Theory of Computing Systems manuscript No.
(will be inserted by the editor)

Searching for Black Holes in Subways

**Paola Flocchini · Matthew Kellett ·
Peter C. Mason · Nicola Santoro**

Submitted: September 2010

Abstract Current mobile agent algorithms for mapping faults in computer networks assume that the network is static. However, for large classes of highly dynamic networks (e.g., wireless mobile ad hoc networks, sensor networks, vehicular networks), the topology changes as a function of time. These networks, called delay-tolerant, challenged, opportunistic, etc., have never been investigated with regard to locating faults. We consider a subclass of these networks modelled on an urban subway system. We examine the problem of creating a map of such a subway. More precisely, we study the problem of a team of asynchronous computational entities (the mapping agents) determining the location of black holes in a highly dynamic graph, whose edges are defined by the asynchronous movements of mobile entities (the subway carriers). We determine necessary conditions for the problem to be solvable. We then present and analyze a solution protocol; we show that our algorithm solves the fault mapping problem in subway networks with the minimum number of agents possible, $k = \gamma + 1$, where γ is the number of carrier stops at black holes. The number of carrier moves between stations required by the algorithm in the worst case is $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$, where n_C is the number of subway trains, and l_R is the length of the subway route with the most stops. We establish lower bounds showing that this bound is tight. Thus, our protocol is both agent-optimal and move-optimal.

Paola Flocchini
School of Information Technology and Engineering, University of Ottawa, Ottawa, Canada

Matthew Kellett, Peter C. Mason
Defence R&D Canada – Ottawa, Government of Canada, Ottawa, Canada

Nicola Santoro
School of Computer Science, Carleton University, Ottawa, Canada

1 Introduction

1.1 Background and problem

Distributed algorithms are often designed under the assumptions that the network on which they run is both reliable and connected. However, these assumptions are often far from reality. In the real world, a reliable network—one that has no faults over the lifetime of the network—is the exception rather than rule. Faults are often undetectable, caused by the failure of computers and network equipment. And while the assumption of connectivity is more or less realistic for wired networks, it is entirely unrealistic for emerging dynamic networks with topologies that change as a function of time. These networks are often disconnected at any point in time but can be connected over some time interval.

There is a large body of research into distributed algorithms for finding faults. By mapping the faults in a network, algorithms that come afterwards can work in the network as if there are no faults. The fault-finding algorithms use mobile agents—autonomous, mobile, computational entities—to do the mapping. The faults, from an agent point of view, include nodes that eliminate agents arriving at them without leaving a discernible trace, often referred to as *black holes*, and links between neighbouring nodes that have the same effect, often referred to as *black links*. The problem of mapping a network with black holes is often referred in the literature to as the black hole search (Bhs) problem, while the problem of finding both black holes and black links is often referred to as the dangerous graph exploration problem (see [8–15, 17, 18, 20, 23–25] for examples of both types of problem). As we discuss below, most of this work is focussed on static networks, some with specific topologies such as the ring.

There is also a large body of research into distributed algorithms for networks with dynamic topologies. There are several classes of these networks that have emerged in the last decade or two. These include, but are not limited to, wireless mobile ad hoc networks where the network's topology may change dramatically over time due to the movement of the network's nodes; sensor networks where links only exist when two neighbouring sensors are awake and have power; and vehicular networks, similar to mobile ad hoc networks, where the topology changes constantly as vehicles move. These networks are often referred to in the literature as *delay-tolerant*, *challenged*, *opportunistic*, *evolving*, etc (e.g., see [6, 7, 21, 22, 29, 31]). However, the work on these networks mostly focusses on broadcasting and routing (e.g., see [3, 4, 21, 26, 27, 30, 31]). There has been little work on mobile agent algorithms for networks with dynamic topologies. One study [19] has looked at how agents can explore one class of these networks: periodically-varying graphs. In the periodically-varying graph (PV graph) exploration problem, agents ride carriers between sites in the network. A link only exists between sites when a carrier is passing between them. The agents explore the network by moving from carrier to carrier when they meet at a site.

We are interested in how to combine these two concepts and search for black holes in dynamic networks. We are also interested in deterministic solutions to the problem. As a result, we look at the black hole search problem in a class of networks similar to PV graphs, based on a subway system. Like PV graphs, the subway model includes carriers that travel on repeating routes amongst the sites in the network. However, unlike PV graphs, agents in the subway model can disembark from a carrier onto a site and, in fact, must do so in order to find the faults in the networks. The faults are black holes that eliminate agents but leave the carriers unaffected.

The subway model gives us a number of benefits as a model of dynamic networks. Subway systems, by their very nature as public transportation systems, are strongly connected directed graphs, allowing passengers at any station to reach any other station in the subway system. However, their connectivity is dynamic in that a link only really exists between two neighbouring stations when a train is moving between them. Mapping the subway model onto real-world computing systems, the subway model can be used to describe *opportunistic* or *parasitic* movement by computational entities, like mobile agents, in a computer network. For example, take a team of mobile agents passively scanning a network for nodes infected with malicious code. The malicious code resides in a node's user space and is agent aware, eliminating all agents it sees. The agents can travel through the network opportunistically using control traffic, which passes safely through kernel space of intermediate nodes. It is only when an agent steps off a control packet into user space that it becomes vulnerable.

In fact, the subway model is more generally applicable than this simple passive scanning example. It allows us to look at the effect of mobile agents moving through a network using other entities' movements. Normally, mobile agent algorithms for black hole search assume that the agent is mobile and can freely move between neighbouring nodes. In the subway model, the agents are at the mercy of the movements of the carriers. As we will show, there are costs for both moving and waiting to move, neither of which can be avoided.

1.2 Our contribution

We introduce the subway model, a way of looking at the effects of mobile agents moving opportunistically or parasitically in a network. The class of networks described by the model is much larger than the set of real subway systems and includes some real-world computer systems. We look at the asynchronous version of the black hole search problem where the agents' calculations and the carriers' movements take a finite but unpredictable amount of time to complete. We introduce a new measure of complexity, carrier moves, that is specific to the subway model. The carrier moves metric combines agents moves, the traditional measure of complexity for asynchronous mobile agent measurements, with agent waits, the cost to the agent of waiting for the carrier to arrive. We show that neither of these agent costs can be avoided in a system where the agents must rely on other entities for movement.

We first investigate, in Section 3, the computability of the Bhs problem and establish necessary conditions for the problem to be solvable. In particular, the number k of agents must greater than the faulty load $\gamma(\mathbf{G})$, which is the number of carrier stops that are black holes. Hence, any minimal solution requires at least $\gamma(\mathbf{G}) + 1$ agents.

We then prove, in Section 4, that the limitations on computability are indeed tight. In fact, we prove that all necessary conditions are also sufficient. We do so by designing a protocol for the Bhs problem in the subway model. We prove its correctness and analyze its complexity. Our solution has a complexity $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, where $k \geq \gamma(\mathbf{G}) + 1$ is the number of agents, n_C is the number of carriers, and l_R is the length of the longest carrier route.

Our solution is worst-case agent and move optimal. In fact, in Section 5, we establish a lower bound on the worst case complexity of carrier moves, proving that $\Omega(\gamma \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves are needed.

1.3 Related work

There is a large amount of work on distributed algorithms for finding faults. There are a number of papers that look at black hole search in networks with specific topology. Dobrev et al. look at algorithms for finding a single black hole in several specific topologies: hypercubes, cube-connected cycles, star graphs, wrapped butterflies, chordal rings, and restricted-diameter multidimensional meshes and tori [13]. Czyzowicz et al. look at finding a single black hole in a bounded-synchronous tree network, where agent moves take an unpredictable but bounded amount of time [12]. Dobrev et al. look at finding a single black hole in an anonymous ring network [15]. These algorithms take advantage of the characteristics of the network's specific topology, most, if not all, of which would be disrupted by the addition or deletion of a single link.

Some work looks at bounds on solutions to the Bhs problem in arbitrary networks. Czyzowicz et al. determine a lower bound for agents finding a single black hole in an arbitrary bound-synchronous network [11]. Klasing et al. improve on the lower bound from [11] and look at the problem for two agents searching for a black hole in an arbitrary synchronous network [23]. In [24], the same authors find a lower bound for the synchronous network problem and further improvements on the lower bound from [11]. Kosowski et al. express the lower bound for the static network problem in terms of the network's maximum degree and look at the same problem in directed graphs [25]. Unfortunately, while all these papers deal with arbitrary networks, they assume that networks remain static and that agent movements are synchronous.

There are also papers that look at Bhs solutions under different assumptions on the amount of knowledge available to the agents, although the assumptions always implicitly include the networks remaining static. In a journal version of one of the earliest conference papers on black hole search by agents, Dobrev et al. look at the effects of topological knowledge—complete ignorance, complete knowledge, and sense of direction—on the search for a single black hole [14]. Flocchini et al. show that pure tokens (single bit) are computationally just as powerful as whiteboards (shared memory on nodes) in solving the single black hole search problem [17]. Glaus shows that it is still possible to solve the single black hole search problem even if the agents have no knowledge of the link from which they arrive on a node [20].

Some papers look at the search for multiple black holes or black holes and black links. For example, Cooper et al. study the multiple black hole search problem in synchronous systems [9]. Chalopin et al. look at the related problem of agent rendezvous in the presence of both black holes and black links in an anonymous network [8]. Under certain assumptions, their rendezvous solution also solves the dangerous graph exploration problem. Flocchini et al. consider specifically the problem of dangerous graph exploration in non-anonymous networks [17]. Recently, the optimality of black hole search has been studied in [2].

