Theory of Computing Systems (2019) 63:1819–1848
https://doi.org/10.1007/s00224-019-09948-6

On-line Search in Two-Dimensional Environment

Dariusz Dereniowski[1] · **Dorota Osula**[1]

Published online: 11 September 2019

Abstract

We consider the following on-line pursuit-evasion problem. A team of mobile agents called searchers starts at an arbitrary node of an unknown network. Their goal is to execute a search strategy that guarantees capturing a fast and invisible intruder regardless of its movements using as few searchers as possible. We require that the strategy is connected and monotone, that is, at each point of the execution the part of the graph that is guaranteed to be free of the fugitive is connected and whenever some node gains a property that it cannot be occupied by the fugitive, the strategy must operate in such a way to keep this property till its end. As a way of modeling two-dimensional shapes, we restrict our attention to networks that are embedded into partial grids: nodes are placed on the plane at integer coordinates and only nodes at distance one can be adjacent. Agents do not have any knowledge about the graph *a priori*, but they recognize the direction of the incident edge (up, down, left or right). We give an on-line algorithm for the searchers that allows them to compute a connected and monotone strategy that guarantees searching any unknown partial grid with the use of $O(\sqrt{n})$ searchers, where n is the number of nodes in the grid. As for a lower bound, there exist partial grids that require $\Omega(\sqrt{n})$ searchers. Moreover, we prove that for each on-line searching algorithm there is a partial grid that forces the algorithm to use $\Omega(\sqrt{n})$ searchers but $O(\log n)$ searchers are sufficient in the off-line scenario. This gives a lower bound on $\Omega(\sqrt{n}/\log n)$ in terms of achievable competitive ratio of any on-line algorithm.

Keywords Connected search · Distributed searching · On-line searching · Partial grid · Pursuit-evasion

This article is part of the Topical Collection on *Special Issue on Approximation and Online Algorithms (2017)*

✉ Dariusz Dereniowski
deren@eti.pg.edu.pl

Dorota Osula
dorurban@student.pg.edu.pl

[1] Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Gdańsk, Poland

1 Introduction

A team of mobile autonomous robots wants to search an area with the goal of finding a mobile intruder (or lost entity). The intruder has several properties that dictate how a search should be conducted. First, the intruder is invisible and therefore the robots may conclude its potential locations only from the history of their own moves. Second, it is assumed that the speed of the intruder is unknown and therefore the robots build their search strategy assuming that the intruder is very fast: it may traverse arbitrarily long distance between any two actions of a robot. Third, the intruder is very clever, i.e., it will avoid being captured as long as possible; in other words we may imagine that it knows locations of robots and their future movements at any point. This assumption enforces robots to consider the worst case scenario for them since they want to have a search strategy that guarantees interception. The above problem is usually restated in discrete terms, naturally expressing the search game using graph-theoretic notation. Following the widely used terminology, the mobile entities performing the search are called *searchers*.

In this work we focus on the graph-theoretic problem statement, where the searchers operate in a given graph in which they move along edges. Moreover, what greatly influences algorithmic approaches is the assumption of whether the searchers know the graph in advance (off-line version of the problem) or whether the graph is unknown and the searchers learn its structure while conducting the search (on-line or distributed setting). We shortly review both approaches, giving later a formal statement of the problem we study in this work. In all cases we are interested in minimizing the number of searchers needed to clear the given network.[1] We discuss briefly later a possibility for our algorithm to be adopted to operate in distributed asynchronous setting, with searchers having local communication and polynomial memory. From the point of view of this work, the terms on-line and distributed are used exchangeably because we do not impose any communication, memory or synchronization restrictions on the agents (a more detailed discussion on this topic is provided in Section 2).

Off-line searching Off-line graph searching models are extensively studied and numerous deep results have been obtained, providing insight into not only the problem itself but also enriching the more widely understood graph theory through the connections between graph searching games and many graph parameters, e.g., pathwidth, treewidth, branchwidth, bandwidth, profile, interval thickness, vertex separation number; see, e.g., [24] for a survey and further references. The historically first studied graph searching model is called *edge search* [37, 38]. In this problem, the goal is to construct a search strategy that guarantees capturing a fast and invisible fugitive (thus, the strategy must ensure success regardless of the moves performed by the fugitive) in a graph that is given as an input to an algorithm computing a search

[1] In this work the terms graph and network are used exchangeably.

strategy. A search strategy itself is a sequence of *moves*, where each move is one of the following: (i) placing a searcher on any graph node; (ii) removing a searcher from the node it occupies; (iii) sliding a searcher along an edge in order to clear it. At each point of the strategy one can distinguish a subgraph the is guaranteed to be free of the fugitive. In a valid strategy we require that this subgraph is the input graph at the end of the strategy (thus the capture of the fugitive necessarily occurs at some point) and, additionally, in a *connected search strategy* we require that after each move this subgraph is connected. In a *monotone search strategy* we require that once an edge has been cleared, it must remain clear till the end of the search; in other words, the subgraph composed of edges that may contain the fugitive may only shrink as the search progresses. Since we adopt the monotone connected searching problem in our on-line model, we point out to few recent works on the problem [2, 3, 14, 15, 19, 20].

On-line searching In the distributed, or on-line, version of the problem it is assumed that the network is unknown in advance to the searchers. In this setting, some assumptions need to be made. First, only monotone search strategies are considered. This assumption is dictated by an observation that otherwise the searchers may first learn the structure of the network by exploring it (and thus ignoring the possibility of capturing the intruder at this stage) and once the network is known, they can compute a search strategy by using an off-line algorithm and finally execute the strategy. The problem then reduces to exploration and map construction, well studied problems in distributed computing. Another natural assumption is to forbid placing a searcher on a node that has not been visited before. Recall that we are interested in minimizing the number of searchers. On-line algorithms are formulated usually in such a way that the algorithm is adding new searchers whenever necessary and in the analysis one counts how many searchers will be added in the worst case — we will follow this route.

We consider *connected* search strategies in this work, i.e., strategies that guarantee that at any given time point the subgraph that is clear is connected. Note that this allows us to assume that all searchers start at some node called the *homebase* and only moves of type (iii) are then made (see the definition of edge search above). Indeed, removing a searcher from a node u and placing it on another one v (i.e., *jumping*) may be replaced[2] by a sequence of sliding moves along a path from u to v consisting of clear edges only (such a path must exists due to connectedness and monotonicity).

1.1 Related Work

Off-line problems One of the central questions raised in the context of various graph searching problems is if the search problem is *monotone*, i.e, if there exists a mono-

[2]Note that this observation is true as long as the considered optimization criterion is to minimize the number of searchers. For other criteria, like, e.g., search time, the exclusion of jumping may be potentially limiting.

tone search strategy solving it. Note that proving monotonicity is a tool that allows to conclude membership in NP for a given problem. It is known that the edge search problem is monotone [31]. On the other hand, the connected search is not monotone [44]. A question related to the latter searching model is: how many extra searchers one needs to ensure connectivity. It turns out that each monotone edge search strategy can be converted (in polynomial time) into a monotone connected one by approximately 'doubling' the number of searchers [16]. Thus, for asymptotic results, like the one in this work, this gives another reason that justifies restricting attention to monotone connected search strategies.

On-line searching In most cases, when designing distributed searching algorithms, the monotonicity requirement is adopted. (See [6] for an example how an optimal connected search strategy can be constructed in a distributed fashion when recontamination is allowed.) During construction of a monotone strategy in an on-line way, there is naturally some 'cost' involved in terms of increased number of searchers required for guarding — this cost measured as the ratio of number of searchers that each on-line algorithm needs to use for some n-node graph and its monotone connected (off-line) search number is know to be $\Omega(n/\log n)$ [28]. In the realm of distributed algorithms, natural questions arise with respect to the amount (and type) of additional information regarding the underlying network given a priori to an algorithm. In [36] is was proved that $O(n \log n)$ bits of *advice* are sufficient to construct an optimal connected monotone search strategy (the concept of such quantitative approach to advice analysis was introduced in [25]). An example of an algorithmic approach in a very weak computational model see, e.g., [5, 13].

Grid networks were studied in searching models, where the concepts of temporal and threshold immunities were used. In the first one, a node after clearing remains protected (even if unguarded) against recontamination for a certain amount of time t. A tight upper bound for the grid of size $m \times n$, $m \geq n$, is equal to $\min\{\lceil m/\lceil t/2\rceil\rceil, \lceil(2m-1)/t\rceil\}$ [11]. In [21, 22] d-dimensional meshes were investigated in the threshold immunity model, where a node becomes recontaminated when a specified number m (or greater) of its neighbors is contaminated. Especially, it has been proved that for $d = 2$ (i.e., grids) and $m \geq 2$ one searcher is enough. For other searching works involving threshold immunity see, e.g., [12, 23, 32].

For other distributed searching models and algorithms for specific network topologies see [8, 20, 26, 35].

Applications in robotics We note that our results may be of particular interest not only by providing theoretical insight into searching dynamics in distributed agent computations, but may also find applications in the field of robotics. Most investigations oriented towards algorithms that can be applied on physical devices need to deal with the problem of modeling of the real world. This can be done either by discretizing it (usually through graphs) or by building algorithms that work in continuous search space and need to address the geometric issues that emerge. In Section 7.1,

we add a brief discussion on this subject from the point of view of our results. Having in mind the vast literature on the subject we point the interested reader to a few references to recent works in this field [10, 17, 27, 30, 39–43].

1.2 Outline of This Work

The next section provides the notation used in this work and the problem statement. It is subdivided so that Section 2.1 defines the graph searching problem we study while Section 2.2 introduces the terminology related to the partial grid networks we consider in this paper. Section 3 gives a construction of a class of n-node networks such that each on-line algorithm, which produces a monotone connected strategy, uses $\Omega(\sqrt{n})$ searchers for some network in the class which turns out to be $\Omega(\sqrt{n}/\log n)$ times more than an optimal off-line algorithm would use (recall that by off-line algorithm we refer to the case when the entire network is given as an input and hence is known in advance to the algorithm).

Section 4 describes an on-line (i.e., agents do not have any knowledge about the graph *a priori*) algorithm that performs a monotone connected search in partial grids where it is assumed that the algorithm is given an upper bound n on the size of the network. We assume a 'sense of direction' in our model, that is, the grid is embedded into a two-dimensional space by assigning integer coordinates to the nodes. Then, an agent knows the coordinates of each neighbor of the currently occupied node. More details are given in Section 2.2. We point out that this algorithm uses an on-line procedure from [7] as a subroutine that is called many times to clear selected parts of a grid and it can be seen as a generalization from a 'linear' graph structure studied in [7] to a 2-dimensional structure discussed in this work. Also, although both algorithms are conducted via some greedy rules which dictate how a search should 'expand' to unknown parts of the graph, the analysis of our algorithm is different from the one in [7].

Then, in Section 5 we prove the correctness of the algorithm and provide an upper bound on its performance: it is using $O(\sqrt{n})$ searchers for any partial grid network. In Section 6 we consider a modified version of the algorithm, which receives no information on the underlying graph in advance, and we prove that the algorithm also uses $O(\sqrt{n})$ searchers. This result, stated in Theorem 4, is our main contribution. We finish with conclusions in Section 7, giving a few remarks on how our work relates to searching two-dimensional environments, like polygons with holes. As there are many open problems and research directions related to the subject, we list some of them also in Section 7.

