The Design and Analysis of an Efficient Local Algorithm for Coverage and Exploration Based on Sensor Network Deployment

Maxim A. Batalin, *Member, IEEE*, and Gaurav S. Sukhatme, *Senior Member, IEEE*

***Abstract*—We present the design and theoretical analysis of a novel algorithm termed least recently visited (LRV). LRV efficiently and simultaneously solves the problems of coverage, exploration, and sensor network deployment. The basic premise behind the algorithm is that a robot carries network nodes as a payload, and in the process of moving around, emplaces the nodes into the environment based on certain local criteria. In turn, the nodes emit navigation directions for the robot as it goes by. Nodes recommend directions least recently visited by the robot, hence, the name LRV. We formally establish the following two properties: 1) LRV is complete on graphs and 2) LRV is optimal on trees. We present experimental conjectures for LRV on regular square and cube lattice graphs and compare its performance empirically to other graph exploration algorithms. We study the effects of the *order* of the exploration and show on a square lattice that with an appropriately chosen *order*, LRV performs optimally. Finally, we discuss the implementation of LRV in simulation and in real hardware.**

***Index Terms*—Coverage, deployment, exploration, mobile robots, sensor network.**

I. Introduction

THE COVERAGE problem has been defined [3] as the maximization of the total area covered by robot's sensors. The *static* coverage problem is addressed by algorithms [4]–[6] which are designed to deploy robot(s) in a static configuration, such that every point in the environment is under the robots' sensor shadow (i.e., covered) at every instant of time. For complete static coverage of an environment, the robot group should be larger than a critical size (depending on environment size, complexity, and robot sensor ranges). Determining the critical number is difficult or impossible if the environment is unknown *a priori*. *Dynamic* coverage, on the other hand, is addressed by algorithms which explore and hence "cover" the environment with constant motion and neither settle to a particular configuration [7], nor necessarily to a particular pattern of traversal.

Manuscript received December 14, 2005; revised October 6, 2006. This paper was recommended for publication by Associate Editor J. Wen and Editor F. Park upon evaluation of the reviewers' comments. This work was supported in part by the National Science Foundation under Grant ANI-0082498, Grant IIS-0133947, Grant EIA-0121141, and Grant CCR-0120778. Portions of this paper have appeared previously in [1] and [2].

The authors are with the University of California, Los Angeles, CA 90095 USA (e-mail: maxim@cens.ucla.edu) and the University of Southern California, Los Angeles, CA 90089 USA (e-mail: gaurav@usc.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TRO.2007.903809

This paper simultaneously addresses the problems of coverage, exploration, and sensor network deployment via a single algorithm called least recently visited (LRV). LRV is based on a robot which can carry network nodes as payload. As the robot moves, it deposits nodes into the environment based on certain local criteria. These nodes, once placed in the environment, emit navigation directions for the robot as it goes by. Nodes recommend directions least recently visited by the robot, hence the name LRV. In this paper, two formal properties of LRV are established: completeness on graphs and optimality on trees. Experimental conjectures for LRV on regular square and cube lattice graphs are given. We also empirically compare the performance of LRV to other graph exploration algorithms. The effects of the order of the exploration are studied. LRV is shown to perform optimally on a square lattice with an appropriately chosen order. Finally, we discuss the design and implementation of LRV in simulation and in real hardware.

II. Related Work and Assumptions

In this paper, we consider a single robot in a bounded environment whose layout is unknown. The environment is assumed to be large enough, so that complete *static* coverage of the environment is not possible with one robot. The robot must thus continually move in order to observe all points in the environment frequently. In other words, we address the *dynamic* coverage problem with a single robot.

A recent survey of coverage algorithms is provided by Choset [8]. This survey distinguishes between *online* algorithms, in which the map of the environment is not available *a priori*, and *offline* algorithms, in which the map is available (hence, an optimal assignment is possible). Choset [8] further distinguishes between the algorithms based on *approximate cellular decomposition*, where the free space is approximated by a grid of equally spaced cells, and *exact decomposition*, where the free space is exactly partitioned.

Exploration, a problem closely related to coverage, has been extensively studied [9], [10]. The frontier-based approach [9] concerns itself with incrementally constructing a global occupancy map of the environment. The map is analyzed to locate the "frontiers" between the free and unknown space. Exploration proceeds in the direction of the closest "frontier." The multi-robot version of the same problem was addressed in [11].

