Finding Good Coffee in Paris

Paola Flocchini[1], Matthew Kellett[2], Peter C. Mason[2], and Nicola Santoro[3]

[1] School of Electrical Engineering and Computer Science, University of Ottawa
[2] Defence R&D Canada – Ottawa, Government of Canada
[3] School of Computer Science, Carleton University

Abstract. Finding a good cup of coffee in Paris is difficult even among its world-renowned cafés, at least according to author David Downie (2011). We propose a solution that would allow tourists to create a map of the Paris Métro system from scratch that shows the locations of the cafés with the good coffee, while addressing the problem of the tourists losing interest in the process once they have found good coffee. We map the problem to the black hole search problem in the subway model introduced by Flocchini et al. at Fun with Algorithms 2010. We provide a solution that allows the tourists to start anywhere and at any time, communicate using whiteboards on the subway trains, rely on much less information than is normally available to subway passengers, and work independently but collectively to map the subway network. Our solution is the first to deal with scattered agents searching for black holes in a dynamic network and is optimal both in terms of the team size and the number of carrier moves required to complete the map.

1 Introduction

Paris is filled with cafés, and Parisian culture is synonymous with café culture. The coffee served in these cafés, however, is not necessarily the best, at least according to American author and longtime Paris resident David Downie. In his book *Paris,* Paris*: Journey into the City of Light* [11], Downie devotes an entire chapter to lamenting the dearth of good coffee to be found among the otherwise fabulous cafés. Assuming that Downie is correct in his assessment, is it possible to map the locations of the good coffee in the city from scratch? Parisians can find their own good coffee over time. The problem is really one for tourists visiting the city. The tourists use maps of the city to guide themselves from landmark to museum, but most of their trips begin and end on the Paris Métro subway system. If they had a map of the Métro that showed which stops are closest to the best coffee, they may have the chance to go just a little out of their way for a guaranteed good *café crème* or *café express*.

Perhaps the tourists themselves can work together to build a map of the subway system that shows the stops with the best coffee. Tourists, however, don't come to Paris for the coffee, they come for the Eiffel tower, or the Louvre, or the Musée d'Orsay. We have a problem of motivation. The tourists are happy to help in the mapping until they find a café with good coffee, but once they have

E. Kranakis, D. Krizanc, and F. Luccio (Eds.): FUN 2012, LNCS 7288, pp. 154–165, 2012.

found their café they go back to their sightseeing. The problem is not dissimilar to finding faults in computer networks. The faults are often caused by computers that have crashed undetectably or network equipment that has malfunctioned or been misconfigured. Solutions to mapping these types of networks generally focus on the mobile agent model, where a mobile agent is a computational entity that can move autonomously in the network, not unlike our tourists moving around the subway system. When an agent encounters a fault—or a tourist encounters a good coffee—it ceases to participate in the solution without the other agents being aware. A computer or network node with such a fault is referred to as black hole, while a network link that behaves in the same way is referred to as a black link. A great deal of research has focussed on finding or mapping these faults in static computer networks under the names of black hole search (Bhs) [2–10, 12, 17, 19–22] or dangerous graph exploration (Dge) [13]. The Bhs problem is for a team of agents to locate the black holes in a network, have at least one agent survive, and have all surviving agents know the locations of the black holes. The Dge problem is similar but includes both black holes and black links.

Unfortunately, the Paris Métro has little in common with static computer networks. The tourists are not free to walk from station to station as they would in a more standard network model, they have to wait for the subway train to arrive. Fortunately, we have a model that captures the dynamics of a subway network, which was presented at FUN 2010 [14] and is fully described in [15]. Based on urban subway systems like the Paris Métro, the *subway model* allows us to model dynamic network environments where the edges of the network appear in cyclical routes. The edge appearance is abstracted as a subway train or carrier. The subway model covers a large class of networks that are characterized in [1] as having a "periodicity of edges" and the model is related to the carrier-only movement of agents in the periodically varying graphs described in [16, 18]. It is even possible to solve problems in subway networks where there is much less information than the average urban subway system; for instance, it assumes that no map exists and that all stations are more or less identical, two things that would make a real subway system extremely difficult to use.