We briefly remark on a potential practical motivation of our setting. Partial grids, which can be seen as a grids with obstacles (formally defined later), are a way of modeling two-dimensional shapes, e.g., polygons. Every search strategy for a polygon can be used to obtain a search strategy for its underlying partial grid and vice versa. The number of searchers in both cases are withing a constant factor of each other. Thus in particular, searching strategies for continuous scenarios like polygons can be obtained by first getting the underlying partial grid and then computing a (discrete) search strategy for the grid by the algorithm we propose in this work. For more details see Section 7.1.

2 Definitions and Terminology

In this section we state our problem formally and present the notation we use.

2.1 Problem Statement

Let G be a simple, undirected, connected graph. A *monotone connected k-search strategy* $\mathcal{S}$ for a network G is defined as follows. Initially, k searchers are placed on a node h of G, called the *homebase*. (We also say that $\mathcal{S}$ *starts at* h.) Then, $\mathcal{S}$ is a sequence of moves, where each *move* consists of selecting one searcher present at some node u and sliding the searcher along an edge $\{u, v\}$. (Thus, the searcher moves from its current location to one of the neighbors.)

Initially, all edges are *contaminated*. After each move of sliding a searcher along an edge $\{u, v\}$ it is declared to be *clear*. It becomes contaminated again (*recontaminated*) if at any time during execution of the strategy $\mathcal{S}$ at least one of its endpoints is not occupied by a searcher and is incident to a contaminated edge. We consider only strategies in which recontamination does not occur and we call such strategies *monotone*. Note that this in particular implies that the homebase h remains clear throughout the entire strategy. Moreover, we require that the *clear* subgraph, that is, the subgraph consisting of all clear edges, is connected after each move of the search strategy. Finally, we require that after the last move of $\mathcal{S}$ all edges are clear. Throughout, we say that a node is *clear* if it is incident to a clear edge.

The minimum k such that there exists a node h and a monotone connected k-search strategy that starts at h is called the *monotone connected search number* of G and denoted by $\texttt{mcs}(G)$.

Having defined a search strategy, we now state the on-line model we use. All searchers start at the homebase and the network itself is not known in advance to the searchers (except for the fact that the searchers may expect that the network is a partial grid). In fact, the searchers have no information about the network. We assume that nodes are anonymous and searchers have identifiers. The edges incident to each node are marked with unique labels (port numbers) and because only partial grids are considered in this work (for a definition see Section 2.2) we assume that labels naturally reflect all possible directions for each edge (i.e., left, right, up and down).

For the searchers, we assume that they communicate *locally* by exchanging information when present at the same node. Our algorithm is stated as if there existed *global* communication but it can be easily turned into required one with local communication as follows: we can designate one extra searcher called the *leader* who will be performing the following actions at the beginning of each move of the search strategy to be executed. First, the leader visits all nodes of the subgraph searched to date and gathers complete information about its structure and positions of all other searchers, then the leader computes the next move and finally visits all searchers to pass the information about the next move. Then, the move is performed by the agents.[3]

[3]Note that the actions of the leader clearly contain a lot of excess work in terms of the number of moves it performs; since the criteria as time or cost (number of sliding moves) are out to scope of this work, we will leave the reader with such a simple leader implementation.

Our algorithm is described for the synchronous model in which time is divided into steps, each step having the same unit length duration allowing each searcher to perform its local computations and slide along an edge if the searcher decides to move. We note that this assumption can be lifted and the algorithm can be easily restated to be asynchronous. Indeed, having one agent that is the leader one can simulate synchronous behavior of the agents in such a way that the leader waits for the completion of the current move of another searcher and then informs the searcher that is supposed to perform the next move, dictated by the search strategy, to start the move. As to the memory model, our algorithm requires that the memory size of the searchers is polynomial in the size of the network, and we do not attempt to optimize this parameter.

For any on-line algorithm A, let $A(G, h)$ be the number of searchers that it uses to clear a network G in a monotone connected way starting from the homebase h. We say that an algorithm A is $f(n)$-*competitive*, for some function f, if

$$\max_h \frac{A(G,h)}{\texttt{mcs}(G)} \leq f(n)$$

for each n-node network G.

2.2 Partial Grid Notation

We define a *partial grid* $G = (V, E)$ with a set of n nodes V and edges E as a connected subgraph of an $n \times n$ grid. We consider each partial grid to be embedded into two-dimensional Cartesian coordinate system with a horizontal x-axis and vertical y-axis, where each node of G is located at a point with integer coordinates and two nodes are adjacent if and only if the distance between them equals one (in Euclidean metric). This embedding is considered for two reasons. The first one is technical, as it simplifies some statements when we refer to coordinates when pointing nodes of G. The second is that our on-line algorithm relies on the underlying geometric structure. For convenience, the homebase is located at the point $(0, 0)$. In order to refer to a node that corresponds to a point with coordinates (x, y) we write $v\,(x, y)$. In this work n denotes an upper bound on the number of nodes of a partial grid, such that $\sqrt{n}$ is an integer.

Informally speaking, our algorithm will conduct a search by expanding the clear part of the graph from one 'checkpoint' to another. These checkpoints (defined formally later) will be subsets of nodes and their potential placements on the partial grid are dictated by the concept of a *frontier*. Take any $x = i\sqrt{n}$ for some integer i, $y = j\sqrt{n}$ for some integer j and take $i', j' \in \{0, 1\}, i' \neq j'$. Then, the line segment with endpoints (x, y) and $(x + \sqrt{n}i', y + \sqrt{n}j')$ is called a *frontier* and denoted by $F\left((x, y), \left(x + \sqrt{n}i', y + \sqrt{n}j'\right)\right)$. Whenever the endpoints of a frontier are clear from the context or not important we will omit them. The frontier $F\left((0, 0), \left(\sqrt{n}, 0\right)\right)$ that contains the origin is called the *homebase frontier* and the set of all frontiers is denoted by $\mathcal{F}$. We will also divide frontiers into *vertical* and *horizontal* ones, where coordinates of two extreme nodes do not differ on first and second coordinate, respectively.

Given any graph $H = (V', E')$ and $X \subseteq V'$, $H[X]$ is the subgraph of H *induced by* X: its node set is X and consists of all edges $\{u, v\}$ of H having both endpoints in X. The subgraph induced by all nodes that belong to a frontier F of a partial grid G is denoted by $G[F]$.

For $i \in \{1, \ldots, \sqrt{n}\}$ and some frontier $F = F\left((x, y), \left(x', y'\right)\right)$, where $x \leq x'$ and $y \leq y'$, we define the *i-th rectangle of* F, denoted by $\mathcal{R}(F, i)$, as the rectangle with corner vertices $(x - i, y - i)$, $(x - i, y' + i)$, $(x' + i, y' - i)$, $(x' + i, y' + i)$ if F is horizontal and as the rectangle with corner vertices $(x - i, y - i)$, $(x + i, y - i)$, $(x' - i, y' + i)$, $(x' + i, y' + i)$ if F is vertical. See Fig. 1 for an example.

Informally speaking, the two above concepts, namely frontiers and rectangles, provide a template on how the search may progress. However, due to the structure of a partial grid it may be possible that only certain nodes, but not all, that lie on a frontier have been reached at some point of a search strategy. For this reason, our notation needs to be extended to subsets of nodes that lie on frontiers and the corresponding rectangles. Any subset C of nodes of G that belong to some frontier F is called a *checkpoint*. *The 0-th expansion of a checkpoint* C is C itself and is denoted by $C\langle 0\rangle$. For $i \in \{1, \ldots, \sqrt{n}\}$ we define *the i-th expansion of* C, denoted by $C\langle i\rangle$, recursively as follows: the set $C\langle i\rangle$ consists of all nodes $v \notin C\langle 0\rangle \cup C\langle 1\rangle \cup \cdots \cup C\langle i-1\rangle$ for which there exists a node $u \in C\langle i-1\rangle$, such that there exists a path between v and u in the subgraph of G induced by nodes that lie on the rectangles $\mathcal{R}(F, 0), \mathcal{R}(F, 1), \ldots, \mathcal{R}(F, i)$. Define

$$C^{+}\langle i\rangle = C\langle 0\rangle \cup \ldots \cup C\langle i\rangle, \quad i \in \{0, \ldots, \sqrt{n}\}.$$

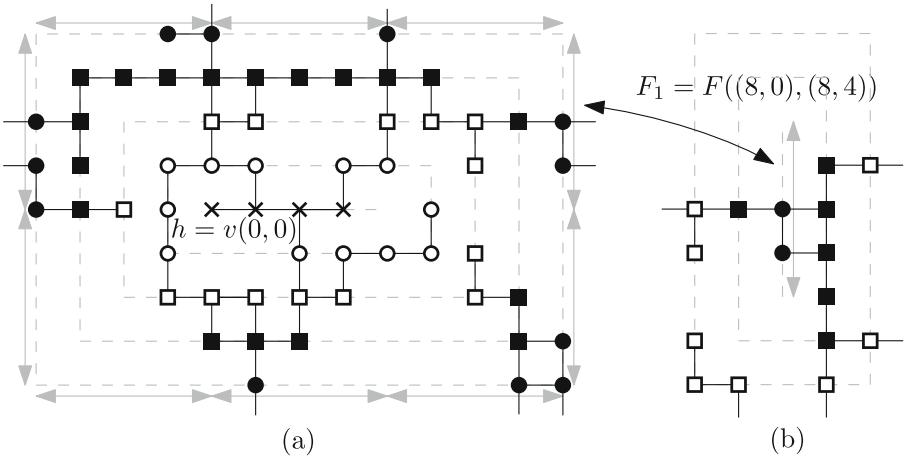


Fig. 1 An illustration of the concept of rectangles (here $\sqrt{n} = 4$). In **a** *crosses* denote nodes that lie on the homebase frontier $F = F((0,0),(4,0))$, *empty circles* denote nodes that lie on $\mathcal{R}(F, 1)$, *empty squares* the ones on $\mathcal{R}(F, 2)$, *dark squares* the ones on $\mathcal{R}(F, 3)$ and *dark dots* denote nodes that lie on $\mathcal{R}(F, 4)$. *Gray arrows* stand for the 10 frontiers, that lie on the $\mathcal{R}(F, 4)$ (six horizontal and four vertical ones). We denote one of the vertical frontiers that lie on $\mathcal{R}(F, 4)$ as $F_1 = F((8,0),(8,4))$. In **b** *dark dots* denote nodes that lie on F_1, *dark squares* the ones on $\mathcal{R}(F_1, 1)$ and *empty squares* the ones on $\mathcal{R}(F_1, 2)$

Informally, $C\langle i\rangle$ consists of only those nodes that belong to the rectangle $\mathcal{R}(F,i)$ that are connected to nodes of C by paths that lie 'inside' of $\mathcal{R}(F,i)$ — this definition captures the behavior of searchers (in our algorithm) that guard the nodes of C and 'expand' from C in all directions: then possible nodes that belong to any of the rectangles $\mathcal{R}(F,0), \mathcal{R}(F,1), \ldots, \mathcal{R}(F,i)$ but do not belong to $C^+\langle i\rangle$ will not be reached by the searchers. See Fig. 2 for an exemplary checkpoint with its expansions.