Unlike the work on finding faults, there is less work on dynamic networks that is relevant to ours. Most looks at routing and broadcast in dynamic networks that use message passing. Bui Xuan et al. developed many of the metrics used to measure message passing dynamic networks, including developing the idea of journeys—the paths of dynamic networks—and cost measures such as hop count, arrival time, and time span [3]. O'Dell and Watenhoffer look at the algorithmic limits for broadcasting information in dynamic networks where the links change but the underlying network remains connected, not unlike the subway model [27]. Three papers propose routing protocols specifically for delay-tolerant networks, where the connections can be very unpredictable [4, 26, 31]. Zhang et al. actually look at real-world performance of routing protocols for dynamic networks on UMass DieselNet, a network made up of WiFi nodes on buses at the University of Massachusetts Amherst [30].

There is relatively little research into agents or agent-like entities working in dynamic networks. Avin et al. look at the cost of random walks in evolving graphs where the topology of the network changes each time step as links are inserted or deleted [1]. As we mentioned in the introduction, Flocchini et al. look at agents exploring a dynamic network deterministically [19]. However, the solution proposed deals only with exploration and, because the agents never leave the carriers, it is difficult to introduce the idea of a fault into their graph model. Furthermore Casteigts et al. [5] study the problem of broadcasting with termination detection in highly dynamic networks under unstructured mobility, that is when the edges of the dynamic graph may appear infinitely often but without any (known) pattern. The class of dynamic networks that we consider is the periodic subclass corresponding to carrier graphs.

2 Definitions and Terminology

We consider a set C of n_C *carriers* that move among a set S of n_S *sites*. A carrier $c \in C$ follows a route $R(c)$ between all the sites in its *domain* $S(c) = \{s_0, s_1, \ldots, s_{n_S(c)-1}\} \subseteq S$, where $n_S(c) = |S(c)|$. A carrier's *route* $R(c) = \langle r_0, r_1, \ldots, r_{l(c)-1} \rangle$ is a cyclic sequence of *stops*: after stopping at site $r_i \in S(c)$, the carrier will move to $r_{i+1} \in S(c)$, where all operations on the indices are modulo $l(c) = |R(c)|$ called the *length* of the route. Carriers move *asynchronously*, taking a finite but unpredictable amount of time to move between stops. We call a route *simple* if $n_S(c) = l(c)$. A *transfer site* is any site that is in the domain of two or more carriers. A *terminal stop* is denoted by a bar over the stop (e.g., $R(c) = \langle \ldots, \bar{r}_i, \ldots \rangle$) is a stop where all passengers on a carrier are forced to disembark while the carrier continues on its route.[1] Note that excluding terminal stops would have no effect on the model discussed here or the algorithm presented in the next section.

A carrier's route $R(c) = \langle r_0, r_1, \ldots, r_{l(c)-1} \rangle$ defines an edge-labeled directed multigraph $\mathbf{G}(c) = (S(c), \mathbf{E}(c), \lambda(c))$, called a *carrier graph*, where $S(c)$ are the nodes, $\mathbf{E}(c)$ are the edges[2], and $\lambda(c)$ the set of labels, and where there is an edge labeled $(c, i+1)$ from r_i to r_{i+1}, and the operations on indices and inside labels are modulo $l(c)$. The entire network is then represented by the edge-labelled directed multigraph $\mathbf{G} = (R, \mathbf{E}, \lambda)$, called a *subway graph*, where $R = \cup_{c \in C} R(c)$, $\mathbf{E} = \cup_{c \in C} \mathbf{E}(c)$, and $\lambda = \{\lambda(c) : c \in C\}$. Associated to the subway graph is the *transfer graph* of $\mathbf{G}$, which we define as the edge-labeled undirected multigraph $H(\mathbf{G}) = (C, E_T)$ where the nodes are the carriers and, $\forall c, c' \in C, c \neq c', s \in S$, there is an edge between c and c' labeled s iff $s \in S(c) \cap S(c')$, i.e., s is a transfer site between c and c'. In the following, where no ambiguity arises, we will omit the edge labels in all graphs.

Working in the network is a team A of k computational *agents* that start at unpredictable times from the same site, called the *homebase*. The agents move opportunistically around the network using the carriers. An agent can move from a carrier to a site (*disembark* at a stop) or from a site to a carrier (*board* a carrier), but not from one carrier to another directly.[3] An agent on a transfer site can board any carrier stopping at it. When travelling on a carrier, an agent can count the number of stops that the carrier has passed, and can decide whether or not to disembark at the next stop.

Agents communicate with each other using shared memory, available at each site in the form of a *whiteboard*, which is accessed in fair mutual exclusion. The agents are *asynchronous* in that they take a finite but unpredictable

[1] We include terminal stops to allow us to model real life subway systems where staying indefinitely on the same train is not possible because trains are often switched out at end stations.

[2] Although self-loops have no real-world equivalents, we allow them for theoretical completeness.

[3] As a result, agents forced off at a terminal stop, somewhat unrealistically, must wait for the carrier to return to that stop before they can board the same carrier again.

amount of time to perform computations at a site. All agents execute the same protocol.

Among the sites there are $n_B < n_S$ *black holes*: sites that eliminate agents disembarking on them without leaving a discernable trace; black holes do not affect carriers. The *black hole search* (Bhs) problem is that of the agents determining the locations of the black holes in the subway graph. A protocol solves the Bhs problem if within finite time at least one agent survives and all surviving agents enter a terminal state and know which *stops* are black holes. Let $\gamma(c) = |\{i : r_i \in R(c) \text{ is a black hole}\}|$ be the number of black holes among the stops of c; and let $\gamma(\mathbf{G}) = \sum_{c \in C} \gamma(c)$, called the *faulty load* of subway graph $\mathbf{G}$, be the total number of stops that are black holes. Note that, since a site might appear in more than one route, in general $\gamma(\mathbf{G}) \geq b(\mathbf{G})$, where $b(\mathbf{G})$ is the number of black holes in $\mathbf{G}$.

As in traditional mobile agent algorithms, the basic cost measure used to evaluate the *efficiency* of a Bhs solution protocol is the *size* of the team, that is, the number k of agents needed by the protocol. To solve Bhs, it is obviously necessary to have more agents than the faulty load of the network, i.e. $k > \gamma$. A solution protocol is *agent optimal* if it solves the Bhs problem for $k = \gamma + 1$.

The other cost measure is the number of *carrier moves*, which is a combination of the traditional mobile agent algorithm metric of agent moves with the cost of waiting imposed on the agent by its use of opportunistic movement in the network. When an agent is riding on a carrier, which we call *agent moves*, or waiting for a carrier, which we call *agent waits*, we count each move made by that carrier as a carrier move for the agent. A solution protocol is *move optimal* in the worst case if the total number of carrier moves incurred by all agents in solving the Bhs problem is the best possible.

Throughout the rest of the paper, we use the subway graph presented in Table 1 as an example to help explain how our proposed solution works. Fig. 1 shows a view of the subway graph $\mathbf{G}$. The routes of carriers c_3 and c_5 are simple while the other carriers' routes are not. Sites 9 and 10 are terminal stops on carrier c_4 but normal stops on carrier c_2. The same for site 15, which is terminal on carrier c_5 but normal on carrier c_3. Fig. 2 shows the transfer graph associated with the example. Note that the transfer graph remains connected when the black holes are removed, including transfer site 13. Fig. 3 shows the routes $R(c)$ for each carrier c. Even though there are only three black holes (nodes 6, 12, 13), the routes show that there are $\gamma = 8$ black hole stops in the network.

3 Basic Limitations and Assumptions

There are some basic limitations for the Bhs problem to be solvable in subway graphs; these in turn dictate some necessary assumptions.

Since solving the Bhs problem requires visiting all carrier stops, some immediate limitations follow from those existing for the easier *safe exploration* Exp problem: all sites are safe; within finite time all the exploring agents enter

Carrier	**Route**	**Length**	**Domain**	**Size**
c	$R(c) = \langle r_0, \ldots, r_{l(c)-1} \rangle$	$l(c) = \|R(c)\|$	$S(c) = \{s_0, \ldots, s_{n_S(c)-1}\}$	$n_S(c) = \|S(c)\|$
c_1	$\langle 1, 3, 3, 2, \bar{4}, 4, 2, 6, 1, 2, 5, 6 \rangle$	12	$\{1, 2, 3, 4, 5, 6\}$	6
c_2	$\langle 1, 9, 10, 9 \rangle$	4	$\{1, 9, 10\}$	3
c_3	$\langle 5, 7, 8, 15, 13 \rangle$	5	$\{5, 7, 8, 13, 15\}$	5
c_4	$\langle \bar{9}, 11, 13, 14, 12, \bar{1}0, 12, 14, 13, 11 \rangle$	10	$\{9, 10, 11, 12, 13, 14\}$	6
c_5	$\langle 14, 13, \bar{1}5, 16 \rangle$	4	$\{13, 14, 15, 16\}$	4
Homebase: $s = 1$; Black holes: $6, 12, 13$, $n_B = 3$; Transfer sites: $1, 5, 9, 10, 13, 14, 15$				
Terminal sites: 4 (r_4 on c_1), 9 (r_0 on c_4), 10 (r_5 on c_4), and 15 (r_2 on c_5)				

Table 1 Example subway graph.

