$, k even, agents can locate the treasure in time $T(k, x) \le x/(k-2) + O(k + \sqrt{x}) \le OPT(k-2, x) + O(k + \sqrt{x})$.*

Proof. Divide the sequence of boxes into k' classes modulo $k' = (k-2)/2$ called *rows*. Each row is divided into *blocks*, with i-th block being of length $2(k' + i)$. Two *explorer* agents LE_r and RE_r are assigned to each row. In addition, there are two *checking* agents: LC and RC.

Let $s_{r,i}$, $t_{r,i}$ be the first and the last box of i-th block of row r. The algorithm works in phases; the aim of each phase is to clear one block in each row. At the beginning of phase i, the explorer agents in each row scan the block in opposite directions, i.e. the agent LE_r opens boxes starting by $s_{r,i}$ and finishing just before $t_{r,i}$ (i.e. without touching $t_{r,i}$), while RE_r symmetrically starts at $t_{r,i}$ and proceeds leftwards. If an explorer opens an already opened box, its task in the current phase is finished; it then proceeds to the next phase (to clear the next block). Let $T_i = 1 + \sum_{j=1}^{i-1}(k' + j + 1)$; note that without any black hole, all explorers start phase i in time step T_i.

The activity of the checking agents in block i is also counted in the phase i: The left checking agent LC scans the $s_{r,i}$ boxes. If all of them have already been opened, it becomes RC for the next phase. However, if it finds an unopened box in row r, it continues as LE_r in the current phase, and onwards. The RC agent works in a symmetrical way. Moreover, the checking agents make sure not to start any activity in block i before time $T_i + 1$; this ensures that they do not to interfere with the explorers.

First, let us consider an execution without a black hole. In this case, the explorers are always ahead of checkers (who don't play any role in the analysis yet), and they start exploring each block i at time T_i. In each phase, all explorers open a new box in every step. The only exception is the last step of the phase, when already opened boxes are probed by all explorers. Hence, if the treasure is located in a box x in i-th block, the delay w.r.t. $OPT(k-2, x)$ is at most the number of blocks (i.e. $i-1$) plus the length of the block (i.e. $2(k'+i)$). As $x \in \Theta((k+i)^2)$, the theorem holds.

Now let us consider a black hole located in a row r. All other rows are unaffected, and proceed as in the case without black holes. Moreover, if the treasure is found before the black hole, again, the analysis of the case without black holes applies. Let us focus only on row r, and let us assume that the black hole is encountered before the treasure. If the black hole is in the interior of the block (i.e. not the first or last box), both explorer agents die, with no unopened boxes left in this block; both checkers find unopened boxes in the next phase, become corresponding explorers for row r, and the algorithm proceeds without further delays and faults. If the black hole is in the first box of row r at phase i, both LE_r and LC die there. RE_r checks the whole block i of row r, and proceeds to the next phase, as well as RC. However, RC continues as LC in phase $i+1$, detects unopened first box and becomes LE_r. Analogously, if the black hole is in the last box of a block, the right explorer and checker die there, and the left checker eventually becomes the right explorer for the rest of the computation.

Note that the presence of the black hole in row r delays the exploration of row r by the delay between the time the explorer would have started exploring the block, and the time the checker checks whether it has indeed done so. As mentioned above, this delay is at most $r + i$. ◀

Observe that the technique used in the proof of Theorem 10 cannot be extended to more black holes. In fact, it is unclear what kind of speedup is possible in the presence of up to f black holes.

4 Conclusions and Open Problems

We have shown that adding even very weak indirect coordination (namely, the ability to detect whether a box has been previously opened, opening it and destroying the information in the process) allows deterministic agents to solve treasure hunt (a) with competitive ratio 1 in the failing agents models, and (b) with asymptotically optimal number of agents in the black hole model. This is somewhat surprising, as this is in some sense the weakest natural form of coordination.

Several research questions remain open:

- what is the hunt time in the black hole model? We have some partial results for 1 and 2 black holes, but not a general solution. It is not clear whether it is possible to have speedup of $\Omega(k-f)$.
- how to solve treasure hunt with black holes using randomized non-coordinated agents? How many agents are needed and what would be the speedup? What about allowing both weak coordination and randomization?
- our solutions are not robust w.r.t. to timing and ordering of boxes. How do you make them robust and at what cost?
- what is the weakest coordination (a) that still allows competitive ratio of 1 in the failing agents model? (b) that allows to solve treasure hunt by deterministic agents?

Acknowledgement. We thank the anonymous referee for pointing out several inaccuracies in the manuscript.

References

1 Steve Alpern and Shmuel Gal. *The Theory of Search Games and Rendezvous*. International Series in Operations Research & Management Science. Springer US, 2006.

2 David P. Anderson. Boinc: A system for public-resource computing and storage. In *5th IEEE/ACM International Workshop on Grid Computing*, pages 4–10, 2004. URL: `http://boinc.berkeley.edu`.

3 Ricardo Baeza-Yates and René Schott. Parallel searching in the plane. *Computational Geometry*, 5(3):143–154, 1995. `doi:10.1016/0925-7721(95)00003-R`.