In this paper, we map the problem of finding good coffee in Paris to the more general problem of black hole search in subway networks. Using pseudocode to specify our algorithm and the coffee problem to illustrate it, we propose a solution to the black hole search problem that works when the agents start scattered in the network. Although there has been work on scattered agent black hole search, it focusses on ring networks [8, 10] and static arbitrary networks [2, 13, 22]. Our solution is the first to look at black hole search by scattered agents in a dynamic network.

Like the existing solution for co-located agents [14, 15], our solution for scattered agents uses an optimal number of agents and its complexity matches the lower bound proven in [15] of $\Omega(k \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves, where k is the number of agents, n_C is the number of carriers, l_R is the length of the longest route, and carrier moves is a measure specific to the subway model that

combines how much an agent moves on a carrier and how many carrier moves it waits for a carrier to arrive.

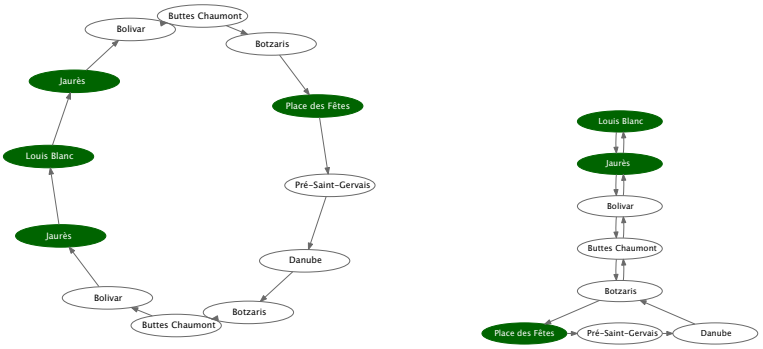
In the next section, we discuss how the Paris Métro can be mapped onto the subway model. We describe our algorithm from the perspective of a tourist in Section 3. We conclude with the correctness and complexity of our algorithm in Section 4. Due to space limitations, some of the proofs are omitted.

2 Modelling the Paris Métro

Our solution to the black hole search problem is applicable to any network that meets the basic limitations on exploration and black hole search in the subway model. In this section, we describe the model and how it applies to the Paris Métro. Although we describe the model here in terms of Paris' subway system, the model could easily be applied to other urban subway systems and, more importantly, to processes and communications in computer networks that create the same dynamics described by the model. For instance, it may be possible to piggyback agent movements on regular network communications, such as web traffic, routing table updates, or client/server database queries.

The subway model [14, 15] defines a subway network as a collection of stations or *sites* and the subway trains or *carriers* that run between them. Each carrier follows an ordered route of stops among the sites in its domain. Notice that there is no concept of a subway line here, which is a collection of trains that for the most part follow the same route. Each subway train is treated separately. Formally, we have a set C of $n_C = |C|$ carriers and a set S of $n_S = |S|$ sites. A carrier c's *domain* $S(c)$ is the set of all the sites it visits and its *route* $R(c)$ is the order in which the sites are visited. Let carrier c be a train from the small 7bis line on the Paris Métro. Its domain is $S(c) = \{$*Louis Blanc, Jaurès, Bolivar, Buttes Chaumont, Botzaris, Place des Fêtes, Pré-Saint-Gervais, Danube*$\}$ and the size of its domain is $n_S(c) = |S(c)| = 8$. The route it takes is $R(c) = \langle$*Louis Blanc, Jaurès, Bolivar, Buttes Chaumont, Botzaris, Place des Fêtes, Pré-Saint-Gervais, Danube, Botzaris, Buttes Chaumont, Bolivar, Jaurès*$\rangle$ and the length of its route is $l(c) = |R(c)| = 12$. After stopping at stop r_i the carrier moves to stop r_{i+1}, where all operations on the indices are modulo $l(c)$. A carrier moves asynchronously in that it takes a finite but unpredictable amount of time to reach the next stop. In other words, we make no assumptions about the trains running on time. We define l_R to be the length of the longest route in the network. We define a *transfer site* as any site that is in the domain of two or more carriers. On the 7bis line, Louis Blanc, Jaurès, and Place des Fêtes are all transfer sites to trains on other subway lines.

