Future Generation Computer Systems 16 (2000) 915–926

ANTS: Agents on Networks, Trees, and Subgraphs

Israel A. Wagner[a,*], Michael Lindenbaum[b], Alfred M. Bruckstein[b]

[a] *IBM Haifa Research Lab, Matam, Haifa 31905, Israel*

[b] *Department of Computer Science, Technion City, Haifa 3200, Israel*

Abstract

Efficient exploration of large networks is a central issue in data mining and network maintenance applications. In most existing work there is a distinction between the active 'searcher' which both executes the algorithm and holds the memory and the passive 'searched graph' over which the searcher has no control at all. Large dynamic networks like the Internet, where the nodes are powerful computers and the links have narrow bandwidth and are heavily-loaded, call for a different paradigm, in which a noncentralized group of one or more lightweight autonomous agents traverse the network in a completely distributed and parallelizable way. Potential advantages of such a paradigm would be fault tolerance against network and agent failures, and reduced load on the busy nodes due to the small amount of memory and computing resources required by the agent in each node. Algorithms for network covering based on this paradigm could be used in today's Internet as a support for data mining and network control algorithms. Recently, a vertex ant walk (VAW) method has been suggested [I.A. Wagner, M. Lindenbaum, A.M. Bruckstein, Ann. Math. Artificial Intelligence 24 (1998) 211–223] for searching an undirected, connected graph by an a(ge)nt that walks along the edges of the graph, occasionally leaving 'pheromone' traces at nodes, and using those traces to guide its exploration. It was shown there that the ant can cover a *static* graph within time nd, where n is the number of vertices and d the diameter of the graph. In this work we further investigate the performance of the VAW method on *dynamic* graphs, where edges may appear or disappear during the search process. In particular we prove that (a) if a certain spanning subgraph S is stable during the period of covering, then the VAW method is guaranteed to cover the graph within time nd_{s}, where d_{s} is the diameter of S, and (b) if a failure occurs on each edge with probability p, then the expected cover time is bounded from above by $nd((\log\Delta/\log(1/p)) + ((1+p)/(1-p)))$, where Δ is the maximum vertex degree in the graph. We also show that (c) if G is a static tree then it is covered within time $2n$. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Dynamic graph search; Edge failure model; Vertex ant walk; Edge ant walk; Cover time

1. Introduction

Ant algorithms are computing methods that make use of ideas from the world of real ant colonies in the context of optimization problems. Some examples for such applications are the traveling salesman problem [11], graph coloring [6], quadratic assignment [22] and routing in communication networks [7,21]; see [8,10] for a list of references to such applications. Both theoretical [1] and experimental [16] studies show that ants continuously and very effectively forage uncharted territories in search for food. Hence, a natural place to apply their search strategies is when the need arises to either learn about the state of the environment or to search for specific items of information in a large, time-varying and rather chaotic environment. As

* Corresponding author. Tel.: +972-4-8296331;
fax: +972-4-8296116.
E-mail address: wagner@il.ibm.com (I.A. Wagner)

explained in [1,16], real ants cope with such challenges by creating a network of information in which the vertices represent meetings between ants, and the information is spread by means of traces of a chemical, called pheromone, that are either left on the ground or transferred directly from one ant to another. In our very limited imitation of real ants behavior we assume that the searching agent moves on a graph from a vertex to a neighbor vertex, and reads and modifies the traces left at the visited vertex; we assume no direct communication between agents. Rather, an indirect communication is achieved by allowing agents to read and use the traces left by others to the benefit of their common mission.

An example of a chaotic environment in which an efficient search is desired is a network of connected computers like the Internet, where the links are made of various kinds of hardware that may suffer frequent failures. In most existing search algorithms for the Internet (for example, those embedded in search engines like 'Yahoo' and 'Alta-Vista') there is a distinction between the active 'searcher', which both executes the algorithm and has a huge amount of memory, and the passive 'searched network' or environment, over which the searcher has no control at all (see, e.g. [14,20]). Such methods put a heavy load on network links and may require a long time to gather the results of a thorough search. In huge and dynamic networks like the Internet, where nodes are usually powerful computers while the communication links are heavily-loaded and mostly have a low capacity, there seems to be a need for a different search paradigm, in which a noncentralized group of one or more lightweight autonomous agents traverse the network in a completely distributed and parallelizable way. Potential advantages of such a paradigm are fault tolerance against network and agent failures, and reduced load on the network links. Algorithms for network covering based on this paradigm could be used in today's Internet as a support for data mining and network control algorithms. Although not every searcher needs a complete covering of the network, it may be needed for either maintenance (e.g. a continuous search for viruses) or for a really exhaustive search — one that is not supported by the existing search engines which rely on predefined indexes and thus cannot find all web-pages that exist at a given moment.

More specifically, we consider a memoryless a(ge)nt that searches a simple, undirected graph G for food; that is, it attempts to explore the whole graph in a short time. A graph G is generally defined by a pair $G = (V, E)$, where V is a set of vertices (or points) and E is a set of edges (or lines), where each edge connects two of the vertices. In our paradigm, the ant has the ability to leave pheromone traces on vertices and to sense the traces at the current location and its immediate neighbors, which are those vertices connected to the current vertex by an edge. By 'sensing the trace' at a vertex we mean that the ant senses both the pheromone traces that have been left on the vertex, and the time at which the trace was last updated. Our goal is an efficient method for *covering* the graph, that is: visiting every vertex in it. Formally, a vertex v at time t is marked by a pair $(\sigma_t(v), \tau_t(v))$, where $\sigma_t(v)$ is the mark left on v, and $\tau_t(v)$ is the time of the most recent mark left there up to time t. Initially we set both marks to zero for all vertices in the graph. Being at a vertex u at time t, the ant smells around and chooses among $N(u)$, the set of neighbors[1] of u, a neighbor v with the minimum mark on it, where marks are ordered lexicographically by the (σ, τ) values on the vertices. Assuming the minimum was found on neighbor v, the ant then sets $\sigma(u)$ to $\sigma(v) + 1$, and $\tau(u)$ to the current time. Then it goes to v. The reason for taking the value of $\sigma(v)$ into account when setting $\sigma(u)$ is that we wish to keep the local difference in $\sigma(\cdot)$ as small as possible; as will be shown later, this is needed in order to guarantee a short cover time. Intuitively, this rule of motion behaves like a steepest-descent optimizer that attempts to find the minimum of the function $\sigma(\cdot)$, with the additional option to dynamically alter its value, thereby avoiding being stuck in a local minimum like steepest-descent methods.

