Algorithmica (2003) 37: 165–186
DOI: 10.1007/s00453-003-1030-9

Algorithmica

A Distributed Ant Algorithm for Efficiently Patrolling a Network

Vladimir Yanovski,[1] Israel A. Wagner,[1,2] and Alfred M. Bruckstein[1]

Abstract. We consider the problem of *patrolling*—i.e. ongoing exploration of a network by a decentralized group of simple memoryless robotic agents. The model for the network is an undirected graph, and our goal, beyond complete exploration, is to achieve close to uniform frequency of traversal of the graph's edges. A simple multi-agent exploration algorithm is presented and analyzed. It is shown that a single agent following this procedure enters, after a transient period, a periodic motion which is an *extended Eulerian cycle*, during which all edges are traversed an identical number of times. We further prove that if the network is Eulerian, a single agent goes into an Eulerian cycle within $2|E|D$ steps, $|E|$ being the number of edges in the graph and D being its diameter. For a team of k agents, we show that after at most $2(1 + 1/k)|E|D$ steps the numbers of edge visits in the network are balanced up to a factor of two. In addition, various aspects of the algorithm are demonstrated by simulations.

Key Words. Ant algorithms, Euler cycle, Blanket time, Graph algorithms.

1. Introduction

1.1. *The Problem.* Suppose several simple memoryless robotic guards have to patrol a large art gallery having many halls connected by corridors. In order to complete the patrolling of a corridor, it should be traversed in both directions. The gallery can be represented by an undirected graph $G(V, E)$, with the edge set E corresponding to corridors and the vertex set V to halls. The guards want to patrol the gallery in a way that guarantees that each corridor will be traversed with approximately the same frequency.

In case of a single agent, an optimal solution for this problem is an Eulerian cycle in a graph G' obtained from the undirected graph G by substituting each undirected edge with two directed ones. An *Eulerian cycle* in a graph $G(V, E)$ is a cycle that visits each edge in E exactly once, see, e.g. [4] for some basic theory of such cycles. An Euler cycle is necessarily optimal since its length is $|E|$ which is the minimum possible length of a complete patrol. For the multi-agent case one wonders whether a cooperative effort could lead agents into disjoint cycles of equal lengths whose union covers all the graph's edges. Unfortunately, such a "cooperative Euler cycle" in which all edges are traversed the same number of times is not always possible. For example, it is not always possible to partition an Eulerian graph into two disjoint Euler cycles of the same size. However, with our solution, the frequency of visiting the edges evolves to a uniform one via an "extended Euler cycle" that will be explained later.

[1] Computer Science Department, Technion IIT, Haifa 32000, Israel. {volodyan|freddy}@cs.technion.ac.il.
[2] IBM Haifa Research Laboratory, MATAM, Haifa 31905, Israel. wagner@il.ibm.com.

Received September 29, 2002; revised January 28, 2003. Communicated by N. Megiddo.
Online publication July 25, 2003.

We first consider several possibilities of approaching this problem via known algorithms. A random walk solution for the guards eventually achieves uniform distributions of corridor visits (see [19] for a stochastic analysis of this process), but it exhibits high variability and its performance can only be estimated with probability. Also, as experiments show, for some corridors there will be very long periods when no guards enter there. Another potential solution is the depth-first search (DFS) algorithm, see [12] and [11]. In case of a single guard, the agent can store, in every hall, the information necessary for backtracking from the hall and information about the corridors that emanate from the hall that were already patrolled. It is possible to forbid a DFS agent to move to an already visited vertex, effectively separating the graph among the agents into several sub-graphs, but these sub-graphs will generally be of vastly different sizes causing some edges to be traversed less frequently than others. Another disadvantage of DFS for guarding or patrolling is that only $|V| - 1$ out of $|E|$ graph edges will be DFS tree edges, the rest will be traversed backward immediately after being traversed forward, causing the agents to visit a corridor twice in a row almost always in the case of a dense graph.

Our goal is to find a way to traverse the graph continuously (that is, again and again), as opposed to the one-time DFS search. Indeed, one can use DFS repeatedly by restarting it immediately after each completion, but this approach is clearly vulnerable to malicious tampering of backtracking pointers that may cause the agent either to run forever or stop before completion. Furthermore, this solution cannot be readily extended to the multi-agent case.

1.2. *Our Solution.* We shall show that the problem of patrolling can effectively be solved with the so-called Edge Ant Walk (EAW) algorithm. The agents working according to this algorithm always choose for the next move to leave the vertex through the edge that was traversed the longest time ago. Referring to the description of the problem as the problem of cleaning a gallery (instead of patrolling) by multiple cleaners, this algorithm has a very intuitive practical motivation—since dirt accumulates with time, the agent should always move to clean the dirtiest corridor. It is shown that the algorithm, after a "stabilization period," enters a cycle during which the agents complete an "extended Eulerian" cycle, a cycle for which each edge is traversed the same number of times. We also prove that adding more agents increases the frequency of traversals of any edge while never increasing the time it takes to cover a graph for the first time.

Our algorithm is evolutionary: it defines a process that eventually emerges into a solution, rather than attempt to solve the problem directly and immediately. This evolutionary approach implies another useful property of EAW: its robustness. Since every state of the algorithm is valid, the algorithm will not fail in case of a link or vertex failure, as long as the connectivity of the network is preserved—after some transient period the algorithm adapts to the new network. Furthermore, the algorithm is modular in the sense that agents can be added on the fly, possibly improving the overall performance of the team.

The contribution of this paper is in showing that:

1. A single EAW agent enters an Euler cycle within time $|E|D$, where E denotes the edge-set of the graph and D is its diameter.

2. Many agents are not worse, and are sometime better, than a single agent in terms of both the cover time and achieving an Euler cycle.
3. k agents achieve an almost uniform coverage of the graph ("blanket time") within time inversely proportional to k.

Although our analysis is described for directed Eulerian graphs, it should be noted that any undirected graph can be made directed Eulerian if each undirected edge is considered as a pair of directed edges in two directions.

1.3. *Related Work.* The EAW algorithm was first described in [13] and [14], where it was proven that the algorithm covers the edges of a graph within time $O(|V||E|)$. In [3] the authors prove that in case of a single agent running on an Eulerian graph $G(V, E)$ the algorithm enters an Eulerian cycle in time $O(|V||E|)$. Recall that an *Euler Cycle* is a cycle that traverses each edge exactly once, and an *Eulerian graph* is a graph with such a cycle, characterized as a connected graph such that the outdegree of every vertex is equal to its indegree (see [4] for more properties of Eulerian Graphs). In this paper we prove that an EAW agent enters an Euler cycle after no more than $|E|D$ steps.

In [15] the authors also present an algorithm called Vertex Ant Walk (VAW) and prove its ability to cover the vertices of a graph by means of sensing the labels at neighboring vertices, while in the EAW model an agent needs to see the labels of outgoing edges only at its current location. The VAW approach is similar in nature to the RTA* (*Real-Time A**) and LRTA* (*Learning RTA**) presented by Korf [10], which are two variations on the famous A* heuristic search, with a cost function that takes the current searcher's location into account, thus making the algorithm more realistic for field applications like robotics. Koenig et al. [9] show that in a strongly connected directed graph the cover time by LRTA* is bounded from above by $2|V|D$ for any number of agents. On the other hand, Knight [6] shows that RTA* demonstrates a better multi-agent behavior compared with LRTA* since it tends to disperse the agents more effectively. In [7] and [8] it is shown that an edge-counting procedure, which is a simpler version of EAW, covers a graph (i.e. visits all edges) within time $O(|E|D)$. Here we show a proof which is much simpler, and which also yields a new result: after at most $O(|E|D)$ steps, the agent goes into a period of length $|E|$ which is an Euler cycle.