A carrier's route $R(c)$ defines an edge-labelled directed graph $\boldsymbol{G}(c)$ called a *carrier graph*. The nodes of the carrier graph are the stops $r_i \in S(c)$ of the route and an edge labelled $(c, i+1)$ exists between any two stops r_i and r_{i+1}, where all operations on indices and inside labels are modulo $l(c)$. The resulting graph is always a cycle, often virtual, and shows a view of the network from the perspective of someone riding on the carrier. Fig. 1(a) shows the carrier graph



(a) Carrier graph.

(b) Portion of Paris Métro subway graph.

Fig. 1. (a) Carrier graph and (b) subway graph portion for a carrier on the 7bis line of the Paris Métro. Node labels are the station names. The green filled nodes are transfer sites.

for a train on the 7bis line. The entire network can be represented as a edge-labelled directed multigraph $\boldsymbol{G}$ called a *subway graph*. The nodes of the subway graph are the sites $s_i \in S$ and an edge exists between any two sites for each edge connecting the two sites in the carrier graphs of all the carriers in the network. The subway graph is essentially a directed version of the subway map provided for users of major subway systems such as the Paris Métro. Fig. 1(b) shows the subset of the Paris Métro subway graph that corresponds to a carrier on the 7bis line and, except for the directed links, is similar to the individual line maps displayed prominently on most subway trains. Note that we omit edge labels where no ambiguity arises.

Associated with a subway graph $\boldsymbol{G}$ is a *transfer graph* $H(\boldsymbol{G})$. The nodes of the transfer graph are the carriers and an edge exists between any two carriers for *each* transfer site they share. Fig. 2 shows the transfer graph for the entire Paris Métro system. To reduce the complexity of the graph and our examples in the rest of the paper, we assume that only one carrier is associated with each subway line. The edge labels are omitted, but the edges associated with one transfer site—Châtelet in central Paris—are highlighted in thick blue.

The tourists running the algorithm are represented by the set A of $k = |A|$ agents (or computational entities). The agents start scattered in the network (like the tourist's hotels would be) and at varying times. An agent moving in the system can disembark from a carrier to a site or board a carrier from a site but cannot directly go from one carrier to another. An agent takes a finite but unpredictable amount of time to do computations, which practically means that it can take any amount of time to go looking for coffee. All agents follow the same protocol as laid out in the next section. They communicate using shared memory in the form of whiteboards located on the carriers, which they access

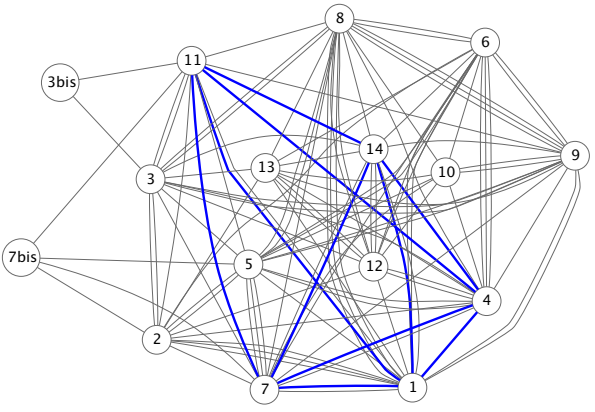


Fig. 2. Transfer graph associated with the Paris Métro assuming one subway train per line. Node labels are line numbers. Edges represent separate transfer sites with the edges associated with Châtelet station in central Paris highlighted in thick blue.

in fair mutual exclusion; in other words, on each carrier they must line up and only one tourist gets the pen at a time.