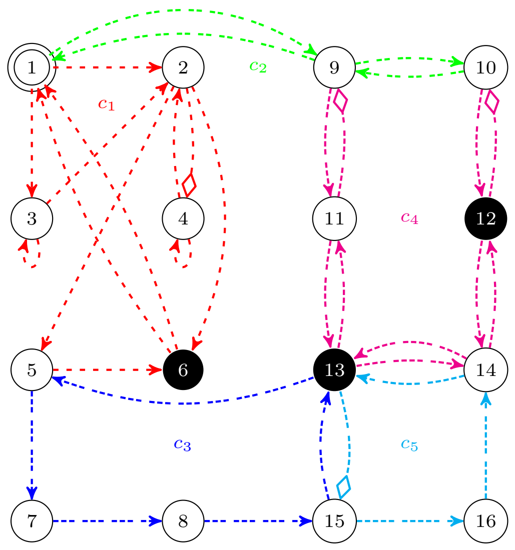

Fig. 1 Subway graph of example. Nodes in black are black holes; doubly-circled node is the homebase.. Diamond arrow head indicates next stop is terminal.

a terminal state, and all sites have been visited by at least one agent. First of all, the Exp problem is deterministically *unsolvable* if the carriers do not have distinct ids visible to the agents.

Lemma 1 *If the carriers do not have distinct ids visible to the agents, the* Exp *problem is deterministically* unsolvable. *This result holds regardless of the number $k \geq 1$ of agents and even if the agents know n_C, n_S, and the length of the routes.*

Proof By contradiction, let P be a deterministic protocol that always allow a team of $k > 0$ agents to explore all the sites of all subway graphs in which

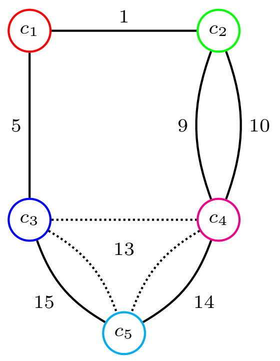

Fig. 2 Transfer graph $H(\mathbf{G})$ of example. Edge labels are corresponding transfer site ids. Transfer site 13 is a black hole.

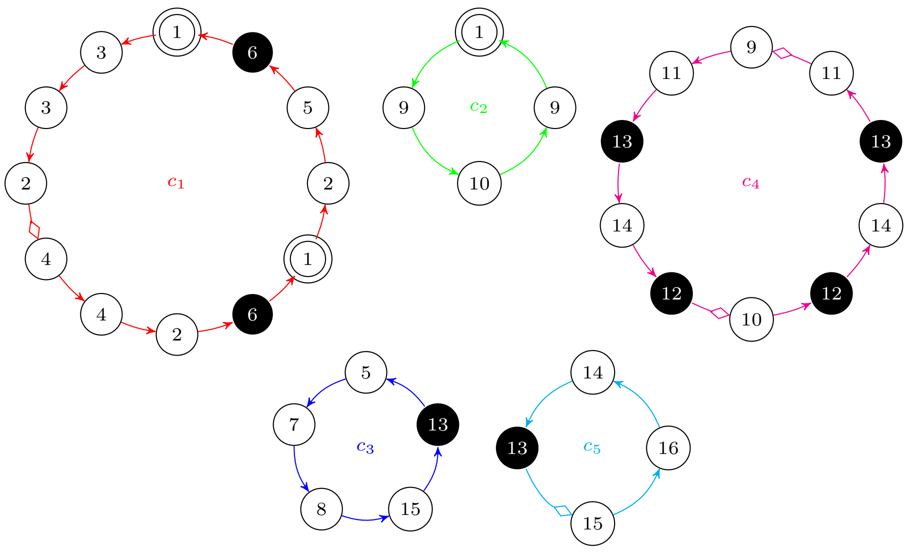

Fig. 3 Carrier routes for carriers in example. Diamond arrow head indicates that the next stop is a terminal stop.

there are no black holes but the carriers do not have distinct identities visible to the agents. Consider now the subway graph $\mathbf{G}$, corresponding to the routes $R(c_1)$ and $R(c_2)$ of only two carriers c_1 and c_2, in which the only transfer site is the homebase s and $S(c_1) = S(c_2) > 1$. Since the two carriers have no visible identifiers, an agent at s cannot distinguish whether an arriving carrier is c_1 or c_2. An adversary can clearly choose the speed of the two carriers in such a way that, whenever an agent a at s decides to board the next carrier, the carrier arriving there is c_1. In other words, none of the agents will ever board c_2 and visit the sites reachable only through that route, contradicting the correctness of P. $\square$

In other words, the carriers must have distinct identities visible to the agents for the problem to be solvable.

Next notice that some metric information, either the number of carriers n_C or the number of sites n_S, must be be available to the agents for exploration, and thus black hole search, to be possible.

Lemma 2 *If the agents have no knowledge of n_C nor of n_S, the* Exp *problem is deterministically* unsolvable. *This result holds regardless of the number $k \geq 1$ of agents and even if the carriers have distinct visible ids.*

Proof By contradiction, let P be a deterministic protocol that always allows a team of $k > 0$ agents to explore all the sites of all subway graphs in which there are no black holes without knowledge of n_C nor of n_S. Consider now an execution of P in the subway graph $\mathbf{G}$ corresponding to the route of a single carrier c. Since the protocol is correct, within finite time T all sites will have been visited and all agents enter a terminal state. Consider now the subway graph $\mathbf{G'}$ corresponding to the routes $R(c_1)$ and $R(c_2)$ of two carriers c_1 and c_2, where $R(c_1) = R(c)$, $S(c_2) > 1$, and the only transfer site is the homebase s, Consider now in $\mathbf{G'}$ precisely the same execution of P as in $\mathbf{G}$, in which the adversary delays the arrival of c_2 at s until time $T' > T$. Notice that by time T none of the other agents will discover the existence of c_2. Since neither n_C nor n_S are known, at time T the agents will notice no difference with the previous setting and will thus enter a terminal state without visiting the sites on the route of c_2, contradicting the correctness of the protocol. $\square$

In other words, at least one of n_C and n_S must be known to the agents. for the problem to be solvable.

Transfer sites play a crucial role in the connectivity of the system, with and without black holes. Knowing that a site is a transfer site is necessary but not sufficient; in fact an agent disembarking there must also know the number of distinct carriers stopping there.

Lemma 3 *If the agents have no knowledge of the number of carriers stopping at a site, the* Exp *problem is deterministically* unsolvable. *This result holds regardless of the number $n_S - 1 > k > 0$ of agents, even if the carriers have distinct visible ids, and the agents know n_C, n_S and the length of the routes.*

Proof By contradiction, let P be a deterministic protocol that always allows a team of $n_S - 1 > k > 0$ agents to explore all the sites of all subway graphs, in which there are no black holes, when the agents can detect whether a site is a transfer site but without knowledge of the number of carriers stopping there. Consider now the subway graph $\mathbf{G}$ corresponding to the route of three carriers c_1, c_2 and c_3. The routes of c_1, c_2 coincide except for the direction; i.e., $R(c_1) = \langle r_0, r_1, \ldots, r_{l-1} \rangle$ and $R(c_2) = \langle r_0, r_{l-1}, \ldots, r_1 \rangle$ where $l = |R(c_1)| = |R(c_1)| = n_S - 1 > 1$. The routes of c_1, c_2 meet with the route of c_3 at a single transfer site x, where $l(c_3) = 2$. Notice that all the sites $r_0, r_1, \ldots, r_{l-1}$ are transfer sites between two carriers, except for x which is a transfer site between all three carriers. The agents can detect whether a site is a transfer site, but not the number of carriers stopping there. The adversary allows the agents to board both c_1 and c_2 and visit all the l sites on their route. Now, to complete the exploration, at least one agent must board c_3; to do so, an agent must wait at x until c_3 arrives; because of asynchrony, this might take a finite but unpredictable amount of time. Since the agents do not know which stop is x and since $k < n_S - 1 = l(c)$, they cannot wait at all the sites on the route of c_1 (or c_2) simultaneously. Hence the adversary can make the agents wait forever, contradicting the termination of every execution of P. □

In other words, the agents must be able to determine the number of carriers stopping at a site for the problem to be solvable.

The next set of limitations are directly related to the nature of black hole search and are a direct extension to the subway graph model of the limitations existing for standard network models.

Lemma 4 *For the* Bhs *problem to be deterministically* solvable

(i) the homebase and the terminal stops must be safe sites;
(ii) the transfer graph must stay connected once the black holes are removed;
(iii) $k > \gamma$*;*
(iv) γ *must be known to the agents.*

This result holds even if the carriers have distinct visible ids and the agents know n_C *and* n_S*, and the length of the routes.*

Proof Conditions (i) and (ii) trivially follow from the fact that, since solving the Bhs problem requires visiting all carrier stops, it is clearly necessary that the safe stops are reachable from the homebase. Condition (iii) expresses the obvious fact that to solve Bhs, it is necessary to have more agents than the number of stops at black holes. The necessity of condition (iv) follows from the requirement of knowledge of the stops leading to black holes when an agent enters a terminal state. Because of asynchrony, slow computation by an agent exploring a safe stop is indistinguishable from an agent having been eliminated by a black hole stop; hence if γ is not known to the agents, a surviving agent cannot decide whether to wait or enter a terminal state. □

We will refer to this set of conditions as *standard* for the Bhs problem, and assume that they hold. Note that a corollary to condition (ii) is that if there is more than one carrier, i.e. $n_C > 1$, then each carrier must have at least one safe transfer site in its domain.

Another important condition for solvability with an optimal team of agents refers again to knowledge; this time it is about knowledge by the agents of the length of each route.