In this paper, however, we are mainly concerned with *patrolling* rather than *covering*, meaning that we are more interested in the long-term periodic behavior of the process, and especially in its load-balancing property: we want the frequency of visits to tend to be uniform on all edges. One way to quantify such uniformity is by means of the *blanket time*, originally defined by Winkler and Zuckerman [19] for random walks. The *blanket time* for a walk on a graph, denoted by B, is defined as the first time at which the ratio between the number of traversals of any two edges in the graph does not exceed two. In what follows we prove an upper bound on B_{EAW}, the blanket time for the EAW.

1.4. *Structure of the Paper.* In Section 2 we formally define the EAW algorithm for a single agent and show that an agent working according to the algorithm eventually traverses all edges of a graph. We present a new proof of the fact that if the agent works in an Eulerian graph, at the moment it completes the first traversal of the graph—i.e. traverses the last unvisited edge—it also completes an Eulerian cycle, as first proved in [3].

In Section 2.1 the algorithm is extended to the multi-agent case. It is proven there that the algorithm has a property of balanced traversals of the graph's edges: the difference between the numbers of traversals of any edge cannot become too large, regardless of whether the agents are synchronized or not. Also, we prove that no edge is left unvisited for a time longer than $|E|$. As a consequence of this property we improve the time bound it takes EAW to find an Eulerian cycle in an Eulerian graph $G(V, E)$ to $O(|E|D)$. We then show that for a team of k agents running on a general graph, after at most $2(1 + 1/k)|E|D$ steps the ratio between edge visit numbers of any two edges is at most two. In Section 3 we give the results of simulations using EAW. Results of multi-agent random walk simulations are provided there for comparison. We conclude in Section 4 with a summary and a discussion.

2. Edge Ant Walk—the Single Agent Case.

The EAW algorithm of Wagner et al. [13] works in a way inspired by the behavior of real ants while foraging for food. Studies on ants (e.g. [1] and [5]) show that the pheromone-based search strategies used by ants in foraging for food in unknown terrains tend to be very efficient. It is believed that ants build a network of information with vertices represented by points of encounter between ants and the information is either passed between ants at a vertex or via pheromone traces that are left on the ground. In our clearly over-simplified model we assume that the information network is a graph, and the role of a pheromone is taken by labels on each vertex and/or edge, that our search a(ge)nts can read and modify. The basic rule of motion is: when in vertex v, an "ant" always chooses the edge traversed earliest and moves through it. In the EAW algorithm of [13], it is assumed that upon traversing an edge e, an ant leaves on the exit from a vertex corresponding to e a pheromone marking which is proportional to the current time. Hence, an ant always chooses an edge with the weakest pheromone level. Since we consider each endpoint (direction) of an edge separately, it always makes sense to consider a directed graph for our model of the network. In case of undirected edges we substitute them with two directed edges—one in each direction. The problem with this method of marking is that the required memory amounts to $O(|E| \log T)$, where T is the running time. However, the same behavior can be achieved by a simpler method of marking, in which we assume that for each vertex v, there is some order between the edges emanating from v, and the ant always exits v through the next edge in this order. Thus, assuming that for every vertex v the edges emanating from the vertex are initially labeled with numbers from 1 to *degree*(v), the EAW algorithm of [13] can be rewritten as follows:

```
Initialization:
    for every vertex $u$
        set next exit pointer in $u$ to point to the first edge emanating
        from $u$

Rule Edge-Ant-Walk
(1)  Let $k$ be the location of the next exit pointer in $u$ and $e = u \to v$ be
         the corresponding edge
(2)  Move next exit pointer in vertex $u$ to the edge $(k + 1)$ mod $degree(u)$
(3)  $u := v$;
(4)  Go to (1).
end Edge-Ant-Walk.
```

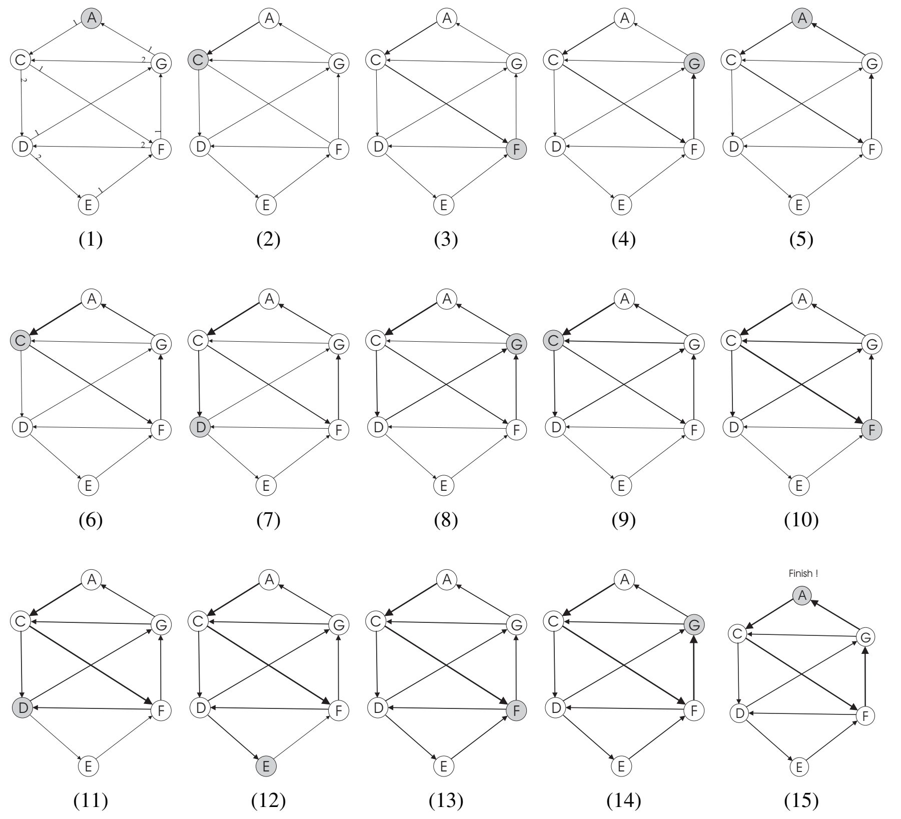


Fig. 1. An example of the EAW algorithm exploring a graph. Sketch (1) shows the graph. Numbers show the order of exits from each node. Sketches (2)–(14) show stages of the algorithm. Heavier edges correspond to a larger number of traversals. A shaded node is the current location of the agent.

See Figure 1 for an example of EAW traversing all edges of a graph.

It was proved in [13] and [14] that EAW covers a graph within time $O(|V||E|)$. In [7] and [8] it is shown that an edge-counting procedure, which is a simpler version of EAW, covers a graph (i.e. visits all edges) within time $|E|D$. Later it was shown by Bhatt et al. [3] that this process not only covers the graph, but also leads the ant to traverse an Euler cycle, if one exists. In this paper we show that:

1. A single EAW agent enters an Euler cycle within time $|E|D$.
2. Many agents are not worse, and are sometime better, than a single agent in terms of both the cover time and achieving an Euler cycle.
3. k agents achieve an almost uniform coverage of the graph ("blanket time") within time inversely proportional to k.

For purpose of completeness and clarity, and in order to lay the foundations for the multi-agent analysis in the next section, we first present a simplified proof that in $O(|V||E|)$ time the agent traverses all edges of the graph and enters an Eulerian cycle.

One can imagine the set of edges traversed by the EAW algorithm as a snake running away from its tail, the agent being the head of the snake. When the head is in a vertex with yet unused edges, the agent goes to one of them and the snake grows. At some step t_1, this process of growing must end since there are only a finite number of untraversed edges. At the next, $(t_1 + 1)$st, step the agent goes through an already traversed edge. Assuming that the "growing" started at step t_0, we call the time interval $[t_0 \cdots t_1]$ an *exploratory stage*. Clearly, all edges traversed at steps $[t_0 \cdots t_1]$ were new—that is, with a zero σ-value. Note that there may be several exploratory stages, and between them there are stages of re-visiting parts of the graph that have already been explored. The following lemma shows that during a non-exploratory stage, the partial graph that has already been visited is always Eulerian. We say that a moment t is *within an exploratory stage* $[t_0 \cdots t_1]$ if $t_0 \leq t \leq t_1$. We denote the set of edges explored during the ith exploratory stage by $\overline{E_i}$, and the set of all edges traversed by time t as E_t.