We now have to deal with the black holes. Formally, a *black hole* is a site that eliminates agents disembarking on it without leaving a discernible trace. Mapping that to our coffee problem, a tourist finding a café with good coffee near the station she is searching goes back to sightseeing. There are $n_B < n_S$ black hole sites. The number of stops a carrier c makes at black hole sites is called its *faulty load*, $\gamma(c)$, while all the stops at black hole sites of all the carriers is the network's faulty load $\gamma(\boldsymbol{G})$, or simply γ where no ambiguity arises.

The subway *black hole search* (Bhs) problem is for a team of agents to explore a subway graph and within finite time to determine which stops are at black hole sites. We say that the problem is solved if at least one agent survives, all surviving agents enter a terminal state, and all surviving agents know which *stops* are black holes.

The following additional assumptions are proven in [15] to be necessary for any solution to the Bhs problem in the subway model. Each carrier has a distinct id. The agents know the number of carriers n_C. Each transfer site is labelled with the number of carriers stopping there. Since the agent knows the number of carriers and can wait for all the carriers to pass by, we assume that each transfer site is also labelled with the ids of the carriers. The additional cost for this assumption is not significant. We assume that the standard assumptions for Bhs apply: the agent's starting site is safe (not a black hole), the transfer graph remains connected when the black holes are removed, there are more agents than faulty stops ($k > \gamma$), and the agents know the number of faulty stops γ.

Because the whiteboards themselves are moving on the carriers, we make the following additional assumptions. We assume that there is a function $r_{curr}(c)$ that returns a distinct id for the current stop when the agent is on carrier c.

Since the agent can record every stop during the carrier's traversal, we assume that there is a second function $R(c)$ that returns the set of all stops on the route, which can also be used to determine the route's length.

The model is slightly different from the reality of the Paris Métro. For instance, we make no assumptions about the availability of a map of the system or the agents being able to distinguish between stations before disembarking, two things that most subway designers go out of their way to provide. Nevertheless, the subway model describes a large class of networks that include—but are not limited to—urban subway systems, and the lack of these assumptions makes the solution we describe in the next section more widely applicable.

3 Finding the Good Coffee

In this section, we describe our solution to the BHS problem in the subway model from the perspective of a tourist following our algorithm to find good coffee. We start with an overview and then describe briefly each type of task performed by the tourist. We provide a formal and complete specification of the algorithm in pseudocode in Algorithms 1-4.

To get an intuition of how the algorithm works, we start with an overview. The tourists work independently but collectively to find when a carrier stops at a site near a café with good coffee, which are the black holes. The tourists each carry a map, initially blank, that records what they know about the subway network. As the tourists move around the network, they synchronize their maps with the maps kept on whiteboards on the subway trains, which are the carriers. Any new carrier that they learn about because of this merging of maps is included in the next iteration of their search. At the start of each iteration, each tourist constructs a spanning tree on the transfer graph of the subset of the subway graph in her map. The tourist then traverses that tree from carrier to carrier looking for work. Work in the algorithm is the visiting of a previously unexplored stop to look for good coffee. The work is coordinated using the whiteboard on the carrier to ensure that at most one tourist searches each stop of a carrier's route. If the tourist finds no carriers with unexplored stops and her map does not yet contain all the carriers, she waits on her home carrier until she is notified of new carriers to be searched. A tourist terminates the algorithm when her map contains all the carriers and she found no new work on her last search.

The algorithm starts with the tourist visiting the station closest to her hotel, her starting site. She takes the first carrier that arrives at the site and this becomes her home carrier. If she is the first tourist running the algorithm to ride the carrier, she sets up the whiteboard. Once that is done, the tourist starts looking for work. Notice that we ignore the fact that most subway stations have separate platforms for trains on separate lines and many have trains with multiple cars. Algorithm 1 shows the pseudocode for starting Algorithm *Find Good Coffee*.