Lemma 5 *Let the standard conditions for the* Bhs *problem hold. If the agents have no knowledge of the length of each route in the subway graph, the* Bhs *problem is deterministically* unsolvable *by a set of* $k = \gamma + 1$ *agents. This result holds even if the carriers have distinct visible ids and the agents know* n_C *and* n_S*.*

Proof By contradiction, let P be a deterministic protocol that always allows a team of $k = \gamma + 1$ agents to solve the BHS problem in all subway graphs under the standard conditions without knowledge by the agents the length of each route in the subway graph. Consider the subway graph **G** consisting of a single route defined on-line by an adversary as follows. The carrier route is initially a sequence of distinct sites starting from s: $R =< s, s_1, s_2 \ldots >$; the adversary will execute P until each agent descends at a stop (they must); let $x_1, ..., x_\gamma, x_{\gamma+1}$ be these stops, with x_1 the closest to s. The adversary sets $x_1, ..., x_{\gamma-1}$ to be black hole stops. Clearly only the agents a stopping at $x_\gamma = x$ and b stopping at $x_{\gamma+1} = y$ survive (for the moment) while all others are destroyed. When ready to move, agent a will make a decision on where to go based on algorithm P; since the length of the route is not known, for any algorithm, this decision can be only of the form: wait for the w_a-th carrier passage, board the carrier, descend at the m_a-th stop. Similarly for b. The adversary adds to the route the stops $(s, x)^{w_a}, z_1, z_2, \ldots, z_{(m_a-1)}, z, (s, y)^{w_b}, z_1, z_2, \ldots, z_{(m_b-1)}, z$ where the z_i are additional sites, and α^f denotes f consecutive occurrences of the subsequence α. In other words, the adversary finalizes the route as follows:

$$\begin{aligned} R =<& s, \ldots, x, \ldots, y, \\ & (s, x)^{w_a}, z_1, z_2, \ldots, z_{(m_a-1)}, z, \\ & (s, y)^{w_b}, z_1, z_2, \ldots, z_{(m_b-1)}, z > \end{aligned}$$

The adversary then makes a and b ready to move before the carrier reaches x for the second time; in this way both agents stop at z which is chosen by the adversary as a black hole stop. Hence all agents are destroyed, contradicting the correctness of P. □

Even if the length of each route is known to the agents, this condition alone is not sufficient for black hole search with an optimal number of agents. In fact, once disembarked, an agent must be able to board the same carrier at the same point in its route, otherwise the problem is unsolvable.

Lemma 6 *Let the standard conditions for the* BHS *problem hold. Unless each agent, once disembarked, is able to board the same carrier at the same point in its route, the* BHS *problem is deterministically* unsolvable *by a set of $k = \gamma + 1$ agents. This result holds even if the carriers have distinct visible ids and the agents know n_C, n_S and the length of each route.*

Proof By contradiction, let P be a deterministic protocol that always allows a team of $k = \gamma + 1$ agents to solve the BHS problem in all subway graphs under the standard conditions even if the agents have no means, once disembarked from a carrier, to board the same carrier at the same point in its route. Consider a subway graph **G** consisting of a single route $v_0 v_1 \ldots v_{(l(c)-1)}$ where there is only one black hole stop; the team thus consists of two agents, a and b both starting from the homebase $v_0 = s$. The location of the black hole stop and the nature of the other sites in the route is decided on-line by an adversary as follows. The adversary executes P until each agent descends at a stop (they must); let a descend at v_i and b at v_j, where without loss of generality $i < j$. If ready to move, agent a will make a decision on where to go based on algorithm P; since it is unable to board the carrier at the same point in its route, this decision will be of the form: wait for w_a-th passage, board the carrier, descend at the m_a-th stop. Similarly for b.

If $m_a \neq j$ then the adversary defines that, in the route, $v_{(j-m_a)} = x = v_i$, where the operations on the indices are modulo $l(c)$; that is $v_{(j-m_a)}$ is the same site where a is currently stopped. This means that by activating a again when the carrier reaches $v_{(j-m_a)}$, a will stop at v_j, the site where b is currently stopped. By choosing v_j as the black hole site, the adversary makes both agents disappear, contradicting the correctness of P. The same argument can be used if $m_a = j$ but $m_b \neq i$ (just exchange a and b, and i and j).

If both $m_a = j$ and $m_b = i$, let $p \in \{1, ..., n_S - 1\} \setminus \{i, j\}$ be such that $p - i \neq j$ and $p - j \neq i$ where the operations are modulo $l(c)$; such an index p always exists for $l(c) > 9$. Notice that by definition of p, $v_{(p-j)} \neq s \neq v_{(p-i)}$ and, since $m_b = i < j = m_a$, then $v_{(p-j)} \neq v_{(p-i)}$. The adversary then defines that, in the route, $v_{(p-j)} = x = v_i$ and $v_{(p-i)} = y = v_j$; in other words, $v_{(p-j)}$ and $v_{(p-i)}$ are the same sites where a and b are currently stopped, respectively; This means that, by activating a again when the carrier reaches $v_{(p-j)}$ and activating b when the carrier reaches $v_{(p-i)}$, both a and b will stop at v_p; notice that by definition v_p is neither s, nor x nor y. By choosing v_p as the black hole site, the adversary makes both agents disappear, contradicting the correctness of P. $\square$

Summarizing, in light of the above impossibility results, we make the following necessary assumptions. The standard assumptions for black hole search hold (necessary by Lemma 4). Each carrier is labelled with a distinct id and with the length of its route, and when at a site an agent can read the labels and the route length of any carrier stopping there (necessary by Lemmas 1 and 5). Each transfer site is labelled with the number of carriers stopping there (necessary by Lemma 3). Once disembarked, an agent is able to board the same carrier at the same point in its route (necessary by Lemma 6). Furthermore, we assume that, while the number of sites n_S might be unknown, the agents know the number of carriers n_C (one of the two values is necessary by Lemma 2).

4 Exploration algorithm

In this section we present the proposed algorithm *SubwayExplore*; as we will show later, our algorithm works correctly with any number of agents $k \geq \gamma + 1$ and is cost optimal. We first describe how the algorithm works in general, followed by a more detailed description.

4.1 Overview

Algorithm *SubwayExplore* works as follows. The agents start at unpredictable times from the same site s, called the *homebase.* and collectively search all the carriers,[4] by visiting all their stops, looking for black holes. Each agent performs a series of search tasks; each task, called *work*, involves visiting a previously unexplored stop on a carrier's route and returning, if possible, to report what was found there.

Every carrier is searched starting from a *work site*; the work sites are organized into a logical *work tree* that is rooted in the homebase, The first agent to access the homebase's whiteboard initializes the homebase as a work site (Section 4.2). It and the agents awaking after it then begin to work by visiting the stops of the carriers stopping at the homebase (Section 4.3). If an exploring agent finds a previously unexplored transfer site, the agent "competes" to add the transfer site to the work tree. If the agent succeeds, the transfer site becomes a work site for some or all of the other carriers stopping at it and the work site from which it was discovered becomes the work site's parent in the work tree (Section 4.4).

When the carrier that the agent is exploring has no more unexplored stops, the agent tries to find work by walking the work tree looking for another carrier with work to be done. The agent looks for work in the subtree rooted in its current work site and if there is no work available it moves to the work site's parent and tries again (Section 4.5). An agent terminates if it is at the homebase, there is no work, and there are n_C carriers in the work tree. Whenever an agent takes a carrier, it is possible for it to encounter terminal stops where the agent is temporarily kicked off the carrier. If this happens and the agent needs to continue on that route, the agent simply gets back on the same carrier.

4.2 Initialization

When an agent awakes for the first time on the homebase, it tries to initialize the homebase as a work site. Only the first agent accessing the whiteboard succeeds and executes the Initialize Work Site procedure. All other agents proceed directly to trying to find work.

The Initialize Work Site procedure is used to set up each work site in the work tree. The procedure takes as input the parent of the work site and the carriers to be worked on or serviced from the work site. For the homebase, the parent is *null* and the carriers to be accessed are all those stopping at s. The procedure initializes the work site's whiteboard with the information needed to find work, do work, and compete to add work. More precisely, when a work site ws is initialized, its parent is set to the work site from which it was discovered (*null* in the homebase's case) and its children are initially *null*.

[4] We use the terminology of searching a *carrier* to mean the searching of the route travelled by that carrier.

Algorithm 1 Initialization

```
    Agent $a$ awakes on starting site $s$.
    ▷ Initialize work information on whiteboard
 1: if whiteboard is blank then
 2:     INITIALIZE WORK SITE($null$, carriers stopping at $s$)
 3: end if
 4: FIND WORK

    Agent $a$ is initializing the whiteboard of work site $ws$ with information needed to find
    work, do work, and compete to add work.
 5: procedure INITIALIZE WORK SITE(parent $p$, carriers $C$)
 6:     for $c \in C$ do
            ▷ Work site information (Do work)
 7:         $U_c \leftarrow \{r_1, r_2, ...r_{l(c)-1}\}$                ▷ Set of $c$'s unexplored stops
 8:         $D_c \leftarrow \emptyset$                                  ▷ Set of $c$'s stops being explored
 9:         $E_c \leftarrow \{r_0\}$              ▷ Set of $c$'s explored stops where $r_0$ is the work site
            ▷ Work competition information (Compete to add work)
10:         $C_{subtree} \leftarrow C_{subtree} \cup \{c\}$    ▷ Set of carriers in subtree rooted in current node
            ▷ Work tree information (Find work)
11:         $C_{local} \leftarrow C_{local} \cup \{c\}$    ▷ Add $c$ to set of carriers worked on from this work site
12:         $C_{work} \leftarrow C_{work} \cup \{c\}$ ▷ Add $c$ to set of carriers in subtree with unexplored stops
13:     end for
        ▷ Work tree information (Find work)
14:     $parent \leftarrow p$
15:     $children \leftarrow \emptyset$
16: end procedure
```

The carriers it will service are added to $C_{subtree}$, the set of carriers in the work tree at and below this work site. The same carriers are also added to C_{work}, the set of carriers in the subtree with unexplored stops, and C_{local}, the set of carriers serviced by this work site. For each carrier c added to C_{local}, the agent setting up the whiteboard creates three sets U_c, D_c, and E_c. The set E_c of *explored stops* is initialized with the work site at $r_0 = ws$ (r_0 is always the work site servicing the carrier). The set U_c of *unexplored stops* is initialized with the rest of the stops on the carrier's route $\{r_1, r_2, \ldots, r_{l(c)-1}\}$, which is possible because each carrier is labelled with its length as well as its id. The set D_c of *stops being explored* (and therefore potentially dangerous sites) is initially empty. The pseudocode for initialization is in Algorithm 1.