LEMMA 1 (Eulerian Subgraphs). *At any moment t not within an exploratory stage, the graph induced by E_t is an Eulerian graph.*

PROOF. By definition, and since no new edges are discovered between exploratory stages,

$$E_t = \bigcup_i \overline{E_i},$$

where the union is taken over exploratory stages that occurred before time t.

We now prove by induction on m, the number of exploratory stages, that the graph induced by $\bigcup_{i \leq m} \overline{E_i}$ is an Eulerian graph.

- For $m = 0$ the statement is trivial—E_0 is an empty set.
- We define $\widehat{E_m} = \bigcup_{i \leq m} \overline{E_i}$, and assume that the induced graph $\widehat{G_m} = (\widehat{V_m}, \widehat{E_m})$ is also an Eulerian graph. Then, since G is assumed to be Eulerian, a graph $G - \widehat{G_m}$, containing edges of G that are not in $\widehat{G_m}$, is a set of one or more Eulerian graphs. Hence, if the $(m + 1)$st exploratory stage started at some vertex v, it must also end in v. Indeed, at any time during the $(m + 1)$st exploratory stage, when the agent is in some vertex u other than v, u was entered during the $(m + 1)$st exploratory stage until this time one more time than exited. Since the set of edges $G - \widehat{G_m}$ that were "new" prior to the $(m + 1)$st exploratory stage is an Eulerian graph, the exploratory stage is continued past vertex u. The only vertex where it can stop is v. Thus, during the $(m + 1)$st exploratory stage the number of entrances to any vertex is equal to the number of exits from it. Hence $\widehat{E_{m+1}}$ induces an Eulerian graph. Therefore, the graph induced by

$$\widehat{E_{m+1}} = \widehat{E_m} \cup \overline{E_{m+1}}$$

is an Eulerian graph as a union of Eulerian graphs having common vertices. □

In particular, Lemma 1 implies that each exploratory stage by itself is an Euler cycle.

COROLLARY 1. *The set $\overline{E_m}$ of edges explored during the* mth *exploratory stage, ordered according to the time of exploration, is an Eulerian cycle on the graph induced by $\overline{E_m}$.*

For any set $E = \{e_1, \dots, e_k\}$ of edges traversed at least once by the algorithm we define $\mathit{Timed}_t(E)$ to be the permutation of E ordered according to the time of the most recent visit up to time t. Hence, the latest traversed edge of E is at the last place in $\mathit{Timed}(E)$.

LEMMA 2 (Eulerian Cycles). *At any moment t not within an exploratory stage, $\mathit{Timed}_t(E_t)$ is an Eulerian cycle on the graph induced by E_t.*

PROOF. Assume that the statement of the lemma holds for $t \in [1 \cdots t_0]$, i.e. that

$$\mathit{Timed}_{t_0}(E_{t_0}) = \langle e_1, \dots, e_{|E_{t_0}|} \rangle$$

is an Eulerian cycle and e_1 emanates from some vertex u and $e_{|E_{t_0}|}$ is incident on u. Two cases should be considered:

1. The edge traversed at time $t_0 + 1$ is not new. This edge is e_1, since e_1 is the first edge in $\mathit{Timed}(E_{t_0})$ and clearly has the least σ-value among the edges emanating from u. In this case the set of visited edges remains unchanged, $E_{t_0+1} = E_{t_0}$, but their timed permutation is cyclically shifted by one place:

$$\mathit{Timed}(E_{t_0+1}) = \langle e_2, \dots, e_{|E_{t_0}|}, e_1 \rangle.$$

   Hence, the edge that was first in $\mathit{Timed}(E_{t_0})$ became last in $\mathit{Timed}(E_{t_0+1})$.
2. The mth exploratory stage starts from vertex u at time t_0 and ends in u at time t_1. By Corollary 1, $\mathit{Timed}_{t_1}(\overline{E_m}) = \langle \overline{e_1}, \dots, \overline{e_{|\overline{E_m}|}} \rangle$ is an Eulerian cycle in a graph induced by $\overline{E_m}$. By the definition of σ-indices, for every edge e in $\overline{E_m}$, its σ-index at time t_1 is greater than that of any edge not in $\overline{E_m}$. Thus,

$$\mathit{Timed}_{t_1}(E_{t_1}) = \mathit{Timed}_{t_0}(E_{t_0}) \oplus \mathit{Timed}_{t_1}(\overline{E_m}),$$

   where $\oplus$ stands for the concatenation of the two sequences. Hence, the lemma holds for $t = t_1$. $\square$

It is the case when the agent traverses an already traversed edge that brought up the analogy of a snake: in this case the "snake" ($\mathit{Timed}_t(E_t)$) does not grow, it just follows its tail. Once the whole graph is already explored, the agent will be following its "tail" indefinitely in a cycle. Thus, if we know that by time t_0 the agent has traversed all the edges, then for every $t \geq t_0$, $\mathit{Timed}_t(E)$ is an Eulerian cycle. All such cycles are identical: $\mathit{Timed}_{t+1}(E)$ is obtained from $\mathit{Timed}_t(E)$ by a cyclic left shift.

COROLLARY 2. *An Eulerian cycle is a limit cycle of the algorithm—once completed it will be repeated indefinitely.*

Our next theorem shows that in $O(|V||E|)$ time the agent traverses all edges of the graph and enters an Eulerian cycle. The first proof of this fact appeared in [3]. Our different method of analysis yields a simpler proof as well as some new results that will be described later.

THEOREM 1. *If $G = (V, E)$ is a directed Eulerian graph, then in time $O(|V||E|)$ the EAW agent will traverse all edges of the graph and enter an Eulerian cycle.*

PROOF. Assume that from time t_0 and on, E_t does not change. That is, for all $t \geq t_0$, $E_t = E_{t_0}$. Hence, by Corollary 2 and Lemma 2, starting at time t_0 the agent repeats the Eulerian cycle defined by $\mathit{Timed}_{t_0}(E_{t_0})$. By the definition of the algorithm, if the agent enters a vertex having untraversed edges emanating from it, the agent should use an untraversed edge. However, our assumption is that there is no change to E_{t_0}, thus $E_{t_0} = E$ meaning that all edges of the graph have been traversed.

Now let us find a bound on the time t_0 when the last unvisited edge is traversed. The time spent in all exploratory stages is equal to $|E|$—every edge is discovered exactly once during some exploratory stage. The walk taken by the agent between two exploratory stages is Eulerian, hence the time interval between two such stages clearly cannot be longer than $|E|$. Since an exploratory stage starting at a vertex u ends only after all edges emanating from u have been traversed, there is at most one exploratory stage per vertex and the total number of exploratory stages is bounded from above by $|V|$. Thus,

$$t_0 \leq |E| + |E||V| = O(|E||V|). \qquad \square$$

COROLLARY 3. *After the last unvisited edge of G is traversed at some time t, the agent will always complete the Eulerian cycle $\mathit{Timed}_t(E_t)$.*

PROOF. By Lemma 2 at time t, $\mathit{Timed}_t(E_t)$ is an Eulerian cycle containing all graph edges. □

It is argued in [3] that the agent can "notice" that all the graph was explored and halt if between two consecutive passages through the same edge no new edge was discovered. This halting condition requires only $O(1)$ additional space to label one edge. This halting condition follows immediately from Corollary 3

2.1. *Multi-Agent Edge Ant Walk.* In order to speed up the exploration of a graph or a network one may want to have several agents running simultaneously. In this section we discuss and analyze the performance of such a multi-agent system of explorers. We consider two models of synchronization: synchronous and asynchronous. In the synchronous model there is a clock common to all agents and the agents move at its ticks. In the asynchronous model the agents may move whenever they want. In the case that a claim holds only in the synchronous case, we state this explicitly.