Each time our tourist finds a carrier that has not been previously visited, she sets up the whiteboard with a map and the information needed to search that

Algorithm 1. *Find Good Coffee*

Agent a awakes on a safe site s.

1: $a.M \leftarrow$ info about s ▷ Agent's map M of network
2: board first carrier c_s arriving at s ▷ Board home carrier c_s
3: **if** whiteboard is blank **then**
4: INITIALIZE WORK INFO (c_s)
5: **end if**
6: FIND WORK

carrier's stops. There are three lists of stops on the whiteboard: unexplored stops, stops being explored, and explored stops. The list of unexplored stops initially includes all the carrier's stops except the one from which the tourist boarded the carrier, which is added to the list of explored stops. The list of stops being explored is initially empty. The pseudocode for initializing a carrier's whiteboard is in Algorithm 2.

Our tourist searches for work by calculating a spanning tree on the transfer graph of her map. She traverses this tree looking for carrier stops remaining in the list of unexplored stops on its whiteboard. If she finds such a carrier, she works on the carrier using the procedure outlined below. During the traversal, she synchronizes or merges her map with the maps of all the carriers she passes. Since all tourists do this synchronization, at the end of her traversal she checks to see if there are carriers in her map that are not in the tree she has been traversing. If so, she does another traversal. If not, after one more traversal to check for new work, she waits on her home carrier. She periodically checks the map on her home carrier until new carriers appear and then she continues to look for work. She terminates the algorithm when her map contains all the carriers and there is no work left. The termination is implicit with the algorithm being complete when the last tourist terminates. The termination can be made explicit by having every tourist wait until its home carrier's map shows that the number of stops being explored is equal to the number of faults, γ, which is known. The pseudocode for finding work is in Algorithm 3.

Algorithm 2. Initialize work information

Agent a is initializing the whiteboard of carrier c with information needed to do work on the carrier.
Functions used:
▷ $R(c)$ returns the set of stops for current carrier c
▷ $r_{curr}(c)$ returns the current stop on current carrier c

7: **procedure** INITIALIZE WORK INFO (carrier c)
8: $U \leftarrow R(c) \setminus r_{curr}(c)$ ▷ Set of c's unexplored stops
9: $D \leftarrow \emptyset$ ▷ Set of c's stops being explored
10: $E \leftarrow r_{curr}(c)$ ▷ Set of c's explored stops
11: $M \leftarrow a.M$ ▷ Carrier's map of network
12: **end procedure**

Algorithm 3. Find work

Agent a is looking for work starting from its home carrier c_s. The agent knows n_C, the number of carriers, which is needed for termination. The operator $\oplus$ denotes the merger of two maps.
Functions used:
▷ $H(M)$ returns the transfer graph of a map M
▷ $T(H)$ returns a spanning tree of a transfer graph H
▷ $C(M)$ or $C(T)$ returns the set of carrier ids in a map M or tree T

13: **procedure** FIND WORK
▷ Main loop
14: **repeat**
15: $a.worked \leftarrow false$ ▷ Work flag
16: $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$; ▷ Synchronize maps M and $a.M$
17: $a.T \leftarrow T(H(a.M))$ ▷ Compute spanning tree of transfer graph of agent's map
18: **while** depth-first traversal of $a.T$ **do** ▷ Preorder traversal
▷ On each carrier c in the traversal
19: $a.M \leftarrow a.M \oplus$ info about carrier c
20: **if** $U \neq \emptyset$ **then**
21: $a.worked \leftarrow true$
22: DO WORK (c)
23: **end if**
24: $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$; ▷ Synchronize maps M and $a.M$
25: take transfer link to next carrier in traversal of $a.T$
26: **end while**
▷ Back on home carrier c_s
27: **if** $\neg a.worked \wedge C(a.M) = C(a.T) \wedge |C(a.M)| < n_C$ **then** ▷ No new work
28: wait until $|C(a.M)| < |C(M)|$ ▷ Awake periodically to check c_s's map M
29: **end if**
30: **until** $\neg a.worked \wedge C(a.M) = C(a.T) \wedge |C(a.M)| = n_C$
31: **end procedure**