We shall call this process 'vertex ant walk' (VAW) to distinguish it from a similar 'edge ant walk' process that has been previously introduced by the authors in [27,28], where the edges, rather than the vertices, were marked. It has been shown there that a group of k trace-oriented ants that evolve in an edge process can cover a static graph in a (worst case) time of $O(\Delta n^2/k)$, where Δ is the maximum vertex-degree

[1] $N(u)$, the set of neighbors of vertex u in a graph $G(V, E)$, is the set $\{v | u \neq v, (u, v) \in E\}$.

and n the number of vertices. The vertex ant walk for static graphs was described in [29], where it was proved that such a process covers a connected static graph using no more than nd steps, where d is the diameter of G, defined as the maximum distance between any pair of vertices in the graph, that is

$$d = \max_{u,v\in V} \{\text{length of the shortest path connecting } u \text{ and } v\}.$$

This is an improvement compared to the edge process, since $nd < \Delta n^2$.

In this paper we extend our analysis of the VAW method to *dynamic* graphs, where edges may stop functioning or recover during the execution of the algorithm. We show that cover time can still be bounded, under two models of dynamic graphs. In particular we prove that (a) if a certain spanning subgraph S is stable during the period of covering, then the VAW method is guaranteed to cover the graph within time nd_s, where d_s is the diameter of S, (b) if a failure occurs on each edge with probability p, then the expected cover time is bounded from above by $nd((\log\Delta/\log(1/p))+((1+p)/(1-p)))$, where Δ is the maximum vertex degree in the graph, and (c) if G is a tree then it is covered within time $2n$.

The VAW process, beyond its theoretical interest, may find application in searching a large network of WWW sites, where the physical links between the sites suffer from frequent hardware failures. In such a network, it sometimes makes sense to send one or more searching agents to do the job rather than apply a static agent that can access a site only if there is a functional path to that site at the moment. A VAW agent can move from site u to site v by copying its code to a (temporary) area of memory in v, then deleting itself from u, leaving there only a small 'signature' that will later help the agent and its fellows in navigation.

The rest of the paper is organized as follows. First we mention related work in Section 2; then, in Section 3, the VAW process is defined and some examples are given. The cover time is analyzed in Section 4 for two models of dynamic graphs, followed by a simulation experiment that demonstrates the ability of VAW agents to cooperate. Then in Section 5 we prove an upper bound on the cover time of trees. We conclude in Section 6 with a summary and some open questions.

2. Related work

Graph exploration is an old problem; a paper on the subject was published as early as 1895 [24]. The goal of covering (also known as exploring or searching) a graph is to visit all its vertices while going along the edges, possibly using markers to store information. In the sequel we cite a sample of covering algorithms; the interested reader can find more using the references therein. Perhaps the most famous search algorithm is *depth-first search* (DFS), first introduced in [24] and further discussed in [12,13,17,23]. In DFS the agent looks for an unvisited neighbor. If one exists, it goes there; otherwise it goes back along the first edge used to enter the current vertex. This edge is marked the first time a vertex is entered, and this marking should persist through the process, or else the searcher cannot backtrack and thus may get stuck. The covering time of a static graph by a simple DFS is $2m$, m being the number of edges in the graph. However, the lack of tolerance to edge failures is a drawback of DFS. Using randomness in resolving ties, the average cover time by DFS can be reduced to $1.5m$ [15]. Another, memoryless, method is the *random walk* — just choose a random neighbor of the current vertex and go there. Clearly, covering by a random walk is rather slow; in [2,3] it was shown to cover a graph within expected time $O(mn)$, where m is the number of edges and n the number of vertices. It was also shown that, under a proper initial distribution of k agents, their simultaneous random walk covers the graph in expected time $O(mn/k)$ [5]. More sophisticated methods use memory (or field marking) to keep a certain record of the history, but the usage of that record is flexible; instead of determining a single good continuation, the agent prioritizes the neighbors and then takes the best among many ways to continue its search; thus, a faulty edge does not cause a terminal disturbance but only extends cover time. The *real-time A^** (RTA*) and *learning RTA** (LRTA*) [18] 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. The worst case cover time by LRTA* is $O(mn)$. In [25,26] a *counter-based exploration* method is described in which a counter holds the number of visits to each vertex so far. The cover time by this method is bounded by $O(n^2d)$, where d

Table 1
Upper-bounds on cover times for a sample of graph-covering algorithms, assuming the graph to be static and to have n vertices, m edges, diameter d and maximum vertex degree Δ[a]

Method	Reference	Cover time
Depth-first search	[24]	$2m$
Random walk	[2]	$O(mn)$ (expected)
k Random-walkers	[5]	$O(mn/k)$ (expected)
Semi-random	[15]	$1.5m$ (expected)
Learning real-time A*	[18]	$O(mn)$
Counter-based	[25]	$O(n^2d)$
Nearest neighbor	[19]	$O(m \log n)$
k Edge ant walkers	[27]	$O(\Delta n^2/k)$
Vertex ant walk	[29]	$O(nd)$

[a]For randomized algorithms the *expected* bound is given.

is the diameter of the graph. In the nearest neighbor approach (NNA) method of [19], a graph is learned by moving towards the nearest un-visited edge, and an upper bound is proved on the cover time of $O(m \log n)$, where m is the number of edges in the graph, which can be as large as $O(n^2)$. An interesting source of ideas for search algorithms is the world of animals, especially insects like ants that use distributed pheromone traces to help their navigation [4,9]. To the best of our knowledge, no upper bounds have been reported on covering by ant-based systems, but extensive simulations have shown them to be efficient heuristics for various search problems. Table 1 summarizes a sample of cover-time results for static graphs.

All the algorithms and cover times described so far are suitable for *static* graphs, that is — graphs which do not change during the execution of the algorithm. If the graph is *dynamic* the algorithms either do not cover it or their cover time is not guaranteed. The contributions of the current paper are in proving that ant walk is efficient not only for static graphs but also for dynamic graphs of various kinds, and in the analysis of the case of searching trees.