4.3 Do work

We now discuss how the agents do their exploration of unexplored stops. To limit the number of agents eliminated by black holes, we use a technique similar to the cautious walk technique used by black hole search algorithms in static networks. Consider an agent a on the work site ws of a carrier c that still has unexplored stops, i.e. $U_c \neq \emptyset$. The agent does the following. It chooses an unexplored stop $r \in U_c$ for exploration, removes r from U_c, and adds it to the set D_c of stops being explored. It then takes c to r and disembarks.

If r is a black hole, the agent disappears.

Algorithm 2 Do work

```
    Agent $a$ is working on carrier $c$ from work site $ws$.
17: procedure DO WORK(carrier $c$)
18:     while $U_c \neq \emptyset$ do
19:         choose a stop $r$ from $U_c$
20:         $U_c \leftarrow U_c \setminus \{r\}$            ▷ Remove $r$ from the set of unexplored stops
21:         $D_c \leftarrow D_c \cup \{r\}$            ▷ Add $r$ to the set of stops being explored
22:         take $c$ to $r$ and disembark
            ▷ If not eliminated by black hole
23:         if whiteboard is blank then
24:             mark node as *visited*
25:             if $r$ is a transfer site then
26:                 $a.newC \leftarrow \emptyset$            ▷ Initialize agent's set of new carriers
27:                 $a.existingC \leftarrow \emptyset$            ▷ Initialize agent's set of existing carriers
28:                 for each carrier $c$ stopping at $r$ do
29:                     record $c$ on whiteboard
30:                     $a.newC \leftarrow a.newC \cup \{c\}$   ▷ Add carrier to agent's set of new carriers
31:                 end for
32:             end if
33:         end if
34:         take $c$ to $ws$ and disembark
35:         $D_c \leftarrow D_c \setminus \{r\}$            ▷ Remove $r$ from the set of stops being explored
36:         $E_c \leftarrow E_c \cup \{r\}$            ▷ Add $r$ to the set of explored stops
37:         if $r$ was a transfer site then
38:             COMPETE TO ADD WORK
39:         end if
40:     end while
41: end procedure
```

If r is not a black hole, the agent marks the site as *visited* on its whiteboard and returns to ws using the same carrier c and disembarks. The agent can make the trip back to ws because it knows the index of r and the length of c's route, $l(c)$, and can therefore calculate the number of stops between r and ws. At ws, it removes r from D_c and adds it to the set E_c of explored stops. At this point, the agent also adds the site id and any other information of interest.

If r is a transfer site and a is the first to visit it (its whiteboard is not marked *visited*), then, before returning to ws, the agent proceeds as follows. It records on r's whiteboard all the carriers that pass by r including their id and lengths of their route. It initializes two sets in its own memory: the set of *new carriers* initially containing all the carriers stopping at r; and the set of *existing carriers*, initially empty. Upon returning to ws and updating the local information, the agent then executes the next procedure that we discuss: competing to add work. The DO WORK procedure is in Algorithm 2.

Fig. 4 shows an example of the work site information at some instant used by agents to do work from the root in our example subway graph. At this point in the execution, carrier c_1 has 5 unexplored stops, 4 stops being explored that could potentially be black holes, and 4 stops that have been explored and are known not to be black holes. Carrier c_2 has no unexplored stops, 3 stops being explored, and one stop that has been explored, which happens to be the homebase in this case.

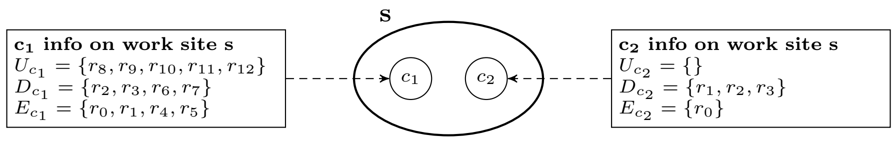
c_1 info on work site s
$U_{c_1} = \{r_8, r_9, r_{10}, r_{11}, r_{12}\}$
$D_{c_1} = \{r_2, r_3, r_6, r_7\}$
$E_{c_1} = \{r_0, r_1, r_4, r_5\}$
s
c_1
c_2
c_2 info on work site s
$U_{c_2} = \{\}$
$D_{c_2} = \{r_1, r_2, r_3\}$
$E_{c_2} = \{r_0\}$

Fig. 4 Work site information for doing work in our example subway graph at some instant.

4.4 Compete to add work

When an agent a discovers that a stop r is an unvisited transfer site, that stop is a potential new work site for the other carriers stopping at it. There is a problem, however: other agents may have independently discovered some or all of those carriers stopping at r. To ensure that to each carrier there is only one associated work site in the work tree, in our algorithm agent a must compete with all those other agents to add r as the new work site in the tree for these carriers. We use $C_{subtree}$ on the work sites in the work tree to decide the competition (if any).

Let us describe the actions that agent a performs; let a have just finished exploring r on carrier c_{ws} from work site ws and found that r is a new transfer site. The agent has a set of *new carriers* that initially contains all the carriers stopping at r, a set of *existing carriers* that is initially empty, and is currently on its work site ws. The agent walks up the work tree from ws to s checking the set of new carriers against $C_{subtree}$ on each work site. If a new carrier is not in $C_{subtree}$, the agent adds it. If a new carrier is in $C_{subtree}$, the agent moves it to the set of existing carriers. The agent continues until it reaches s or its set of new carriers is empty. The agent then walks down the work tree to ws. It adds each carrier in its set of new carriers to C_{work} on each work site on the way down to ws. For each carrier in its set of existing carriers, it removes the carrier from $C_{subtree}$ if it was the agent that added it. When it reaches ws, it removes the existing carriers and if there are no new carriers, it continues its work on c_{ws}. If there are new carriers, the agent adds r as a child of ws and goes to r. At r, the agent initializes it as a work site using the Initialize Work Site procedure with ws as its parent and the set of new carriers as its carriers. The agent then returns to ws and continues its work on c_{ws}. The pseudocode for the Compete to Add Work procedure is in Algorithm 3.

Fig. 5 shows the work competition information at some instant used by two agents to compete to add a new work site servicing carrier c_5 to the work tree for our example subway graph. Agents a_1 and a_2 have discovered carrier c_5 independently from carriers c_3 and c_4 respectively. The agents will walk up the tree towards s adding c_5 to $C_{subtree}$ on the way. Only one of the agents will be able to add c_5 to s's $C_{subtree}$ and that agent's site will become the new work site for carrier c_5

The Compete to Add Work procedure ensures the acyclic structure of the work-tree and that all new work is reported to the root:

Algorithm 3 Compete to Add Work

Agent a has found a new transfer site r while exploring carrier c_{ws} from work site ws and is competing to add it to the work tree with ws as r's parent.

```
42: procedure COMPETE TO ADD WORK
        ▷ Walk up tree
43:     repeat
44:         take the appropriate carrier to $parent$ and disembark
45:         for $c \in a.newC$ do
46:             if $c \in C_{subtree}$ then
47:                 $a.newC \leftarrow a.newC \setminus \{c\}$          ▷ Remove from agent's set of new carriers
48:                 $a.existingC \leftarrow a.existingC \cup \{c\}$ ▷ Add to agent's set of existing carriers
49:             else
50:                 $C_{subtree} \leftarrow C_{subtree} \cup \{c\}$
51:             end if
52:         end for
53:     until (on $s$) $\vee$ ($a.newC = \emptyset$)
        ▷ Walk down tree
54:     while not on $ws$ do
55:         for $c \in a.newC$ do
56:             $C_{work} \leftarrow C_{work} \cup \{c\}$               ▷ Add new carriers with work in subtree
57:         end for
58:         for $c \in a.existingC$ do
59:             if $a$ added $c$ to $C_{subtree}$ then
60:                 $C_{subtree} \leftarrow C_{subtree} \setminus \{c\}$            ▷ Remove carrier from subtree set
61:             end if
62:         end for
63:         take appropriate carrier to $child$ in direction of $ws$ and disembark
64:     end while
        ▷ Remove any existing carriers on $ws$
65:     for $c \in a.existingC$ do
66:         if $a$ added $c$ to $C_{subtree}$ then
67:             $C_{subtree} \leftarrow C_{subtree} \setminus \{c\}$                ▷ Remove carrier from subtree set
68:         end if
69:     end for
        ▷ Add any new carriers to the tree with $r$ as their work site
70:     if $a.newC \neq \emptyset$ then
71:         $children \leftarrow children \cup \{r\}$
72:         for $c \in a.newC$ do
73:             $C_{work} \leftarrow C_{work} \cup \{c\}$
74:         end for
75:         take carrier $c_{ws}$ to $r$ and disembark
76:         INITIALIZE WORK SITE($ws$, $a.newC$)
77:         take carrier $c_{ws}$ to $ws$ and disembark
78:     end if
79:     DO WORK($c_{ws}$)                                   ▷ Keep working on original carrier
80: end procedure
```