3 Lower Bound

First note that a regular $\sqrt{n} \times \sqrt{n}$ grid requires $\Omega(\sqrt{n})$ searchers even in the off-line setting [18], that is, when the network is known in advance and the searchers may decide on the location of the homebase. Therefore, our on-line algorithm is asymptotically optimal with respect to this worst case measure.

We aim at proving that for *each* on-line algorithm A there exists an n-node partial grid network G with homebase h such that $\max_h A(G,h)/\mathtt{mcs}(G) = \Omega(\sqrt{n}/\log n)$.[4]

Define a class of partial grids

$$\mathcal{L} = \bigcup_{l\geq 0} \mathcal{L}_l,$$

where $\mathcal{L}_l$ for $l \geq 0$ is defined recursively as follows. We take $\mathcal{L}_0$ to contain one network that is a single node located at $(0,0)$. Then, in order to describe how $\mathcal{L}_{l+1}$ is obtained from $\mathcal{L}_l$, $l \geq 0$, we introduce an operation of *extending* $G \in \mathcal{L}_l$ *at* i, for $i \in \{0,\ldots,l\}$. In this operation, first take G and add $l+2$ new nodes located at coordinates:

$$(0, l+1), (1, l), \ldots, (j, l+1-j), \ldots, (l+1, 0).$$

Call these coordinates the $(l+1)$*-th diagonal*. For each $j \in \{0,\ldots,i\}$ add an edge connecting the nodes $v(j, l-j)$ and $v(j, l-j+1)$, and for each $j \in \{i,\ldots,l\}$ add an edge connecting the nodes $v(j, l-j)$ and $v(j+1, l-j)$. Then, obtain $\mathcal{L}_{l+1}$ as follows: initially take $\mathcal{L}_{l+1}$ to be empty and then for each $G \in \mathcal{L}_l$ and for each $i \in \{0,\ldots,l\}$, obtain a network G' by extending G at i and add G' to $\mathcal{L}_{l+1}$. Notice here that a graph constructed this way is not only a partial grid, but also a tree.

Figure 3 shows a network that was obtained from the corresponding network in $\mathcal{L}_7$ by extending it at 6.

For a network $G \in \mathcal{L}_l$, $l \geq 0$, we define a *characteristic sequence of* G, $\sigma(G)$, as follows. If $l = 0$, then the characteristic sequence of G is empty. If $l > 0$, then take the network G' such that G has been obtained by extending G' at i. The

[4]We remark that we defined the competitive ratio by taking the worst case homebase for A and in the definition of $\mathtt{mcs}(G)$ the most favorable homebase is selected. However, we note that this does not weaken the result of this section as, informally speaking, one may take two copies of each grid obtained in this section, rotate one copy by 180 degrees and merge the two copies at their homebases. Then, we obtain that for each choice of the homebase any algorithm is forced to use $\Omega(\sqrt{n})$ searchers for some grids since in one copy the search is conducted as in our following analysis.

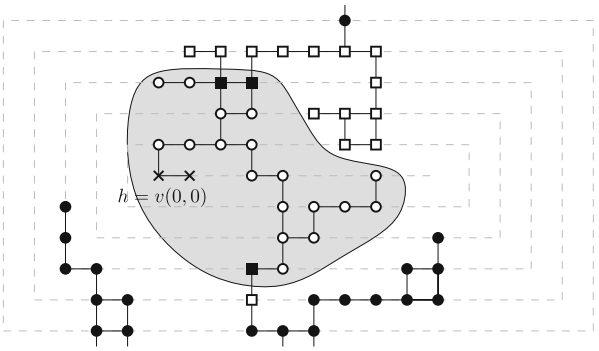


Fig. 2 Some expansions of a checkpoint C (here $\sqrt{n} = 9$); *crosses* denote $C = C\langle 0\rangle$, *thw gray area* covers nodes that belong to $C^+\langle 3\rangle$, *empty squares* denote nodes in $C\langle 4\rangle$ and *dark squares* denote the ones that need to be guarded provided that the gray area consists of the clear nodes. The horizontal dotted line that contains h is the considered frontier

characteristic sequence of G is $\sigma(G)$, constructed by appending to $\sigma(G')$ a new element $v\,(i, l-i-1)$. Note that the characteristic sequence uniquely defines the corresponding network. In other words, G is a binary tree rooted at $v(0,0)$ with $l+1$ leaves, where only the vertices from $\sigma(G)$ have two children. The network introduced in Fig. 3 has characteristic sequence ($v\,(0,0)$, $v\,(1,0)$, $v\,(1,1)$, $v\,(0,3)$, $v\,(3,1)$, $v\,(2,3)$, $v\,(1,5)$, $v\,(6,1)$).

Lemma 1 *For any integer l and for each on-line algorithm A computing a connected monotone search strategy there exists $G \in \mathcal{L}_l$ such that for homebase $v\,(0,0)$ we have $A(G, v\,(0,0)) \geq (l+1)/2$.*

Proof Consider any algorithm A producing a connected monotone search strategy. Run A for each network in $\mathcal{L}_l$ with the homebase $v\,(0,0)$. Note that for each network in $\mathcal{L}_l$, there exist distinct moves $m_1, \ldots, m_l$ such that till the beginning of move m_j, $j \in \{1, \ldots, l\}$, no node on the j-th diagonal has been occupied by a searcher and at the end of m_j some node $v\left(x_j, y_j\right)$ of the j-th diagonal is occupied by a searcher.

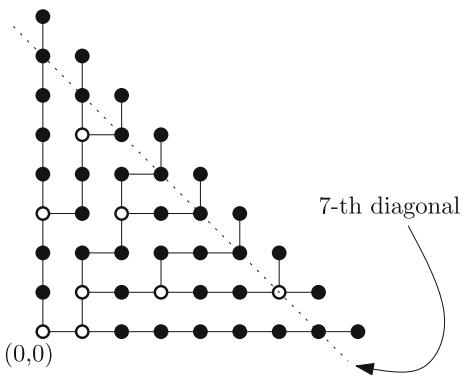


Fig. 3 A network from $\mathcal{L}_8$ obtained from the corresponding network in $\mathcal{L}_7$ by extending it at 6

Consider $G \in \mathcal{L}_l$ such that $\sigma(G) = (v(0,0), v(x_1, y_1), \ldots, v(x_{l-1}, y_{l-1}))$. Informally speaking, whenever the algorithm reaches for the first time a node $v(i, j-i)$ in the j-th diagonal, an *adversary* decides to extend at i the network explored so far, thus always forcing the situation that the first node reached on a diagonal is of degree three.

Note that at the beginning of move m_j, $j \in \{1, \ldots, l\}$, no node of the j-th diagonal has been reached by a searcher and the first j nodes of the characteristic sequence have been reached by searchers. Recall that G is a binary tree.

We analyze the explored part of any graph $G \in \mathcal{L}_l$ at the beginning of the move m_l. All edges incident to the leaves in G are contaminated at this point. On the other hand, all nodes of the characteristic sequence have been visited by searchers till the end of the move $m_l - 1$. Therefore, the contaminated subgraph of G at this point is a collection of paths leading from nodes that are guarded to the leaves. Since there are $l+1$ leaves in G, there are $l+1$ such paths, each such a path needs to have a searcher placed at one of its endpoints (the one that is not a leaf in G) and, by construction of G, any searcher can be present on at most two such endpoints. Thus, at least $(l+1)/2$ nodes need to be occupied by searchers, as required by the lemma. □

Theorem 1 *For each on-line algorithm A computing a connected monotone search strategy there exists an n-node network G with homebase h such that*

$$\frac{A(G,h)}{\mathtt{mcs}(G)} = \Omega(\sqrt{n}/\log n).$$

Proof Observe that each network G in $\mathcal{L}$ is a tree and therefore $\mathtt{mcs}(G) = O(\log(n))$, $n = |V(G)|$ [2, 34]. The theorem follows hence from Lemma 1 and the fact that the length of the characteristic sequence of each network in $\mathcal{L}_l$ is $\Omega(\sqrt{n})$. □

4 The Algorithm

In this section we describe our algorithm that takes an upper bound on the size of the network as an input. Section 4.1 deals with the initialization performed at the beginning of the algorithm. Then, Section 4.2 introduces two procedures used by the algorithm and finally Section 4.3 states the main algorithm.

We point out that the strategy to be computed is monotone. This means that whenever a new node has been reached by some searcher, the node will be guarded as long as it has some incident contaminated edges. After each move performed by searchers, each searcher that occupies a node that does not need to be guarded is said to be *free*. Each node that needs to be guarded is occupied by at least one searcher; if more searchers occupy such a node then all of them except for one are also *free*. Once all incident edges of a guarded node v become clear, the searcher that has been guarding v becomes immediately free. So we do not express this fact explicitly in the algorithm as the above rule is sufficient to partition the searchers into the free and guarding ones at any point of the strategy computed by the algorithm. Before

we start the description of the algorithm, we stress out how we 'reuse' searchers that are free. Whenever the algorithm decides that a searcher needs to perform some action the following decision takes place. If there exists a searcher that is free, then the action is made by an arbitrary such searcher. If there is no free searcher, then a new one is introduced by the algorithm to perform the action. Thus, in our analysis we will count the number of searchers introduced throughout the execution of the algorithm.

If, at some point, no node of the last expansion of some checkpoint needs to be guarded, then we say that the expansion is *empty*.

4.1 Initialization

We start presenting our algorithm by describing initial conditions. Recall that the origin v $(0, 0)$ of the two-dimensional xy coordinate system is situated in the homebase. The initial checkpoint C_0 is the set of nodes of the connected component of $G[F]$ that contains h, where F is the homebase frontier. Thus, initially $|C_0|$ searchers place themselves on all nodes of C_0 (note that the nodes of C_0 induce a path in G). See Fig. 4 for an example.

4.2 Procedures

4.2.1 Procedure `ClearExpansion`

We start with an informal description of the procedure. When a new checkpoint C has been reached, our search strategy 'expands' from C by successively clearing subgraphs $G[C^+\langle i\rangle]$ for $i \in \{1, \ldots, \sqrt{n}\}$. Once all nodes in $C^+\langle i-1\rangle$ are clear for some $0 < i \leq \sqrt{n}$, the transition to reaching the state in which all nodes in $C^+\langle i\rangle$ are clear requires clearing all nodes of the i-th expansion of C. This is done by calling for every guarded node u from $C^+\langle i-1\rangle$ a special procedure (`ModConnectedSearching`, described below), which clears nodes which belong to $C\langle i\rangle$ and 'can be accessed' from u. Procedure `ClearExpansion` makes the above-mentioned calls to `ModConnectedSearching` and uses $O(\sqrt{n})$ searchers in the process.

For clearing all nodes of the i-th expansion of C, provided that $G[C^+\langle i-1\rangle]$ is clear we will use a procedure from [7]. That procedure is more general and it is stated in [7] as Procedure `ConnectedSearching` with its performance stated in Theorem 1 in [7]. Here we give its following reformulation that uses our notation.