3. Vertex ant walk

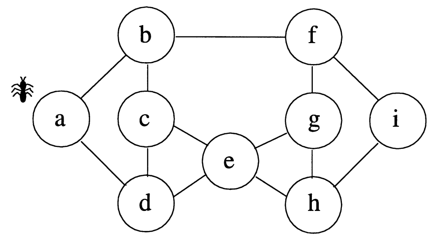
The VAW process is formally defined by the VAW rule of motion given in Fig. 1. Initially, all (σ, τ) values are set to $(0, 0)$ and a starting vertex is chosen for each agent, and then the rule is applied repeatedly. There is no explicit stopping condition for this algorithm; however one can use the upper bound on cover time to stop it after a sufficient period that guarantees covering. Note that our rule applies for any number of ants; if multiple ants occupy the same vertex at the same time, they all attempt to write the same numbers (Steps 3 and 4) so there is no logical conflict. Their next step, however, might be different since the random decision (Step 1) could differ between ants. See Figs. 2 and 3 for examples of the VAW evolution on static and dynamic graphs. In the sequel we shall investigate the case of a single VAW ant by analysis, and of multiple ants by simulation.

```
Rule Vertex-Ant-Walk(u vertex; (σ(.),τ(.)) for u's neighborhood)
(1) $v := u$'s neighbor with minimal
        lexicographic value of $(\sigma(.),\tau(.))$;
        (in case of a tie make a random decision)
(2) $\sigma(u) := \sigma(v) + 1$;
(3) $\tau(u) := t$;
(4) $t := t + 1$;
(5) go to $v$.
end Vertex-Ant-Walk.
```

Fig. 1. The vertex ant walk (VAW) rule.

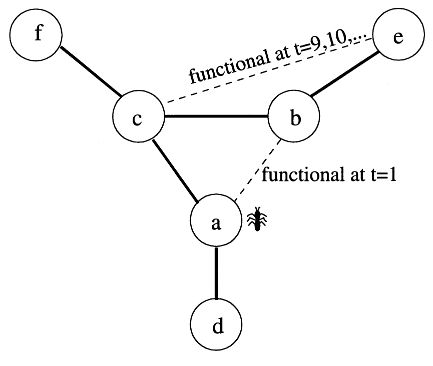
4. Vertex ant walk on dynamic graphs

It is known that a single VAW works well for static graphs; as has recently been shown in [29], a VAW



(a,1,1)	(b,1,2)	(c,1,3)	(e,1,4)	(h,1,5)	(i,1,6)
(f,1,7)	(g,2,8)	(e,1,9)	(d,2,10)	(a,2,11)	(b,2,12)
(c,2,13)	(e,2,14)	(h,2,15)	(i,2,16)	(f,3,17)	(g,3,18)
(e,3,19)	(d,3,20)	(a,3,21)	(b,3,22)	(c,4,23)	(e,3,24)
⋮					⋮

Fig. 2. A VAW ant covers a static graph, starting from vertex 'a', within 10 steps. Here $n = 9$ and the diameter is equal to 4. The numbers in parentheses represent the vertex and its σ, τ values upon leaving the vertex.



(a,1,1)	(b,1,2)	(c,1,3)	(f,2,4)	(c,2,5)	(a,1,6)
(d,2,7)	(a,3,8)	(c,1,9)	(e,2,10)	(b,2,11)	(c,3,12)
⋮					⋮

Fig. 3. The first 12 steps in a VAW tour of a dynamic graph with six vertices, diameter $d = 3$ and stable diameter $d_s = 4$, starting from vertex 'a'. Edges (a, b) and (c, e) are not stable; (a, b) is only functional at time 1 while (c, e) begins to function at time 9. The corresponding $(u_t, \sigma(u_t), \tau(u_t))$ evolution is shown in the table. The process starts at time $t = 1$, and $\sigma(u_t)$ is the value after the tth step has been completed. Note that the graph is covered in the 10th step upon first visiting vertex 'e', while the analytical upper bound is $nd_s = 24$.

ant covers *any* static connected graph within *nd* steps, where n is the number of vertices and d is the diameter of the graph. How efficient is the process for a graph with a dynamic structure? The analytical results from [29] do not hold for that case, and it is our goal here to generalize those results and achieve an upper bound on the covering time of the VAW process for dynamic graphs. We consider two models of dynamic graphs: the *stable spanning subgraph* model, and the *probabilistic failure* model. In the first we assume that there exists a spanning subgraph of G which is always functional, while in the second model we consider each edge to be faulty with some probability p, and establish an upper bound on the expected cover time as a function of p and some topological properties of G: its diameter, the number of vertices and the maximum vertex degree.

4.1. G_S — a dynamic graph with a stable subgraph

Let G_S be a graph with a stable spanning subgraph S. The subgraph S is a subset of the edge-set E that covers all vertices and does not fail in the process. This kind of subgraph is sometimes called the *backbone* of the network. We shall now show that such a dynamic graph is covered within time nd_s, where d_s is the diameter of S. For this purpose we extend the analysis from [29], using the following definitions.

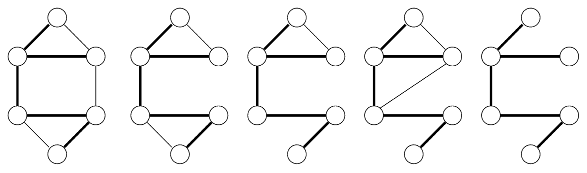


Fig. 4. A possible evolution of a dynamic graph G_S on six vertices with a stable spanning subgraph S (in bold lines).

Definition 1. An edge $e \in E$ is called *stable* if it does not fail during the process.

Definition 2. A spanning subgraph (V, S) is called a *stable subgraph* of $G = (V, E)$ if $S \subset E$ and all edges in S are stable.

Definition 3. A graph G_S is *dynamic with a stable subgraph* if not all its edges are stable, but the edges of the spanning subgraph S are stable.

See Fig. 4 for an example of such a dynamic graph.

We shall now show that, as long as the graph has not yet been covered, the σ-values of already-visited vertices cannot grow beyond a certain limit. For that purpose, let us denote by U_t the set of all vertices with $\sigma = 0$ at time t, that is

$$U_t = \{u \in V \,|\sigma_t(u) = 0\}.$$

Clearly, $U_0 = V$ and if the graph is covered at time t then U_t is empty. We shall also denote by $\text{dist}_s(u, v)$ the distance[2] between u and v using only edges of S, the stable subgraph of G. Similarly, if Q is a set of vertices in G, then $\text{dist}_s(u, Q)$ will denote the minimum distance between u and any member of set Q, that is

$$\text{dist}_s(u, Q) = \min_{v \in Q} \{\text{dist}_s(u, v)\}.$$

If Q is empty then $\text{dist}_s(u, Q) = \infty$.