Joint exploration by several agents can be achieved by having all agents share the same set of next-exit pointers; thus the agents cooperate via the shared markers. When several agents want to exit from a vertex simultaneously they can either choose an arbitrary order in which they exit from the vertex or they can use an order predefined by their IDs. Since the set of the edges visited each step and the locations of the agents afterwards are the same independently of the order of the agents, these two approaches are equivalent and in what follows we assume the agents are identical—without unique IDs.

Denote by $d_t(u \to v)$ the number of times edge $u \to v$ was traversed until time t. Denote by $d_t(u) = \min_v(d_t(u \to v))$, taken over all edges emanating from u. By the definition of the algorithm, $0 \leq d_t(u \to v) - d_t(u) \leq 1$. Denote by $N_t(u)$ the number of visits of all agents in vertex u until time t. It would be convenient to think that an agent initially located at some vertex visits that vertex at time $t = 0$. In this notation we can say that for every vertex u the number of times u was exited until time t, i.e. the sum of $d_t(u \to v)$ over all edges emanating from u, equals $N_{t-1}(u)$. Also, $N_t(u)$ equals the sum of $d_t(v \to u)$ over all edges incident on vertex u plus the number of agents initially in u.

LEMMA 3 (Effect of Adding an Agent). *Adding an agent increases the total number of edge traversals without decreasing the number of traversals of any particular edge and the number of visits of any vertex at any given time.*

PROOF. The first part of the lemma is trivial—the number of edge traversals by time t by k agents is kt—increases with k.

We prove the second part by induction on time t. We want to show that both $N_t(v)$ and $d_t(v \to u)$ for any edge $v \to u$ and vertices v and u cannot decrease when the $(k+1)$st agent is added.

- For $t = 1$ the lemma is obvious—neither the number of traversals of any edge nor the number of visits of any vertex decreases when a new agent is added.
- Assume that the statement of the lemma holds for $t = i$ and prove it for $t = i+1$. First, let us see that $d_{i+1}(u \to v)$ cannot decrease when the $(k+1)$st agent is added. By the induction hypothesis, the number of visits of vertex u till time $t = i$ inclusive—$N_i(u)$—does not decrease as a result of adding a new agent. Thus, we have that, assuming identical initializations of the next-exit pointers, the number of exits through each edge emanating from the vertex u by time $t+1$, cannot decrease after adding a new agent. By definition, $N_{i+1}(v)$ equals the number of agents initially located in v plus the sum of $d_{i+1}(u \to v)$ over all edges incident on v. Since the number of agents initially in v clearly did not decrease when the $(k+1)$st agent was added and, as we just proved, $d_{i+1}(u \to v)$ also does not decrease, we have that $N_{i+1}(v)$ cannot decrease upon adding an agent. □

In addition to the result of Lemma 3 we want to have a guarantee that there cannot be a very long delay between two traversals of any edge. In the single agent case, when an agent enters an Eulerian cycle, each edge will be revisited within a period of $|E|$. In the following lemma we show a similar result for a multi-agent case:

LEMMA 4 (Revisiting Edges). *Consider k agents synchronously exploring a network. Assume that edge $u \to v$ was visited by some agent at time t. This edge must be revisited at least once again until time $t + |E| + 1$ by some, perhaps different, agent.*

PROOF. Let us look at the behavior of the agents since some agent a traversed edge $u \to v$. Without loss of generality, we can define the time of this traversal to be zero. We want to prove that by time $t = |E| + 1$ the edge $u \to v$ will be traversed again by some agent.

First, assume there is only one agent—the agent a. After completing the first exploratory stage started by agent a at time $t = 0$ the agent will traverse edge $u \to v$—the first edge traversed during the exploratory stage. By definition of an exploratory stage all edges traversed by the agent during this exploratory stage were not traversed before. Thus, the length of the exploratory stage and the time interval between two visits of edge $u \to v$ cannot be more than $|E|$. Applying Lemma 3 we conclude that by time $t = |E| + 1$ edge $u \to v$ will be visited at least twice, implying the lemma. $\square$

Lemmas 3 and 4 show us that adding agents cannot decrease the number of visits of a vertex by the algorithm and that a vertex already visited will not be neglected for a long time. Next we prove that the vertices are visited more or less uniformly—the number of visits of the vertices of the graph cannot differ by too much. Define the *flow* through an edge $u \to v$ till time t as the number of times the edge was traversed till that time. Given a cut $\langle S \mid T \rangle$ in a graph G (the set of edges connecting a vertex in S to a vertex in T, when $V = S \cup T$), we denote by $F(S \to T)$ the sum of flows through the edges of the cut. $F(S \to T)$ equals the number of agents that moved from S to T minus the number of agents that moved in the opposite direction. By definition $|F(S \to T)| \leq k$, where k is the number of agents.

THEOREM 2. *Assume a k-agent EAW algorithm runs on a directed Eulerian graph. Then at any time t for any edge $e = u \to v$ it holds that*

$$-k \leq d_t(e) - d_t(v) \leq k + 1.$$

(The proof does not assume that the network is synchronous.)

PROOF. We separate the set V of vertices of the graph into disjoint subsets V_i, such that $v \in V_i$ implies that $d_t(v) = i$. First we show that

$$d_t(e) - d_t(v) \leq k + 1, \tag{1}$$

then we prove

$$-k \leq d_t(e) - d_t(v). \tag{2}$$

- Assume, on the contrary, that $v \in V_j$ and there exists an edge $e = u \to v$ such that $d_t(e) \geq j + k + 2$. Consider a cut $\langle S \mid T \rangle$, $S = \bigcup_{m>j} V_m$ and $T = \bigcup_{m \leq j} V_m$. Since the graph of the network is Eulerian, there is the same number—call it l—of edges going in each direction of the cut. Since $d_t(u \to v) \geq j + k + 2$, we have that $d_t(u) \geq j + k + 1$ (recall that the d_t values of two edges emanating from the same vertex cannot differ by more than one), and hence $u \in S$, thus for the total directed flow $F(S \to T)$ from S to T it holds that

$$F(S \to T) \geq (l - 1)(j + 1) + j + k + 2 = l(j + 1) + (k + 1).$$

On the other hand, by the definition of V_j, the flow from any vertex in T is bounded from above by $j+1$. Hence, the flow $F(T \to S)$ is bounded from above by $l(j+1)$, yielding

$$F(S \to T) - F(T \to S) \geq k+1,$$

which is impossible since there are only k agents.

- Now we assume that $v \in V_j$ and there exists an edge $e = u \to v$, $d_t(e) \leq j-k-1$. Consider a cut $\langle S \mid T \rangle$ where $S = \bigcup_{m \geq j} V_m$ and $T = \bigcup_{m<j} V_m$. Since $d_t(u \to v) \leq j-k-1$, we have that $d_t(u) \leq j-k \leq j-1$, hence $u \in T$. In this case out of l edges in direction $T \to S$ at least one—edge e—contributes to the flow $F(T \to S)$ at most $j-k-1$, the remaining l edges contribute at most j each. Thus,

$$lj - (k+1).$$

On the other hand, the flow through any l edges in direction $S \to T$ is at least j and the total flow in this direction of the cut is at least lj. Thus,

$$F(S \to T) - F(T \to S) \geq k+1.$$

However, this is impossible—the flow cannot be greater than the number of agents. □

Corollary 4 follows from Theorem 2:

COROLLARY 4. *Assume a k-agent EAW algorithm runs on a network. Then at any time t for any two edges $u_1 \to v$ and $v \to u_2$ it holds that*

$$|d_t(u_1 \to v) - d_t(v \to u_2)| \leq k+1.$$

PROOF.

$$d_t(u_1 \to v) - d_t(v \to u_2) \leq d_t(u_1 \to v) - d_t(v) \leq k+1,$$
$$d_t(u_1 \to v) - d_t(v \to u_2) \geq d_t(u_1 \to v) - (d_t(v)+1) \geq -k-1. \qquad \square$$

COROLLARY 5. *Assume a k-agent EAW algorithm runs on a network. Then at any time t for any two edges $e_1 = u_1 \to v$ and $e_2 = u_2 \to v$ it holds that*

$$|d_t(e_2) - d_t(e_1)| \leq 2k+1.$$

PROOF. By Theorem 2,

$$-k \leq d_t(e_1) - d_t(v) \leq k+1,$$
$$-k-1 \leq d_t(v) - d_t(e_2) \leq k.$$

Summing up the two inequalities we obtain

$$-(2k+1) \leq d_t(e_1) - d_t(e_2) \leq 2k+1.$$

This completes the proof of the corollary. □

Using Corollary 5 for $k = 1$ one can see that for a single agent at any time the difference between the numbers of traversals of any two edges that enter the same vertex is upper-bounded by three. We next show an upper bound on the difference in d_t between any two vertices in the graph.