Lemma 7 *If a new carrier is discovered, within finite time it is added to the work tree and the new work is reported to the root.*

Proof By construction, all the steps taken by an agent after finding a new carrier are safe. They involve either moving on the carrier, which is immune to black holes, to a work site in the work tree, which must be safe. Hence each move will be completed in finite time. Let c be a newly discovered carrier. If

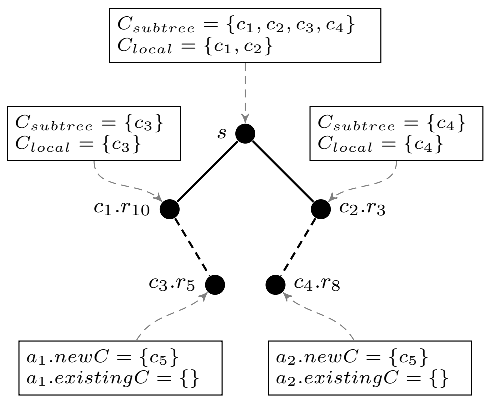

Fig. 5 Work competition information for competing to add work at some instant after Fig. 4.

it is discovered by only a single agent a, working from work site ws, then the agent a clearly reaches s, adding c to $C_{subtree}$ on ws up to s. Consider now the case when two or more agents $a_1, a_2, \ldots a_i$ from work sites $ws_1, ws_2, \ldots ws_i$ independently discover carrier c; let w be the closest common ancestor in the work tree of the work sites $ws_1, ws_2, \ldots ws_i$. Each such agent competes to add the new information to the work tree. However, mutual exclusion access to the whiteboards ensures that only one, say a_j, will be able to proceed from w to s, and ws_j will become the only work site for carrier c. □

4.5 Find work

Now that we have seen work being done and new work added to the tree, it is easy to discuss how an agent a finds work. When a work site is initialized, its parent is set to the work site from which it was discovered (*null* in the homebase's case) and its children are initially *null*. As mentioned before, each work site has a set C_{work} that contains the carriers in its subtree with unexplored stops.

An agent a looking for work at a work site checks the local set C_{work}. If it is not empty a chooses a carrier $c \in C_{work}$ and walks down the tree until it reaches the work site ws servicing c, or it finds that c is no longer in C_{work}. Assume that agent a reaches ws without finding c missing from C_{work}. Then a works on c until it is either eliminated by a black hole or U_c is empty. If the agent survives and is the first agent to discover that U_c is empty, it walks up the tree from ws to s removing c from C_{work} along the way. So, it is possible for an agent descending to do work on c to find out before it reaches ws that the work on c is finished. In that case, the agent starts over trying to find work.

If agent a looking for work finds that C_{work} at the current work site is empty, it moves to the work site's parent and tries again. If it reaches the

Algorithm 4 Find work

Agent a is looking for work in the work tree. The agent knows n_C, the number of carriers, which is needed for termination. Let ws be the current work site.

```
81: procedure Find Work
        ▷ Main loop
82:     while (not on $s$) $\vee$ ($C_{work} \neq \emptyset$) $\vee$ ($|C_{subtree}| < n_C$) do    ▷ Termination conditions
            ▷ Choose carrier to work on and go there
83:         if $C_{work} \neq \emptyset$ then
84:             choose carrier $c$ from $C_{work}$
85:             while ($c \notin C_{local}$) $\wedge$ ($c \in C_{work}$) do ▷ While not $c$'s work site and $c$ has work
86:                 take appropriate carrier to child in direction of $c$ and disembark
87:             end while
88:             if $c \in C_{local}$ then                    ▷ On the work site servicing $c$
89:                 Do Work($c$)
                    ▷ Carrier $c$ has no more work so remove it from the tree
90:                 if $c \in C_{work}$ then           ▷ The first agent to find no work left on $c$
91:                     while not on $s$ do
92:                         $C_{work} \leftarrow C_{work} \setminus \{c\}$     ▷ Remove $c$ from all $C_{work}$ on way to $s$
93:                         take appropriate carrier to parent and disembark
94:                     end while
95:                     $C_{work} \leftarrow C_{work} \setminus \{c\}$                ▷ Remove $c$ from $C_{work}$ on $s$
96:                 end if
97:             end if
            ▷ No work in subtree
98:         else
99:             take appropriate carrier to parent and disembark
100:        end if
101:    end while
102: end procedure
```

root without finding work but the termination condition is not met (there are fewer than n_C carriers in the work tree), the agent waits (loops) until new work arrives or the termination condition is finally met. The pseudocode for the Find Work procedure is in Algorithm 4.

Figure 6 shows the work tree information at some instant used to find work in our example subway graph. Note the agent from carrier c_4 won the competition to add carrier c_5 to the tree. Assume that an agent has just finished exploring carrier c_3 from stop r_{10} on carrier c_1's route and the agent finds no more unexplored stops. There are no other carriers in this part of the work tree, so the agent moves to the parent in the tree, s. It finds that three carriers, c_1, c_4, and c_5, still have work. It randomly chooses a carrier from C_{work} and safely traverses the work tree to that carrier's work site, where it starts to do work.

The Find Work procedure ensures the following property:

Lemma 8 *Within finite time, an agent looking for work either finds it or waits on the root.*

Proof By Lemma 7, all work gets reported to the root within finite time. By construction, if an agent does not find work on the current work site, it moves to the work site's parent. Both checking for work and moving to the parent

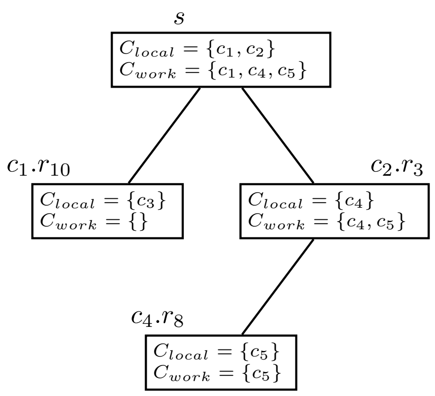

Fig. 6 Work tree information for finding work at some instant after Fig. 5.

take finite time. Therefore, within finite time, an agent looking for work either finds it or reaches the root and waits there until work arrives or the termination conditions are satisfies. □

4.6 Correctness

Now that we have seen how algorithm *SubwayExplore* works, we can show that it works correctly.

To do so, we need to establish some additional properties of the Algorithm:

Lemma 9 *Let $r_i \in R(c)$ be a black hole. At most one agent is eliminated by stopping at r_i when riding c.*

Proof By contradiction, assume that two agents have chosen to explore a black hole stop r_i on the same carrier c. By Lemma 7, each carrier is worked on from a single work site, so both agents must have chosen r_i from U_c on the work site. However, since U_c is kept on the whiteboard which is accessed in mutual exclusion, it is impossible for both agents to choose the same stop for exploration. Hence, the lemma follows. □

Lemma 10 *There is at least one agent alive at all times before termination.*

Proof By Lemma 9, we know that only one agent can die per black hole stop. Since we have $\gamma(\mathbf{G}) + 1$ agents working, where $\gamma(\mathbf{G})$ is the number of black hole stops in $\mathbf{G}$, there is always at least one more agent than can be eliminated by the black holes in the network. □

Lemma 11 *An agent that undertakes work completes it within finite time.*

Proof An agent works by visiting an unexplored stop on a carrier's route. The carrier arrives within finite time and takes finite time to deliver the agent to the stop. If the agent is eliminated then its work is completed and other agents are protected from being eliminated by the same stop on the same carrier. If the agent survives then within finite time it takes the carrier back to the work site to report on the work. Hence, the lemma follows. □

Lemma 12 *If there is work available, some agent eventually does it.*

Proof By contradiction, assume that there is work available but no agent does it. By construction, an agent is either trying to do work, competing to add work, or find work and by Lemmas 7, 8, and 11, the agent completes each activity in finite time. Furthermore, by Lemma 8, if there is work available an agent finds it. Therefore, the only reason that work would be available but no agent does it is if all the agents have been eliminated by a black hole. But, we know from Lemma 10 that there is always at least one agent alive, a contradiction. Hence, the lemma follows. □

Lemma 13 *All carriers are eventually added to the tree.*

Proof By Lemma 12, available work is eventually done. As a result, every carrier is eventually discovered. When an agent doing work finds a new carrier, by Lemma 11, it is added to the tree in finite time. Therefore, all carriers are added within finite time and the lemma follows. □

We can now state the correctness of our algorithm:

Theorem 1 *Protocol* SubwayExplore *correctly and in finite time solves the mapping problem with* $k \geq \gamma(\mathbf{G}) + 1$ *agents in any subway graph* $\mathbf{G}$.

5 Bounds and Optimality

We now analyze the costs of our algorithm, establish lower bounds on the complexity of the problem and prove that they are tight, showing the optimality of our protocol.

Theorem 2 *The algorithm solves black hole search in a connected dangerous asynchronous subway graph in* $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ *carrier moves in the worst case.*

Proof First let us examine the *do work* procedure. Each time an agent chooses a node for exploration, it waits up to $l(c) - 1$ carrier moves for the carrier to return to the work site. The agent then takes at most $l(c)$ carrier moves to reach the stop and return to the work site. Once the stop has been explored, the agent waits at most $l(c)$ carrier moves for the carrier to return. Therefore, the cost for exploring one stop is at most $3l(c) - 1$ carrier moves. Since $l(c) \leq l_R$, the cost for exploring all the stops on a route is therefore $O(l_R^2)$ carrier moves and the cost for searching all carriers is $O(n_C \cdot l_R^2)$.