Our algorithm differs from these approaches in a number of ways. We use neither a map, nor localization in a shared frame of reference. Our algorithm is based on the deployment of static, communication-enabled, sensor nodes into the environment by

the robot. For purposes of analysis, we treat this collection of sensor nodes as the vertices of a graph even though no explicit adjacency lists are maintained at each node. The graph is thus purely an aid to our *analysis* of the coverage and exploration algorithm, not an entity *used* by the algorithm itself.

qThe problem of exploration using passive nodes (READ-only devices) was considered from the graph theoretic viewpoint in [12] and [13]. In both cases the authors studied the problem of *dynamic* single robot coverage on a graph world. The key result was that the ability to tag a limited number of vertices (in some cases, only one vertex) with unique passive nodes dramatically improved the cover time. We note that [12] and [13] consider the coverage problem, but in the process, also create a topological map of the graph being explored. References [12] and [13] also show that in certain environments exploration is impossible without tagging. There are four key differences between our algorithm and the work reported in [12] and [13].

1) We do not assume the robot can navigate from one node to another in any reliable fashion. The robot does not localize itself, nor has a map of the environment (the structure of the graph corresponding to the environment is not known to the robot, nor does it construct it on the fly).
2) We assume the number of sensor nodes available for dropoff is unlimited; in [12] and [13], a limited number of nodes are used.
3) We assume that each node being dropped off is capable of simple computation and communication—the nodes are active; in [12] and [13], the nodes are passive—they neither compute nor communicate.
4) We do not assume that nodes need to be retrieved; in [12] and [13], retrieval and reuse of nodes by the robot is implied.

Our paper is closely related to the ant robots literature [14]–[20], where the idea of a node with decaying intensity (a semi-active node) is used. The robots sense the change in intensity and are able to change the direction of exploration to cover environment efficiently. Our algorithm differs from these approaches—we assume that each deployed node is capable of sensing, simple computation and communication. We exploit the computation and communication capabilities of the nodes to address problems beyond coverage and exploration.

The nodes we use, act as a support infrastructure which the mobile robot uses to solve the coverage problem efficiently. The robot explores the environment, and based on certain *local criteria*, drops a node into the environment, from time to time. Each node is equipped with sensing, a small processor, and a radio of limited range. The ensemble of nodes forms a sensor network. Our algorithm performs the coverage task successfully using only local sensing and local interactions between the robot and the sensor network.

The problem of coverage and deployment in the sensor network community was considered from a different perspective. For example, [21] considers quality of service of the deployed network, [22] discusses algorithms to achieve low energy deployment. Collaborative target tracking and surveillance is considered in [23] and [24].

Algorithm 1 Least Recently Visited (LRV) Algorithm—Robot Loop

$\mathbf{n}, \mathbf{d}$—current node and suggested direction;
$\mathbf{R}$—set containing data received from nodes in robot's vicinity (node id, signal strength, suggested direction);
SHORT—communication range threshold used to determine when to deploy new nodes;
$\mathbf{Opposite}(\mathbf{d})$—function returning direction opposite to d

$R =$ receive NODE_INFO messages from nodes in vicinity
if out of SHORT communication range with n **then**
  $(n_{\text{closest}}, d_{\text{closest}}) =$ node and corresponding direction in R withlargest signal strength}
  if $n! = \text{NULL}$ **then**
    Send(UPDATE_DIR, n_{closest}, d_{closest})
    Send(UPDATE_DIR, n_{closest}, Opposite(d_{closest}))
  else
    deploy sensor node n' with suggested direction d'
    $(n_{\text{closest}}, d_{\text{closest}}) = (n', d')$
  if no obstacles detected in direction d_{closest}
    $(n, d) = (n_{\text{closest}}, d_{\text{closest}})$
  else
    Send(UPDATE_DIR, n_{closest}, d_{closest})
    Wait for response, repeat the check
if moving and obstacle detected $\leq$ OBSTACLE_AVOIDANCE_RANGE **then**
  if obstacle is large and no nodes in vicinity
    deploy sensor node n' with suggested direction d'
    $(n, d) = (n', d')$
    if obstacles detected in direction d **then**
    Send(UPDATE_DIR, n, d)
    Wait for response, repeat the check
  else
    avoid the obstacle
if $d =$ NULL **then**
  Move in direction d

Algorithm 2 LRV Algorithm—Sensor Node i Loop

$\mathbf{D}(\mathbf{i})$—set of directions incident to node i; $\mathbf{W}(\mathbf{d})$—number of times direction d traversed from this node;
$\mathbf{ANY\_OF}(\mathbf{G})$—function returns member of set G according to arbitrary rule

Repeat:
if received UPDATE_DIR message from robot with direction d_{update} **then**
  $W(d_{\text{update}} = W(d_{\text{update}}) + 1$
Send(NODE_INFO, n, ANY_OF($\arg\min_{\forall\, d \in D(i)} W(d)$))

III. LRV Algorithm

In this section, we present the LRV algorithm for sensor network deployment and maintenance, coverage, and exploration.