When our tourist finds a carrier with unexplored stops, she works on it until there are no more unexplored stops or until she finds a stop with good coffee. To explore a stop, she moves it from the list of unexplored stops to the list of stops being explored. She then disembarks and exits the station looking for a café with good coffee. If she finds one, she enjoys the coffee and has no incentive to spend the rest of her vacation traipsing around the Paris Métro. If she does not find one, she returns to the station, gets back on the carrier where she is working, and moves the stop from list of stops being explored to the list of explored stops.

The most complicated part of the algorithm is what happens when the stop the tourist explores is a transfer site. We want to ensure that we add any new carriers that might stop at the site. The tourist boards any carrier that is not in her map and either sets up its whiteboard or merges her map with the one on the carrier's whiteboard. After one circuit of the carrier's route, she disembarks back onto the transfer site. She does this for every carrier not in her map. She then

Algorithm 4. Do work

Agent a is working on carrier c.

32: **procedure** DO WORK (carrier c)
33: **while** $U \neq \emptyset$ **do**
34: $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$; ▷ Synchronize maps M and $a.M$
35: choose a stop $r \in U$
36: $U \leftarrow U \setminus \{r\}$ ▷ Remove r from the set of unexplored stops
37: $D \leftarrow D \cup \{r\}$ ▷ Add r to the set of stops being explored
38: disembark when $r_{curr}(c) = r$
▷ If not eliminated by black hole
39: $a.M \leftarrow a.M \oplus$ info about r
▷ If r is transfer site check the carriers passing by for information
40: **for** each carrier $c_i \notin a.M$ stopping at r **do** ▷ If r is a transfer site
41: board c_i
42: **if** whiteboard is blank **then**
43: INITIALIZE WORK INFO (c_i)
44: **else**
45: $U \leftarrow U \setminus \{r\}$ ▷ Remove r from set of unexplored stops
46: $E \leftarrow E \cup \{r\}$ ▷ Add r to set of explored stops.
47: **end if**
48: $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$; ▷ Synchronize maps M and $a.M$
49: disembark when $r_{curr}(c_i) = r$
50: **end for**
51: board c
52: $D \leftarrow D \setminus \{r\}$ ▷ Remove r from the set of stops being explored
53: $E \leftarrow E \cup \{r\}$ ▷ Add r to the set of explored stops
▷ If previously unknown carriers found distribute new map
54: **if** $|C(a.M)| > |C(M)|$ **then**
55: $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$; ▷ Synchronize maps M and $a.M$
56: $a.T_{notify} = T(H(a.M))$ ▷ Compute spanning tree for notification
57: **while** depth-first traversal of $a.T_{notify}$ **do** ▷ Preorder traversal
▷ On each carrier in the traversal
58: $a.M \leftarrow a.M \oplus$ info about carrier
59: $M \leftarrow M \oplus a.M$; $a.M \leftarrow M$; ▷ Synchronize maps M and $a.M$
60: take transfer link to next carrier in traversal of $a.T_{notify}$
61: **end while**
▷ Back on carrier c
62: **end if**
63: **end while**
64: **end procedure**

boards the carrier that she is working from and compares her map to the one on the carrier. If her map has new carriers then there may be some agent waiting to find out about them. After synchronizing her map with the carrier's, she calculates a new spanning tree—separate from the one she had calculated to find work—and traverses the new tree to synchronize her map with all the carriers' maps. When she gets back to the carrier she is working from, she continues her work and traversal as normal. If there were new carriers found, she will include

them in the spanning tree she calculates for her next traversal. The pseudocode for doing work is in Algorithm 4.