The following lemma shows that $\sigma_t(u)$ cannot exceed u's distance from the set U_t.

[2] The *distance* between two vertices in a graph is the number of edges in a shortest path connecting the vertices.

Lemma 1. *At all times t it holds for all vertices $u \in V$ that*

$$\sigma_t(u) \le \mathrm{dist}_{\mathrm{S}}(u, U_t), \tag{1}$$

where U_t is the set of unvisited vertices at time t.

Proof. The lemma is clearly true when $t = 0$ since all $\sigma(\cdot)$ values are being preset to zero and since $U_0 = V$. Assuming it is also true at time t, we claim that the $(t + 1)$th step of VAW does not cause any harm, that is, at time $t + 1$ it holds for all $u \in V$ that

$$\sigma_{t+1}(u) \le D, \tag{2}$$

where D is the stable distance of u from U_{t+1}, that is, $D = \mathrm{dist}_{\mathrm{S}}(u, U_{t+1})$. If u is not the vertex which is visited at time $t + 1$, then $\sigma_{t+1}(u) = \sigma_t(u)$. The unvisited set is monotonically nonincreasing, that is, $U_{t+1} \subseteq U_t$, implying that

$$\mathrm{dist}_{\mathrm{S}}(u, U_t) \le D,$$

hence

$$\sigma_{t+1}(u) = \sigma_t(u) \le \mathrm{dist}_{\mathrm{S}}(u, U_t) \le D,$$

and Eq. (2) holds. Otherwise, u is a vertex to which the ant has arrived at time $t+1$. We need to prove that, after the $t+1$th step, $\sigma_{t+1}(u) \le D$. Because $u \notin U_{t+1}$ then $D \ge 1$, and u must have a neighbor, say w, such that

$$\mathrm{dist}_{\mathrm{S}}(w, U_{t+1}) = D - 1,$$

(w is the first vertex in a shortest path from u to U_{t+1}), and it follows that, because the lemma was true at time t,

$$\sigma_t(w) \le D - 1.$$

According to the VAW rule, the ant should always go to a vertex with the minimum value of σ among its neighbors; hence the σ_t value of the vertex visited at time $t + 2$ cannot exceed $D - 1$, since at least one neighbor of u (namely w) certainly has this value. Hence, the new value of u at time $t+1$ will not exceed D, which proves the Lemma. □

Our next step is to show that under the VAW rule, the sum of all σ-markers at the end of the tth step is at least t. To this end, let us define the total amount of traces on the graph at time t, $\sigma_t(G)$, to be the sum of all the σ values on the vertices at that time.

Lemma 2. *At all times t,*

$$\sigma_t(G) \triangleq \sum_{u\in V} \sigma_t(u) \ge t.$$

Proof. Let us denote by u_i the node visited at time i, such that the sequence of nodes visited up to time t is $u_1, u_2, \dots, u_t$, and the σ-value of a vertex u upon completion of time-step t, by $\sigma_t(u)$. The only change to σ may occur at the vertex currently visited, hence

$$\sigma_t(G) = \sigma_{t-1}(G) + \delta_t,$$

where δ_t stands for the addition to $\sigma(G)$ at time t, that is

$$\delta_t = \sigma_t(u_t) - \sigma_{t-1}(u_t). \tag{3}$$

According to the VAW rule, upon moving from u_i to u_{i+1}, the value of $\sigma_i(u_i)$ is set to $\sigma_i(u_{i+1}) + 1$; hence

$$\delta_i = \sigma_i(u_{i+1}) - \sigma_{i-1}(u_i) + 1. \tag{4}$$

From Eq. (3)

$$\sigma_t(u_t) = \sigma_{t-1}(u_t) + \delta_t, \tag{5}$$

and, substituting i by $t - 1$ in Eq. (4),

$$\sigma_{t-1}(u_t) = \sigma_{t-2}(u_{t-1}) + \delta_{t-1} - 1. \tag{6}$$

Applying Eqs. (5) and (6) and recursively one gets

$$\begin{aligned} \sigma_t(u_t) &= \sigma_{t-1}(u_t) + \delta_t \\ &= \sigma_{t-2}(u_{t-1}) + \delta_{t-1} + \delta_t - 1 \\ &= \sigma_{t-3}(u_{t-2}) + \delta_{t-2} + \delta_{t-1} + \delta_t - 2 \\ &\vdots \\ &= \sigma_0(u_1) + \textstyle\sum_{i=1}^{t} \delta_t - (t - 1), \end{aligned}$$

and hence, since $\sigma_t(G) = \sum_{i=1}^{t} \delta_i$, we get that

$$\sigma_t(G) = \sigma_t(u_t) - \sigma_0(u_1) + t - 1.$$

But, according to our rule, $\sigma_0(u_1) = 0$ and $\sigma_t(u_t) \ge 1$, and the Lemma follows. □

Now let us combine the smoothness of $\sigma(\cdot)$ (Lemma 1) with its temporal accumulation (Lemma 2) to get

Theorem 1. *Denote by d_{S} the diameter of a stable spanning subgraph S of a dynamic graph G, and by n the number of vertices. Then after at most nd_{S} steps the graph G is covered.*

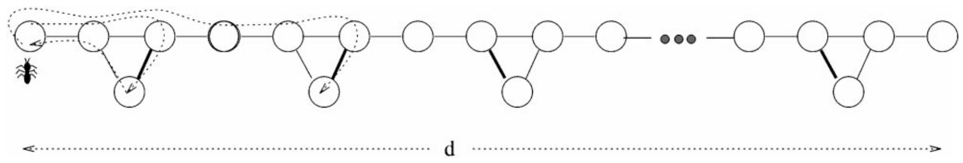


Fig. 5. A hard case for the VAW rule. There are n vertices, and about $1.25n$ edges. The diameter is about $0.8n$, and the time needed to traverse it may be as long as $O(nd) = O(n^2)$. The dotted arrows show the worst case where each triangle of vertices is a 'trap' that causes the ant to go back to its starting point. Note that removing the bold-line edges would make it a tree, thus reducing the cover time to at most $2n$.

