Theoretical Computer Science 605 (2015) 129–145

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Explore and repair graphs with black holes using mobile entities ☆, ☆☆

Mattia D'Emidio [a,b,*], Daniele Frigioni [a], Alfredo Navarra [c]

[a] Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via Gronchi, 18, I-67100, L'Aquila, Italy
[b] Gran Sasso Science Institute (GSSI), Viale Francesco Crispi, I-67100, L'Aquila, Italy
[c] Department of Mathematics and Computer Science, University of Perugia, Via Vanvitelli, 1, I-06123, Perugia, Italy

ARTICLE INFO

Article history:
Received 9 April 2014
Received in revised form 31 August 2015
Accepted 1 September 2015
Available online 8 September 2015
Communicated by C. Kaklamanis

Keywords:
Mobile entities
Black-holes
Exploration algorithms
Time complexity
Experiments

ABSTRACT

In this paper, we study the problem of mobile entities that synchronously have to explore and repair a graph with faulty nodes, usually called *black-holes*, that destroy any entering entity. We consider the scenario where the destruction of an entity by means of a black-hole also affects all the entities within a fixed range r (in terms of number of edges), while the black-hole disappears. Clearly, if there are b black-holes in the graph, then $k \geq b$ entities are necessary to remove all of them from that graph. We ask for the minimum number of synchronous steps needed to make safe all the graph.

The results of this paper are both theoretical and experimental, and can be summarized as follows. From the theoretical point of view, first we show that the problem is NP-hard even for $b = k = 1$. Then, we provide a general lower bound holding when $r \geq 0$ and a higher one for the case of $r > 0$. We then consider the case of $r \leq 1$. We propose an optimal solution holding when k is unbounded, that is, an infinite number of robots is available. Then, we provide three different exploration strategies, named *snake*, *scout*, and *parallel-scout*, respectively, for the case of bounded k, that is, the number of robots is fixed a priori. The three strategies are then analyzed according to the time complexity with respect to the lower bound. From the experimental point of view, we implemented the three strategies and tested them on different scenarios with the aim of assessing their practical performance. The experiments confirm the theoretical analysis and show that *parallel-scout* is always by far the best exploration strategy in practice.

1. Introduction

The exploration task of unknown graphs by means of mobile entities has been widely considered during the last years. The increasing interest to the problem comes from the variety of applications that it meets. In robotics, it might be very useful to let a robot or a team of robots to explore dangerous or impervious zones. In networking, software agents might

☆ A preliminary version of this paper has been presented at the 12th International Conference on Ad Hoc Networks and Wireless (ADHOC-NOW), July 8–10, 2013, Wroclaw, Poland [1].

☆☆ Research partially supported by the Research Grant 2010N5K7EB 'PRIN 2010' ARS TechnoMedia (Algoritmica per le Reti Sociali Tecno-mediate) from the Italian Ministry of University and Research.

* Corresponding author at: Department of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Via Gronchi, 18, I-67100, L'Aquila, Italy. Tel.: +39 0862 434466; fax: +39 0862 434403.

E-mail addresses: mattia.demidio@univaq.it (M. D'Emidio), daniele.frigioni@univaq.it (D. Frigioni), alfredo.navarra@unipg.it (A. Navarra).

http://dx.doi.org/10.1016/j.tcs.2015.09.002
0304-3975/

automatically discover nodes of a network and perform updates and/or refuse their connections. In this paper, we are interested in the exploration of a graph with faulty nodes, i.e. nodes that destroy any entering entity. Such nodes are called *black-holes*, and the exploration of a graph in such kind of networks is usually referred to as *black-hole search*. According to the assumed initial settings of the network, the knowledge, and the capabilities of the involved entities, many results have been provided in the literature.

1.1. Related works

Pure exploration strategies, without dealing with black-holes, have been widely addressed (see, for instance, [2–5] and references therein). In that case, the requirement is usually to perform the exploration as fast as possible. When black-holes are considered, along with the time (or equivalently the number of edge traversals) required for a full exploration, the main goal consists in minimizing the number of entities that may fall into the black-holes. A full exploration in this case means that, at the end, all the edges which do not lead to any black-hole must be "marked" as safe edges.

In this research area, many different models have been investigated. The system might be either synchronous [6,7] or asynchronous [8,9]. The input graph might be undirected [7,9] or directed [10,11]. It can be known in advance [7,12] or just a bound on the number of nodes is provided [11]. It can refer to a specific topology like rings [9,13], trees [6], hypercubes [14], tori [15]. Entities may communicate only when they meet [16], or by means of white-boards associated to the nodes of the graph [13], or simply by opportunely disposing available pebbles [17] or tokens [18]. The objective function may ask for the minimum number of entities, the minimum number of steps performed by all the entities, the minimum number of synchronous steps.

An interesting variation to the black-hole search problem has been recently introduced in [19], where the network must be decontaminated from *black-viruses*. A black-virus is a harmful process which destroys any agent arriving at the node where it resides; when that occurs, a black-virus moves, spreading to all the neighboring nodes, thus increasing its presence in the network. If however one of these nodes contains an entity, then that clone of the black-virus is destroyed. The initial location of the black-virus is unknown. The objective is to permanently remove any presence of the black-virus from the network with minimum number of node infections. The main cost measure is the total number of entities needed to solve the problem.

A hybrid model in between exploration and black-hole search has been introduced in [16], named the *Explore and Repair* problem. Given a graph G with n nodes, some of which are black holes, and a number k of entities, the idea is to perform the exploration of G as fast as possible with the constraint that if an entity enters in a black-hole, then it is destroyed and the black-hole disappears. The objective is to provide an exploration strategy that ensures to make safe the whole graph (i.e. removing all the black-holes) in the fastest way. A particular assumption is made in [16] on the consequences of entering in a black-hole. In fact, if more than one entity enter in a black-hole, only one gets destroyed while the others continue the exploration. With this assumption, a deterministic algorithm for the *Explore and Repair* problem is proposed in [16] which terminates within $O(\frac{n}{k} + \frac{\log f}{\log\log f} D)$ steps, where $f = \min\{\frac{n}{k}, \frac{n}{D}\}$, and D is the diameter of G. This algorithm is also shown to be worst-case asymptotically optimal, by giving a network such that for any deterministic algorithm, there is a placement of the faulty nodes forcing the algorithm to work for $\Omega(\frac{n}{k} + \frac{\log f}{\log\log f} D)$ steps. A general lower bound of $\Omega(\frac{n}{k} + D)$ steps is also provided.

It is possible to think about different scenarios where the model fits the assumption of [16], like in the case that software agents move along a network and, when they find a virus (represented by a black-hole), only one of them prolongs its stay for repairing purposes. However, it is also possible to think about scenarios where the assumption of [16] cannot model reality. For instance, in the case that entities are mobile robots exploring an impervious area disseminated of land-mines, if more than one robot incur concurrently in an explosion, then all of them get involved. Furthermore, the explosion may also affect robots within a fixed range.

1.2. Contribution of the paper

In this paper, we consider the Explore and Repair problem, and extend the model of [16] in two ways: (i) if more than one entity enters a black hole, then all of them are destroyed; (ii) given an integer number r, all entities within distance r (in terms of number of edges) from an entity entered in a black-hole instantaneously disappear from the network along with the black-hole. We call this the *Explore and Repair* problem with *radius* r. Clearly, if there are b black-holes, then $k \geq b$ entities are necessary to remove all the black-holes from the network. It might happen that the network is not explored completely, but an exploration algorithm must guarantee that all the black-holes have been removed as long as $k \geq b$.

The results of this paper are both theoretical and experimental, and can be summarized as follows. From the theoretical point of view, first we show that the problem is NP-hard even for $b = k = 1$; second, we consider the case of $r = 0$ and propose a simple variation of the strategy proposed in [16] that can be applied to give a worst-case asymptotically optimal solution when $k \geq 2b$; third, we consider the case of $r > 0$, and provide a lower bound of $\Omega(\frac{n}{k} + D + r\min\{D, b\} + |K_{\max}|)$ time steps, where b is the number of black holes, D is the diameter of the input graph, and $|K_{\max}|$ is the size of its maximum clique; fourth, we consider the case of $r \leq 1$ with an unbounded or a bounded number of entities k. In the unbounded case, that is, an infinite number of robots is available, we give an optimal strategy. For the bounded case,

that is the number of robots is fixed a priori, we propose three different exploration strategies, named *snake*, *scout*, and *parallel-scout*, respectively, with different peculiarities. In particular, *snake* is independent of the graph topology, *scout* is independent of the number of black holes, while *parallel-scout* introduces some parallel moves in the *scout* strategy, thus allowing a reduction in the number of time steps with respect to *scout*. We show that the three strategies require in the worst case a number of time steps linear in the number of nodes of the input graph, and then discuss their behavior with respect to the given lower bound. Moreover, all the proposed strategies can be easily shown to be worst-case asymptotically optimal, by providing simple instances where any algorithm requires $\Omega(n)$ time steps. Finally, we provide some hints on how to extend the proposed strategies to the case of $r > 1$.

From the experimental point of view, we implemented the three new strategies for the $r \leq 1$ case, and tested them on different graph classes by varying on their size, on the number of black-holes, and on the number of available entities. These experiments confirm the theoretical analysis of the proposed strategies and clearly show, as expected, that *parallel-scout* is always by far the best exploration strategy in practice.

1.3. Outline

In Section 2, we introduce definitions and notation needed for the formalization of the Explore and Repair problem with radius r. In Section 3, we provide results on the complexity of the problem. In Section 4, we propose different exploration algorithms for both the unbounded and the bounded case of k, when $r \leq 1$, and analyze their complexity and correctness. In Section 5, we compare the behavior of the proposed algorithms for the bounded case of k by means of extended experiments on different scenarios. In Section 6, we provide some hints on how to extend the proposed strategies to the case of $r > 1$. Finally, in Section 7, we conclude the paper with some possible future research directions.