4 Proving It Works on Every Subway-Like Network

We assert a series of lemmas that leads to the theorem about the correctness of our algorithm. First, we show that the number of agents that can be eliminated by a black hole stop is bounded.

Lemma 1. *Let $r \in R(c)$ be a black hole. At most one agent is eliminated by stopping at r when riding carrier $c \in C$.*

Since we have more agents than faults, $k > \gamma$, we get the following.

Lemma 2. *There is at least one agent alive at all times.*

An agent does work by visiting a previously unexplored stop.

Lemma 3. *Within finite time, an agent that undertakes work completes it.*

Next, we look at an agent looking for work.

Lemma 4. *Within finite time, an agent looking for work either finds it, waits on its home carrier, or terminates the algorithm.*

While working and terminating are useful, waiting would seem to be something that agent could do indefinitely. We prove that that does not happen.

Lemma 5. *Within finite time, a waiting agent learns of a new carrier.*

Proof. By Lemma 4, we know that a waiting agent found neither work nor new carriers on its last traversal. This situation can only arise if all the links leading out of the subgraph described by the agent's map are either explored or being explored. If the waiting agent's map contained all the carriers, it would have terminated, so there must be carriers that do not appear in the agent's map.

There must be at least one stop being explored that is a safe transfer site that connects to a carrier that is not in the map. By contradiction, assume that no such transfer site exists. All the stops being explored must be black holes, safe non-transfer sites, or transfer sites that do not connect to new carriers. However, that would mean that the transfer graph of the network is disconnected, a contradiction.

By Lemma 3, we know that a working agent exploring a transfer site that connects to new carriers finishes its work within finite time. Its work includes synchronizing its map with the map of every carrier in the subgraph its map describes. Since the working agent's map has been synchronized with the map of the carrier from which it was working, and that carrier's map includes the waiting agent's home carrier c_s because it was synchronized with the map of the waiting agent on its last traversal, the working agent within finite time must update c_s with a map that contains new carriers. Hence, the lemma follows. □

In fact, if work is available, it gets done.

Lemma 6. *If there is work available, some agent eventually does it.*

As a consequence, we get the following corollary:

Corollary 1. *All carriers are eventually added to each carrier's map.*

We can now state the correctness of our algorithm based on the preceding lemmas. Note that our algorithm uses an optimal number of agents.

Theorem 1. *Algorithm* Find Good Coffee *correctly and in finite time solves the mapping problem with $k \geq \gamma(\boldsymbol{G}) + 1$ agents in any subway graph $\boldsymbol{G}$.*

We can now assert an upper bound on the complexity of our solution.

Theorem 2. *Algorithm* Find Good Coffee *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.*

The lower bound on any solution to the BHS problem in the subway model is proven in [15] to be $\Omega(\gamma(\boldsymbol{G}) \cdot n_C^2 \cdot l_R + n_C \cdot l_R^2)$ carrier moves. The optimality of the protocol with respect to the number of agents and the number of carrier moves now follows.