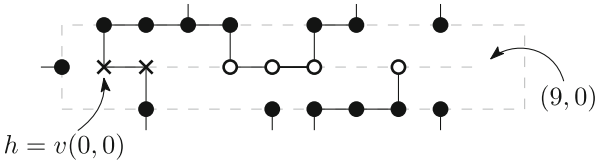


Fig. 4 Exemplary initialization for $\sqrt{n} = 9$; *crosses* denote nodes belonging to the initial checkpoint C_0 and *empty circles* denote nodes that belong to the homebase frontier, but do not fall into C_0.

Theorem 2 [7] *Let F be any frontier and let G' be any connected partial grid whose nodes lie entirely on the rectangles $\mathcal{R}(F,0), \mathcal{R}(F,1), \ldots, \mathcal{R}(F,i)$, $i \geq 0$. There exists an on-line procedure* `ConnectedSearching` *that, starting at an arbitrarily chosen homebase in G', clears G' in a connected and monotone way using $6i+4$ searchers.*

We stress out that the above theorem assumes that the partial grid is entirely contained in the area covered by the rectangles. In other words, the subgraph G' in Theorem 2 has no vertices 'outside' of the specified area. However, while using procedure `ConnectedSearching`, we will be clearing a subgraph of $G[C^+\langle i\rangle]$ that is embedded into the entire partial grid and thus some nodes v of $G[C^+\langle i\rangle]$ have edges leading to neighbors that lie outside of $G[C^+\langle i\rangle]$. If such an edge is already clear, then no recontamination happens for the node v and moreover no searcher used by `ConnectedSearching` for the subgraph of $G[C^+\langle i\rangle]$ needs to stay at v. On the other hand, if such an edge is contaminated (and thus not reached yet by our search strategy), then v needs to be guarded and for that end we place an extra searcher on it that guards v during the remaining execution of `ConnectedSearching`. Note that in the latter case, the node v belongs to $\mathcal{R}(F,i)$, where F is the frontier that contains the nodes of C and therefore there exist $O(\sqrt{n})$ such nodes v. In other words, `ConnectedSearching` is called to clear a certain subgraph contained within $\mathcal{R}(F,i)$ and whenever a node on the rectangle $\mathcal{R}(F,i)$ has a contaminated edge leading outside of the rectangle $\mathcal{R}(F,i)$, then an extra searcher, not accommodated by `ConnectedSearching` in Theorem 2, is introduced to be left behind to guard v. The modification of `ConnectedSearching` that leaves behind a searcher on each such newly reached node of $\mathcal{R}(F,i)$ will be denoted by `ModConnectedSearching`. Note that this procedure is invoked for every guarded node from $C^+\langle i-1\rangle$ in order to clear $C^+\langle i\rangle$, see Fig. 5 for an example.

It follows that it is enough to provide as an input to `ModConnectedSearching`: a node v in $C^+\langle i-1\rangle$ that plays the role of homebase for `ModConnectedSearching`, the frontier F and i. We stress out that

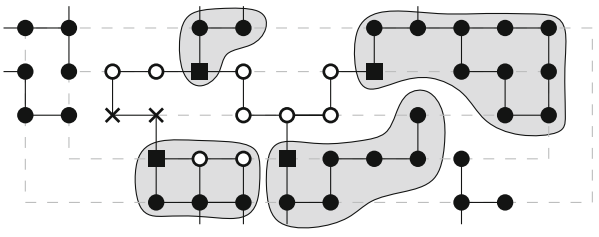


Fig. 5 Example of an execution of procedure `ClearExpansion`; *crosses* denote $C = C\langle 0\rangle$, *empty circles* denote nodes that belong to $C^+\langle 1\rangle$, *dark squares* denote the one that belongs to $C^+\langle 1\rangle$ and for which procedure `ModConnectedSearching` is invoked, *gray areas* show nodes that will be cleared in four calls of `ModConnectedSearching` in order to clear $C\langle 2\rangle$. Note that the empty circles that lie on a gray area are guarded at first, but after one of the calls of `ModConnectedSearching` there is no need to guard them any more, so the procedure is not invoked for them

there are possibly many such nodes v and once one of them is selected, some other such nodes in $C^+\langle i-1\rangle$ may no longer have an incident edge that is contaminated since the call to `ModConnectedSearching` did clear such an edge. However, we assume that `ModConnectedSearching` clears only the maximal connected subgraph that contains v and is induced by contaminated edges only. Thus, once its execution is completed, there may exist another vertex v for which a new call to `ModConnectedSearching` will be made to clear another maximal connected subgraph induced by contaminated edges. See Fig. 5 that illustrates this process: the shaded areas indicate which subgraphs have been actually cleared by subsequent calls to `ModConnectedSearching`. We point out that, alternatively, a single call to `ModConnectedSearching` would suffice if the procedure 'processed' the entire subgraph contained in the expansion $C^+\langle i\rangle$ but this approach would ignore that some subgraph of $C^+\langle i\rangle$ is already clear and hence we present the procedure as having multiple calls to `ModConnectedSearching` that work on contaminated edges only. We note that each checkpoint used in our final algorithm is obtained as follows: some frontier F is selected and then a checkpoint C is created as some set of nodes that belong to F; thus we assume that with C such a unique frontier F is associated.

Thus, this approach guarantees us using at most $6i+4$ searchers to clear $G[C\langle i\rangle]$ and, in addition to those, $2\sqrt{n}+8i$ searchers for guarding nodes lying on $\mathcal{R}(F,i)$, which will be analyzed in more details in Section 5.

To summarize, we give a formal statement of our procedure.

Procedure `ClearExpansion`

Input: An expansion $C\langle i-1\rangle$ with C contained in the frontier F, $i \geq 1$.
Result: Clearing all nodes of $C\langle i\rangle$.
while there exists a node $v \in C\langle i-1\rangle$ with a contaminated neighbor u in $C\langle i\rangle$ **do**
Place $6i+4+2\sqrt{n}+8i$ free searchers on v to be used by `ModConnectedSearching`.
Call `ModConnectedSearching` for v as the homebase, frontier F and integer i.

The following observation summarizes the outcome of an execution of procedure `ClearExpansion`.

Lemma 2 *Suppose that $C\langle i-1\rangle$, that is an expansion contained in a frontier F, where $i \geq 1$, is an input to procedure* `ClearExpansion`. *Suppose that G' is the maximal subgraph contained in $G[C\langle i\rangle]$ and induced by all nodes v such that there exists a path contained in $G[C\langle i\rangle]$ connecting v with a vertex of C. Then, a call to* `ClearExpansion` *with the above input provides the following:*

- *exactly the edges of G' that are contaminated prior to the call are cleared during this call to* `ClearExpansion`,
- *after the call, each vertex of G' with an incident contaminated edge is guarded by a searcher,*
- *all of the nodes from $C\langle i-1\rangle$ are cleaned and do not have to be guarded.*

We point out that there may be an indirect interaction between different checkpoints. Consider an execution of procedure `ClearExpansion` with an input $C\langle i-1\rangle$. At the point of performing this call, there may exist a different checkpoint C' and a corresponding expansion $C'\langle i'\rangle$ such that some searcher is guarding a node v of $C'\langle i'\rangle$ because v has (assuming for simplicity) a single contaminated edge e incident to it. It may happen that during the execution of `ClearExpansion` the edge e becomes clear as it belongs to $C^+\langle i\rangle$. Therefore, this results in a situation that v is not guarded (since it has no incident contaminated edges) and the corresponding searcher becomes free.

4.2.2 Procedure `UpdateCheckpoints`

By definition, if F is some frontier, then $\mathcal{R}(F,\sqrt{n})$ contains 10 frontiers (see Fig. 1). Thus, reaching the $\sqrt{n}$-th expansion $C\langle\sqrt{n}\rangle$ of a checkpoint of F provides a possibility of creating one new checkpoint for each of the above frontiers. Procedure `UpdateCheckpoints`, which takes as an input $C\langle\sqrt{n}\rangle$ and a collection $\mathcal{C}$ of currently present checkpoints, generates these new checkpoints and adds them to $\mathcal{C}$ and removes C from $\mathcal{C}$. Also, if it happens that some newly constructed checkpoint belongs to the same frontier as some existing checkpoint in $\mathcal{C}$ and no expansion for the existing one has been performed yet, then both checkpoints are merged into one. Finally, any checkpoint in $\mathcal{C}$ whose lastly performed expansion is empty is removed from $\mathcal{C}$. We remark that procedure `UpdateCheckpoints` only modifies the collection of checkpoints $\mathcal{C}$ and this procedure performs no clearing moves.

Procedure `UpdateCheckpoints`

Input: $C\langle\sqrt{n}\rangle$ and the collection of all checkpoints $\mathcal{C}$
Result: Updated collection $\mathcal{C}$

```
C ← C \ {C}
C_new ← ∅
for each frontier F on √n-th rectangle of the frontier containing C do
    Let C' consist of all guarded nodes in F.
    If C' ≠ ∅, then C_new ← C_new ∪ {C'}.
for each C'' in C do
    if there exists C' ∈ C_new that is a subset of the same frontier as C'' then
        if C'' is in 0-th expansion then
            C ← C \ {C''}
            Replace C' with C'' ∪ C' in C_new.
C ← C ∪ C_new
for each C' in C do
    if no node in the last expansion of C' is guarded then
        C ← C \ {C'}
```

Thus, to summarize, the 'lifetime' of a checkpoint is as follows. Once the 1-st expansion of C is performed, the checkpoint will remain in the collection $\mathcal{C}$ and

possibly more expansions of C are made (in total at most $\sqrt{n}$ expansion are possible for each checkpoint). A checkpoint C may disappear from $\mathcal{C}$ in three ways:

- when C is in its 0-th expansion and another checkpoint C' appears in the same frontier (thus, C' is in its 0-th expansion) and then the nodes of C are added to C', or
- some expansion of C becomes empty (then C is not removed from $\mathcal{C}$ right away but during the subsequent call to `UpdateCheckpoints`), or
- C reaches its $\sqrt{n}$-th expansion and procedure `UpdateCheckpoints` is called for C (in which case C possibly 'gives birth' to new checkpoints during the execution of `UpdateCheckpoints`).

Our algorithm maintains a collection $\mathcal{C}$ of currently used checkpoints.

4.3 Procedure `GridSearching`

`GridSearching` is the main algorithm, whose aim it is to clear the entire partial grid G in a connected and monotone way. We start with an informal introduction of the algorithm. The search strategy it produces is divided into phases, which will formally be defined in the next section. In each step of the algorithm, a checkpoint with the highest number of nodes that need to be guarded is chosen and the next expansion is made on it. When one of the checkpoints reaches its $\sqrt{n}$-th expansion, then the current phase ends and the procedure `UpdateCheckpoints` is invoked. Thus, the division of search strategy into phases is dictated by consecutive calls to procedure `UpdateCheckpoints`. For an expansion C, in the pseudocode below we write $\delta(C)$ to refer to the set of nodes that belong to the last expansion of C and need to be guarded at a given point.