COROLLARY 6. *Assume a k-agent EAW algorithm runs on a network. Then at any time t and for any two vertices u and v it holds that*

$$|d_t(u) - d_t(v)| \leq d(u, v)(k + 1),$$

where $d(u, v)$ stands for the length of the shortest path between vertices u and v.

PROOF. Consider a shortest path from u to v, say $u = x_1, x_2, \ldots, x_p = v$. The number of edges on this path is $p - 1 = d(u, v)$. Now apply the definition of $d_t(u)$ and (1) repeatedly to obtain

$$\begin{aligned} d_t(u) &\leq d_t(x_1 \to x_2) \\ &\leq d_t(x_2) + k + 1 \\ &\leq d_t(x_2 \to x_3) + k + 1 \\ &\leq d_t(x_3) + 2(k + 1) \\ &\vdots \\ &\leq d_t(v) + d(u, v)(k + 1), \end{aligned}$$

implying the corollary. □

Since $d(u, v) \leq D$, for any two vertices u and v we have

$$|d_t(u) - d_t(v)| \leq D(k + 1).$$

In [15] it is proved that

$$|d_t(u) - d_t(v)| \leq (|V| - 1)(1 + k/\lambda(G)),$$

where $\lambda(G)$ stands for edge connectivity as defined in [4]. For most graphs the diameter of a graph is much smaller than $|V|$. In such cases the $O(Dk)$ result of Corollary 6 can improve the result from [15]. In [7] and [8] it is shown that an edge-counting procedure, which is a simpler version of EAW, covers a graph (i.e. visits all edges) within time $|E|D$. In the following we show that this process not only covers the graph but also goes into an Euler cycle within time $O(|E|D)$.

Before proving the new result, let us observe that it is very tempting to try to improve $D(k + 1)$ in Corollary 6 to some expression that is a slower function of k, for instance to $D + k$. However, the example with the chain-graph shown in Figure 2 proves that this cannot be done in a general *asynchronous* network. Here we dispatch from S one agent at time. When an agent arrives at T, it stops there and the next agent is dispatched. Note

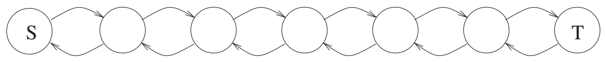


Fig. 2. All the agents are initially in vertex S of a chain-graph G, and all next-exit pointers initially point to the left. The motion is asynchronous; thus we let the next agent exit from S only when the previous agent arrives at vertex T. When all k agents are in T, the edge emanating from S was exited kD times, while the edge emanating from T was never traversed.

that after an agent comes to T the next-exit pointers are back in the original position; so for each agent it will take D exits from S until it arrives at T. This is repeated k times. At the end the edge exiting from S is traversed kD times, while the edge exiting from T was not traversed at all.

Using Corollary 6 we can improve the $O(|V||E|)$ bound on the time it takes a single agent to enter an Eulerian cycle in a graph. For the purpose of the new bound, we first prove the following corollary.

COROLLARY 7. *Assume a k-agent EAW algorithm runs on a network. Then at any time t and for any edge e it holds that*

$$d_t(e) \geq \frac{tk}{|E|} - (k+1)D,$$

where D is the diameter of the graph G.

PROOF. We denote by $d_{\min}$ and $d_{\max}$ the minimum and maximum numbers of visits to edges in G at time t. It is implied by Corollary 6 that

$$d_{\max} - d_{\min} \leq D(k+1).$$

Also, since $\sum_{e\in E} d_t(e) = tk$, we have that

$$\begin{aligned} tk &= \sum_{e\in E} d_t(e) \\ &\leq d_{\min} + (|E|-1)d_{\max} \\ &\leq d_{\min} + (|E|-1)(d_{\min} + D(k+1)) \\ &\leq d_{\min}|E| + (k+1)D|E|, \end{aligned}$$

leading to

$$d_{\min} \geq \frac{tk}{|E|} - D(k+1).$$ □

THEOREM 3. *If $G = (V, E)$ is a connected Eulerian graph, then in time $O(|E|D)$ the agent will traverse all edges of the graph and enter an Eulerian cycle.*

PROOF. By substituting $k = 1$ in Corollary 7, it is implied that if $t > 2|E|D$, then at time t, $d_{\min} > 0$, thus all edges of the graph are traversed in $O(|E|D)$ time. By Corollary 3 this is the time it takes an agent to enter an Eulerian cycle. □

An important property that we require of an algorithm intended to traverse a network repeatedly is some sort of uniformity or balance in the traversal of edges. One way to quantify such uniformity is by means of the *blanket time*, originally defined by Winkler and Zuckerman [19] for random walks. The *blanket time* for a walk on a graph, denoted by B, is defined as the first time in which the ratio between the number of traversals of any two edges in the graph does not exceed two. We now prove an upper bound on B_{EAW}, the blanket time for EAW.

THEOREM 4. *Assume that a team of k EAW agents runs on a network. Then at any time* $t \geq 2(1 + 1/k)|E|D$, *for any two edges* $e_1, e_2 \in E$, *it holds that*

$$\frac{d_t(e_1)}{d_t(e_2)} \leq 2,$$

or in other words, the blanket time for EAW is upper bounded as follows:

$$B_{\text{EAW}}(G) \leq 2\left(1 + \frac{1}{k}\right)|E|D. \tag{3}$$

PROOF. Let e_m be the edge with the minimal value of $d_t(\cdot)$ at time t. Using Corollary 7, $d_t(e_m) \geq tk/|E| - (k+1)D$. Now consider edges e_1, e_2 at time t. By Corollary 6, $|d_t(e_2) - d_t(e_1)| \leq (k+1)D$, hence

$$\left|1 - \frac{d_t(e_1)}{d_t(e_2)}\right| = \left|\frac{d_t(e_2) - d_t(e_1)}{d_t(e_2)}\right| \leq \frac{(k+1)D}{d_t(e_m)} \leq \frac{(k+1)D}{tk/|E| - (k+1)D}, \tag{4}$$

implying that in order to have

$$\frac{d_t(e_1)}{d_t(e_2)} \leq 2$$

it is sufficient that

$$\left|1 - \frac{d_t(e_1)}{d_t(e_2)}\right| \leq 1,$$

or in other words, by applying (4), that

$$\frac{tk}{|E|} \geq 2(k+1)D,$$

implying

$$t \geq 2\left(1 + \frac{1}{k}\right)|E|D. \qquad \square$$

An interesting question regarding k-agent EAW is whether it enters a periodic motion for $k > 1$. A motion of k agents is considered *periodic* if the sequence of k-tuples of edges visited by the group is periodic. The answer to this question is given by the following lemma:

LEMMA 5. *After a sufficiently long time in which the k-agent EAW runs on a synchronous network, it enters a periodic motion. During a period each edge is traversed the same number of times. The length T of the period is a multiple of $|E|/k$.*

PROOF. The number of states of the algorithm—locations of the agents and the next-exit pointers in all vertices—is finite, since the process is deterministic, hence it must eventually enter a period. Assume there are two edges that are traversed a different number of times during the period. Then after a sufficiently long time the difference between the number of traversals of these two edges can grow arbitrarily large, in contradiction to Corollary 6.