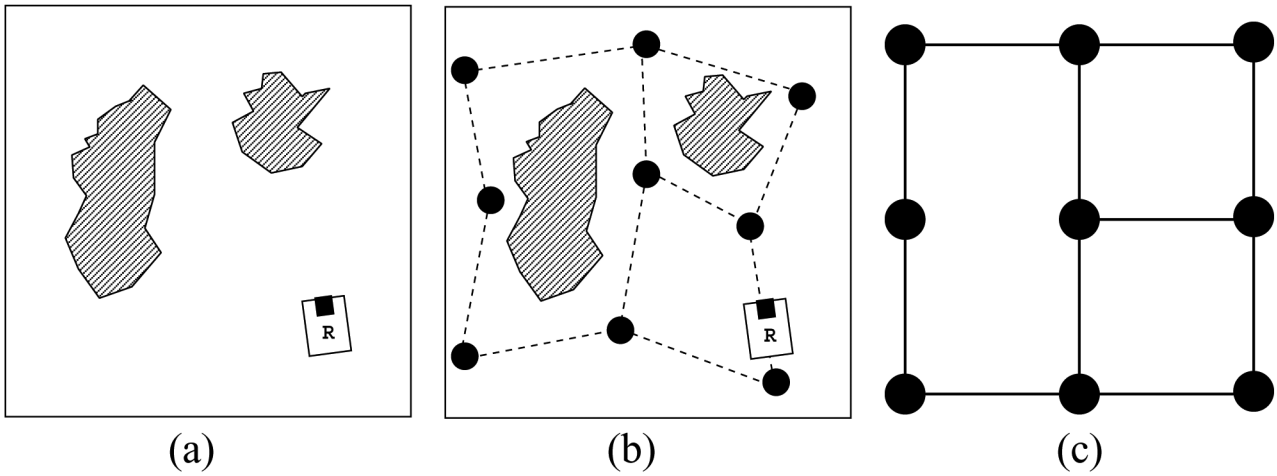


Fig. 1. Modeling the network as a graph. (a) Initial environment. (b) Deployed sensor network and a mobile robot. (c) Graph representation of (a).

As shown in Algorithms 1 and 2, LRV is the concurrent execution of two algorithms—one on a robot (robot loop) and another on every node (sensor node loop). For every node i, let $D(i)$ be the set of directions possible to traverse from i. Then, $\forall\ d \in D(i), W(i,d)$ is the weight (cached on a node) maintaining the number of times d was traversed from i. In some cases, we will refer to the weight of a direction d as $W(d)$ if the node id is implicit. The function $\mathrm{ANY\_OF}(T)$ returns a single element of a set T according to some arbitrary rule (i.e., in order, random, etc).

When a robot is deployed into the environment initially, according to Algorithms 1 and 2, it deploys a node because there is no sensor node within communication range. Over time, LRV causes a network of nodes to be deployed since every time a new node is deployed it must be able to communicate with at least one other sensor node in the network. As shown in Section IX, maintaining network connectivity is the minimal requirement that the deployment function should have.

Once deployed, each sensor node starts to emit the locally least recently visited direction (hence the name LRV), which is one of the directions with smallest weight W (if there are multiple directions of the same weight, one is picked according to the function $\mathrm{ANY\_OF}(T)$ which can be ordered or random). In practice the number of directions per node is often bounded and application dependent. In our experimental work (see Section VI), we set this bound to 4.

Another important aspect of the algorithm is the update of the weight W. The weight of a direction is incremented in two cases: right before a direction is traversed and on the destination node right after a direction is traversed. Suppose the robot is in the vicinity of a node i and is suggested to move in direction d. The weight $W(i,d)$ is incremented right before direction d is traversed. Suppose the robot enters node i through direction d. The weight $W(i,d)$ is incremented right after direction d is traversed.

IV. Graph Model

For the purpose of analysis, consider an open (no obstacles) bounded environment. In this case, given our node deployment algorithm (LRV) described in Section III, we can model the steady-state spatial configuration of the nodes as a finite graph $G = (V, E)$, where V is a set of vertices (the deployed nodes) and E is a set of edges such that $\forall\ i, j \in V$ there is an edge between i and j iff 1) i and j are within communication range and 2) there is a physical path between i and j. Consider the schematic of the environment in Fig. 1(a). We represent the LRV-deployed network in this environment as a graph $G = (V, E)$ [shown in Fig. 1(c)]. A graph model is a natural choice because of its flexibility and ubiquity of usage in such problems.