Next let us examine the cost of *competing to add work.* Consider when an agent finds a transfer site where a previously unknown carrier c causing the resulting work to have to be added. Note that several agents, possibly all, might discover c independently and must compete to add the new work to the tree. Each of these agents tries to move towards the root, moving from its current work site to its parent, each move costing up to $l(c) - 1 \leq l_R - 1$ carrier moves. However, for each work site, at most one agent wanting to add the work of c is allowed to proceed to its parent work site. Hence the total number of moves up the tree incurred to add the work of c is at most one for each edge in the tree, i.e. $n_C - 1$, for a total of at most $O(n_C \cdot l_R)$ carrier moves.

Finally, let us consider the cost of *finding work*. When finding work, an agent a first looks for information of where work can be found, moving up the work tree if needed; once the information is found, agent a moves down the tree following the specified direction. Notice that it is possible that, when looking for information, agent a finds none, ending up at the root of the work tree; if there are still some routes to be explored (a condition that a can verify), eventually some other agent will add a new carrier to be explored (i.e., new work to be performed) to the list at the root and a will follow the indication. All this movement will elicit at most $2n_C - 1$ moves up and down the tree, each costing at most l_R carrier moves. An agent looks for an indication only when it has no work; i.e., when the stops of the local carrier it was working for have all been explored; this means that this process can occur at most n_C times. In other words, the total number of carrier moves caused by finding work will be at most $O(k \cdot n_C^2 \cdot l_R)$.

Summarizing, the total number of carrier moves required by the algorithm in the worst case is at most $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$. □

We now establish some *lower bounds* on the worst case complexity of any protocol using the minimal number of agents.

Theorem 3 *For any α, β, γ, where $\alpha, \beta > 2$ and $1 < \gamma < 2\alpha\beta$, there exists a simple subway graph $\mathbf{G}$ with α carriers with maximum route length β and faulty load γ in which every protocol $\mathcal{P}$ solving the* Exp *problem requires $\Omega(\alpha^2 \cdot \beta \cdot \gamma)$ carrier moves in the worst case.*

Proof Consider a subway graph $\mathbf{G}$ whose transfer graph is a line graph; all α routes are simple and have the same length β; there exists a unique transfer stop between neighbouring carriers in the line graph; no transfer site is a black hole, and the number of black holes is γ. The agents have all this information, but do not know the order of the carriers in the line (the agents do not know in what order the carriers will arrive).

Let $\mathcal{P}$ be a subway mapping protocol that always correctly solves the problem within finite time with the minimal number of agents $k = \gamma + 1$. Consider an adversary $\mathcal{A}$ playing against the protocol $\mathcal{P}$. The power of the adversary is the following: 1) it can choose which stops are transfers and which are black holes; 2) it can "block" a site being explored by an agent (i.e., delay the agent exploring the stop) for an arbitrary (but finite) amount of time; 3) it can choose the order of the carriers in the line graph. The order of

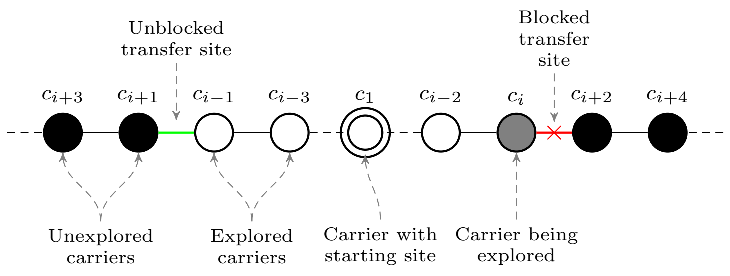

Fig. 7 Transfer graph in the lower bound proof

the carrier will be revealed to the agents incrementally, with each revelation consistent with all previous ones; at the end the entire order must be known to the surviving agents.

Let the agents start at the homebase on carrier c_1. Let $q = \lceil \frac{k-2}{\beta-2} \rceil$. Assume that the system is in the following configuration, which we shall call *Flip(i)*, for some $i \geq 1$: (1) carrier c_1 is connected to c_2, and carrier c_j $(j < i)$ is connected to c_{j+2}; (2) all stops of carriers $c_1, c_2, \ldots, c_i$ have been explored, except the transfer stop r_{i+1}, leading from carrier c_{i-1} to carrier c_{i+1}, and the stop r_{i+2} on carrier c_{i+1}, which are currently being explored and are blocked by the adversary; and (3) all agents, except the ones blocked at stops r_{i+1} and r_{i+2}, are on carrier c_i. See Fig. 7.

If the system is in configuration *Flip(i)*, with $i < \alpha - q$, the adversary operates as follows.

1. The adversary unblocks r_{i+1}, the transfer site leading to carrier c_{i+1}. At this point, all $k-1$ unblocked agents (including the $k-2$ currently on c_i) must move to r_{i+1} to explore c_{i+1} without waiting for the agent blocked at r_{i+2} to come back. To see that all must go within finite time, assume by contradiction that only $1 \leq k' \leq k-2$ agents go to explore c_{i+1} within finite time, while the others never go to r_{i+1}. In this case, the adversary first reveals the order of the carriers in the line graph by assigning carrier c_j to be connected to c_{j+1} for $\alpha > j > i$. Then the adversary chooses to be black holes: r_{i+2}, the first k' non-transfer stops visited by the k' agents, and other $k - k' - 2$ non-transfer stops arbitrarily chosen in those carriers. Notice this can be done because, since $q = \lceil \frac{k-2}{\beta-2} \rceil$, the number of non-transfer stops among these carriers is $q(l-2)+1 \geq k-1$. Thus all k' agents will enter a black hole. Since none of the other agents will ever go to c_{i+1}, the mapping will never be completed. Hence, within finite time all $k-1$ non blocked agents must go to r_{i+1}, with a total cost of $O(k \cdot i \cdot \beta)$ carrier moves.
2. The adversary blocks each stop of c_{i+1} being explored, until $k-1$ stops are being explored. At that point, it unblocks all those stops except one, r_{i+3}. Furthermore, it makes r_{i+2} the transfer stop leading to carrier c_{i+2}.

Notice that after these operations, the system is precisely in configuration *Flip(i+1)*. Further observe now that, from the initial configuration, when all agents are at the homebase and the protocol starts, the adversary can create configuration *Flip(0)* by simply blocking the first two stops of c_1 being explored, and making one of them the transfer to c_2.

In other words, within finite time, the adversary can create configuration *Flip(0)*; it can then transform configuration *Flip(i)* into *Flip(i+1)*, until configuration *Flip* $(\alpha - q - 1)$ is reached.

At this point the adversary reveals the entire graph as follows: it unblocks $r_{\alpha-q+1}$, the transfer site leading to carrier $c_{\alpha-q+1}$; it assigns carrier c_j to be connected to c_{j+1} for $\alpha > j > \alpha - q$; finally it chooses $k-1$ non-transfer stops of these carriers to be black holes; notice that they can be chosen because, since $q = \lceil \frac{k-2}{\beta-2} \rceil$, the number of non-transfer stops among these carriers is $q(l-2)+1 \geq k-1$.

The transformation from *Flip(i)* into *Flip(i+1)* costs the solution protocol $\mathcal{P}$ at least $\Omega(k \cdot i \cdot \beta)$ carrier moves, and this is done for $1 \leq i \leq \alpha - q$; since $\alpha(l-2) \geq (k-2)$ it follows that $\alpha - q = \alpha - \lceil \frac{k-2}{\beta-2} \rceil \geq \alpha - \frac{k-2}{\beta-2} \geq \frac{\alpha}{2}$; hence, $\sum_{1 \leq i \leq \alpha - q} i = O(\alpha^2)$. In other words, the adversary can force any solution protocol to use $\Omega(\alpha^2 \cdot \beta \cdot \gamma)$ carrier moves. □

Theorem 4 *For any α, β, γ, where $\alpha, \beta > 2$ and $1 < \gamma < \beta - 1$, there exists a simple subway graph* $\mathbf{G}$ *with α carriers with maximum route length β and faulty load γ in which every protocol $\mathcal{P}$ solving the* Exp *problem requires $\Omega(\alpha \cdot \beta^2)$ carrier moves in the worst case. This result holds even if the transfer graph* $H(\mathbf{G})$ *is known to the agents.*

Proof Consider a subway graph $\mathbf{G}$ whose transfer graph is a line graph, where c_i is connected to c_{i+1}, $1 \leq i < \alpha$; all α routes are simple and have the same length β; there exists a unique transfer stop between neighbouring carriers in the transfer graph; no transfer site is a black hole and the number of black holes is γ. The agents have all this information, including the order of the carriers in the line. Let $\mathcal{P}$ be a subway mapping protocol that always correctly solves the problem within finite time. Correctness of $\mathcal{P}$ implies that all stops are explored. If the stop is not a black hole, after exploring it, the agent must take a carrier; the adversary can delay this operation ensuring that $\beta - 1$ carrier moves are elapsed from the time the agent starts waiting to the time the carrier arrives. Thus the adversary can ensure that the execution of protocol $\mathcal{P}$ costs at least $m\ (\beta-1)$ carrier moves, where m is the number of safe stops in the subway graph. Since the total number of stops is $\alpha(\beta-2)+2$ and $\gamma \leq \beta-2$, it follows that $m\ (\beta-1) = (\alpha(\beta-2)+2-\gamma)(\beta-1) \geq (\alpha(\beta-2)+2-(\beta-2))(\beta-1) = ((\alpha-1)(\beta-2)+2)(\beta-1)$, and the theorem holds. □

The optimality of the protocol with respect to carrier moves now follows.