2. Definitions and notation

We represent the exploration area as an undirected graph $G = (V, E)$ where V is a finite set of nodes and E is a finite set of edges. An edge in E between two nodes $u, v \in V$ is denoted as $\{u, v\}$. Given $v \in V$, $N(v)$ denotes the set of neighbors of v in G, and $\deg(v) = |N(v)|$ denotes the degree of v. A path P in G between nodes u and v is denoted as $P = (u, \ldots, v)$. The length of P, denoted as $l(P)$ is equal to the number of edges in P. A *shortest path* between nodes u and v is a path from u to v with the minimum length. The *distance* $d(u, v)$ from u to v is the length of a shortest path from u to v. The eccentricity $ecc(v)$ of a node v is the maximum distance between v and any other node in G. The diameter D of G is the maximum distance between two nodes in G, i.e. $D = \max_{v \in G}\{ecc(v)\}$. By $K_{\max}$, we denote the maximum clique contained in G. Given G and a source node $s \in V$, a Breadth First Search (BFS) tree of G, denoted from now on as $T = (V_T, E_T)$, is the result of a Breadth First Search algorithm on G, starting at s. Given T and a node v of T other than the source s, the *parent* of v, denoted as $parent(v)$, is the neighbor of v belonging to the path $P = (v, \ldots, s)$ in T from v to the source s. We assume $parent(s) = s$, so every node has a unique parent in T. The set of *children* of a node v in T is denoted by $children(v)$, and contains the neighbors of v in T with the exception of $parent(v)$. The *level* of v in T, denoted by $\ell(v)$, equals the distance $d(s, v)$ between s and v. A given tree is *ordered* if and only if an ordering is specified for the children of each node.

We study the *Explore and Repair* problem of [16] with radius r (r-ER problem from now on), which is defined as follows. We are given:

- An undirected graph $G = (V, E)$, with n nodes and m edges.
- A set of $0 < b \leq n - 1$ *black-holes*, each of them being posed at a different node of G. *Parameter b, and the locations of the black-holes are unknown.*
- A specific node $s \in V$, referred to as the *home-base* of G, which is assumed to be always safe, that is, entities in s cannot be destroyed by black-holes.
- A set of k entities, numbered from 1 to k (the entities' identifiers), initially placed at s. *Feasibility constraint requires* $k \geq b$.
- A parameter $r \in \mathbb{N}$, named *radius*, which represents the maximum distance (in the number of edges) from a black-hole at which an entity is affected by such a black-hole.

The goal is to define an *exploration strategy* that exploits the k entities, which synchronously move from s through G by traversing one edge per time step, with the goal of repairing all the black holes of G. If an entity is at the beginning of the current step at a node v, then its next move and its next state are determined by the exploration algorithm based on the topology of the network, the entity's identity, the entity's state (the state of its memory), the states of all other entities which are at the same time step at node v. We consider the general case where the repair of a black-hole affects also its neighborhood within distance r. The exploration strategy has to satisfy the following rules:

- If an entity enters in a black-hole residing at node v, then it is destroyed while repairing the black-hole. We assume that the repair of a node by an entity is instantaneous.
- If several entities enter in the same black-hole concurrently, then they are all destroyed and the black-hole is repaired.

- If an entity enters in a node where initially there was a black-hole that subsequently has been repaired, then the entity does not notice any differences. In fact, after the repair, the node acts normally and other entities can pass through it as if the black-hole never existed.
- If an entity enters in a black-hole residing at node v, then all entities not in s within distance r from v are destroyed during the repair of such a black-hole.
- Entities can communicate only when they meet at a common node of G, i.e. they can access the states of all other entities currently at the same node.

During the exploration, entities may decrease in number as some of them can enter in black-holes. On the one hand, since it is assumed that $k \geq b$, entities suffice to repair all the black-holes. On the other hand, while an entity is acting to repair a black-hole, it is a must to avoid that other entities get involved (in particular if $k = b$).

An exploration strategy is *correct* if it ensures to repair *all* the b black-holes in a network by means of $k \geq b$ entities, within a finite number of time steps, for any possible placements of the black-holes. It is *optimal* if it requires the minimum number of time steps among all possible correct strategies. The definition of the r-ER problem can be summarized as follows.

r-ER problem

Input: An undirected graph $G = (V, E)$ with black-holes, a home-base node $s \in V$ where a set of $k > 0$ ordered entities is initially placed, a radius $r \geq 0$.

Goal: A correct exploration strategy performed by the k entities which minimizes the overall number of synchronous time steps.

3. Complexity results

In this section, we first show that the r-ER problem is NP-hard even for $b = k = 1$. Then, we consider the case of $r = 0$ and show that a simple variation of the strategy proposed in [16] can be applied to give an asymptotically optimal solution. Finally, we provide a general lower bound for the case of $r > 0$.

Theorem 3.1. *The r-ER problem is NP-hard for $b = 1$ and $k = 1$.*

Proof. It is enough to observe that even the fastest way to explore a graph in order to detect where the unique black-hole resides must involve all the nodes of the input graph G. Hence, $n - 1$ time steps are required for the full exploration by means of the unique entity. This corresponds to the problem of finding a Hamiltonian Path in G, if possible. Since the Hamiltonian Path problem is known to be NP-complete [20], the claim holds also for the r-ER problem. □

The r-ER problem is subject to a simple lower bound concerning the time steps required by any exploration algorithm with $k \geq b$. In fact, the input instance might need an entity to move from the home-base s to the farthest possible node whose distance from s is the diameter D of G. Moreover, if k entities are available, then each of them can explore a portion of the graph equal to $\frac{n}{k}$ nodes, concurrently. The next lemma follows.

Lemma 3.2. *Any correct exploration strategy for the r-ER problem requires $\Omega(\frac{n}{k} + D)$ time steps.*

3.1. Case $r = 0$

When $r = 0$ the r-ER problem is very similar to that presented in [16]. The only differences are the following: we relax the constraint $k \geq 2b$ to $k \geq b$; we need to avoid that more than one entity at the same time enters in a black-hole as otherwise all of them would be destroyed. As lower bound, the one provided for the general case by Lemma 3.2 holds. As upper bound for the case $k \geq 2b$, we can exploit the solution proposed in [16] with a slight modification. Assume a bunch of entities has to move from a node u to a node v concurrently (in one time step). The same movement can be obtained in two time steps as follows: first move only one entity (say the one with the smallest identifier) from u to v; then, in the subsequent time step, move all the remaining entities from u to v, while the first entity, if survived, waits for them at v. In this way, if node v is occupied by a black-hole, only one entity is affected. The case where a node is entered concurrently from different neighbors cannot occur. In fact, all moves of the algorithm in [16] are performed on the edges of a fixed spanning tree. Therefore, a black-hole can be entered only through one edge (the one leading to its parent). Hence, we can state the next lemma by simply exploiting the exploration algorithm proposed in [16] that holds for $k \geq 2b$, with the only modification of applying the considerations given above.

Lemma 3.3. *If $n \geq k \geq 2b$, the 0-ER problem can be solved in $O(\frac{n}{k} + \frac{\log f}{\log \log f} D)$ time steps, where $f = \min\{\frac{n}{k}, \frac{n}{D}\}$.*

More general upper bounds holding for any $k \geq b$ will be discussed in Section 4.

3.2. Case $r > 0$

In general, when $r > 0$, the exploration strategy of [16] enriched as described in Section 3.1 is no longer sufficient to guarantee that all black-holes of the graph will be repaired. In fact, when the first entity moves from a node u to a node v, if v is a black-hole, then this entity and all the entities waiting on u will die. Hence, we need to develop a completely new strategy for the case of $r > 0$. The following lemma provides a lower bound on the number of time steps required.

Lemma 3.4. *Any correct exploration strategy for the r-ER problem requires* $\Omega(\frac{n}{k} + D + r\min\{D, b\} + |K_{\max}|)$ *time steps, for any* $r > 0$ *and* $b \leq k < n - 1$.

Proof. The term $\Omega(\frac{n}{k} + D)$ comes from Lemma 3.2. The other terms are obtained as follows.

Assume G contains a simple path P with p black holes and the home-base s as one end-point. Since each black-hole affects nodes at distance at most r, once an entity enters a black-hole v on P, the next entity e that keeps on exploring P must reside at distance greater than r from v. This means that e can continue the exploration of P after $r + 1$ steps, and this happens for each black-hole on P but the first. Hence, $\ell(P) + (r + 1)(p - 1) - \frac{r(r+1)}{2}$ time steps are required. The last term is due to the fact that, for each node of the path at distance $h \leq r + 1$ from s, h steps are needed instead of $r + 1$. If $\ell(P) = D$ and $p = b$ then $\Omega(D + r(\min\{D, b\}))$ time steps are required for exploring G.

Furthermore, let us assume G contains a clique K_p of p nodes, one of which is the home-base s. As parameter b is unknown, any exploration algorithm must work also for the case $k = b$. Since each entity can repair at most one black-hole, if the repairing of one black-hole causes the destruction of more than one entity, then there will not remain enough entities to repair all the black-holes. In order to find out all the b black-holes, any algorithm can allow at most one entity to reside in K_p apart from those in s. In fact, since $r > 0$, it is sufficient that such an entity enters in a black-hole to destroy all the other entities in K_p but not in s. Since there is no knowledge on the possible locations where the black-holes reside, any algorithm can allow to explore K_p at most one node per time step, until all the clique gets repaired, thus requiring at least $p - 1$ time steps for K_p. It follows that $\Omega(|K_{\max}|)$ time steps are required for exploring the maximum clique contained in G.

The hypothesis on $k < n - 1$ is required for the case in which G is a clique of n nodes. In such a case, in fact, if $n - 1$ entities are available, then they can move concurrently from s to all the other nodes of G, hence repairing all the black-holes in one time step. □

The above lemma reveals that $\Omega(n)$ time steps are required if the number of available entities is constant or $D = \Omega(n)$, or $|K_{\max}| = \Omega(n)$. This implies that in the two limit cases of G being a path or a complete graph, $\Omega(n)$ time steps are required. In general, it is not so clear how to spread entities over G in order to exploit possible concurrency. In the next section, we tackle the problem for $r \leq 1$ and provide some efficient exploration strategies. Some hints on how to extend the proposed strategies to the case of $r > 1$ will be given in Section 6.