Proof. Assume that at time t the graph has not yet been fully covered. Then there exists at least one node, say u, for which $\sigma(u) = 0$. According to Lemma 1, none of u's neighbors can have $\sigma > 1$, none of the neighbors' neighbors can have $\sigma > 2$, etc. Hence, the maximum σ value on any node at that time cannot exceed d_S, the diameter of S, and the sum $\sigma_t(G)$ cannot exceed nd_S. On the other hand, we know from Lemma 2 that $\sigma_t(G) \geq t$; hence $t < nd_S$; or, in other words if t exceeds nd_S, then the σ-values of all nodes should be at least 1 and the graph is covered. □

Note that although $\sigma_t(G) \geq t$, the change is not always smooth; for example, in the case depicted in Fig. 3, $\sigma_7(G) = 8$, $\sigma_8(G) = 10$, but $\sigma_9(G) = 9$.

Also note that Theorem 1 from [29] is achieved as a special case when G is static, that is all edges are stable and hence $S = G$. Also note that the bound of $O(nd_S)$ is tight; the example in Fig. 5 depicts a case where the cover time required by VAW is indeed $O(nd_S)$. Another insight gained from this example is that more edges do not always imply faster coverage; in this case if all bold edges are deleted from the graph, the cover time reduces to $2n$, as will be proven in Section 5.

4.2. G_p — the faulty edge model

Let us now consider a different type of dynamic graphs. We shall denote by G_p a connected graph with diameter d and n vertices, such that each edge may be faulty with probability p, and functional with probability $1 - p$, independently of any other edge, and independently in each time step. See Fig. 6 for an example; note that, as opposed to the G_S model, the graph is not always connected. If we use the VAW rule as is, it will cover the graph with some probability but it is hard to predict its performance because the upper

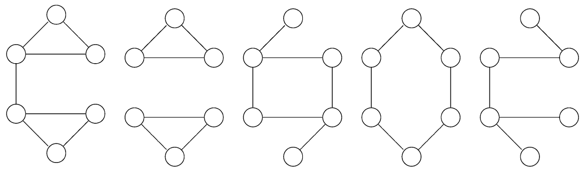


Fig. 6. 5 steps in the life of a dynamic graph G_p with all edges being prone to faults with some probability $p > 0$. Note that under this model, the graph is not always connected.

bound on σ (Lemma 1) no longer holds, and there may be sharp discontinuities in the σ function that can cause the ant to get stuck for a long time in a small area of the graph. Hence, we slightly modify the VAW rule by instructing our ant to behave as before but *refrain* from moving or marking until all neighbors of the current vertex are accessible at least *once* for deciding on the best neighbor (Step 1), and then a second wait may be needed for the chosen edge to be functional for making the move (Step 5). All other steps remain as before. The Waiting VAW rule is shown in Fig. 7. Clearly, not all neighbors should be *simultaneously* accessible, so the average waiting time (in Steps 1 and 5), as we shall see, is not too long. For example, if a vertex u has five emanating edges that randomly come and go, the ant visiting u has to see each of the five neighbors before choosing a move, although not all five edges should be up at the same time because we assume (in the single ant case) that vertices other than u do not change while the ant is in u. After the ant saw all neighbors and decided to go, say, to neighbor v, it may happen that v has again disappeared and the ant may need to wait until a second edge (u, v) is up again and Step 5 can be executed. Working this way, there is no harm to Lemma 1, but the value of $\bar{\sigma}_t(G_p)$, the average value of the total σ after t steps, will decrease as p increases, as is formulated in the following Lemma.

```
Rule Waiting Vertex-Ant-Walk(u vertex; $(\sigma(.),\tau(.))$ for u's neighborhood)
(1) probe all neighbors of $u$ and find $v$ such that
    $v := u$'s neighbor with minimal
        lexicographic value of $(\sigma(.),\tau(.))$;
        (in case of a tie make a random decision)
(2) $\sigma(u) := \sigma(v)+1$;
(3) $\tau(u) := t$;
(4) $t := t+1$;
(5) wait until $v$ is accessible and go there.
end Waiting Vertex-Ant-Walk.
```

Fig. 7. Waiting Vertex-Ant-Walk rule.

Lemma 3.

$$\bar{\sigma}_t(G_p) \geq \frac{t}{\log\Delta/\log(1/p)+(1+p)/(1-p)},$$

where σ_t is averaged over all possible evolutions of G_p and Δ is the maximum vertex degree in G.

Proof. Under the Waiting VAW rule there are two kinds of time steps: those in which the ant is passive (that is, waiting in a vertex until each neighbor is accessible at least once) and those in which it is active (that is, moves according to the rule). In order to estimate the ratio of active time to the total time, we first achieve an upper bound on $\bar{W}$, the average waiting time at a vertex u until each neighbor is accessible at least once. Since we do not require *all* edges to be functional at the same time, we can bound the expected waiting time at a vertex with a 'tail' analysis as follows. Assuming that all vertex degrees are less than or equal to Δ, we have

$$\begin{aligned}
\bar{W} &= \textstyle\sum_{t=0}^{\infty} t \,\text{Prob}\\
&\quad \{\text{all neighbors were seen at least once until time } t\}\\
&= \textstyle\sum_{t=1}^{\infty} \text{Prob}\\
&\quad \{\text{not all neighbors were seen until time } t\}\\
&= \textstyle\sum_{t=1}^{\infty}\left[1-(1-p^t)^{\Delta}\right]\\
&= \textstyle\sum_{t=1}^{\lfloor\log_{1/p}\Delta\rfloor}\left[1-(1-p^t)^{\Delta}\right]\\
&\quad +\textstyle\sum_{t=\lfloor\log_{1/p}\Delta\rfloor+1}^{\infty}\left[1-(1-p^t)^{\Delta}\right]\\
&\quad (\text{using}(1-q)^M \geq 1-Mq \text{ for } q<1)\\
&\leq \lfloor\log_{1/p}\Delta\rfloor + \Delta\textstyle\sum_{t=\lfloor\log_{1/p}\Delta\rfloor+1}^{\infty} p^t\\
&\leq \log_{1/p}\Delta + \Delta\left(p^{\lfloor\log_{1/p}\Delta\rfloor+2}\right)/(1-p)\\
&\leq \log_{1/p}\Delta + \Delta\left(p^{\log_p(1/\Delta)+1}\right)/(1-p)\\
&= \log_{1/p}\Delta + (p/(1-p))\\
&= (\log_2\Delta/\log_2(1/p)) + (p/(1-p)).
\end{aligned}$$