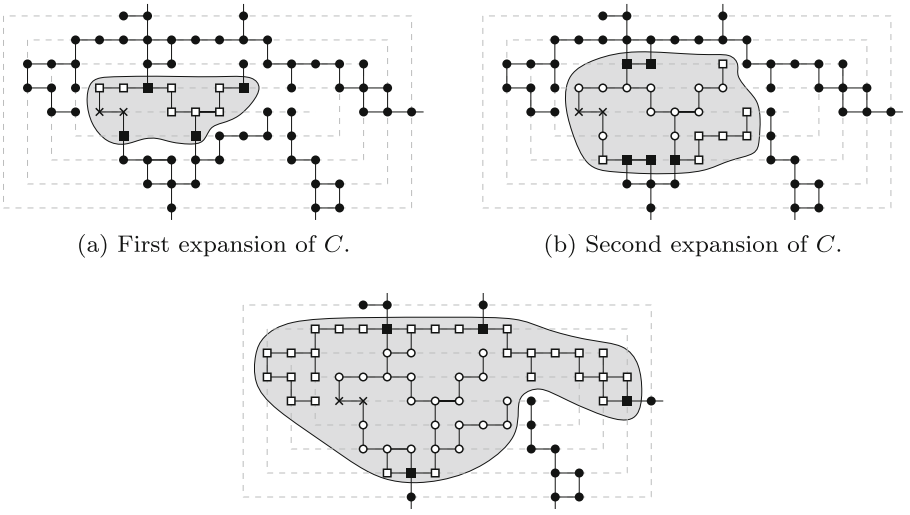
Procedure `GridSearching`

Input: An integer n providing an upper bound on the size of the partial grid G.
Result: A monotone connected search strategy for G.
  Perform the initialization (see Section 4.1).
  while G is not clear **do**
    while no checkpoint has reached its $\sqrt{n}$-th expansion **do**
      Let $C_{\max} \in \mathcal{C}$ be such that $\delta(C_{\max}) \geq \delta(C)$ for each $C \in \mathcal{C}$.
      Let i be the number of expansions of $C_{\max}$ performed so far.
      Invoke `ClearExpansion` for $C_{\max}\langle i \rangle$.
    Invoke `UpdateCheckpoints` for the $\sqrt{n}$-th expansion $C_{\max}\langle\sqrt{n}\rangle$ and for $\mathcal{C}$.

We now introduce a classification of searchers used in our algorithm. This classification will be used in the proof of Theorem 3 but we place it here as it provides another way of describing several actions that take place in the algorithm. We can divide searchers into three groups: *explorers*, *cleaners* and *guards*. Suppose that procedure `ClearExpansion` performs the i-th expansion of a checkpoint $C_{\max}$. Denote by $F_{\max}$ the frontier that contains the nodes in $C_{\max}$. All

searchers located at nodes on the $(i-1)$-th rectangle of $F_{\max}$ that need to be occupied in order to avoid recontamination at the beginning of the call to procedure `ClearExpansion` are named to be guards. The explorers and cleaners are used by algorithm `ModConnectedSearching` called during the execution of procedure `ClearExpansion`. Each time `ModConnectedSearching` reaches a node v on the i-th rectangle of $F_{\max}$ such that v needs to be guarded, the searcher used for guarding v is called an explorer. The searchers used in `ModConnectedSearching` that mimic the movements of searchers in algorithm `ConnectedSearching` are the cleaners. We point out that we do not alter here the behavior of `ClearExpansion` and `ModConnectedSearching` but just assign one of the three categories to each searcher they use. Informally speaking, when explorers protect nodes lying on the i-th rectangle and the guards protect the ones lying on the $(i-1)$-th rectangle of $F_{\max}$, cleaners clear nodes inside the i-th rectangle of $F_{\max}$ (i.e., the remaining nodes of the i-th expansion of $C_{\max}$).

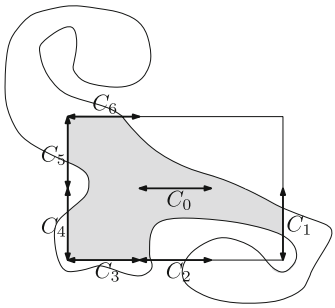
We close this chapter with giving examples of the first three expansions of some checkpoint C, see Fig. 6, and showing how our algorithm clears an exemplary partial grid network, see Fig. 7 (for a formal definition of a phase see the first paragraph of Section 5).



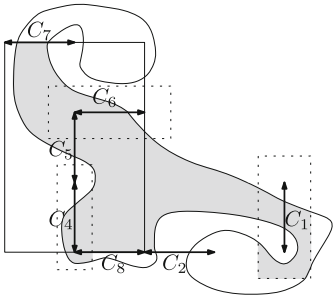
(a) First expansion of C. (b) Second expansion of C.

(c) Third expansion of C.

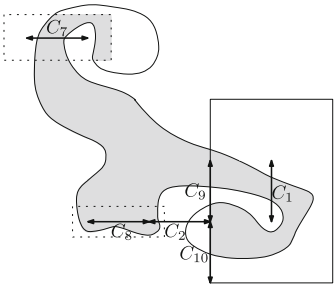
Fig. 6 First three expansions for some checkpoint C (here $\sqrt{n} = 9$); *crosses* denote $C = C\langle 0\rangle$, *empty circles* denote nodes cleared in previous expansions; *squares* denote nodes explored in the current expansion; *dark circles* are nodes not reached yet by the searchers; and *dark squares* denote nodes that need to be guarded at the end of current expansion. *Gray areas* show the clear part of the graph, i.e., $C^{+}\langle i\rangle$ for $i \in \{1, 2, 3\}$



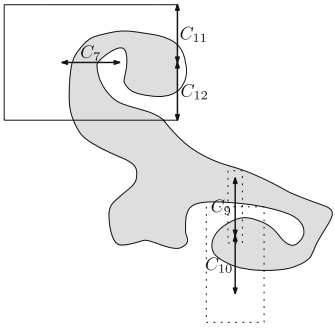
(a) At the end of the first phase C_0 (initial checkpoint) reaches its $\sqrt{n}$-th expansion. Procedure `UpdateCheckpoints` creates 6 new checkpoints and removes C_0 from $\mathcal{C}$, i.e., $\mathcal{C} = \{C_1, C_2, C_3, C_4, C_5, C_6\}$.



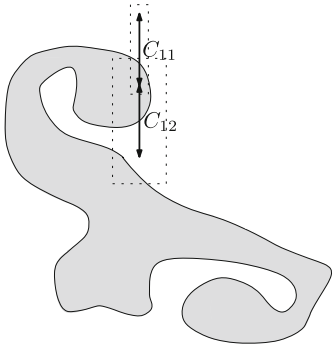
(b) The $\sqrt{n}$-th expansion of C_5 ends the second phase. Checkpoints C_4 and C_6 are removed from $\mathcal{C}$ because (in our example) there is no need to guard any node on theirs expansions. C_3 is in 0-th expansion, so when a new checkpoint C_8 is created on the same frontier, C_3 is removed from $\mathcal{C}$; $\mathcal{C} = \{C_1, C_2, C_7, C_8\}$.



(c) Checkpoint C_1 ends the third phase. Notice that a new checkpoint C_9 emerged 'inside' the already cleared area by C_0; C_2 is removed from $\mathcal{C}$ even if $\sqrt{n}$-th expansion has not been reached but its last expansion has no nodes to be guarded; $\mathcal{C} = \{C_7, C_9, C_{10}\}$.



(d) Checkpoint C_7 ends the fourth phase. Note that a new checkpoint C_{12} emerged on an edge of C_5's $\sqrt{n}$-th expansion; it was not created at the end of the second phase because then there was no access to the contaminated part; $\mathcal{C} = \{C_{11}, C_{12}\}$.



(e) Last phase, in which the rest of the graph is cleared.

Fig. 7 Clearing an exemplary partial grid by procedure `GridSearching`; *gray areas* denote the clear part, *arrows* denote frontiers on which the marked checkpoints lie, *dotted rectangles* around checkpoints denote their current expansions and *solid rectangles* denote the $\sqrt{n}$-th expansions, which end phases

5 Analysis of the Algorithm

By a *step of the algorithm*, or simply a *step*, we mean all searching moves performed during a single iteration of the internal 'while' loop of procedure `GridSearching`. Thus, one step of the algorithm includes all moves produced by one call to procedure `ClearExpansion`. A *phase* of an algorithm consists of all its steps between two consecutive calls to procedure `UpdateCheckpoints`. Note that phases may differ with respect to the number of steps they are made of.

We say that a checkpoint is *present* in a given phase if its last expansion is not empty at the beginning of this phase, i.e., if this checkpoint belongs to $\mathcal{C}$ at the beginning of the phase. Similarly, a checkpoint is *present* in a given step if it is present in the phase to which the step belongs. Thus, in particular, a checkpoint is present in none or in all steps of a given phase. Note that some checkpoints may have empty expansions during a part of a the phase, but they still remain present to the end of the phase; this assumption is made to simplify the analysis of the algorithm.

Let t be a step and v be a node, which needs to be guarded at the beginning of step t. We say that the checkpoint C *owns* v *in step* t if:

- either C owns v in step $t-1$ or
- no checkpoint owns v in step $t-1$ and v belongs to the last expansion of C performed till the end of step $t-1$.

(Intuitively, if a node v is reached by searchers in a step in which an expansion of C occurred, then C owns v as long as v is guarded.) We note that any vertex v can be owned by only one checkpoint. This follows from the fact that our strategy is monotone. More precisely, once v is owned by some checkpoint C in some step, then in the following steps it either continues to be owned by C or v does not need to be guarded. In the latter case v will not be owned by any checkpoint till the end of the strategy. Given a checkpoint C present in a step t, we write $\mathcal{E}(C,t)$ to denote the set of nodes that C owns in step t. The *weight of a checkpoint* C present in a step t is $\omega_t(C) = |\mathcal{E}(C,t)|$ and if a checkpoint C is not present in a step t, then we take $\omega_t(C) = 0$. Note that each guarded node is owned by exactly one checkpoint and hence, for a step t, the sum of weights of all checkpoints present in step t equals the number of nodes that need to be guarded.

The checkpoint $C_{\max}$ selected in a step t (see the pseudocode of Procedure `GridSearching`) is called *active in step* t, or simply *active* if the step is clear from the context or not important. All other checkpoints present in this step are called *inactive*. We define an *active interval* of a checkpoint C to be a maximal interval $[t', t'']$ such that C is active in all steps $t \in \{t', \ldots, t''\}$.