4. Exploration algorithms for $r \leq 1$

In this section, we propose various exploration strategies for $r \leq 1$, and show their complexity and correctness analysis. In particular, we first propose an approach for unbounded k (an infinite number of entities is available) which tries to exploit potential parallelism among the entities in order to save execution time. As we will see, this approach is not suitable for the case of bounded k (a finite number of entities is available), for which we propose three different alternative exploration strategies and analyze their performance with respect to the lower bound.

4.1. Unbounded k

Our first approach is based on the observation that more entities may potentially explore different parts of the input graph G, concurrently. This can be done as long as the graph is opportunely divided into "disconnected" parts. In fact, if an entity gets destroyed by means of a black-hole, it might be required that this event does not affect other entities on the graph. We describe a basic greedy algorithm, called BASIC, that works when a sufficiently large number of entities $k \geq n - 1$ is available.

The BASIC algorithm works in stages as follows. At each stage, some entities starting from s move to enlarge the borders of a *Safe-Zone*. This is a connected subgraph of G containing s, whose nodes have been explored and hence repaired from possible black-holes. At the first stage, the safe-zone contains only node s. Then, $N(s)$ entities move concurrently to make safe all the neighbors of s in G, which will be included in the safe-zone. It may happen that some entities survive for the second stage, but since the number of entities is sufficiently large, we can ignore them. At the second stage, the nodes to explore are those neighboring to nodes in the safe-zone but not belonging to it. Keeping working this way, all the graph will be made safe, eventually. The BASIC strategy solves the 1-ER problem in $O(D^2)$ time steps. In fact, at each stage, $\sum_{x \in SafeZone} |N(x) \backslash SafeZone|$ new entities are sent from s to explore as much *unvisited nodes* as possible, where unvisited nodes are nodes that may contain black-holes. Since the number of entities k is assumed to be at least $n - 1$, all the nodes will be explored, eventually. The number of stages required by the algorithm equals $ecc(s)$, and each stage i requires i

time steps as the explored nodes are at distance i from s. In total, the number of time steps required by BASIC is hence $\sum_{i=1}^{ecc(s)} i = ecc(s)(ecc(s)-1)/2 = O(D^2)$.

The above result is not much satisfactory if compared to the lower bound. Actually, this comes from the fact that at each stage, entities reach the border of the safe-zone always starting from s. A simple variant of BASIC, named BFS-BASIC, can be provided requiring an optimal number of time steps. It is based on the observation that, since each entity knows the input graph G, and G is explored in a breadth first way by BASIC, then a BFS tree T of G can be pre-computed and used by the entities for the exploration as follows. Assume entities are at stage $i > 2$, exploring the i-th level of T. Given a node v of the i-th level, the number of entities required to explore the subtree T_v of T rooted at v is known and is equal to $|T_v|$. Hence, BFS-BASIC during the last step of stage $i-1$ can preventively move $|T_v|$ entities to $parent(v)$ in case $r = 0$ or to the ancestor of v in T at distance two from v in case $r = 1$. By observing that all entities necessary to explore the unexplored subtrees are at distance at most two from the root of each subtree, it follows that each stage of BFS-BASIC requires at most two time steps. The overall cost is hence upper bounded by $2D$, and the following lemma holds.

Lemma 4.1. *Given a graph G of n nodes and $k \geq n-1$ entities,* BFS-BASIC *takes $O(D)$ time steps.*

Strategy BFS-BASIC relies on the assumption to have a number of entities k at least equal to $n-1$. If this in not the case, then a more accurate strategy is required. In particular, a generic algorithm should work as long as $k \geq b$. The worst possible scenario is certainly the case of $k = b$. In this case, when an entity meets a black-hole, an exploration algorithm must guarantee that no other entity is affected as otherwise there will not remain enough entities to terminate the exploration. It is worth noting that since parameter b is unknown, any strategy must be designed to work also for the limit case $k = b$.

4.2. Bounded k

In what follows, we first propose two exploration strategies for the case $k \geq b$, named *snake* and *scout*, and show their complexity and correctness. Such strategies do not require any parallel movements by the available entities. Then, we introduce a variant of the *scout* strategy named *parallel-scout* which is able to take advantage of parallelism.

In all strategies, we assume that an ordered BFS tree T of the input graph G, rooted at the home-base s, has been pre-computed and that all the entities know it. Given an edge $\{u, v\}$ in T, such that the level of u is smaller than the level of v, then an entity is said to move in the *down* (*up*, respectively) direction if it moves from u to v (v to u, respectively).

Given a BFS tree T of a graph G with n nodes, we call *exploration walk* of T the path obtained by visiting T by a Depth First Search algorithm. Clearly, this visit traverses each edge of the ordered tree T twice, once in the *down* direction and once in the *up* direction. Then, it follows that the *exploration walk* of T has length equal to $2(n-1)$.

The strategies we propose require that entities move along the edges of the *exploration walk* of T by giving priority to the visit (and possibly to the repair) of unvisited nodes, i.e. nodes that are not safe, and by avoiding to visit nodes that have already been visited. For implementing this behavior, entities have to keep trace of which part of T has already been made safe by the exploration strategy at hand. To this aim, we define, at each time step i, the *unvisited part* of T, denoted by $U_i = (V_{U_i}, E_{U_i})$, which is a tree obtained from T as follows: at the beginning, U equals T, i.e. $U_0 \equiv T$. At the generic time i, E_{U_i} is obtained by removing from E_{U_0} edges of the following types:

- $\{x, y\} : \ell(x) < \ell(y)$, there exists a time instant $0 < j < i$ such that y is a leaf node of U_j, and edge $\{x, y\}$ has been traversed both *down* and *up* by an entity;
- $\{x, y\} : \ell(x) < \ell(y)$, there exists a time instant $0 < j < i$ such that y is a leaf node of U_j, and a black-hole on y was repaired (and hence edge $\{x, y\}$ has been traversed *down* by a single entity).

Trivially, every edge removal, as described above, induces a zero degree node, which is also removed from the unvisited tree.

Since the unvisited tree changes as the exploration proceeds, each entity e maintains, at every time step i, a local copy of the unvisited tree, denoted by U_i^e and updates it, in order to perform the correct exploration strategy, every time either it detects one of the events that modify U or it meets an entity with a more recent copy of U. In particular, if some entities meet at node x at time t, then each of them updates its own unvisited tree with that known by the entity with the smallest id among those residing at x. This update policy follows from the fact that, in our strategies, when a bunch of entities reside at a node and has to agree about who leaves the node first, the entity with the smallest id is chosen. Hence, the entity with the smallest id always carries the most recent version of U, as it is the first to detect edges to be removed from T, as described above. More details about this issue will be given within the descriptions of the proposed strategies.

For the sake of readability, we introduce the following notation that characterizes nodes of the graph and that is used within the pseudo-code. Given a node x in G and a time i:

- $home(x)$ is a boolean value which is true if and only if x is the home-base, i.e. the root of T (and the root of every U, consequently);
- $\min(x)$ is the smallest id among entities residing at x;

```
Procedure: REGULAR
Event: REGULAR is invoked by an entity with id equal to e, at node x, at time i.
Input: U_i^e, H(x), R(x) and B(x)
1  if home(x) ∧ leaf(x) then
2  |  Exploration terminated;
3  else
4  |  if H(x) then
5  |  |  Move to parent(x);
6  |  |  Call BACKWARD;
7  |  else if leaf(x) then
8  |  |  Move to parent(x);
9  |  |  Call HEAD                          /* No head has been met on the path to x */
10 |  |  else
11 |  |  |  if leaf(next(x)) ∧ home(x) then
12 |  |  |  |  if time ≠ 0 then              /* head has died while visiting next(x)  */
13 |  |  |  |  |  Update U_i^e by removing (x, next(x));
14 |  |  |  |  if e = min(x) then
15 |  |  |  |  |  Call HEAD                                /* Election of the new head */
16 |  |  |  |  else
17 |  |  |  |  |  Wait 2 time steps        /* Keep distance from head greater than 1 */
18 |  |  |  |  |  Call REGULAR;
19 |  |  |  else
20 |  |  |  |  if B(x) = 0 then
21 |  |  |  |  |  if e = min(x) then
22 |  |  |  |  |  |  Move to next(x);
23 |  |  |  |  |  else
24 |  |  |  |  |  |  Wait 2 time steps     /* Keep distance from head greater than 1 */
25 |  |  |  |  |  Call REGULAR;
26 |  |  |  |  else
27 |  |  |  |  |  Call BACKWARD;
```

Fig. 1. Pseudo-code of procedure REGULAR.

- *parent*(x) is the parent of x in T;
- *leaf*(x) is a boolean value which is true if and only if x is a leaf of $U_i^{\min(x)}$ at time i;
- *next*(x) is the first child of node x in the ordered tree $U_i^{\min(x)}$.

Moreover, we denote by *time* the positive integer representing the number of elapsed time steps. Every entity is aware of the synchronous increase of *time* and uses it to determine the action to be performed. In our strategies, we assume that in one time step an entity can traverse at most one edge. Any other computation performed by the entity is executed instantaneously.

4.2.1. The snake *strategy*

In this strategy, we distinguish three different types of entities: *regular*, *head*, and *backward*. The pseudo-code of the procedures executed by the entities for the case $r = 1$ is given in Figs. 1, 2, and 3. The type of each entity is determined by the procedure it executes. An example of execution is given in Fig. 4 and will be discussed later. Clearly, the strategy can be simplified to also work for the case $r = 0$.

Given a node x in G, we define the following variables which are used by entities during the exploration:

- $H(x)$ is a boolean value which is true if and only if the *head* is residing at x;
- $R(x)$ ($B(x)$, respectively) is a non-negative integer equal to the number of *regular* (*backward*, respectively) entities residing at x.

Note that, variables $H(x)$, $R(x)$ and $B(x)$ can be easily computed by an entity while entering in or residing at node x.