Now recall that the modified algorithm waits this time ($\bar{W}$) in order to see all neighbors and then another period of (average) $1/(1-p)$ in order to make the proper move. Let us denote the average total waiting time at a vertex by $\overline{W_{\text{tot}}}$; our discussion implies

$$\overline{W_{\text{tot}}} \leq \frac{\log\Delta}{\log(1/p)} + \frac{1+p}{1-p}.$$

(Note that if $p=0$ the waiting time is 1 as in a nonfaulty graph). Hence, after t steps, an average of $t/(\overline{W_{\text{tot}}})$ steps were active ones. Using Lemma 2 and the fact that no change occurs to the (σ,τ) marks in 'passive' times, the Lemma is proved. □

Going along the lines of Theorem 1 and its proof, one can prove that $\bar{t}(p)$, the average cover time under edge-fault probability p, is bounded from above

Corollary 1.

$$\bar{t}(p) \leq nd\left(\frac{\log\Delta}{\log(1/p)} + \frac{1+p}{1-p}\right).$$

Proof. Consider a run of the Waiting VAW rule. Lemma 1 still holds, so if at time t the graph has not yet been fully covered, then there exists at least one node, say u, for which $\sigma(u)=0$; none of u's neighbors can have $\sigma>1$, none of the neighbors' neighbors can have $\sigma>2$, etc. Hence, the maximum σ value of any node at that time cannot exceed d, the diameter of G, and the sum $\sigma_t(G)$ cannot exceed nd. On the other hand, we know from Lemma 3 that the average sum of σ at time t is bounded from below by $\bar{\sigma}_t(G_p) \geq t/(\log\Delta/\log(1/p)+(1+p)/(1-p))$. Thus, in an average run, t cannot exceed $nd(\log\Delta/\log(1/p)+(1+p)/(1-p))$. □

Remark 1. *In some cases it makes more sense to consider faulty vertices rather than faulty edges. For* *example, one may assume that each vertex is functional with probability* $1 - p$; *otherwise it does not function and all its emanating links are considered dysfunctional, too. Under such an assumption, our analysis holds as before with the only exception being* $\bar{W}$, *the average waiting time at a vertex. Let us call the waiting time under vertex failures* $\bar{W}_v$. *Using the same* Waiting VAW *rule of behavior, the ant has to probe all neighbor vertices before it can proceed. The calculation of the waiting time is like in the proof of Lemma 3, but there is yet another source of delay – the functionality of the current node. The ant cannot execute its code if the processor hosting it is not working. Thus we have to stretch the waiting time by a factor of* $1/(1 - p)$, *getting* $\bar{W}_v \leq \log \Delta / \log(1/p) + (1 + p)/(1 - p)$, *and*

$$\bar{t}_v(p) \leq nd \frac{1}{1-p} \left(\frac{\log \Delta}{\log(1/p)} + \frac{1+p}{1-p} \right).$$

4.3. Experiments in co-operation

In the multiagent setting we assume each of the agents to apply the same VAW rule. If two or more agents meet at a vertex, only one of them (say the one with highest ID) is allowed to set the new σ value for that vertex. Our analysis as reported above is only valid for a single agent, and we do not yet have a clear way for extending it to multiple agents working together to cover a dynamic graph. One problem that arises when multiple ants are used is the lackness of ‘global’ instantaneous knowledge, e.g. when an ant A_1 is at node u_1 and reads the state of node u_2 and decides to go there because it is a minimum. In the meantime ant A_2, that moved from u_2 to u_3, writes in u_2 the value $\sigma(u_3) + 1$. So, at the time ant A_1 moves from u_1 to u_2, u_2 is no longer the minimum but ant A_1 does not know about it.

In order to get an empirical result for the multiagent case, a simulation program was written and run on an IBM *PowerPC 604e* machine. We simulated sets of 1–12 agents, all starting at the same vertex, and covering a graph under both the stable subgraph (G_S) and the faulty edge (G_p) models of noise. The reason for starting all agents at the same vertex is to make their life as hard as possible in order to get an estimate of the worst-case behavior.

See Fig. 8 for an example depicting the results of 480 simulations (20 per each number of agents between 1 and 12), each on a graph with 500 vertices that was created by first generating a random tree and then adding randomly edges until a density of 0.5% is reached. The cover times are the average over 20 simulation runs for each case. It can be seen that as the number of agents grows, the cover time decreases up to an asymptotic point where more agents do not help much. The exact location of this point clearly depends on the number of vertices n; as n grows, the maximum useful number of agents increases. It also seems to depend on p; as the uncertainty p grows, the advantage of additional agents is more significant. Note that even networks with high fault ratios (e.g. $p = 0.75$) can be covered as fast as a faultless network if the number of agents is increased (for $p = 0.75$, three agents are needed).

5. VAW **on trees**

We shall now consider a special case, where the graph G is a static tree, that is it is connected but free of cycles.

Lemma 4. *Assume that* T *is a tree in* G *(that is, a subset of* V *such that all edges between them make a tree) and* T *is connected to* $G \setminus T$ *by a single edge* $e = (x, y)$, *with* $x \in G \setminus T$ and $y \in T$ *(See Fig. 9). Then the edge* e, *once traversed from* x *to* y, *is not traversed again before* T *is completely covered.*

Proof. By induction on r, the number of vertices in T. If $r = 1$ the lemma is obviously true. Now let us assume its correctness for all trees of size r or smaller, and consider a tree of size $r + 1$. Denote by $v_1, v_2, \ldots, v_p$ and $T_1, T_2, \ldots, T_p$ the neighbors of y in T and their respective subtrees, respectively. Note that the size of each subtree is less than or equal to r. According to our assumption, once the ant has moved from y to, say v_i, it will not get back to y before it has covered the whole subtree T_i. Once the ant has returned to y, it will go to another not-yet-visited neighbor, and so on. Clearly, the ant will not travel back from y to x before all y’s neighbors (and the respective subtrees) have been visited, and the lemma follows.□

Corollary 2. *A tree over* n *vertices is covered by a* VAW *process in at most* $2(n - 1)$ *steps.*