Assuming that during a period of length T each edge is traversed m times, we have $kT = m|E|$. Hence, $T = m|E|/k$. □

In our extensive testing of k-agent EAW in a synchronous network we have experimentally noticed the following fact: after a very short time—less than $3|E|$ iterations in most tests—the algorithm entered a period of length $T = m|E|/k$ for $m \leq 2k$. For instance, in all the tests of the algorithm for $k = 2$ on graphs of different topologies and sizes, the algorithm quickly entered a period of length $|E|$. We could neither prove this nor find a counterexample.

It can be mentioned that, although in case of a synchronous network we assumed the length of any edge is one, the results can be generalized to integer lengths of edges—we could simply think there are intermediate vertices on edges having a length longer than one.

3. Results of Simulations.

We have tested the EAW algorithm with different numbers of agents on Eulerian graphs of various densities. For each experiment, we plot the average over 1000 random graphs, selected according to various criteria as described below. In Figure 3 one can see plots of the time it takes EAW to cover a graph for the first time. On each plot, the performance of a random walk is provided for comparison—in all cases EAW is many times faster and the difference grows for denser graphs. The ratio between the cover time by EAW and the cover time by a random walk grows with the density of the graph: EAW covered graphs with 150 edges for the first time on average 4.54 times faster than a random walk with the same number of agents, 5.10 times faster on graphs with 300 edges, 5.48 times faster for 600 edges and 5.63 times faster on graphs with 1200 edges. The number of vertices in the graphs was 50 in all our tests. As expected, the speedup from adding agents decreases as the number of agents becomes comparable with the number of vertices.

In another set of simulations we used *small world* random graphs; these are relatively sparse graphs with a small number (about 10%) of high degree vertices, called *hubs*. Such graphs are scale-free and are considered to mimic, although simplistically, the topology of real-life communication networks in both the living [18] and the electronic [17], [2] media. In this type of graphs, EAW seems to be much (10–100 times) faster than random walk in terms of both cover time (Figure 4) and blanket time (Figure 5).

We also ran a set of simulations on random graphs with average density 0.15, in order to compare the actual blanket time with our upper bound. Results show that this bound is

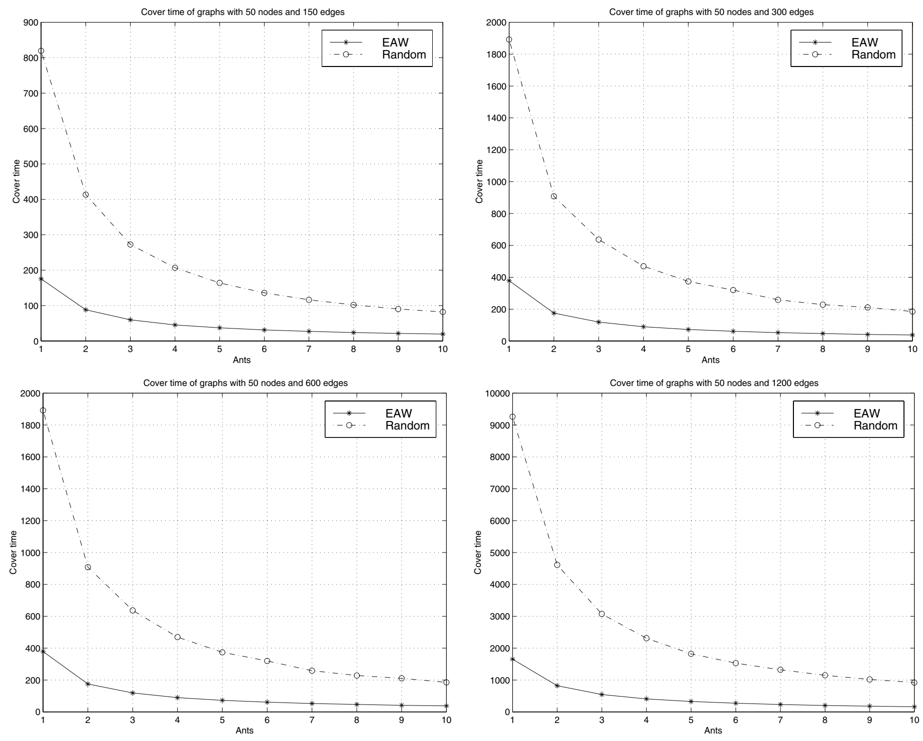


Fig. 3. Cover time of graphs with 50 vertices and different numbers of edges by multi-agent EAW and random walk, drawn against the number of participating ants.

far from being tight; see Figure 6. However, when looking at the results of simulations of blanket time versus the number of vertices n (Figure 7), one can see that the blanket time behaves pretty much as n^2, which is basically in line with the prediction of formula (3).

The difference between the numbers of traversals of different edges was less than five in all experiments at all times and, on average, in 20 times the cover time, the ratio between number of traversals of any two different edges was within 0.1 of 1. Plotting this time against the time it takes a multi-agent random walk to reach the same quotient is not useful since, even for very small graphs with 10 vertices and 50 edges, this time for random walk was around 10^6 iterations.

Tables 1 and 2 display another interesting property of EAW—the algorithm always enters a short cycle, with length almost always less than the number of edges and always less than twice the number of edges. Especially extensive testing of the algorithm was performed for two agents. Despite being run on millions of random graphs of different models and our attempts to build "hard" examples by hand, no cycle longer than the number of edges has been found. We can also see in Table 2 that although the cycle length decreases as new agents are added, it jumps up at seven. This can be explained by the fact that seven is co-prime with every edge set size we tested yielding a lower bound

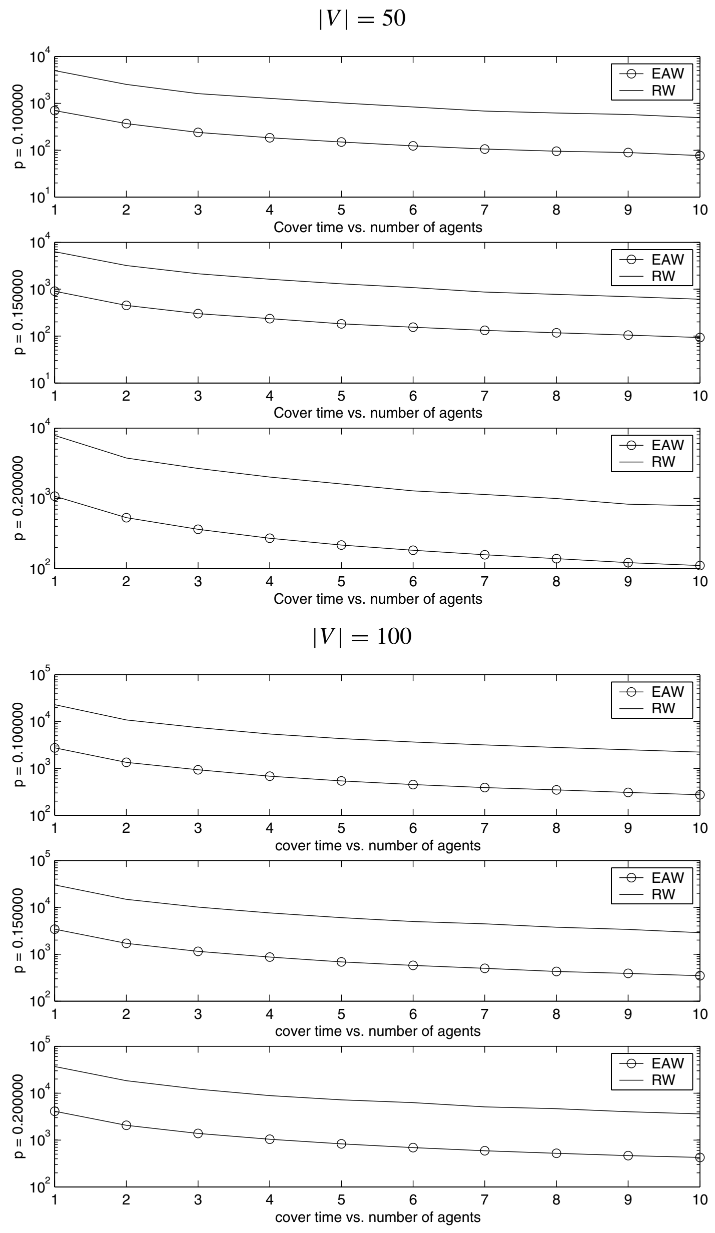


Fig. 4. Cover time (time until all edges are visited) for EAW and random walk on random small world graphs, with 50 (top) and 100 (bottom) vertices, with density from 0.1 to 0.2 and 10% hubs; thus, density of 0.1 means that there are $0.1 \cdot \binom{100}{2}$ edges in a graph with 100 vertices.