At the beginning, all the entities are of *regular* type, reside in the home-base s, and execute REGULAR as first routine. Then, on the basis of their identifiers, a *head* entity is elected. In particular, an easy way of computing the *regular* entity that has to turn its type to *head*, is that of choosing the entity with the smallest id (see Line 14 of Fig. 1). By now, no *backward* entity exists and U_0^e equals T, for every entity e at s. The *snake* strategy allows at each time step, the existence of at most one *head* entity. The entities move synchronously through the *exploration walk* starting from s, according to their

```
Procedure: HEAD
Event: HEAD is invoked by an entity with id equal to e, at node x, at time i.
Input: U_i^e, R(x) and B(x)
1  if home(x) ∧ leaf(x) then
2  |  Exploration terminated;
3  else
4  |  if entity e reached x by an up move then
5  |  |  Update U_i^e by removing (x, next(x));
6  |  |  Share U_i^e with other entities residing at x;
7  |  if leaf(x) then
8  |  |  Move to parent(x);
9  |  else
10 |  |  Move to next(x);
11 |  Call HEAD;
```

Fig. 2. Pseudo-code of procedure HEAD.

```
Procedure: BACKWARD
Event: BACKWARD is invoked by an entity with id equal to e, at node x, at time i.
Input: U_i^e, H(x), R(x) and B(x)
1  if home(x) ∧ leaf(x) then
2  |  Exploration terminated;
3  else
4  |  if H(x) then
5  |  |  Move to parent(x) ;                          /* If x = s, stay on s */
6  |  else
7  |  |  if e = min(x) then
8  |  |  |  Share U_i^e with other entities residing at x;
9  |  |  |  Move to next(x);
10 |  |  |  Call REGULAR;
11 |  |  else
12 |  |  |  Wait 2 time steps          /* Wait until you are the minimum id entity */
13 |  Call BACKWARD;
```

Fig. 3. Pseudo-code of procedure BACKWARD.

type. In general, as the strategy name suggests, the entities proceed in line in a *snake*-like visit as follows: the *head* entity starts moving from s, edge by edge in the *down* direction through the *exploration walk* (see Line 10 of Fig. 2), and every $r+1$ time steps (see Line 17 of Fig. 1) a *regular* entity leaves s along the same walk as the *head* (see Line 22 of Fig. 1). As well as in the *head* election step, the *regular* entity that has to leave s can be decided by choosing that with the smallest id among those residing at s (see Line 21 of Fig. 1). This implies that there are always r empty nodes between each pair of consecutive entities composing the *snake*, when they are moving in the *down* direction. Usually, entities do not meet, except for the following cases.

1. If the *head* visits, at step t, a leaf node x of T without entering in a black-hole, then it is back to $parent(x)$ at step $t+1$ where it meets the first *regular* entity following it. This *regular* entity changes its type to *backward* (see Line 4 of Fig. 1) and updates its own version of U by replacing it with that carried by the *head* (see Line 6 of Fig. 2).
   In such a case, if node x has no children in U, then entities start traversing the *exploration walk* in the *up* direction one edge per time step (Line 8 of Fig. 2 and Line 5 of Fig. 3). During this traversing the entities can meet other *regular* entities which turn to *backward* type as well and join this group (see Line 4 of Fig. 1).
   The visit continues until the entities reach a node v of U with at least one child (that is v is unvisited). At this point, the exploration continues as follows: the *head* visits the first child of v (Line 10 of Fig. 2), while the *backward* entities move together to $parent(v)$ in the *up* direction (Line 5 of Fig. 3), where they meet a *regular* entity (if any), and stay there. In this case, the *regular* entity turns its type to *backward* (see Line 27 of Fig. 1). Now, every two time steps, a *regular* entity (if any) meets the *backward* entities group and is made aware of the updated U, making the entities able to follow the *head* according to the knowledge of U. In particular, this is done as follows: every two time steps *regular* entities entering the node turn into *backward* entities (Line 27 of Fig. 1) while a *backward* entity, made aware of the most recent version of U, turns its type into *regular* (see Line 9 of Fig. 3), leaves the node, follows the *head* entity and continues the visit as usual (see Line 10 of Fig. 3).

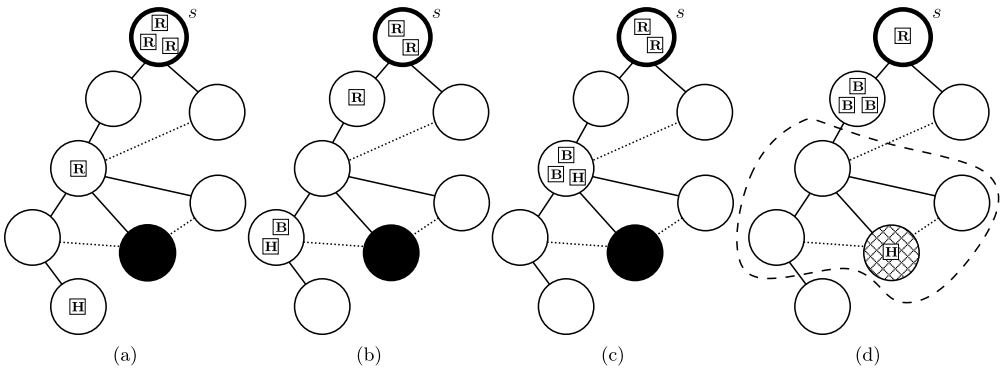

Fig. 4. Four consecutive snapshots during the execution of the *snake* strategy for $r = 1$. Entities are represented by squares of types *head* (H), *regular* (R), and *backward* (B). The bold node s is the safe home-base. Black nodes represent black-holes. Solid edges represent BFS tree edges, while dotted edges represent graph edges. Nodes rounded by the dashed line are those affected by the black-hole repair.

2. A *regular* entity visits a leaf node of U without entering in a black-hole. This can happen if the *head* entity and possibly some *regular* entities have entered in black-holes. In this case, the leading *regular* entity immediately changes its type to *head* and the visit proceeds as previously described (see Line 14 of Fig. 1).
3. The surviving entities (if any) have gone throughout all the *exploration walk* and are back in s, that is, the exploration has terminated (see Line 2 of Figs. 2, 1 and 3).

In Fig. 4(a), it is shown a possible situation with $r = 1$ where the *head* has reached a leaf of T. In the subsequent step, shown in Fig. 4(b), the *head* has made an up move and has met a *regular* entity which has become of *backward* type. Moreover, one further entity has moved from s along the *exploration walk*. In Fig. 4(c), the *head* and the *backward* entities have made an *up* move since there was no other child to explore from the situation of Fig. 4(b). There, they also meet the other *regular* entity on the exploration walk, which now has become of *backward* type. In Fig. 4(d), the *head* has died in the black-hole, while the other entities have moved up in the exploration walk in order to maintain their distance greater than $r = 1$ from the *head*.

Lemma 4.2. *The* snake *strategy takes* $2(n-1)$ *time steps if* $b = 0$, *and* $2(n-1) + (r+1)(b-1)$ *time steps if* $b > 0$, *for* $r \leq 1$.

Proof. If $b = 0$, then the *head* requires exactly $2(n-1)$ synchronous steps to traverse the $2(n-1)$ edges of the *exploration walk*. If $b > 0$, then the leading entity of the *snake* can be either a *head* or a *regular* entity. Every time the leading entity enters in a black-hole, $r+1$ synchronous steps are required to have the new leading entity in the same position where its predecessor died, and hence to keep continuing with the traversing of the *exploration walk*. Therefore, the *snake* strategy for $r = 0$ ($r = 1$, resp.) requires $2(n-1) + b - 1$ ($2(n-1) + 2(b-1)$, resp.) overall steps. □

4.2.2. The scout *strategy*

In this strategy, we distinguish two different types of entities: *scout* and *team*. The pseudo-code of the procedures executed by the entities for the case $r = 1$ is given in Figs. 5 and 6. We remind that the type of each entity is determined by the procedure it executes. An example of execution is given in Fig. 7 and will be discussed later. Clearly, the strategy can be simplified to also work for the case $r = 0$.

In order to define the procedures we introduce a set of variables that encode specific events that entities can detect while performing the exploration. In particular, the variables are as follows:

- $SC(x)$ is a boolean value which is true if and only if a *scout* entity is residing at x;
- $TM(x)$ is a non-negative integer giving the number of *team* entities residing at node x.

Note that, variables $SC(x)$ and $TM(x)$, can be easily computed by an entity while entering in or residing at node x.

At the beginning, all the entities are of *team* type, reside in s, and execute `TEAM` as first routine. Then, on the basis of the identifiers, a *scout* entity is elected. By now, U_0^e equals T, for every entity e at s. The *scout* strategy allows at each time step, the existence of at most one *scout* entity. The entities move synchronously through the *exploration walk* starting from s, according to their type as follows. An entity of *scout* type traverses $r+1$ edges in the *down* direction (see Line 10 of Fig. 5) and one edge in the *up* direction (see Line 8 of Fig. 5), i.e. it performs a *scout round*. An entity of *team* type waits $r+1$ synchronous steps and then traverses an edge in the *down* direction (see Line 16 of Fig. 6), i.e. it performs a *team round*.