5.1 Single Phase Analysis — How Weights of Checkpoints Evolve

We now prove lemmas that characterize how the weight of a checkpoint changes over time — see Fig. 8 for an exemplary life cycle of a checkpoint. Informally, the weight of a checkpoint C does not grow in intervals in which C is inactive (Lemma 3). Also, the weight of C at the end of an active interval is not greater than at the beginning of

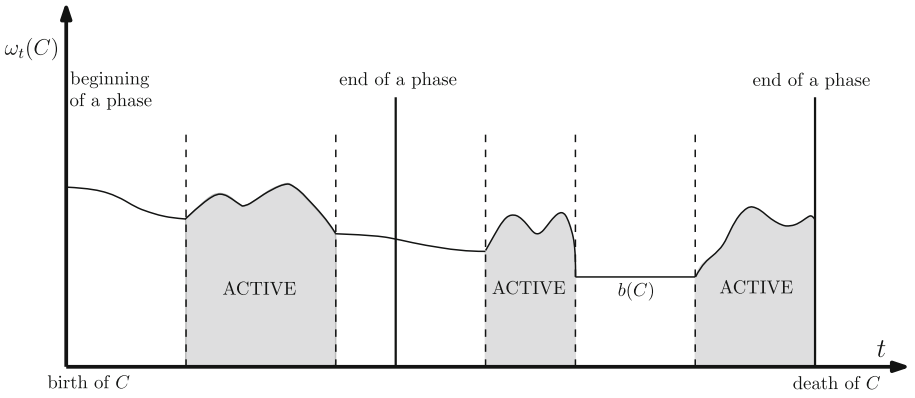


Fig. 8 Exemplary life cycle of a checkpoint C

it (Remark 1); however, no upper bounds except for the trivial one of $O(\sqrt{n})$ can be concluded for the weight of C inside its active interval.

Lemma 3 *If a checkpoint C is present and inactive in a step t, then $\omega_{t+1}(C) \leq \omega_t(C)$.*

Proof It follows directly from the definitions and procedure `ClearExpansion` that the only checkpoint on which an expansion is performed during execution of `ClearExpansion` is the active one. The weight of an inactive checkpoint C can change only in the situation where the active checkpoint in a step t expands on some nodes owned by C. In other words, the weight of C may decrease if C contains in step t nodes that are added to the active checkpoint in step $t+1$. Thus, if t is not the last step of a phase, then the proof is completed.

If t is the last step of some phase, then apart from procedure `ClearExpansion`, procedure `UpdateCheckpoints` is invoked, which affects C in two situations:

- there exists a step t' in the phase that ends such that $\omega_{t'}(C) = 0$. Then, because C cannot be expanded during steps $t', \ldots, t$ of the phase, we get directly that $\omega_{t+1}(C) = \omega_t(C) = 0$.
- C is in its 0-th expansion and a new checkpoint is placed on the same frontier, which implies that C is not present in step $t+1$ and thus $\omega_{t+1}(C) = 0$.

Thus, in all cases we obtain that $\omega_{t+1}(C) \leq \omega_t(C)$. □

We next observe that, informally speaking, once a checkpoint becomes active, it remains active until either the phase ends or its weight decreases. Note that a checkpoint that is active in the last step of the phase is not present in the first step of the next phase, i.e., its weight is then zero, which allows us to state the lemma as follows:

Lemma 4 *Let C be a checkpoint and let $[t', t'']$ be an active interval of C. For every step $t \in \{t', \ldots, t''\}$ it holds $\omega_t(C) \geq \omega_{t''+1}(C)$.*

Proof Obviously, t and t'' must belong to the same phase, because at the end of each phase the active checkpoint is removed from $\mathcal{C}$, i.e., it is no longer present in the next phase.

If t'' is the last step of the phase then the lemma follows, because $\omega_{t''+1}(C) = 0 \leq \omega_{t'}(C)$.

We will now prove that lemma holds when t'' is not the last step of the phase. Let us suppose for a contradiction that $\omega_{t''+1}(C) > \omega_t(C)$. From the assumptions of the lemma and definition of an active interval we get that C is not the active checkpoint in step $t''+1$. Because we are still in the same phase, it means that there must exist a checkpoint C^* such that $\omega_{t''+1}(C^*) \geq \omega_{t''+1}(C)$. Moreover from Lemma 3 we know, that because C^* was inactive from step t' to t'', it holds $\omega_t(C^*) \geq \omega_{t''}(C^*) \geq \omega_{t''+1}(C^*)$. This gives us

$$\omega_t(C^*) \geq \omega_{t''+1}(C^*) \geq \omega_{t''+1}(C) > \omega_t(C),$$

which is in a contradiction to the assumption that C is the active checkpoint in step t. □

Remark 1 Let C be a checkpoint and let $[t', t'']$ be an active interval of C. Then, $\omega_{t''+1}(C) \leq \omega_{t'}(C)$.

We now conclude from the two previous lemmas about the weight of inactive checkpoints in the ends of the consecutive phases.

Lemma 5 *Suppose that a phase ends in a step t' and the next one ends in a step t''. If a checkpoint C is inactive (but present) in steps t' and t'', then $\omega_{t''}(C) \leq \omega_{t'}(C)$.*

Proof Each checkpoint C can be active or inactive in different steps during the whole phase. If in some step $t \in \{t', \ldots, t''\}$ a checkpoint C is inactive then from Lemma 3 we have that its weight will not increase, i.e., $\omega_t(C) \geq \omega_{t+1}(C)$. On the other hand, Lemma 4 guarantees us, that the weight of an active checkpoint cannot be greater after its active interval than at the beginning. □

5.2 How Many Nodes are Explored by a Checkpoint?

Define a *bottleneck* of a checkpoint C, denoted by $b(C)$ to be its minimum weight taken over all steps in which C was present. (Note that a checkpoint may be present in many consecutive phases, see Fig. 8.)

Suppose that a node v has been reached by a searcher for the first time in a step t. Let C be the active checkpoint in step t. We say that v has been *explored by* C.

If an expansion of an active checkpoint C reaches in a step t a node u already explored by some checkpoint C', then in most situations u does not need to be guarded. However there might occur a "corner situation" when u still needs to be guarded in order to avoid contamination. In such case, the algorithm clearly needs one searcher on u to guard it and so it is counted in our analysis due to the 'ownership' relation used in the definition of the weight of a checkpoint.

The next lemma states a lower bound on the number of nodes explored by a checkpoint reaching its last expansion.

Lemma 6 *Suppose that a phase ends in a step t. Let C be the active checkpoint in step t. The number of nodes explored by C in all steps is at least $b(C)\sqrt{n}$.*

Proof First let us make a remark that nodes can be only explored by C during execution of procedure `ClearExpansion` that took C as an input, i.e., when C is active. Let us denote by S the set of all nodes explored by C.

Because C is active in the last step of the phase, it had to be active in exactly $\sqrt{n}$ steps in total, which can be contained in several past phases. Let $t_1, t_2, \ldots, t_{\sqrt{n}} = t$ be all steps in which C is active. Note that

$$\bigcup_{i=1}^{\sqrt{n}} \mathcal{E}(C, t_i) \subseteq S$$

and $\mathcal{E}(C, t_i) \cap \mathcal{E}(C, t_j) = \emptyset$ for $i \neq j$. The latter follows directly from the fact that nodes in $\mathcal{E}(C, t_i)$ and $\mathcal{E}(C, t_j)$ belong to different rectangles of the frontier containing C for $i \neq j$. (Recall that $|\mathcal{E}(C, t)| = \omega_t(C)$ for each step t.) Also from the definition of the bottleneck, we get that $b(C) \leq \omega_{t_i}(C)$ for each $i \in \{1, \ldots, \sqrt{n}\}$ and hence we conclude that:

$$|S| \geq \sum_{i=1}^{\sqrt{n}} \omega_{t_i}(C) \geq b(C)\sqrt{n}.$$

□

We now give an upper bound on the weight of each inactive checkpoint at the end of a phase.

Lemma 7 *Suppose that a phase ends in a step t. Let $C_1, \ldots, C_l$ be all checkpoints present in this phase, where C_1 is the active checkpoint in step t. Then, $b(C_1) \geq \omega_t(C_j)$ for each $j \in \{2, \ldots, l\}$.*

Proof Let us denote by t' the last step in which $\omega_{t'}(C_1) = b(C_1)$. If $t' = t$ then the lemma follows strictly from the definition of an active checkpoint. We will now prove that lemma stands also when $t' < t$.

Suppose that t' and t do not belong to the same active interval of C_1. From the Lemma 4 we know that $\omega_{t''}(C_1) = b(C_1)$ occurs for some t'' that does not belong to an active interval. Moreover from Remark 1 we get that every next active interval will need to start and finish on the same weight as the bottleneck, which is in contradiction that t' is the last step when $b(C_1)$ occurred.

Hence t and t' are part of the same active interval of C_1. Then, we get from Lemma 3 and the fact that C_1 is active in step t':

$$\omega_t(C_j) \leq \omega_{t'}(C_j) \leq \omega_{t'}(C_1) = b(C_1), \quad j \in \{2, \ldots, l\},$$

which finishes our proof. □

Let us introduce a relation $\prec$ on a set of checkpoints. Whenever $C \prec C'$, we say that C is a *predecessor* of C' and C' is a *successor* of C. We stress out that the construction depends on the execution of the algorithm, namely only checkpoints that appear in some step are considered, and the division of the steps into phases shapes the relation. More precisely, the relation is defined only for checkpoints added to the set $\mathcal{C}$ during all executions of procedure `UpdateCheckpoints`. To construct the relation we iterate over the consecutive phases of the algorithm. Initially the relation is empty and once the construction is done for each phase smaller than i, we perform the following for phase i. Let C be the active checkpoint in the last step of phase i. Let $C_1, \ldots, C_l$ be all checkpoints, different from C, that have no successors so far and were added to $\mathcal{C}$ till the end of phase $i-1$ (including the last step). Then, let $C_j \prec C$ for each $j \in \{1, \ldots, l\}$.

An important property of our algorithm is that each checkpoint may have only a constant number of predecessors:

Lemma 8 *Each checkpoint has at most* 10 *predecessors.*

Proof A checkpoint C can only once be active in the last step of some phase i, because after that it will not be present in any later phases. At the end of phase i the only checkpoints that do not have any successors are the ones that were constructed by the procedure `UpdateCheckpoints` at the end of phase $i-1$. There are at most 10 such checkpoints. □

5.3 The Algorithm Uses $O(\sqrt{n})$ Searchers in Total

We now bound the total weight of all checkpoints at the end of each phase — note that this bounds the total number of searchers used for guarding at the end of a phase. A high level intuition behind the proof of Lemma 9 is as follows. Due to Lemma 6, each checkpoint C that is active in the last step of a phase explores at least $b(C)\sqrt{n}$ nodes in total. Therefore, the sum of bottlenecks of all such checkpoints C cannot exceed $\sqrt{n}$. Moreover, C can have at most 10 predecessors and hence the sum of weights of those predecessors is bounded by $10b(C)$ according to Lemma 7. Since each checkpoint (except the one that is active in the last step of a given phase) is a predecessor of some checkpoint that is active in the last step of some phase, we bound the sum of all weights of all such checkpoints present in a given phase by $10\sqrt{n}$.