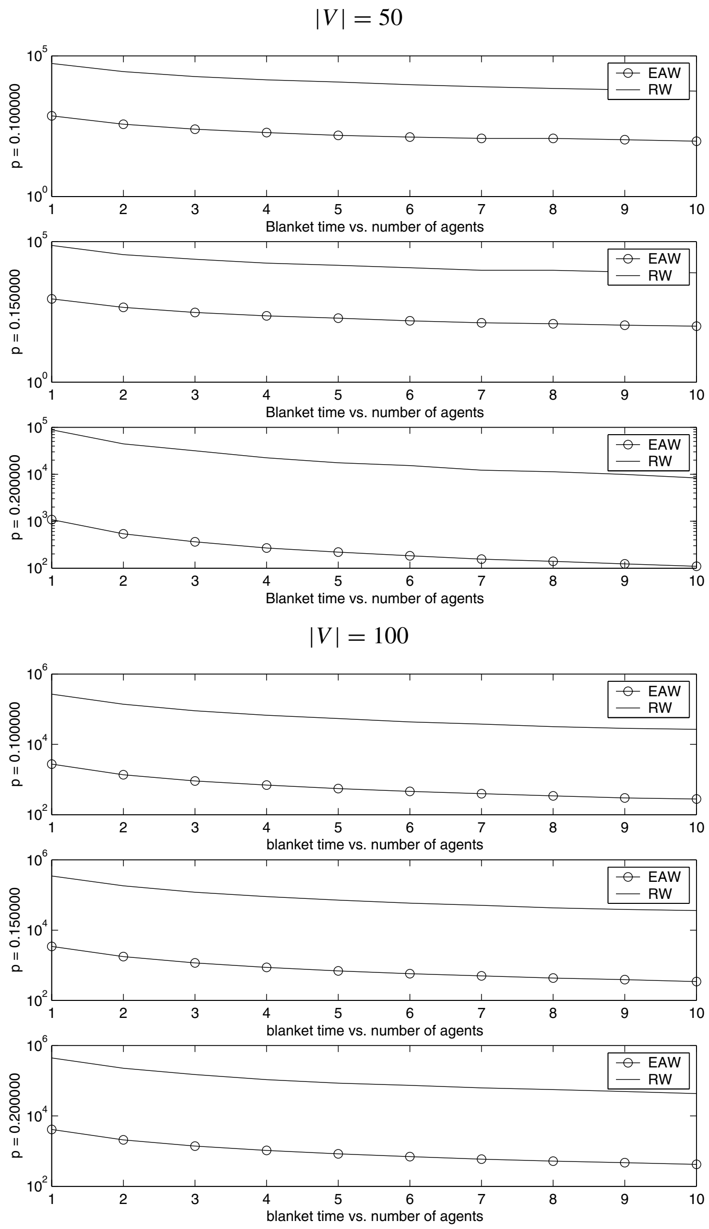
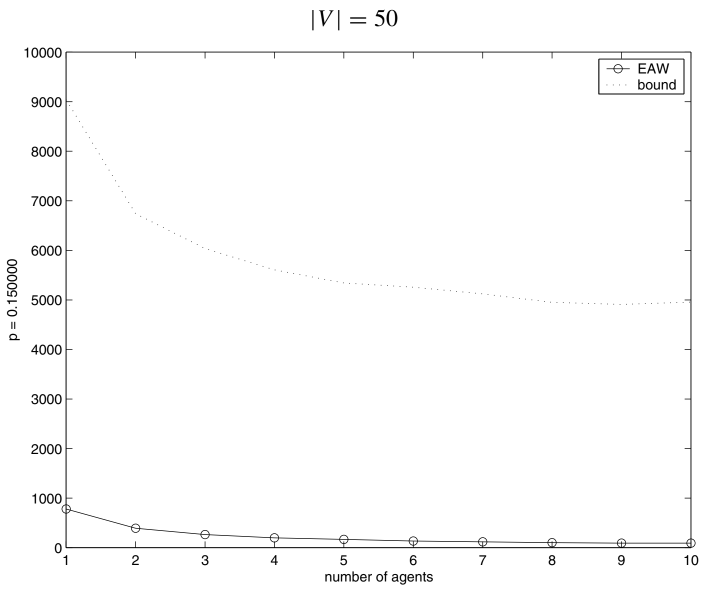


Fig. 5. Blanket time (time until the visit ratio between any two edges is ≤ 2) for EAW and random walk on random small world graphs, with 50 (top) and 100 (bottom) vertices, with density from 0.1 to 0.2 and 10% hubs.



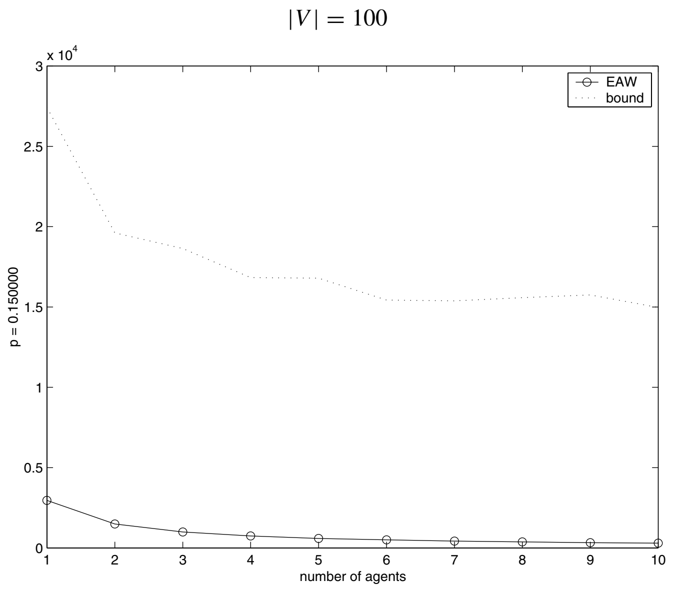
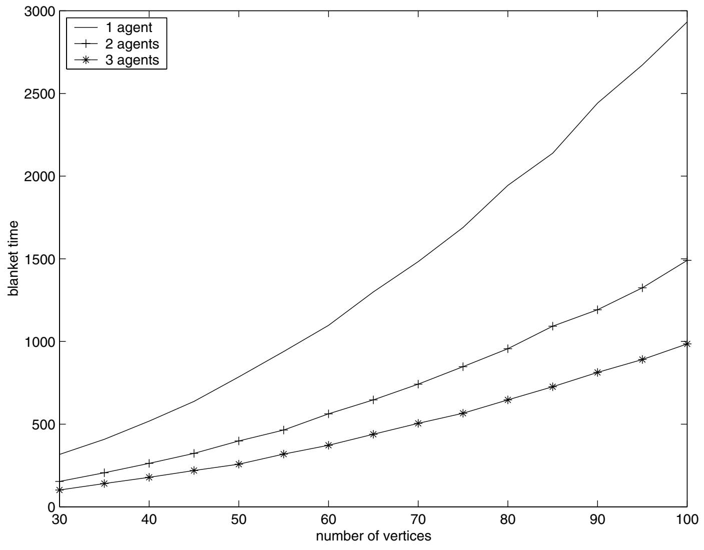
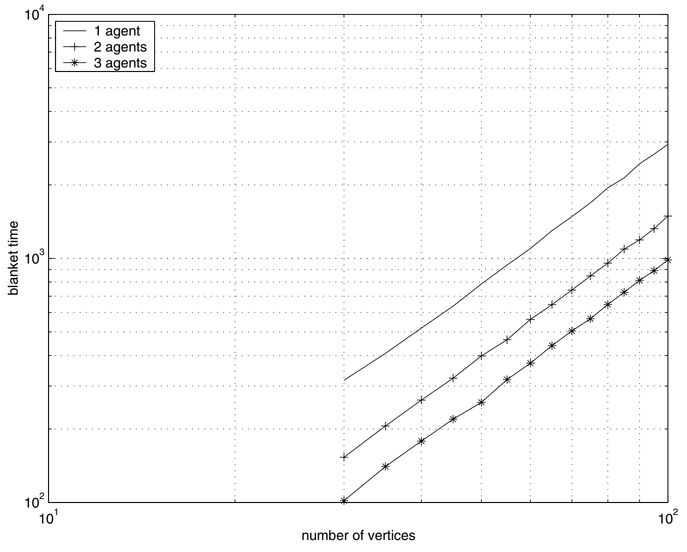