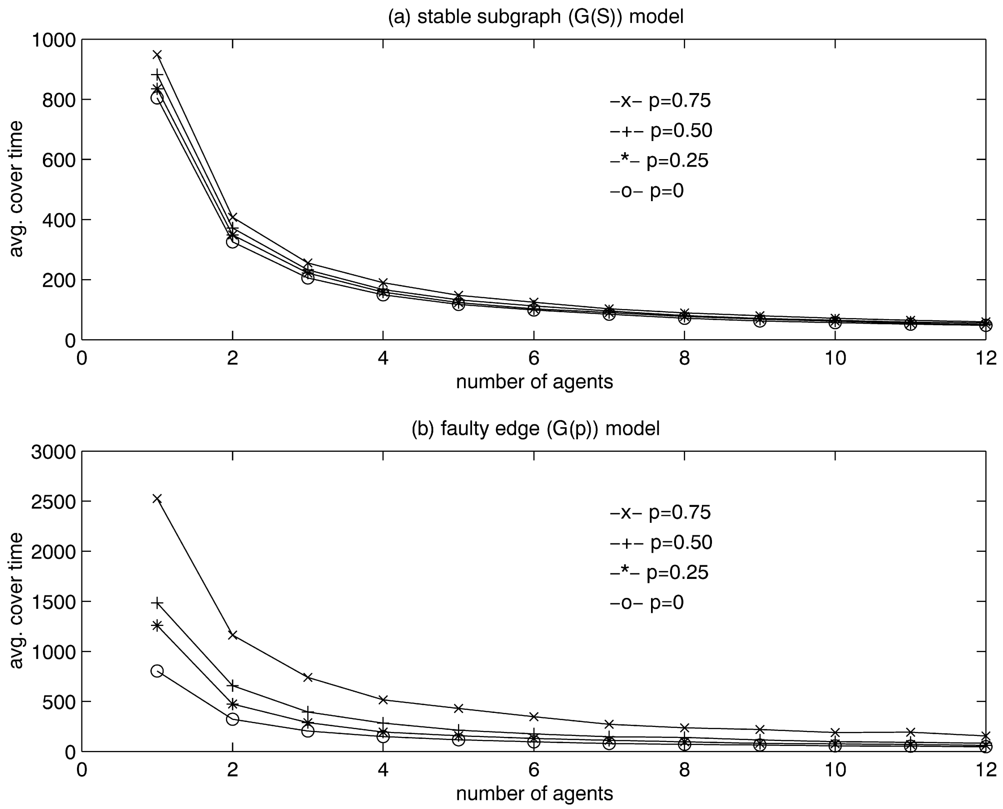


Fig. 8. Cover times by multiple agents that apply the VAW rule on the stable subgraph model (top) and the faulty edge model (bottom). Times are plotted against the number of agents, and parameterized according to edge-failure probability p. For each number of agents and failure probability, 20 experiments were made, each on a graph with 500 vertices that was created by a random tree and additional random edges with 0.5% density.

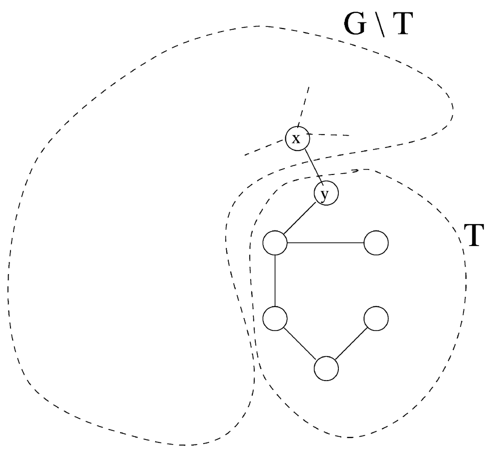


Fig. 9. The tree T is connected to the rest of the graph, $G \setminus T$, by the edge (x, y).

Proof:. Using Lemma 4, one can show that no edge in T is traversed more than twice before covering the whole tree. Since T is a tree, it has exactly $n-1$ edges and the lemma follows. □

Note that the VAW process on a tree evolves in the same way as a depth-first search (DFS) process on the same tree.

6. Summary and open questions

The Vertex Ant Walk, a trace oriented process, was presented and shown to be able to cover a dynamic graph in a bounded time. Two models of connected dynamic graphs were presented. The first model assumes that some spanning subgraph S is always functional, in which case the VAW process is guaranteed to cover the graph within time nd_s, where $n = |V(G)|$, and $d_s = \mathrm{diam}(S)$. In the second model we assume that each edge is functional with probability $1 - p$ and may fail with probability p, independently of all other edges. For the latter case we have proved that a slight modification of VAW results in an upper bound on the expected cover time: $\bar{t}(p) \leq nd(\log \Delta / \log(1/p) + (1+p)/(1-p))$, where d is the diameter of G and Δ is the maximum vertex degree in G.

There are various questions that are open for further research.

1. Our proof of coverage by the VAW process seems to rely on the specific rule of motion. Can similar results be achieved for a more relaxed protocol, for example, if $\sigma(u)$ is *always* increased by one upon visiting u, no matter what its situation relative to the neighbors is? So far we could not prove for this protocol anything better than the exponential upper bound $t_c \leq \Delta^d$, Δ being the maximum vertex degree in G. On the other hand, we do not have any example that does achieve this high bound, and simulations seem to yield much lower times.
2. One problem with a large number of agents in a network is that they may cause an overwhelming amount of traffic that can overload the network and disturb its regular use. A possible cure may be to put a limit on the life time of an agent, or on the allowed rate of agents launching from any point in the network. This bound, however, will clearly increase the cover time, or may even totally impede covering. Hence, further analysis and simulations are needed for understanding the behavior of such mortal agents (as opposed to the immortal model of the current paper), depending on the network parameters.
3. Our analysis holds for a single ant, and, as can be seen by the simulations depicted in Fig. 8, it seems reasonable to say that if the graph is large, more ants will cover it faster. Is it possible to analytically quantify the speed-up ratio as a function of the size of the colony and the structure of the graph?

Acknowledgements

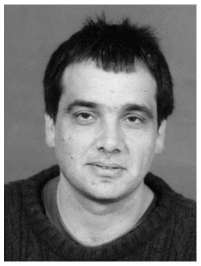
We thank Marco Dorigo, Gianni Di Caro, Thomas Stützle and the anonymous referees for their careful reading and helpful comments. We also wish to thank our alert students in the MultiRobotics class at the Technion for their help in debugging the paper.