Usually, at the $r+2$-th time step of a round, the *scout* and the *team* entities meet at a node v of G and the *scout* communicates to the *team* ones the updated unvisited tree (see Line 6 of Fig. 5). This meeting implies that the *scout* has

```
Procedure: SCOUT
Event: SCOUT is invoked by an entity with id equal to e, at node x, at time i.
Input: U_i^e, SC(x) and TM(x)
1  if home(x) ∧ leaf(x) then
2  |  Exploration terminated;
3  else
4  |  if entity e entered x by an up move then
5  |  |  Update U_i^e by removing (x, next(x));
6  |  |  Share U_i^e with other entities residing at x, if any;
7  |  if leaf(x) then
8  |  |  Move to parent(x);
9  |  else
10 |  |  Move to next(next(x)) ;                    /* Requires two time steps */
11 |  Call SCOUT;
```

Fig. 5. Pseudo-code of procedure SCOUT.

```
Procedure: TEAM
Event: TEAM is invoked by an entity with id equal to e, at node x, at time i.
Input: U_i^e, SC(x) and TM(x)
1  if home(x) ∧ leaf(x) then
2  |  Exploration terminated;
3  else
4  |  if leaf(x) ∨ SC(x) then
5  |  |  Move to parent(x);
6  |  else
7  |  |  if leaf(next(x)) then
8  |  |  |  if time ≠ 0 then          /* scout has died while visiting next(x)  */
9  |  |  |  |  Update U_i^e by removing (x, next(x));
10 |  |  |  if e = min(x) then
11 |  |  |  |  Call SCOUT ;                      /* Election of the new scout */
12 |  |  |  else
13 |  |  |  |  Move to parent(x);
14 |  |  else
15 |  |  |  Wait 2 time steps;          /* Keep distance from scout greater than 1 */
16 |  |  |  Move to next(x);
17 |  Call TEAM;
```

Fig. 6. Pseudo-code of procedure TEAM.

visited only safe nodes and the exploration can continue from v. Note that, if the *scout* has visited a leaf node of the unvisited tree, this can be removed from U itself before sharing it with other entities (see Line 5 of Fig. 5).

Otherwise, if the *team* entities do not meet the *scout* at node v, this means that the *scout* has entered in a black-hole and has died by repairing it. At this point, a new *scout* is elected among the *team* entities in v and the exploration continues from v (see Line 11 of Fig. 6).

The only exception to the above behavior is when the *scout* visits a child node v of s. In this case, the *scout* and the *team* entities meet always at the second step of a round. However, since all entities know how T must be explored, the *team* entities can infer that the *scout* can come back after only two synchronous steps, by checking whether $leaf(next(x))$ is true or not (see Line 7 of Fig. 6). Therefore, they can remove $next(x)$ from U (see Line 9 of Fig. 6) and the exploration can continue as usual.

When the *team* entities reach the parent node of a leaf in T, in two time steps the leaf is explored by the *scout* if $r = 0$. While if $r = 1$, all the entities already know that the leaf has been already visited. In both cases, all entities can now start traversing the *exploration walk* in the *up* direction, an edge per step, until they enter in a node v of T with at least one unvisited child, or they are back to s. In the first case, the *scout* (elected now if needed) moves to the first unvisited child of v and the *team* entities move to the parent node of v if $r = 1$ (see Line 13 of Fig. 6). At this point, the entities start over with their normal behavior. In the second case, the exploration keeps continuing on an unexplored child of s, if any. If all the children of s have been already explored then the exploration is terminated (see Lines 2 of Figs. 5 and 6).

Fig. 7 describes the behavior of the *scout* strategy in a scenario similar to that of Fig. 4 for the *snake* strategy with $r = 1$. In detail, Fig. 7(a) shows the situation where the *scout* entity has explored a branch of a node and is going to inform the *team* entities. Fig. 7(b) shows the time when *scout* and *team* entities meet. From there, the *scout* goes to explore the other

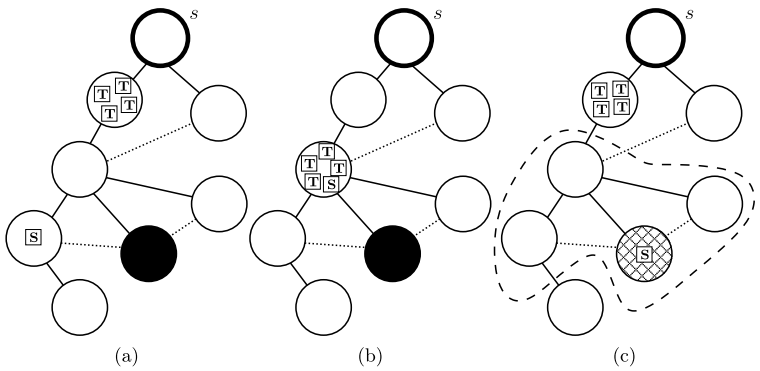

Fig. 7. Three consecutive snapshots during the execution of the *scout* strategy for $r = 1$. Entities are represented by squares of types *scout* (S) and *team* (T). The bold node s is the safe home-base. Black nodes represent black-holes. Solid edges represent BFS tree edges, while dotted edges represent graph edges. Nodes rounded by the dashed line are those affected by the black-hole repair.

branch of the explored node, while the *team* entities move *up* to make them safe from possible black-holes found by *scout*. This occurs in the subsequent step shown in Fig. 7(c).

Let us denote as F the set of leaves in T, as $I = V \setminus \{F \cup \{s\}\}$, and as $deg_T(v)$ the degree of a node v in T. The following lemma can be stated.

Lemma 4.3. *The* scout *strategy takes* $2(n-1) + (r+1)|I|$ *time steps, for* $r \leq 1$.

Proof. In general, only the *scout* entity can enter in black-holes while the *team* is always maintained safe along its walk without additional steps.

When $r = 1$, each node is made safe within three time steps in order to let know the *team* entities about it. Moreover, the *team* entities make one further step for each reached node in the *up* direction. This means that in general, any node different from the home-base s requires four time steps by the whole process. Exceptions are represented by leaves that are never reached by the *team* entities. Moreover, the children of the home-base s are made safe by the *scout* in two steps each, making known also the *team* entities. Finally, whenever the *team* and the *scout* walk together in the *up* direction until reaching a node x with an unexplored child y, the *team* goes towards the parent of x while the *scout* explores y. In the subsequent step, both the *team* entities and the *scout* (if survived) meet at x, hence y requires only two steps to be explored. Note that, for every $x \in I$, there can be at most $deg_T(x) - 2$ unexplored children.

By summing up overall the contributions, the *scout* strategy requires a number of time steps equal to:

$$
\begin{aligned}
&4(n-1) - |F| - deg_T(s) - \sum_{x \in I} (deg_T(x) - 2) \\
&= 4(n-1) - \sum_{x \in F} deg_T(x) - deg_T(s) - \sum_{x \in I} deg_T(x) + \sum_{x \in I} 2 \\
&= 4(n-1) - \sum_{x \in V} deg_T(x) + 2|I| = 2(n-1) + 2|I|.
\end{aligned}
$$

When $r = 0$, the behavior is quite similar but for the fact that each node is made safe in two time steps. Moreover, the *team* entities make one further step for each reached node in the *up* direction. Exceptions are represented by leaves that are never reached by the *team* entities. By summing up overall the contributions, the *scout* strategy requires a number of time steps equal to $2(n-1) + |I|$. □

4.2.3. The parallel-scout *strategy*

The *snake* and the *scout* strategies have been shown to require a number of steps proportional to the number of nodes of the input graph. Although this is an asymptotically optimal behavior with respect to many practical cases as described with the lower bound, none of them exploit possible parallel explorations. The performance (the required number of steps) of the *scout* strategy is independent of b, while that of the *snake* strategy is independent of the input graph topology, and hence on the structure of T. For instance, when $r = 1$ and the input graph is a path with one extreme as the home-base, the *scout* strategy requires $2(n-1) + 2|I| = 4(n-1) - 2$ steps, and this might be close to the double of what is required by the *snake* strategy. Nevertheless, the *scout* strategy reveals an important peculiarity that may help in introducing some parallel moves, hence reducing the number of required time steps. In fact, each time the scout entity explores one new node, it starts moving after having met all the other entities. In this way, all the entities are always aware of the current part of the graph already explored.

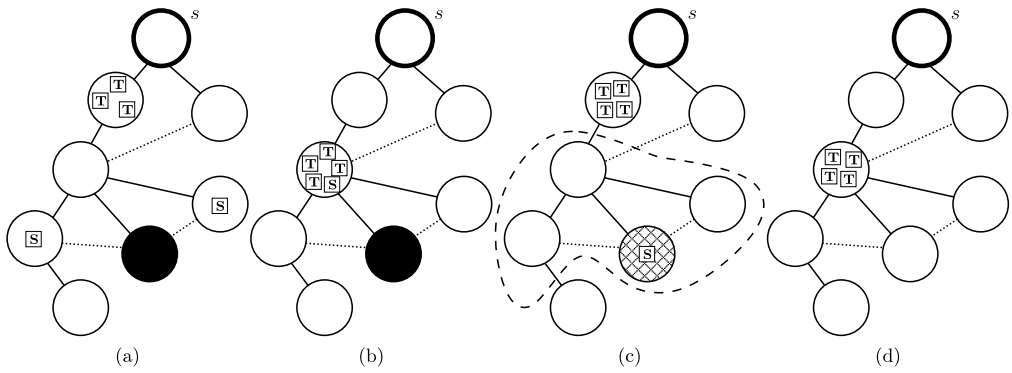

Fig. 8. Four consecutive snapshots during the execution of the *parallel-scout* strategy for $r = 1$. Entities are represented by squares of types *scout* (S) and *team* (T). The bold node s is the safe home-base. Black nodes represent black-holes. Solid edges represent BFS tree edges, while dotted edges represent graph edges. Nodes rounded by the dashed line are those affected by the black-hole repair.

In the general step of the *scout* strategy, when $r = 1$ and the scout entity e wants to explore a node z, it moves from node x, where also all the other entities reside, to a neighbor y which is the parent of z in the BFS tree T. Clearly, node y must have been made safe in some previous step. From y, e can move towards z without damaging any other entity since they are all waiting at distance two in x. If y has many children, then all the ones which are not directly connected to z in G could be explored by other entities in parallel with e without causing multiple destructions if a black-hole is met.

Similarly, when $r = 0$ and the scout entity e wants to explore a node z, it moves from the parent of z, say y, where also all the other entities reside, to z. If y has many children, then all them could be explored by other entities in parallel with e without causing multiple destructions if a black-hole is met.

In order to implement this variant of the *scout* strategy called *parallel-scout*, we need to introduce the following further step: in the general step described above for $r = 1$, the entities compute an independent set in the subgraph of G induced by the children of node y in T; in the general step for $r = 0$, all the children of node y in T are considered as an independent set.