4 Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J. E. Rawlins. *Searching with uncertainty extended abstract*, pages 176–189. Springer Berlin Heidelberg, Berlin, Heidelberg, 1988. `doi:10.1007/3-540-19487-8_20`.

5 Ricardo A. Baeza-Yates, Joseph C. Culberson, and Gregory J.E. Rawlins. Searching in the plane. *Inf. Comput.*, 106(2):234–252, 1993. `doi:10.1006/inco.1993.1054`.

6 Hans-Joachim Böckenhauer, Juraj Hromkovic, Dennis Komm, Richard Královic, and Peter Rossmanith. On the power of randomness versus advice in online computation. In Henning Bordihn, Martin Kutrib, and Bianca Truthe, editors, *Languages Alive - Essays Dedicated to Jürgen Dassow on the Occasion of His 65th Birthday*, volume 7300 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2012. `doi:10.1007/978-3-642-31644-9_2`.

7 Marek Chrobak, Leszek Gąsieniec, Thomas Gorry, and Russell Martin. *Group Search on the Line*, pages 164–176. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. `doi:10.1007/978-3-662-46078-8_14`.

8 Lihi Cohen, Yuval Emek, Oren Louidor, and Jara Uitto. Exploring an infinite space with finite memory scouts. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 207–224. SIAM, 2017. `doi:10.1137/1.9781611974782.14`.

9 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Searching for a black hole in arbitrary networks: optimal mobile agents protocols. *Distributed Computing*, 19(1):1–99999, 2006. `doi:10.1007/s00446-006-0154-y`.

10 Stefan Dobrev, Paola Flocchini, Giuseppe Prencipe, and Nicola Santoro. Mobile search for a black hole in an anonymous ring. *Algorithmica*, 48(1):67–90, 2007. `doi:10.1007/s00453-006-1232-z`.

11 Yuval Emek, Tobias Langner, Jara Uitto, and Roger Wattenhofer. *Solving the ANTS Problem with Asynchronous Finite State Machines*, pages 471–482. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi:10.1007/978-3-662-43951-7_40.

12 Ofer Feinerman, Amos Korman, Zvi Lotker, and Jean-Sebastien Sereni. Collaborative search on the plane without communication. In *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*, PODC '12, pages 77–86, New York, NY, USA, 2012. ACM. doi:10.1145/2332432.2332444.

13 Pierre Fraigniaud, Amos Korman, and Yoav Rodeh. Parallel exhaustive search without coordination. In Daniel Wichs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 312–323. ACM, 2016. doi:10.1145/2897518.2897541.

14 Chryssis Georgiou. *Do-All Computing in Distributed Systems: Cooperation in the Presence of Adversity.* Springer Publishing Company, Incorporated, 1st edition, 2010.

15 Nicolas Hanusse, Dimitris J. Kavvadias, Evangelos Kranakis, and Danny Krizanc. Memoryless search algorithms in a network with faulty advice. *Theor. Comput. Sci.*, 402(2-3):190–198, 2008. doi:10.1016/j.tcs.2008.04.034.

16 Ming-Yang Kao, John H. Reif, and Stephen R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 441–447, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=313559.313848.

17 Dénes König. Über graphen und ihre anwendung auf determinantentheorie und mengenlehre. *Mathematische Annalen*, 77(4):453–465, 1916. doi:10.1007/BF01456961.

18 Amos Korman and Yoav Rodeh. Parallel linear search with no coordination for a randomly placed treasure. *CoRR*, abs/1602.04952, 2016. arXiv:1602.04952.

19 Amos Korman and Yoav Rodeh. Parallel search with no coordination. In *Structural Information and Communication Complexity - 24th International Colloquium, SIROCCO 2017, Porquerolles, France, June 19-22, 2017*, page to appear, 2017.

20 Tobias Langner, Jara Uitto, David Stolz, and Roger Wattenhofer. *Fault-Tolerant ANTS*, pages 31–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. doi:10.1007/978-3-662-45174-8_3.

21 Christoph Lenzen, Nancy Lynch, Calvin Newport, and Tsvetomira Radeva. Searching without communicating: tradeoffs between performance and selection complexity. *Distributed Computing*, pages 1–23, 2016. doi:10.1007/s00446-016-0283-x.

22 Nancy A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

23 Andrzej Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1-2):71–109, 2002. doi:10.1016/S0304-3975(01)00303-6.

24 Mengfei Peng, Wei Shi, Jean-Pierre Corriveau, Richard Pazzi, and Yang Wang. Black hole search in computer networks: State-of-the-art, challenges and future directions. *Journal of Parallel and Distributed Computing*, 88:1–15, 2016. doi:10.1016/j.jpdc.2015.10.006.

25 Amnon Ta-Shma and Uri Zwick. Deterministic rendezvous, treasure hunts, and strongly universal exploration sequences. *ACM Trans. Algorithms*, 10(3):12:1–12:15, 2014. doi:10.1145/2601068.

26 Qin Xin. Faster treasure hunt and better strongly universal exploration sequences. *CoRR*, abs/1204.5442, 2012. arXiv:1204.5442.