References

[1] F.R. Adler, D.M. Gordon, Information collection and spread by networks of patrolling ants, The Am. Naturalist 140 (3) (1992) 373–400.
[2] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovasz, C. Rakoff, Random walks, universal traversal sequences, and the complexity of maze problems, Proc. FOCS '79 (1979) 218–223.
[3] G. Barnes, U. Feige, Short random walks on graphs, SIAM J. Disc. Math. 9 (1) (1996) 19–28.
[4] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, Oxford, 1999.
[5] A.Z. Broder, A.R. Karlin, P. Raghavan, E. Upfal, Trading space for time in undirected $s - t$ connectivity, SIAM J. Comput. 23 (2) (1994) 324–334.
[6] D. Costa, A. Hertz, Ants can colour graphs, J. Operat. Res. Soc. 48 (1997) 295–305.
[7] G. Di Caro, M. Dorigo, AntNet: distributed stigmergetic control for communications networks, J. Artificial Intelligence Res. (JAIR) 9 (1998) 317–365.
[8] M. Dorigo, G. Di Caro, The ant colony optimization meta-heuristic, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, New York, 1999, pp. 11–32.
[9] M. Dorigo, G. Di Caro, L. M. Gambardella, Ant algorithms for discrete optimization, Artificial Life 5 (2) (1999) 137–172.
[10] M. Dorigo (Ed.), The ant colony optimization WWW page. An introduction and compendium of examples, publications and events. Maintained at IRIDIA, Université Libre de Bruxelles, Belgium. http://iridia.ulb.ac.be/+.1667emaã mdorigo/ACO/ACO.html
[11] M. Dorigo, V. Maniezzo, A. Colorni, The ant system: optimization by a Colony of cooperating agents, IEEE Trans. Syst. Man Cybernetics Part B 26 (1996) 29–41.
[12] S. Even, Graph Algorithms, Computer Science Press, Rockville, Maryland, 1979.
[13] A.S. Fraenkel, Economic traversal of labyrinths, Math. Mag. 43 (1970) 125–130, and a correction in 44 (1971) 12.
[14] M. Frauenfelder, The Future of Search Engines, The Industry Standard, 25 September 1998. http://www.thestandard.com/articles/article_print/0,1454,1826,00.html
[15] S. Gal, E.J. Anderson, Search in a maze, in: Probability in the Engineering and Informational Sciences, Vol. 4, Cambridge University Press, Cambridge, 1990, pp. 311–318.
[16] D. M. Gordon, The expandable network of ant exploration, Animal Behaviour 50 (1995) 995–1007.
[17] J. Hopcroft, R. Tarjan, Efficient algorithms for graph manipulation (Algorithm 447), Comm. ACM 16 (6) (1973) 372–378.

[18] R.E. Korf, Real-time heuristic search, Artificial Intelligence 42 (1990) 189–211.
[19] S. Koenig, Y. Smirnov, Graph learning with a nearest neighbor approach, Proc. COLT '96 (1996) 19–28.
[20] S. Lawrence, C.L. Giles, Searching the World Wide Web, Science 280 (5360) (1998) 98–100.
[21] R. Schoonderwoerd, O. Holland, J. Bruten, L. Rothkrantz, Ant-based load balancing in telecommunications networks, Adaptive Behav. 5 (2) (1997) 169–207.
[22] T. Stützle, M. Dorigo, ACO algorithms for the quadratic assignment problem, in: D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, New York, 1999, pp. 33–50.
[23] R. Tarjan, Depth-first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.
[24] G. Tarry, Le probleme des labyrinths, Nouvelles Annales de Mathematiques 14 (1895) 187.
[25] S. Thrun, The role of exploration in learning control, in: Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches, Van Nostrand Reinhold, Florence, Kentucky, 1992.
[26] S. Thrun, Efficient Exploration in Reinforcement Learning, Carnegie-Mellon University, Pittsburgh, Pennsylvania, Tech. Rep. CMU-CS-92-102.
[27] I.A. Wagner, M. Lindenbaum, A.M. Bruckstein, Smell as a computational resource — a lesson we can learn from the ant, Proc. ISTCS '96 219–230. http://www.cs.technion.ac.il/∼wagner
[28] I.A. Wagner, M. Lindenbaum, A.M. Bruckstein, Distributed covering by ant-robots using evaporating traces, IEEE Trans. Robotics Automation 15 (5) (1999) 918–933.
[29] I.A. Wagner, M. Lindenbaum, A.M. Bruckstein, Efficient graph search by a smell-oriented vertex process, Ann. Math. Artificial Intelligence 24 (1998) 211–223.

Israel A. Wanger was born in Isreal, in 1960. He received his B.Sc. degree in computer engineering from the Technion, Israel Institute of Technology, Haifa, in 1987, cum laude, an M.Sc. degree in computer science from the Hebrew University, Jerusalem, Israel, in 1990, cum laude, and a Ph.D. degree in computer sceince from the Technion in 1999. He was a research engineer in General Microwave, Jerusalem, grom 1987 until 1990, when he joined the IBM Haifa Research Laboratory as a staff member. Dr. Wagner is currently an adjunct lecturer in the Computer Science Department at the Technion. His research interests include multiagent robotics, manual and automatic VLSI design, computational geometry and graph theory. He is a member of MAA and AMS.



Michael Lindenbaum was born in Israel in 1956. He received his B.Sc., M.Sc. and D.Sc. in the Department of Electrical Engineering at the Technion, Israel, 1978, 1987 and 1990, repectively. From 1978 to 1985 he served in the IDF. He did his post-doc at the NTT Basic Research Labs in Tokyo, Japan, and from October 1991, he is with the Department of Computer Science, Technion. His main research interest in Computer Vision, and especially statistical analysis of object recognition and grouping processes.

Alfred M. Bruckstein was born in Transylvania, Romania, on January 24, 1954. He received the B.Sc. and M.Sc. degrees in Electrical Engineering, from the Technion, Israel Institute of Technology, in 1977 and 1980, respectively, and the Ph.D. degree in Electrical Engineering from Stanford University, Stanford, CA, in 1984. Since October 1984, he is the faculty at the Technion, Haifa, Israel, where he presently holds the Ollendorff Professorship of Sciences in the Computer Science Department. He is a frequent visitor at Bell Laboratories, Lucent Technologies, in Murray Hill, NJ. Professor Bruckstein's research interests are Image Analysis and Processing, Pattern Recognition, Robotics and Ants, and Computer Graphics. He has also done work in Estimation Theory, Signal Processing, Algorithmic Aspects of Inverse Scattering, Point Processes and Mathematical Models in Neurophysiology. Professor Bruckstein is a member of SIAM, MAA and AMS. He enjoys drawing and is an amateur logo designer.