Fig. 6. Blanket time by EAW compared with the analytical upper bound, averaged over 30 random graphs, with 50 (top) and 100 (bottom) vertices.



(a)



(b)

Fig. 7. Blanket time versus the number of vertices, for one, two and three agents, in linear (a) and logarithmic (b) scales. It can be seen that the blanket time behaves pretty much as n^2.

Table 1. Time to cycle.*

	Number of agents									
Edges	1	2	3	4	5	6	7	8	9	10
150	27	53	63	84	72	71	130	106	99	71
300	80	76	88	92	95	97	135	116	128	96
600	164	139	136	139	137	136	160	142	165	134
1200	454	284	242	227	222	219	230	214	225	212

*The average (over 1000 runs) time it takes the EAW algorithm to enter a cycle—the time when the first cycle is completed minus the length of the cycle itself.

on the length of period of at least $|E|$—more than for the other tested numbers of agents. This can explain the jump in time-to-cycle for seven ants in Table 1.

4. Conclusion

4.1. *Summary and Open Questions*. In this paper the EAW process has been proved to cover the edges of an Eulerian graph and settle into an Eulerian cycle within a number of steps that does not exceed $2|E|D$, $|E|$ being the number of edges in the graph and D being its diameter. Moreover we have shown that for a team of k agents running EAW on a graph, after at most $2(1 + 1/k)|E|D$ steps the edge visits are balanced up to a factor of two, i.e. the so-called blanket time does not exceed this upper bound. Simulations have shown that (a) both cover time and blanket time decrease as k is increased, up to an asymptotic limit, and (b) EAW is much faster than random walk for both cover time and blanket time.

Despite the simplicity of the multi-agent EAW, some of its facets still pose challenging problems. It would be very interesting to know what kind of traversal periods the algorithm exhibits and what is a bound on their length. For instance, in all simulations for two agents running synchronously, the period was of length either $|E|$ or $|E|/2$. Another problem is to try to improve the bound

$$|d_t(u) - d_t(v)| \leq d(u, v)(k + 1)$$

Table 2. Cycle length.*

	Number of agents									
Edges	1	2	3	4	5	6	7	8	9	10
150	150	95	84	94	77	78	150	96	83	96
300	300	174	142	127	122	121	279	131	144	129
600	600	360	304	259	263	232	580	221	307	238
1200	1200	802	750	648	706	567	1165	597	735	581

*The average (over 1000 runs) cycle length of multi-agent EAW for different numbers of agents.

in Corollary 6. Although we have shown (see Figure 2) that in the asynchronous case this bound is tight, our simulations make us hope that in the case of synchronously running agents, the bound $d(u, v)(k + 1)$ can still be improved.

Another interesting question is: how robust is the EAW process against edge or vertex failures? For example, one would wonder whether results similar to those in [16] for the VAW process can be achieved for EAW as well.

References

[1] F. R. Adler, D. M. Gordon, Information Collection and Spread by Networks of Patrolling Ants, *The American Naturalist*, Vol. 140, No. 3, pp. 373–400, 1992.

[2] A. L. Barabasi, *Linked*: *The New Science of Networks*, Perseus, Cambridge, MA, 2002.

[3] S. Bhatt, S. Even, D. Greenberg, R. Tayar, Traversing Directed Eulerian Mazes, *Proceedings of WG* '2000—26*th International Workshop on Graph-Theoretic Concepts in Computer Science*, *June* 15–17, *Konstanz*, *Germany*, Lecture Notes in Computer Science, Vol. 1928, pp. 35–46, Springer-Verlag, Berlin, 2000.

[4] S. Even, *Graph Algorithms*, Addison-Wesley, Reading, MA, 1979.

[5] D. M. Gordon, The Expandable Network of Ant Exploration, Animal Behaviour, Vol. 50, pp. 372–378, 1995.

[6] K. Knight, Are Many Reactive Agents Better than a Few Deliberative Ones, *Proceedings of the International Joint Conference on Artificial Intelligence* (*IJCAI*), pp. 432–437, 1993.

[7] S. Koenig, Complexity of Edge Counting, in: Goal-Directed Acting with Incomplete Information, Technical Report CMU-CS-97-199, School of Computer Science, Carnegie Mellon University, November 1997.

[8] S. Koenig, R. G. Simmons, Easy and Hard Testbeds for Real-Time Search Algorithms, *Proceedings of the National Conference on Artificial Intelligence*, pp. 279–285, 1996.

[9] S. Koenig, B. Szymanski, Y. Liu , Efficient and Inefficient Ant Coverage Methods, *Annals of Mathematics and Artificial Intelligence* (Special issue on Ant-Robotics, editors: I. A. Wagner, A. M. Bruckstein), Vol. 31, No. 1/4, pp. 41–76, 2001.

[10] R. E. Korf, Real-Time Heuristic Search, *Artificial Intelligence*, Vol. 42, pp. 189–211, 1992.

[11] R. Tarjan, Depth-First Search and Linear Graph Algorithms, *SIAM Journal on Computing*, Vol. 1, No. 2, pp. 146–160, 1972.

[12] G. Tarry, Le probléme des labyrinths, *Nouvelles Annales de Mathematiques*, Vol. 14, p. 187, 1895.

[13] I. A. Wagner, M. Lindenbaum, A. M. Bruckstein, Smell as a Computational Resource—a Lesson We Can Learn from the Ants, *Proceedings of ISTCS* '96, *Jerusalem*, pp. 219–230, 1996.

[14] I. A. Wagner, M. Lindenbaum, A. M. Bruckstein, Distributed Covering by Ant-Robots Using Evaporating Traces, *IEEE Transactions on Robotics and Automation*, Vol. 15, No. 5, pp. 918–933, October 1999.

[15] I. A. Wagner, M. Lindenbaum, A. M. Bruckstein, On-Line Graph Searching by a Smell-Oriented Vertex Process, AAAI-97, Workshop on On-Line Search, Providence, RI, 1997.

[16] I. A. Wagner, M. Lindenbaum, A. M. Bruckstein, ANTS: Agents on Networks, Trees and Subgraphs, *Future Generations Computing Systems*, (Special Issue on Ant Optimization, editors: M. Dorigo, G. Di Caro, T. Stutzle), Vol. 16, No. 8, pp. 915–926, June 2000.

[17] D. J. Watts, *Small Worlds—the Dynamics of Networks Between Order and Randomness*, Princeton University Press, Princeton, NJ, 1999.

[18] R. Weiss, T. K. Knight, Jr., Engineered Communications for Microbial Robotics, *DNA*6: *Proceedings of the* 6*th DIMACS Workshop on DNA Based Computers*, *Leiden*, *The Netherlands*, 13–17 June, 2000, Lecture Notes in Computer Science, Vol. 2054, pp. 1–16, Springer-Verlag, Berlin.

[19] P. Winkler, D. Zuckerman, Multiple Cover Time, *Random Structures and Algorithms*, Vol. 9, No. 4, pp. 403–411, December 1996.