Lemma 9 *Suppose that $C_1, \ldots, C_l$ are all checkpoints present in a phase that ends in step t, where C_1 is active in step t. Then,*

$$\sum_{i=1}^{l} \omega_t(C_i) \leq \omega_t(C_1) + 10\sqrt{n}.$$

Proof Suppose that phase j ends in step t. Let t_i be the last step of phase i and let C_i^0 be the active checkpoint in step t_i for each $i \in \{0, \ldots, j\}$. We denote by s the number of nodes visited by searchers till the end of step $t = t_j$. From Lemma 6 and

the fact that the number of all nodes n is at least s we have:

$$n \geq s \geq \sum_{i=0}^{j} b(C_i^0)\sqrt{n} \quad \Rightarrow \quad 10\sqrt{n} \geq 10 \sum_{i=0}^{j} b(C_i^0). \tag{1}$$

From Lemma 8 we have that the checkpoints $C_0^0, \ldots, C_j^0$ can have at most 10 predecessors. From the definition, they are constructed (i.e., added to collection $\mathcal{C}$ during the execution of procedure `UpdateCheckpoints`) at the beginning of the first step of a phase at the end of which their successor is active. Let us denote by $C_i^1, \ldots, C_i^{l_i}, 0 \leq l_i \leq 10$, the predecessors of C_i^0 for each $i \in \{0, \ldots, j\}$ (by $l_i = 0$ we denote that C_i^0 has no predecessors). From Lemma 7 we have:

$$\sum_{k=1}^{l_i} \omega_{t_i}(C_i^k) \leq 10b(C_i^0), \quad i \in \{0, \ldots, j\}. \tag{2}$$

Lemma 5 assures us that weights of inactive checkpoints will not be greater at the end of the next phase than they are in the last step of current phase:

$$\omega_t(C_i^k) = \omega_{t_j}(C_i^k) \leq \omega_{t_{j-1}}(C_i^k) \leq \cdots \leq \omega_{t_i}(C_i^k), \ i \in \{0, \ldots, j\}; \ k \in \{1, \ldots, l_i\}. \tag{3}$$

Because

$$\{C_1, \ldots, C_l\} \subseteq \{C_j^0\} \cup \left\{ C_i^k \middle| k \in \{1, \ldots, l_i\}, i \in \{0, \ldots, j\} \right\},$$

we can conclude from (3), (2) and (1) (in this order) that:

$$\begin{aligned} \sum_{i=1}^{l} \omega_t(C_i) &\leq \omega_t(C_j^0) + \sum_{i=0}^{j} \sum_{k=1}^{l_i} \omega_t(C_i^k) \\ &\leq \omega_t(C_j^0) + \sum_{i=0}^{j} \sum_{k=1}^{l_i} \omega_{t_i}(C_i^k) \\ &\leq \omega_t(C_j^0) + \sum_{i=0}^{j} 10b(C_i^0) \\ &\leq \omega_t(C_j^0) + 10\sqrt{n}. \end{aligned}$$

□

Theorem 3 *Given an upper bound n of the size of the network as an input, the algorithm* `GridSearching` *clears in a connected and monotone way any unknown underlying partial grid network using $O(\sqrt{n})$ searchers.*

Proof At first let us notice that the algorithm `GridSearching` ends with the whole network cleared. Indeed, as long as there are contaminated nodes, it will continue clearing next expansions of the checkpoints. Because no recontamination takes place, it eventually terminates. We will bound the number of searchers s used by a single call to procedure `ClearExpansion` and the total number of searchers s' used

for guarding at the end of any step of the algorithm. Note that $s + s'$ bounds the total number of searchers used by `GridSearching`. In the proof we refer to the classification of searchers into explorers, cleaners and guards introduced in Section 4.

We first analyze procedure `ClearExpansion` to give an upper bound on s. The fact that each rectangle of a frontier contains at most $10\sqrt{n}$ nodes and Theorem 2 give that:

$$\begin{aligned} \text{number of explorers} &\leq 10\sqrt{n}, \\ \text{number of cleaners} &\leq 6\sqrt{n} + 4. \end{aligned}$$

Thus,

$$s \leq 16\sqrt{n} + 4.$$

The guards used to protect nodes lying on the $(i-1)$-th rectangle are accounted for during the estimation of s' below.

We now bound the maximal number of searchers used for guarding at the end of each step t of our search strategy, which we denote by g_t. It is easy to see that $g_t \leq 10\sqrt{n}$ if t belongs to phase 0.

Let us now take any step t that belongs to an i-th phase, where $i > 0$ and denote by t' the last step of the phase $i-1$ and by C the active checkpoint in step t'. From Lemma 9 we know that $g_{t'} \leq \omega_{t'}(C) + 10\sqrt{n} \leq 20\sqrt{n}$. The latter inequality follows from the fact that all nodes in $\mathcal{E}(C, t')$ belong to the j-th rectangle of the frontier that contains C, $j \leq \sqrt{n}$, and the number of nodes in this rectangle is at most $10\sqrt{n}$.

We know now that every phase starts with at most $20\sqrt{n}$ guards. If t is the first step of an active interval of some checkpoint, then by Lemma 3 and Remark 1 we have that $g_t \leq g_{t'} \leq 20\sqrt{n}$. But if t is a step inside some active interval, then an active checkpoint can reach at most $10\sqrt{n}$ new nodes that need to be guarded. Note that by Lemma 2, the nodes of subsequent expansions of a checkpoint that need to be guarded do not accumulate, that is, we only guard the one of the last expansion. Because in one step only one checkpoint can be active that leads us to conclusion that for every step t we have $g_t \leq 30\sqrt{n}$. Therefore, we obtain that $s' \leq 30\sqrt{n}$.

Thus, we obtain $s + s' \leq 46\sqrt{n} + 4 = O(\sqrt{n})$ as required. □

6 Unknown Size of the Graph

The algorithm we have described needs to know an upper bound on the size of the underlying partial grid network G. In this section we design a procedure called `ModGridSearching` that performs the search using $O(\sqrt{n})$ searchers and having no prior information on the network. The procedure is based on a standard technique: guessing an upper bound on n by doubling potential estimate each time. More about applications of the doubling technique in designing on-line and off-line approximation algorithms can be found in [9].

The procedure `ModGridSearching` is composed of a certain number of *rounds*. In round i, procedure `GridSearching` first introduces $c\sqrt{2^i}$ new searchers called *i-th team*, where c is the constant from the asymptotic notation in Theorem 3. Then, a call to `GridSearching` is made, where procedure

`GridSearching` is using only the searchers of the i-th team. The outcome can be twofold. The procedure may succeed in searching the entire graph and in such case the i-th round is the last one and `ModGridSearching` is completed, or the procedure may encounter a situation in which it would be forced to use more than $c\sqrt{2^i}$ searchers to continue. In such case `GridSearching` stops, the i-th round ends and the $(i+1)$-th round will follow. Once the i-th round is completed, the searchers of the i-th team stay idle indefinitely. We point out that during the execution of an i-th round, $i > 1$, procedure `GridSearching` using the searchers of the i-th team is ignoring the fact that the network may be partially clear as a result of the work done in previous rounds. Moreover, the searchers of j-th team for each $j < i$ are not used and thus also ignored during i-th round.

We close this section by giving an upper bound on the number of searchers that need to be used in the presented modified version of our algorithm.

Theorem 4 *The on-line algorithm* `ModGridSearching` *clears (starting at an arbitrary homebase) in a connected and monotone way any unknown underlying partial grid network using $O(\sqrt{n})$ searchers. The algorithm receives no prior information on the network.*

Proof Let n be the number of nodes of the partial grid network, which is unknown to our procedure. The number of rounds m fulfills $2^{m-1} < n \leq 2^m$, i.e., $m = \lceil \log_2 n \rceil$. At the end of i-th round, $c\sqrt{2^i}$ searchers need to stay in their last positions till the end of our procedure and are not used in subsequent rounds. This means that the total number of searcher s is upper bounded by a sum of searchers used in every round:

$$s \leq c\sqrt{2} + c\sqrt{2^2} + \cdots + c\sqrt{2^{\lceil \log_2 n \rceil}} = c \sum_{j=1}^{\lceil \log_2 n \rceil} \left(\sqrt{2}\right)^j$$

$$= \sqrt{2}c \frac{1 - \sqrt{2}^{\lceil \log_2 n \rceil}}{1 - \sqrt{2}} = \frac{\sqrt{2}c}{\sqrt{2} - 1} \left(\sqrt{2^{\lceil \log_2 n \rceil}} - 1\right).$$

Because $\sqrt{n} \leq \sqrt{2^{\lceil \log_2 n \rceil}} < \sqrt{2n}$, we conclude

$$s < \frac{\sqrt{2}c}{\sqrt{2} - 1} \left(\sqrt{2n} - 1\right) \quad \Rightarrow \quad s = O(\sqrt{n}).$$

□

7 Conclusions

7.1 Motivation

There exists a number of studies of graph searching problems in the graph-theoretic context. Much less is known for geometric scenarios. It turns out that the geometric

(or continuous) analogue of graph searching is challenging to analyze. More precisely, in the recently introduced continuous version [29, 33] the input geometric shape is searched by using line segments or curves (that form a barrier separating contaminated and clear area) instead of searchers. The corresponding optimization criterion is then the total length of this barrier. It can be observed that computing optimal strategies even for some simple shapes turns out to be quite non-trivial [33].

The class of graphs we have selected to study in this work is motivated by the following arguments. First, on-line (monotone) searching turns out to be difficult in terms of achievable upper bound on the number of searchers even in simple topologies like trees. This suggest that some additional information is needed to perform on-line search efficiently and our work shows that, informally speaking, a two-dimensional sense of direction is enough to search a graph in asymptotically almost optimal way. Our second motivation comes from approaching the problem of geometric search by considering its discrete analogues, i.e., by modeling via graph theory. We give a short sketch to give an overview as the problem of modeling is out of scope of this work and we only refer to some recent works on the subject [1, 4, 29, 33]. Consider a continuous search problem in which k searchers initially placed at the same location need to capture the fugitive hiding in an arbitrary polygon that possibly has holes. The polygon is not known a priori to the searchers. The fugitive is considered captured in time t when it is located at distance at most r from some searcher at time point t. (The distance r can be related to physical dimensions of searchers and/or their visibility range, etc.)

Consider the following transition from the above continuous searching problem of a polygon to a discrete one. Overlap the coordinate system with the polygon in such a way that the origin coincides with the original placement of the searchers. Then, place nodes on all points with coordinates, which are multiples of r and lie in the polygon. Connect two nodes with an edge if the edge is contained in the polygon. In this way we obtain a partial grid network. In this brief sketch we omit potential problems that may arise in such modeling, like obtaining disconnected networks or having 'blind spots', i.e., points in the polygon that cannot be cleared by using the above nodes and edges only. We say that a partial grid network G *covers* the polygon

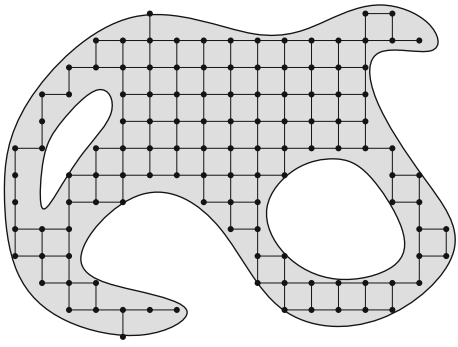


Fig. 9 An example of the construction of a partial grid network

if G is connected and for each point p in the polygon there exist a node of G in distance at most r from p.

See Fig. 9 for an example.