Then, if enough entities are available, all the nodes belonging to the computed independent set can be explored concurrently in one time step (see Fig. 8(a) for the case $r = 1$), and the results made known to all the entities in the subsequent step when they all meet again if survived (see Fig. 8(b)). If not survived (see Fig. 8(c)), still all the other entities can imply that some black-holes have been repaired and that the computed independent set is now safe (see Fig. 8(d)). The acquired information will be used in the subsequent steps of the exploration in order to move safely over nodes already explored, and hence saving in the number of required time steps.

Of course the *parallel-scout* strategy does not give advantages in terms of asymptotic bounds with respect to *scout*. However, it can give a practical gain with respect to the basic *scout* strategy in terms of the effective number of time steps needed, which can make *parallel-scout* very competitive in practice. We will give experimental evidence of all the above observations in Section 5.

4.2.4. Correctness

In this section, we provide the correctness proof of the proposed strategies.

Lemma 4.4. *In all* snake, scout, *and* parallel-scout *strategies, when* $r \leq 1$, *if an entity enters in a black-hole, then it does not affect any other entity.*

Proof. First of all notice that in each of the three strategies, an entity can enter in a black-hole only when moving in the down direction. In fact, when an entity moves in the up direction, it traverses and edge which has been already traversed, and hence possible black-holes have been already repaired.

In the *snake* and the *scout* strategies, there is always a single entity leading the exploration. For the *parallel-scout* strategy, there might be more than one entity leading the exploration, but by definition they visit different nodes composing an independent set, that is their distance is greater than r. Only leading entities can enter in black-holes, while the other entities follow the leading ones at distance greater than r in the exploration walk.

As the exploration walk is induced by the pre-computed BFS tree T, then the distance in G between the leading entities and the closest entity in all strategies remains equal to $r + 1$ by construction, if the leading entity is moving *down*. The only exception to this argument occurs for $r = 1$ when a leading entity finds a black-hole in a child of the home-base s. However, in such a case all the entities in s are safe by definition. □

Theorem 4.5. *The* snake, *the* scout, *and the* parallel-scout *strategies are correct.*

Proof. By Lemma 4.4, the repair of one black-hole affects only one leading entity for each strategy. This implies that $k = b$ entities are sufficient to repair all the black-holes.

Entities move along the *exploration walk* which contains all the nodes of G. By Lemma 4.4, if $k > b$ then $k - b$ entities visit the entire *exploration walk* and come back to the home-base. Whereas, if $k = b$ then the exploration strategy ends with the repair of the last black-hole by means of the last surviving entity. In both cases, all the black-holes are repaired and the exploration strategy is correct. □

5. Experiments

In this section we present the results of our experimental study, which has been performed on a workstation equipped with a Quad-core 3.60 GHz Intel Xeon X5687 processor with 24 GB of main memory. The programs have been compiled with GNU g++ compiler 4.4.3 under Linux (Kernel 2.6.32). The experiments consist of simulations within the OMNeT++ 4.0p1 environment [21], a well known framework for testing networked/distributed systems.

Executed tests We performed simulations for the two different settings of $r = 0$ and $r = 1$. For each setting, we modeled the network and the entities by means of OMNeT++ modules and then implemented the *snake*, *scout* and *parallel-scout* exploration strategies, within the modules. Concerning the heuristic of *parallel-scout* in the $r = 1$ setting, we implemented the $(\Delta + 2)/3$-approximation algorithm of [22] for the computation of the required independent set, where Δ is the maximum degree of the nodes in the network. For each instance of the problem, we simulated each exploration strategy and measured the number of time steps required to explore and repair a network. In particular, we included within the general simulator specific modules to verify that the strategies correctly explore the network, i.e. to guarantee that at most one entity dies every time a black hole is repaired.

As input to our simulations we used both real-world and artificial instances of the problem, with different number of nodes, in order to test how the strategies scale with respect to the size of the network. In detail, we used real-world power-law topologies of the *CAIDA dataset* [23], which are very sparse, random power-law graphs generated by the *Barabási–Albert* algorithm [24], which are slightly denser than the CAIDA ones, and *Erdős–Rényi* random graphs [25] with a number of edges equal to 20% that of the complete graph with the same number of nodes, which makes these graphs denser than the CAIDA and *Barabási–Albert* ones. We used also planar graphs generated by Boltzmann samplers [26], which have density similar to *CAIDA* graphs.

Concerning CAIDA instances, we parsed the files provided by CAIDA to obtain a weighted undirected graph, which consists of almost 35 000 nodes. We extracted seven different subgraphs of this graph, with n ranging from 50 to 1250, with a step of 200. Concerning planar graphs, we used the Boltzmann samplers to generate a 30 000 nodes planar graph. Then, we extracted seven different subgraphs of this graph as follows: for each fixed value x between 50 and 1250, we run a breadth first search from a node chosen at random and by stopping the search when x nodes have been touched; then, we considered the subgraph induced by the visited nodes. We generated random graphs of the same sizes by using also the *Barabási–Albert* and the *Erdős–Rényi* algorithms.

In order to evaluate how the proposed algorithms behave with respect to both the number of available entities and the number of black holes residing within the network, we tested the algorithms on such instances with different values of b and k. In particular, for each graph with n nodes, we have tested the strategies within different scenarios, given by different combinations of b and k that are always kept below n.

On the one side, regarding b, we chose:

- b ranging from $5\%n$ to $50\%n$ with a step of $5\%n$. The aim of this scenario is to evaluate how the strategies behave when the number of black holes within the network is quite high with respect to the size of the network itself;
- $b \in \{\ln(n), \ln^2(n), \sqrt{n}\}$. The aim of this scenario is to evaluate the behavior of the algorithms when the number of black holes within the network is small with respect to the size of the network itself.

On the other side, concerning k, we chose:

- $k = b + 3$. This choice of k is done to compare the strategies in a setting where at least one entity returns to the home-base after the network has been made safe, and to guarantee that in the *scout* strategy at least the *scout* and two *team* entities survive;
- $k \in \{\ln^3(n), n - 1\}$. This choice of k is done to compare the strategies in a setting where the number of available entities exceeds by far the number of black holes of the network. This experimental setup represents the worst possible case for strategies that do not use parallelism like *snake* and *scout*.

For each possible test instance, given by a combination of the above inputs, we choose the position of the b black holes at random, we perform 5 different experiments, and we report average values.

Analysis The results of our experiments are reported in Figs. 9–11. In details, we show the number of time steps required by the three algorithms in the case of *Erdős–Rényi* graphs, *Barabási–Albert* graphs, CAIDA graphs, and planar graphs.

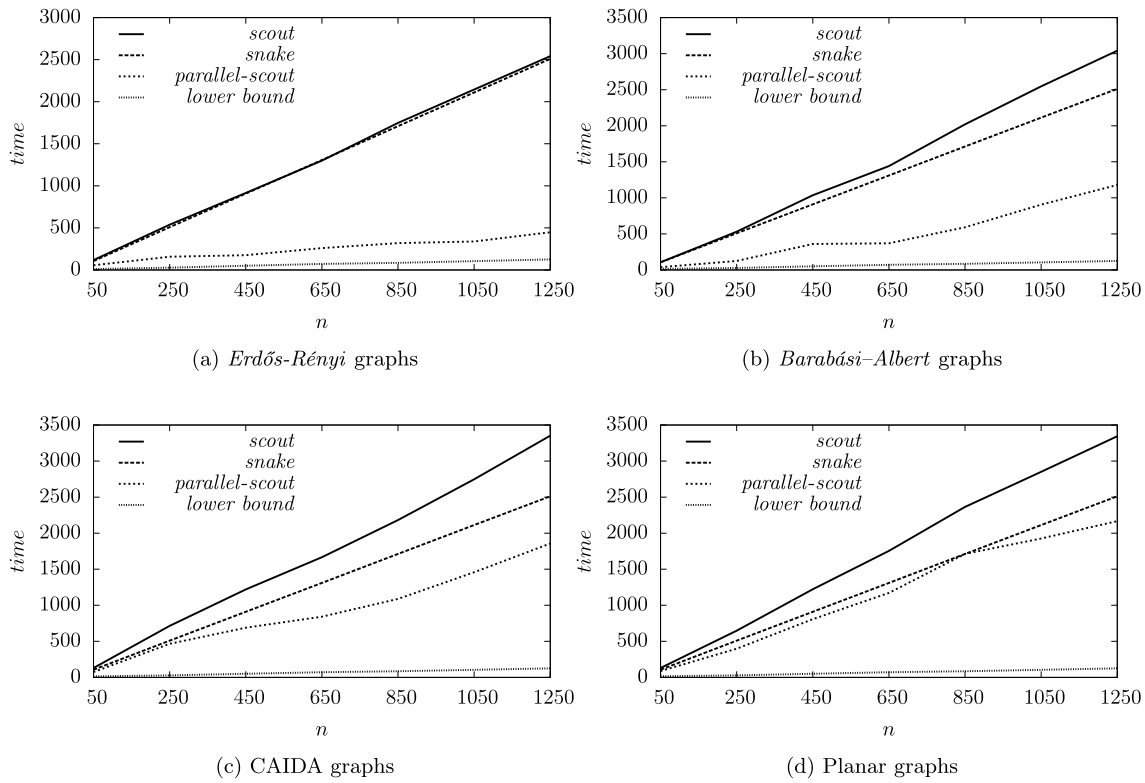

Fig. 9. Number of time steps required by *scout*, *snake* and *parallel-scout* on *Erdős–Rényi* graphs (*a*), *Barabási–Albert* graphs (*b*), CAIDA graphs (*c*) and planar graphs (*d*). The *x*-axis represents the number of nodes n ranging from 50 to 1250, with step 200. The number of black holes b is fixed to $\ln(n)$ while k equals $b+3$.

For each instance of the 1-ER problem, we report also the exact value of the lower bound of Lemma 3.4, which holds for every $r > 0$. According to the analysis in the proof of Lemma 3.4, the value of the lower bound is computed as $\max\{\frac{n-1}{k},\ ecc(s) + (r+1)\cdot(\min\{b, \ell(P)\} - 1) + 2\cdot(|K_{\max}| - 1) - 1\}$, with P being the longest simple path in the input graph.