Theorem 3. *Algorithm* Find Good Coffee *is agent-optimal and move-optimal.*

References

[1] Casteigts, A., Flocchini, P., Quattrociocchi, W., Santoro, N.: Time-Varying Graphs and Dynamic Networks. In: Frey, H., Li, X., Ruehrup, S. (eds.) ADHOC-NOW 2011. LNCS, vol. 6811, pp. 346–359. Springer, Heidelberg (2011)
[2] Chalopin, J., Das, S., Santoro, N.: Rendezvous of Mobile Agents in Unknown Graphs with Faulty Links. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 108–122. Springer, Heidelberg (2007)
[3] Cooper, C., Klasing, R., Radzik, T.: Searching for Black-Hole Faults in a Network Using Multiple Agents. In: Shvartsman, M.M.A.A. (ed.) OPODIS 2006. LNCS, vol. 4305, pp. 320–332. Springer, Heidelberg (2006)
[4] Cooper, C., Klasing, R., Radzik, T.: Locating and Repairing Faults in a Network with Mobile Agents. In: Shvartsman, A.A., Felber, P. (eds.) SIROCCO 2008. LNCS, vol. 5058, pp. 20–32. Springer, Heidelberg (2008)
[5] Czyzowicz, J., Kowalski, D., Markou, E., Pelc, A.: Complexity of searching for a black hole. Fund. Inform. 71(2,3), 229–242 (2006)
[6] Czyzowicz, J., Kowalski, D., Markou, E., Pelc, A.: Searching for a black hole in synchronous tree networks. Combin. Probab. Comput. 16(4), 595–619 (2007)
[7] Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Searching for a black hole in arbitrary networks: Optimal mobile agents protocols. Distrib. Comput. 19(1), 1–19 (2006)
[8] Dobrev, S., Flocchini, P., Prencipe, G., Santoro, N.: Mobile search for a black hole in an anonymous ring. Algorithmica 48(1), 67–90 (2007)

[9] Dobrev, S., Santoro, N., Shi, W.: Locating a Black Hole in an Un-oriented Ring Using Tokens: The Case of Scattered Agents. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 608–617. Springer, Heidelberg (2007)
[10] Dobrev, S., Santoro, N., Shi, W.: Using scattered mobile agents to locate a black hole in an un-oriented ring with tokens. Internat. J. Found. Comput. Sci. 19(6), 1355–1372 (2008)
[11] Downie, D.: Paris, Paris: Journey into the City of Light. Broadway (2011), ISBN 978-0307886088
[12] Flocchini, P., Ilcinkas, D., Santoro, N.: Ping pong in dangerous graphs: Optimal black hole search with pebbles. Algorithmica 1–28 (2011)
[13] Flocchini, P., Kellett, M., Mason, P.C., Santoro, N.: Map construction and exploration by mobile agents scattered in a dangerous network. In: Proceedings of IPDPS 2009, pp. 1–10 (2009)
[14] Flocchini, P., Kellett, M., Mason, P.C., Santoro, N.: Mapping an Unfriendly Subway System. In: Boldi, P. (ed.) FUN 2010. LNCS, vol. 6099, pp. 190–201. Springer, Heidelberg (2010)
[15] Flocchini, P., Kellett, M., Mason, P.C., Santoro, N.: Searching for black holes in subways. Theory of Computing Systems 50(1), 158–184 (2012)
[16] Flocchini, P., Mans, B., Santoro, N.: Exploration of Periodically Varying Graphs. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 534–543. Springer, Heidelberg (2009)
[17] Glaus, P.: Locating a Black Hole without the Knowledge of Incoming Link. In: Dolev, S. (ed.) ALGOSENSORS 2009. LNCS, vol. 5804, pp. 128–138. Springer, Heidelberg (2009)
[18] Ilcinkas, D., Wade, A.M.: On the Power of Waiting When Exploring Public Transportation Systems. In: Fernàndez Anta, A., Lipari, G., Roy, M. (eds.) OPODIS 2011. LNCS, vol. 7109, pp. 451–464. Springer, Heidelberg (2011)
[19] Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Hardness and approximation results for black hole search in arbitrary networks. Theor. Comput. Sci. 384(2-3), 201–221 (2007)
[20] Klasing, R., Markou, E., Radzik, T., Sarracco, F.: Approximation bounds for black hole search problems. Networks 52(4), 216–226 (2008)
[21] Kosowski, A., Navarra, A., Pinotti, M.C.: Synchronization Helps Robots to Detect Black Holes in Directed Graphs. In: Abdelzaher, T., Raynal, M., Santoro, N. (eds.) OPODIS 2009. LNCS, vol. 5923, pp. 86–98. Springer, Heidelberg (2009)
[22] Shi, W.: Black Hole Search with Tokens in Interconnected Networks. In: Guerraoui, R., Petit, F. (eds.) SSS 2009. LNCS, vol. 5873, pp. 670–682. Springer, Heidelberg (2009)