Theorem 5 *Protocol* SubwayExplore *is agent-optimal and move-optimal.*

Proof Agent optimality derives from the fact that the protocol correctly solves the problem in a subway graph $\mathbf{G}$ for any $k > \gamma(\mathbf{G})$, hence also when $k = \gamma(\mathbf{G}) + 1$. As for the number of carrier moves with the minimal number of agents, observe that, in case $\gamma(\mathbf{G}) < l_R - 1$, by Theorems 3 and 4, the lower bound for any agent-optimal solution protocol is $\Omega(\gamma(\mathbf{G}) \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$; hence, by Theorem 2, the protocol is move optimal in that case. Further observe that, when $\gamma(\mathbf{G}) > l_R - 1$ and $k = \gamma(\mathbf{G}) + 1$, the protocol uses $O(\gamma(\mathbf{G}) \cdot n_C^2 \cdot l_R)$ moves, which, by Theorem 3, are needed in the worst case. □

6 Concluding Remarks

In this article, we introduced the subway model, a class of dynamic networks with links that appear and disappear in cyclical patterns. We provided an algorithm that solves the black hole search problem in this new model with $k = \gamma(\mathbf{G}) + 1$ agents and $O(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves in the worst case. We then prove lower bounds on the problem that show that our solution is both agent and carrier move optimal. By Lemma 2, the agents need to know either the number of carriers, n_C, or the number of sites, n_S, in order to terminate. Our algorithm assumes knowledge of n_C but could be easily modified to use n_S.

We showed in Section 2 that there are a number of assumptions that are not only sufficient for solving BHS in the subway model, but also necessary. There are other parameters that can be considered and these lead to a number of open problems. For instance, it may be possible to change the method of inter-agent communications. We chose whiteboards as a simple, unrestricted form of communication between the agents that is often used in solutions to the BHS problem. It is an open question whether it is possible to use some potentially weaker form of communication, such as tokens, to solve BHS in the subway model. The token model has been extensively studied in static networks (e.g. [16, 17]). We speculate that the use of tokens may require that the agents be able to see the presence of a token on a site from the safety of a carrier; however, the change in visibility alone may not be sufficient to implement a variation on the cautious walk technique using tokens.

We considered the standard case where the agents all start on the same homebase. In the case of static networks, some work has also been done on the more difficult *scattered* agent case where the agents start on multiple homebases [8, 15, 16, 18]. This setting is clearly more difficult and little is known. An open question is how can the scattered agent case be solved in the subway model at what cost in agents and moves.

A final open problem is changing the timing of the network to be synchronous. A synchronous subway network would allow for face-to-face agent communications. If agents were allowed to jump from carrier to carrier at the same site, the subway model could be combined with the model of periodically-varying graphs presented in [19].

Acknowledgements This work has been supported in part by NSERC Discovery Grants, and by Dr. Flocchini's University Research Chair. The authors would like to thank the anonymous referees for their helpful comments.

References

1. Avin C, Koucký M, Lotker Z. How to Explore a Fast-Changing World (Cover Time of a Simple Random Walk on Evolving Graphs). In: ICALP 2008: Proceedings of the 35th International Colloquium Automata, Languages and Programming, pp 121–132 (2008)
2. Balamohan B, Flocchini P, Miri A, Santoro N. Time Optimal Algorithms for Black Hole Search in Rings. In: COCOA 2010: Proceedings of the 4th Annual International Conference on Combinatorial Optimization and Applications, pp 58–71 (2010).
3. Bui Xuan B, Ferreira A, Jarry A. Computing shortest, fastest, and foremost journeys in dynamic networks. International Journal of Foundations of Computer Science 14(2):267 (2003)
4. Burgess J, Gallagher B, Jensen D, Levine BN. MaxProp: Routing for vehicle-based disruption-tolerant networks. In: INFOCOM 2006: Proceedings of the 25th IEEE International Conference on Computer Communications, pp 1–11 (2006)
5. Casteigts A, Flocchini P, Mans B, and Santoro N. Deterministic Computations in Time-varying Graphs: Broadcasting under Unstructured Mobility. In TCS 2010: Proceedings of the 6th IFIP International Conference on Theoretical Computer Science, pp 111–124 (2010).
6. Casteigts A., Flocchini P., Mans B., and Santoro N. Measuring Temporal Lags in Delay-Tolerant Networks. In IPDPS 2011: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium, (2011).
7. Chaintreau A., Mtibaa A., Massoulie L., and Diot C., The Diameter of Opportunistic Mobile Networks. Communications Surveys & Tutorials, 10(3), 74-88 (2008).
8. Chalopin J, Das S, Santoro N. Rendezvous of mobile agents in unknown graphs with faulty links. In: DISC 2007: Proceedings of the 21st International Symposium on Distributed Computing, pp 108–122 (2007)
9. Cooper C, Klasing R, Radzik T. Searching for black-hole faults in a network using multiple agents. In: OPODIS 2006: Proceedings of the 10th International Conference on Principles of Distributed Systems, pp 320–332 (2006)
10. Cooper C, Klasing R, Radzik T. Locating and repairing faults in a network with mobile agents. In: SIROCCO 2008: Proceedings of the 15th International Colloquium on Structural Information and Communication Complexity, pp 20–32 (2008)
11. Czyzowicz J, Kowalski D, Markou E, Pelc A. Complexity of searching for a black hole. Fundamenta Informaticae 71(2,3):229–242 (2006)

12. Czyzowicz J, Kowalski D, Markou E, Pelc A. Searching for a black hole in synchronous tree networks. Combinatorics, Probability and Computing, 16(4):595–619 (2007)
13. Dobrev S, Flocchini P, Kralovic R, Ruzicka P, Prencipe G, Santoro N. Black hole search in common interconnection networks. Networks 47(2):61–71 (2006)
14. Dobrev S, Flocchini P, Prencipe G, Santoro N. Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. Distributed Computing 19(1):1–19 (2006)
15. Dobrev S, Flocchini P, Prencipe G, Santoro N. Mobile search for a black hole in an anonymous ring. Algorithmica 48(1):67–90 (2007)
16. Dobrev, S, Santoro, N, Shi, W. Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens. International Journal of Foundations of Computer Science 19(6): 1355–1372 (2008)
17. Flocchini P, Ilcinkas D, Santoro N. Ping pong in dangerous graphs: Optimal black hole search with pure tokens. Algorithmica, to appear. An earlier version appeared in DISC 2008.
18. Flocchini P, Kellett M, Mason P, Santoro N. Map construction and exploration by mobile agents scattered in a dangerous network. In: IPDPS 2009: Proceedings of the 23rd IEEE International Symposium on Parallel & Distributed Processing, pp 1–10 (2009)
19. Flocchini P, Mans B, Santoro N. Exploration of periodically varying graphs. In: ISAAC 2009: Proceedings of the 20th International Symposium on Algorithms and Computation, pp 534–543 (2009)
20. Glaus P. Locating a black hole without the knowledge of incoming links. In: ALGOSENSORS 2009: Proceedings of the 5th International Workshop on Algorithmic Aspects of Wireless Sensor Networks, pp 128–138 (2009)
21. Jain S., Fall K., and Patra R. Routing in a Delay Tolerant Network. In: SIGCOMM 2004: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication , 145–158, 2004.
22. Jones, E.P.C., Li L., Schmidtke J.K., and Ward P.A.S. Practical Routing in Delay-Tolerant Networks, IEEE Transactions on Mobile Computing, 6(8): 943–959, 2007.
23. Klasing R, Markou E, Radzik T, Sarracco F. Hardness and approximation results for black hole search in arbitrary networks. Theoretical Computer Science 384(2–3):201–221 (2007)
24. Klasing R, Markou E, Radzik T, Sarracco F. Approximation bounds for black hole search problems. Networks 52(4):216–226 (2008)
25. Kosowski A, Navarra A, Pinotti MC. Synchronization helps robots to detect black holes in directed graphs. In: OPODIS 2009: Proceedings of the 13th International Conference on the Principles of Distributed Systems, pp 86–98 (2009)
26. Liu C, Wu J. Scalable routing in cyclic mobile networks. IEEE Transactions on Parallel and Distributed Systems 20(9):1325–1338 (2009)

27. O'Dell R, Wattenhofer R. Information dissemination in highly dynamic graphs. In: DIALM-POMC '05: Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing, pp 104–110 (2005)
28. Ramanathan R., Hansen R., Basu P., Hain R., and Krishnan R. Prioritized Epidemic for Routing in Opportunistic Networks. In MobiOpp 2007: Proceedings of the 1st International Workshop on Mobile Opportunistic Networking, pp 62–66. (2007)
29. Ramanathan R., Basu P., and Krishnan R. Towards a Formalism for Routing in Challenged Networks. In CHANTS 2007: Proceedings of the 2nd ACM Workshop on Challenged Networks, pp 3–10. (2007)
30. Zhang X, Kurose J, Levine BN, Towsley D, Zhang H. Study of a bus-based disruption-tolerant network: Mobility modeling and impact on routing. In: MobiCom '07: Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking, pp 195–206 (2007)
31. Zhang Z. Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: Overview and challenges. IEEE Communications Surveys & Tutorials 8(1):24–37 (2006)