We show the results for the following scenarios:

- $b = \ln(n)$ with $k = b + 3$ (Fig. 9), and $b = \ln^2(n)$ with $k = \ln^3(n)$ (Fig. 10). The time steps required by the strategies and the lower bounds are presented as a function of the number of nodes n which ranges from 50 to 1250, with step 200;
- $n = 1250$ and $k = n - 1$ (Fig. 11). The results are shown as a function of the number of black holes b which ranges from 5%n to 50%n with a step of 5%n.

Results for other combinations of n, b and k are omitted, as they are comparable to those of the above ones and lead to similar conclusions.

Our experiments clearly confirm the analysis of Section 3, that is: (i) the number of steps required by *snake* is exactly $2(n-1) + 2b$, while that of *scout* is exactly $2(n-1) + 2|I|$; (ii) the number of steps required by all the strategies directly depends on n (see Figs. 9–10). However, the experiments show how the dependency on n of the time complexity faced by the *parallel-scout* strategy is drastically reduced with respect to *scout* and *snake*. Such a reduction is even larger as the density of the input graph increases. Concerning the number of steps required by *scout* it does not depend on b, whereas that required by *snake* grows linearly as a function of b (see Fig. 11).

Our experiments also show (see Fig. 10) that in the CAIDA graphs and in the planar graphs *snake* is better than *scout*, as $|I| >> b$; in the *Barabási–Albert* graphs *snake* is almost always better than *scout*, as in most of the cases $|I| > b$; in the *Erdős–Rényi* graphs *scout* is better than *snake*, as $|I| < b$. Given a fixed b, this graph-depending behavior is due to the structure of the BFS tree T, whose number of leafs grows with the density of the graph. The same behavior is somehow confirmed by Fig. 11, where we show that *snake* is better than *scout* until a certain value of b, after which *scout* outperforms *snake*. Such value of b linearly depends on $|I|$ and it is around 35%n in the CAIDA instances and planar instances, around 20%n in the *Barabási–Albert* instances, and around 5%n, in the denser *Erdős–Rényi* instances.

Finally, our experiments clearly show that *parallel-scout* is very effective in practice. In fact, its number of steps is always by far smaller than those required by *scout* and *snake*, as shown in Figs. 9–10, thus resulting in the best strategy in all the cases.

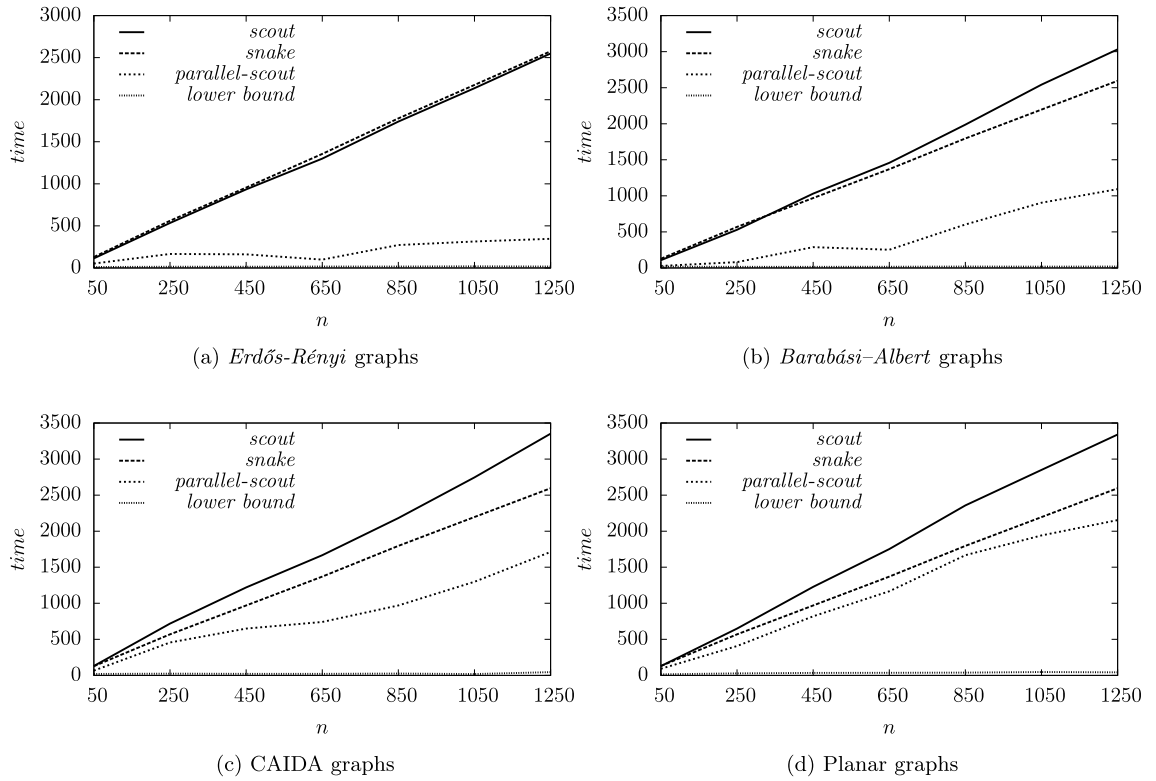

Fig. 10. Number of time steps required by *scout*, *snake* and *parallel-scout* on *Erdős–Rényi* graphs (*a*), *Barabási–Albert* graphs (*b*), CAIDA graphs (*c*) and planar graphs (*d*). The *x*-axis represents the number of nodes n ranging from 50 to 1250, with step 200. The number of black holes b is fixed to $\ln^2(n)$ while k equals $\ln^3(n)$.

Concerning the practical performance of the strategies with respect to the exact value of the lower bound, we distinguish two cases: (i) the dense *Erdős–Rényi* instances, and (ii) the sparser CAIDA, *Barabási–Albert*, and planar instances. In case (i) (see Figs. 9(a)–10(a) and 11(a)), the performance of the *parallel-scout* strategy is quite close to the lower bound, since in dense graphs the parallel moves of this strategy are quite effective. While, in case (ii) (see Figs. 9(b, c, d)–10(b, c, d) and 11(b, c, d)) the gap between the performance of *parallel-scout* and the lower bound increases. This is due to the fact that the average degree of nodes in these instances is quite small. This is more evident in the planar graphs, where the average degree is at most 4. Clearly, this does not affect the validity of the proposed strategies with respect to the asymptotic lower bound.

Another evidence that is pointed out by our experiments is the relationship that exists between the exact value of the lower bound of Lemma 3.4 and the specific scenario given by a combination of n, b and k. On the one side, when k is small with respect to n, the lower bound is dominated by its first term ($\frac{n-1}{k}$). An example of this behavior is shown in Fig. 9, where $b = \ln(n)$ and $k = b + 3$. On the other side, when k is closer to n, the exact lower bound is dominated by the second term which is almost always a small constant. Example of this behavior are shown in Fig. 10, where $b = \ln^2(n)$ and $k = \ln^3(n)$, and in Fig. 11, where $k = n - 1$.

For the sake of completeness, we performed experiments also for instances of the 0-ER problem in similar scenarios. Along with the three strategies we compare the results with the exact value of the lower bound for general r, that is $\max\{\frac{n-1}{k}, ecc(s)\}$. The results of our experiments in this case show similar behavior with respect to the case $r = 1$, and hence we omit them.

6. Implications for the case $r > 1$

In Section 4 we have proposed three different strategies to solve the r-ER problem when $r \leq 1$ and the number k of entities is bounded. In what follows, we provide some hints on how to extend the proposed strategies to the case $r > 1$.

Regarding the *snake* strategy, an idea could be that of interleaving r empty nodes between two consecutive entities composing the "snake". The exploration would still behave as described in Section 4, but at some time, the shape of the subsequent entities released could not necessarily look like a snake. In fact, the first entity follows the exploration walk. Any other entity must be kept at distance $r + 1$ from the position p of its predecessor, in such a way it is able to reach p within $r + 1$ steps. In doing so, $2(n - 1) + 2rb$ steps would be required. However, every entity must be kept also at distance grater than r from any other entity already released. This is still realized by a "snake" if the input graph admits enough