Note that any search strategy $\mathcal{S}'$ for a polygon P can be used to obtain a search strategy $\mathcal{S}$ for underlying partial grid network G as follows. For each searcher s used in $\mathcal{S}'$ introduce four searchers $s_1, \ldots, s_4$ that will 'mimic' its movements by going along edges of G. More precisely, the searchers $s_1, \ldots, s_4$ will ensure that at any point, if s is located at a point (x, y), then $s_1, \ldots, s_4$ will reside on nodes with coordinates $(\lfloor x/r \rfloor, \lfloor y/r \rfloor)$, $(\lfloor x/r \rfloor, \lceil y/r \rceil)$, $(\lceil x/r \rceil, \lfloor y/r \rfloor)$, $(\lceil x/r \rceil, \lceil y/r \rceil)$. In this way, area protected by s in $\mathcal{S}'$ is always protected by four searchers in $\mathcal{S}$. This allows us to state the following.

Observation 1 Let P be a polygon and let G by an underlying partial grid network that covers P. Then, there exists a search strategy for G using k searchers such that its execution in G results in clearing P and $k = O(p)$, where p is the minimum number of searchers required for clearing P (in a continuous way).

7.2 Open problems

In view of the lower bound shown in [28] that even in such simple networks as trees each distributed or on-line algorithm may be forced to use $\Omega(n/\log n)$ times more searchers than the connected search number of the underlying network, one possible line of research is to restrict attention to specific topologies that allow to obtain algorithms with good provable upper bounds. This work gives one such an example. An interesting research direction is to find other non-trivial settings in which distributed or on-line search can be conducted efficiently. Also, we leave a logarithmic gap in our approximation ratio. Since there exist grids that require $\Omega(\sqrt{n})$ searchers the gap can be possibly closed by analyzing the grids that require few (e.g. $O(\log n)$) searchers.

The above questions related to network topologies can be stated more generally: what properties of the on-line model are crucial for such a search for fast and invisible fugitive to be efficient? This work and also a recent one [7] suggest that a 'sense of direction' may be one such a factor. Possibly interesting directions may be to analyze the influence of visibility on search scenarios.

We finally note that the only optimization criterion that was of interest in this work is the number of searchers. This coincides with the research done in off-line search problems where this was the most important criterion giving nice ties between graph searching theory and structural graph theory. However, one may consider adding different optimization criteria like time (defined as the maximum number of synchronized steps) or the total distance (the total number of moves performed by all searchers).

Acknowledgments Research partially supported by National Science Centre (Poland) grant number 2015/17/B/ST6/01887. An extended abstract of this work appeared in Proceedings of The 15th Workshop on Approximation and Online Algorithms (WAOA 2017).

References

1. Altshuler, Y., Yanovski, V., Wagner, I.A., Bruckstein, A.M.: Multi-agent cooperative cleaning of expanding domains. I. J. Robot. Res. **30**(8), 1037–1071 (2011)
2. Barrière, L., Flocchini, P., Fomin, F.V., Fraigniaud, P., Nisse, N., Santoro, N., Thilikos, D.M.: Connected graph searching. Inf. Comput. **219**, 1–16 (2012)
3. Best, M.J., Gupta, A., Thilikos, D.M., Zoros, D.: Contraction obstructions for connected graph searching. Discrete Applied Mathematics. https://doi.org/10.1016/j.dam.2015.07.036 (2015)
4. Bhadauria, D., Klein, K., Isler, V., Suri, S.: Capturing an evader in polygonal environments with obstacles The full visibility case. I. J. Robot. Res. **31**(10), 1176–1189 (2012)
5. Blin, L., Burman, J., Nisse, N.: Perpetual graph searching. Technical report INRIA ⟨hal-00675233⟩ (2012)
6. Blin, L., Fraigniaud, P., Nisse, N., Vial, S.: Distributed chasing of network intruders. Theor Comput. Sci. **399**(1-2), 12–37 (2008)
7. Borowiecki, P., Dereniowski, D., Kuszner, L.: Distributed graph searching with a sense of direction. Distrib. Comput. **28**(3), 155–170 (2015)
8. Cai, J., Flocchini, P., Santoro, N.: Decontaminating a network from a black virus. IJNC **4**(1), 151–173 (2014)
9. Chrobak, M., Kenyon-Mathieu, C.: Sigact news online algorithms column 10: competitiveness via doubling. ACM SIGACT News **37**(4), 115–126 (2006)
10. Chung, T.H., Hollinger, G.A., Isler, V.: Search and pursuit-evasion in mobile robotics - A survey. Auton. Robot. **31**(4), 299–316 (2011)
11. Daadaa, Y.: Network Decontamination with Temporal immunity, Master Thesis. Master Thesis. University of Ottawa, Ottawa (2012)
12. Daadaa, Y., Flocchini, P., Zaguia, N.: Network decontamination with temporal immunity by cellular automata. In: ACRI'10: Proceedings of the 9th International Conference on Cellular Automata for Research and Industry, Ascoli Piceno, pp. 287–299 (2010)
13. D'Angelo, G., Navarra, A., Nisse, N.: Gathering and exclusive searching on rings under minimal assumptions. In: ICDCN '14: Proc. of the 15th International Conference on Distributed Computing and Networking, Coimbatore, pp. 149–164 (2014)
14. Dereniowski, D.: Connected searching of weighted trees. Theor. Comp. Sci. **412**, 5700–5713 (2011)
15. Dereniowski, D.: Approximate search strategies for weighted trees. Theor. Comput. Sci. **463**, 96–113 (2012)
16. Dereniowski, D.: From pathwidth to connected pathwidth. SIAM J. Discret. Math. **26**(4), 1709–1732 (2012)
17. Durham, J.W., Franchi, A., Bullo, F.: Distributed pursuit-evasion without mapping or global localization via local frontiers. Auton. Robot. **32**(1), 81–95 (2012)
18. Ellis, J., Warren, R.: Lower bounds on the pathwidth of some grid-like graphs. Discret. Appl. Math. **156**(5), 545–555 (2008)
19. Flocchini, P., Huang, M.J., Luccio, F.L.: Decontaminating chordal rings and tori using mobile agents. Int. J. Found. Comput. Sci. **18**(3), 547–563 (2007)
20. Flocchini, P., Huang, M.J., Luccio, F.L.: Decontamination of hypercubes by mobile agents. Networks **52**(3), 167–178 (2008)
21. Flocchini, P., Luccio, F., Pagli, L., Santoro, N.: Optimal network decontamination with threshold immunity. In: CIAC'13: Proc. of the 8th International Conference on Algorithms and Complexity, Barcelona, pp. 234–245 (2013)
22. Flocchini, P., Luccio, F., Pagli, L., Santoro, N.: Network decontamination under m-immunity. Discret. Appl. Math. **201**, 114–129 (2016)

23. Flocchini, P., Mans, B., Santoro, N.: Tree decontamination with temporary immunity. In: Algorithms and Computation, 19th International Symposium, ISAAC, 2008, Gold Coast, Australia, pp. 330–341. Proceedings (2008)
24. Fomin, F.V., Thilikos, D.M.: An annotated bibliography on guaranteed graph searching. Theor. Comput. Sci. **399**(3), 236–245 (2008)
25. Fraigniaud, P., Ilcinkas, D., Pelc, A.: Oracle size: a new measure of difficulty for communication tasks. In: PODC'06: Proc. of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, pp. 179–187 (2006)
26. Gonċalves, V.C.F., Lima, P.M.V., Maculan, N., Franċa, F.M.G.: A distributed dynamics for web-graph decontamination. In: ISoLA '10 Proceedings of the 4th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation, Heraklion, Crete, pp. 462–472 (2010)
27. Hollinger, G.A., Singh, S., Djugash, J., Kehagias, A.: Efficient multi-robot search for a moving target. I. J. Robot. Res. **28**(2), 201–219 (2009)
28. Ilcinkas, D., Nisse, N., Soguet, D.: The cost of monotonicity in distributed graph searching. Distrib. Comput. **22**(2), 117–127 (2009)
29. Karaivanov, B., Markov, M., Snoeyink, J., Vassilev, T.S.: Decontaminating planar regions by sweeping with barrier curves. In: CCCG '14: Proceedings of the 26th Canadian Conference on Computational Geometry (2014)
30. Kolling, A., Carpin, S.: Multi-robot pursuit-evasion without maps. In: ICRA'10: Proceedings of IEEE International Conference on Robotics and Automation, pp. 3045–3051 (2010)
31. LaPaugh, A.S.: Recontamination does not help to search a graph. J. ACM **40**(2), 224–245 (1993)
32. Luccio, F., Pagli, L., Santoro, N.: Network decontamination in presence of local immunity. Int. J. Found Comput. Sci. **18**(3), 457–474 (2007)
33. Markov, M., Haralampiev, V., Georgiev, G.: Lower bounds on the directed sweepwidth of planar shapes. Serdica J. Comput. **9**(2), 151–166 (2015)
34. Megiddo, N., Hakimi, S.L., Garey, M.R., Johnson, D.S., Papadimitriou, C.H.: The complexity of searching a graph. J. ACM **35**(1), 18–44 (1988)
35. Moraveji, R., Sarbazi-azad, H., Zomaya, A.Y..: Performance modeling of cartesian product networks. J. Parallel Distrib. Comput. **71**(1), 105–113 (2011)
36. Nisse, N., Soguet, D.: Graph searching with advice. Theor. Comput. Sci. **410**(14), 1307–1318 (2009)
37. Parsons, T.D.: Pursuit-evasion in a graph. In: Theory and Applications of Graphs, Lecture Notes in Mathematics, vol. 642, pp. 426–441. Springer (1978)
38. Petrov, N.N.: A problem of pursuit in the absence of information on the pursued. Differ. Uravneniya **18**, 1345–1352 (1982)
39. Raboin, E., Kuter, U., Nau, D.S.: Generating strategies for multi-agent pursuit-evasion games in partially observable euclidean space. In: AAMAS '12 Proceedings of the International Conference on Autonomous Agents and Multiagent Systems, Valencia, Spain, pp. 1201–1202 (2012)
40. Robin, C., Lacroix, S.: Multi-robot target detection and tracking: taxonomy and survey. Auton. Robot. **40**(4), 729–760 (2016)
41. Rodríguez, S., Denny, J., Burgos, J., Mahadevan, A., Manavi, K., Murray, L., Kodochygov, A., Zourntos, T., Amato, N.M.: Toward realistic pursuit-evasion using a roadmap-based approach. In: ICRA '11: Proceedings of the IEEE international conference on robotics and automation, Shanghai, China, pp. 1738–1745 (2011)
42. Sachs, S., LaValle, S.M., Rajko, S.: Visibility-based pursuit-evasion in an unknown planar environment. I. J Robot. Res. **23**(1), 3–26 (2004)
43. Stiffler, N.M., O'Kane, J.M.: A complete algorithm for visibility-based pursuit-evasion with multiple pursuers. In: ICRA '14: Proceedings of the IEEE International Conference on Robotics and Automation, Hong Kong, China, pp. 1660–1667 (2014)
44. Yang, B., Dyer, D., Alspach, B.: Sweeping graphs with large clique number. Discret. Math. **309**(18), 5770–5780 (2009)