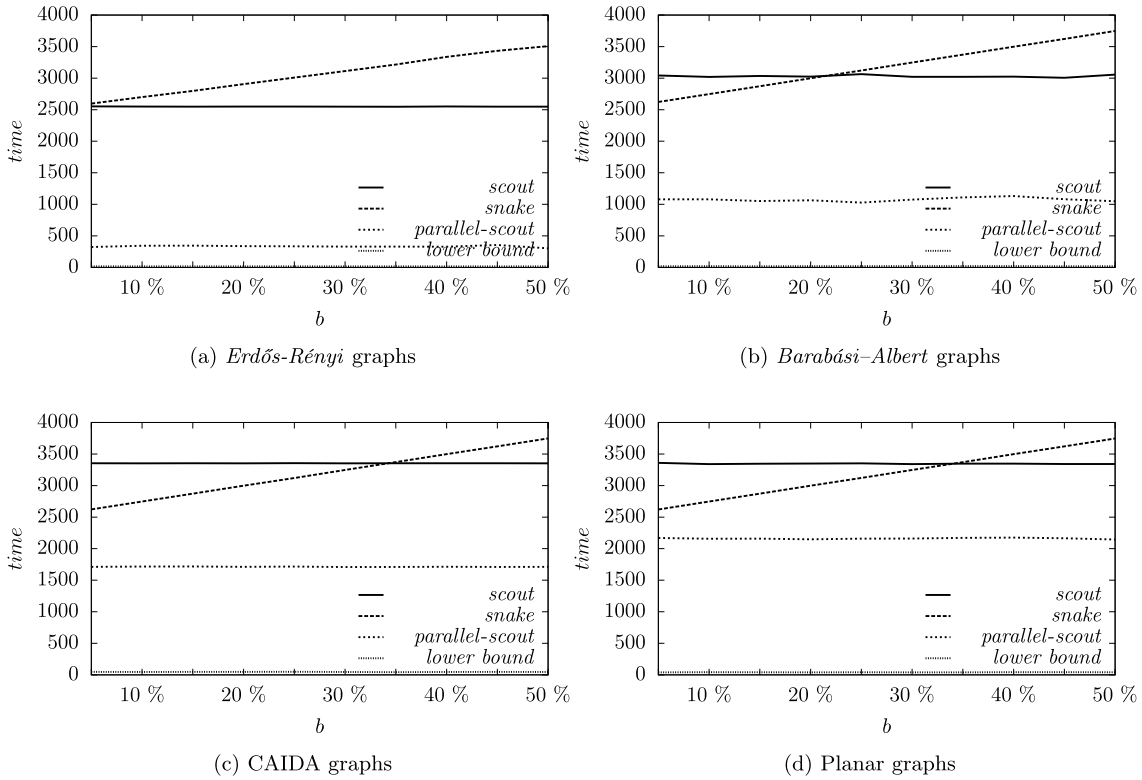

Fig. 11. Number of time steps required by *scout*, *snake* and *parallel-scout* on *Erdős–Rényi* graphs (*a*), *Barabási–Albert* graphs (*b*), CAIDA graphs (*c*) and planar graphs (*d*). The *x*-axis represents the number of black-holes b ranging from 5%n to 50%n, with step 5%n. The number of nodes n is fixed to 1250 while k equals $n-1$.

space, otherwise different movements might be imposed to the entities. Since entities are synchronized and the exploration walk is known to every entity, it should be possible to impose the right movements to each entity accordingly, possibly with the side effect of increasing the number of steps required. This implies that the number of required time steps for the exploration would be greater than $2(n-1)+2rb$.

Concerning the *scout* strategy and its *parallel-scout* variant, we can follow the proof of Lemma 4.3, and we obtain that, instead of three steps required by a general round of *scout* when $r=1$, the strategy would require $\lceil \frac{3}{2}(r+1) \rceil$ time steps. This is obtained by letting move the *scout* entity $r+1$ steps in the down direction, and then meet the *team* entities in the middle of the traversed path which requires further $\lceil \frac{1}{2}(r+1) \rceil$ steps. This implies that the *team* entities never reach leaves nor nodes at distance smaller than $\lceil \frac{1}{2}(r+1) \rceil$ from a leaf. Concerning all the nodes at distance $d \leq r$ from the home-base s, these are made safe by the *scout* in $\lceil \frac{3}{2}(d+1) \rceil$ steps making known also the *team* entities. Furthermore, whenever the *team* and the *scout* entities walk together in the *up* direction until reaching a node x with an unexplored child y, the *team* goes towards the parent of x at distance $r+1$ from the *scout*, while the *scout* waits r time steps before exploring y. Subsequently, the *team* entities and the *scout* (if survived) meet at the middle node of their joining path as described above. Hence y requires $\lceil \frac{3}{2}r \rceil$ steps to be explored.

Concerning the *parallel-scout* strategy, the independent set to compute in general is not among the children of a node, but among the nodes to be explored of the i-th level of T, $i > \lceil \frac{r+1}{2} \rceil$, reachable from the current position of the entities. The set S to compute is a generalization of the classical independent set, called *Distance-d Independent Set* (see, e.g. [27]). This is such that for any pair of nodes u, $v \in S$, the distance between u and v is at least d. In our case, d must be set to $r+1$.

7. Concluding remarks

In this paper, we have studied the problem of using k mobile entities to explore and make safe a graph with an unknown number b of black-holes. In particular, we have considered the scenario where the destruction of an entity by means of a black-hole also affects all the entities within a fixed range r.

We have shown that this problem is NP-hard even when $b=k=1$; we have provided two general lower bounds for the cases of $r \geq 0$ and $r > 0$. For $r \leq 1$, we have proposed an optimal solution for the case of unbounded k, and three different exploration strategies for the case of bounded k. We have provided for all of them the analysis of performance guaran-

tees along with correctness proofs. We have then provided an experimental study to show the practical performance of the proposed strategies with respect to many different scenarios. Finally, we have provided some hints concerning possible extensions of the proposed strategies to the case $r > 1$. These observations show that the extensions are not so straightforward. Then, a first open problem is that of studying how the increase of r impacts on the bounds provided for the cases of $r \leq 1$. Accordingly, the design of completely new exploration strategies for the case $r > 1$ surely deserves to be investigated. Moreover, the study of tighter lower bounds for the cases of $r \leq 1$ remains certainly of main interest.

Other interesting research directions that have to be pointed out are those arising by considering a more general model in which the exploration graph G is weighted and the radius r represents the distance of two nodes in G computed on the basis of the weights associated to its edges.

Finally, another variant of the model that could deserve further investigation is that concerning the case where the repairing of a black-hole also involves other black-holes within the range r, hence producing a sort of "chain explosion".

Acknowledgements

The authors thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] M. D'Emidio, D. Frigioni, A. Navarra, Exploring and making safe dangerous networks using mobile entities, in: 12th International Conference on Ad Hoc Networks and Wireless, ADHOC-NOW, in: Lecture Notes in Computer Science, vol. 7960, Springer, 2013, pp. 136–147.
[2] P. Flocchini, D. Ilcinkas, A. Pelc, N. Santoro, Computing without communicating: ring exploration by asynchronous oblivious robots, Algorithmica 65 (2013) 562–583.
[3] L. Gąsieniec, R. Klasing, R.A. Martin, A. Navarra, X. Zhang, Fast periodic graph exploration with constant memory, J. Comput. System Sci. 74 (5) (2008) 802–822.
[4] L. Gąsieniec, T. Radzik, Memory efficient anonymous graph exploration, in: Graph-Theoretic Concepts in Computer Science, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 14–29.
[5] A. Kosowski, A. Navarra, Graph decomposition for improving memoryless periodic exploration, Algorithmica 63 (1–2) (2012) 26–38.
[6] J. Czyzowicz, D. Kowalski, E. Markou, A. Pelc, Searching for a black hole in synchronous tree networks, Combin. Probab. Comput. 16 (2007) 595–619.
[7] R. Klasing, E. Markou, T. Radzik, F. Sarracco, Approximation bounds for black hole search problems, Networks 52 (4) (2008) 216–226.
[8] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Searching for a black hole in arbitrary networks: optimal mobile agents protocols, Distrib. Comput. 19 (1) (2006) 1–18.
[9] S. Dobrev, P. Flocchini, G. Prencipe, N. Santoro, Mobile search for a black hole in an anonymous ring, Algorithmica 48 (1) (2007) 67–902.
[10] J. Czyzowicz, S. Dobrev, R. Kralovic, S. Miklik, D. Pardubska, Black hole search in directed graphs, in: Proc. of the 16th Int. Colloquium on Structural Information and Communication Complexity, SIROCCO, in: Lecture Notes in Computer Science, vol. 5869, 2009, pp. 182–194.
[11] A. Kosowski, A. Navarra, C. Pinotti, Synchronous black hole search in directed graphs, Theoret. Comput. Sci. 412 (41) (2011) 5752–5759.
[12] S. Dobrev, P. Flocchini, N. Santoro, Improved bounds for optimal black hole search in a network with a map, in: Proc. of the 11th Int. Colloquium on Structural Information and Communication Complexity, SIROCCO, in: Lecture Notes in Computer Science, vol. 3104, 2004, pp. 111–122.
[13] B. Balamohan, P. Flocchini, A. Miri, N. Santoro, Time optimal algorithms for black hole search in rings, Discrete Math. Algorithms Appl. 3 (4) (2011) 457–472.
[14] S. Dobrev, P. Flocchini, R. Kralovic, G. Prencipe, P. Ruzicka, N. Santoro, Black hole search in common interconnection networks, Networks 47 (2) (2006) 61–71.
[15] E. Markou, M. Paquette, Black hole search and exploration in unoriented tori with synchronous scattered finite automata, in: 16th Int. Conf. on Principles of Distr. Systems, OPODIS, in: LNCS, vol. 7702, Springer, 2012, pp. 239–253.
[16] C. Cooper, R. Klasing, T. Radzik, Locating and repairing faults in a network with mobile agents, Theoret. Comput. Sci. 411 (14–15) (2010) 1638–1647.
[17] P. Flocchini, D. Ilcinkas, N. Santoro, Ping pong in dangerous graphs: optimal black hole search with pebbles, Algorithmica 62 (3–4) (2012) 1006–1033.
[18] S. Dobrev, P. Flocchini, R. Královic, N. Santoro, Exploring an unknown dangerous graph using tokens, Theoret. Comput. Sci. 472 (2013) 28–45.
[19] J. Cai, P. Flocchini, N. Santoro, Decontaminating a network from a black virus, Int. J. High. Perform. Comput. Networking 4 (1) (2014) 151–173.
[20] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Company, New York, 1979.
[21] OMNeT++, Discrete event simulation environment, http://www.omnetpp.org.
[22] M. Halldorsson, J. Radhakrishnan, Greed is good: approximating independent sets in sparse and bounded-degree graphs, Algorithmica 18 (1997) 145–163.
[23] Y. Hyun, B. Huffaker, D. Andersen, E. Aben, C. Shannon, M. Luckie, K. Claffy, The CAIDA IPv4 routed/24 topology dataset, http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml.
[24] R. Albert, A.-L. Barabási, Emergence of scaling in random networks, Science 286 (1999) 509–512.
[25] B. Bollobás, Random Graphs, Cambridge University Press, 2001.
[26] É. Fusy, Uniform random sampling of planar graphs in linear time, Random Structures Algorithms 35 (4) (2009) 464–522.
[27] H. Eto, F. Guo, E. Miyano, Distance-d independent set problems for bipartite and chordal graphs, in: G. Lin (Ed.), Combinatorial Optimization and Applications, in: Lecture Notes in Computer Science, vol. 7402, Springer, Berlin, Heidelberg, 2012, pp. 234–244.