Before discussing the theoretical properties of LRV, we provide working definitions for coverage and exploration on graphs and corresponding performance metrics.

Definition (Coverage on a Graph) 1: Coverage on a graph is the act of visiting every vertex of a graph.

The performance of a coverage algorithm is measured using the *cover time* [25] defined as follows.

Definition (Cover Time) 2: Cover time is the number of edges traversed such that every vertex of a graph is visited at least once, i.e., the graph is covered.

In order to cover a graph, a robot needs to at least traverse one edge per node (consider a spanning tree of a graph). This notion is distinct from graph exploration or "complete" graph coverage (where the robot needs to traverse every edge of a graph). This latter notion is called graph exploration defined as follows.

Definition (Exploration on a Graph) 3: Exploration on a graph is the act of traversing every edge of the graph.

An exploration algorithm is evaluated using the *exploration time* metric defined as follows.

Definition (Exploration Time) 4: Exploration time is the number of edges traversed such that every edge is traversed at least once.

It follows from the previous definitions that exploration is a superset of coverage. Therefore, Cover Time $=$ O(Exploration Time).

A. LRV on Graphs

Given the graph model, we exhibit two important properties of LRV. First, we show that LRV is *complete*, and second, we establish a relationship between its cover time and exploration

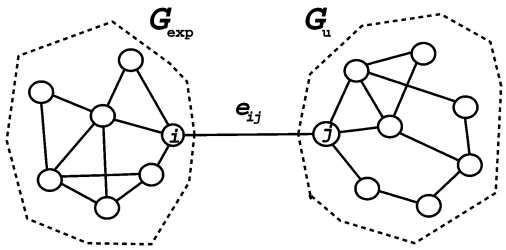


Fig. 2. Illustration for Theorem 1. At every point in time during the execution of LRV a graph $G = G_{\exp} + G_u$, where $G_{\exp}$ is the explored part of G and G_u is an unexplored part. An edge e_{ij} connects $G_{\exp}$ and G_u.

time. For purposes of this analysis, we are interested in the behavior of LRV in the "steady state" when all nodes have been deployed. In this special case, one can consider a simple version of LRV on a graph as follows. For every vertex i, $E(i)$ is the set of edges incident to i. For clarity, we identify an edge in $E(i)$ with the node this edge connects node i to. Then, $\forall\, k \in E(i) : W(i,k)$ is the weight (cached on node i) maintaining the number of times edge (i,k) was traversed from i. Note that in the general case of LRV, described in Algorithms 1 and 2, the weight of an edge is incremented twice: before and after traversal, but on different nodes. This redundancy is required for practical purposes: the weights are cached on nodes and since the environment is dynamic, sensing and actuation are noisy, starting at the same node and traversing the same direction at different points in time does not guarantee that robot would arrive at the same node. In the graph model, we study the steady-state spatial configuration of the nodes on a finite unchanging graph. Hence, for clarity of presentation, $\forall\, i,j \in V$ consider storing the weight associated with $i \to j$ transition on the edge $e_{i,j} \in E$. Note that an edge e_{ij} is undirected. This weight is identical to the one associated with $j \to i$ transition (e.g., $W(i,j) = W(j,i)$). We increment the weight just in one case: right before an edge is traversed and associate it with the edge $e_{i,j} \in E$.

Algorithm 3 shows this simplified version of LRV on a graph. Note that the deployment function is removed since we are in the steady state.

Algorithm 3 LRV Algorithm on a Graph

n—current node; the node the robot is at
n'—next node; the node the robot transitions to

while Covered/Explored the graph = FALSE **do**
  $n' = \text{ANY\_OF}(\arg\min_{\forall\, j \in E(n)} W(n,j))$
  $W(n,n') := W(n,n') + 1$
  $n := n'$

We now state and prove two results: completeness of LRV on finite graphs and the relationship of cover time to exploration time.

Theorem 1 (Completeness): The exploration time of LRV on a finite graph is finite.

Proof: The goal is to show that LRV traverses every edge of any finite graph in finite time. The proof is by contradiction. Suppose the exploration time of LRV is infinite. Therefore, there is a time t after which LRV traverses only those edges that it traverses infinitely many times (edges of the graph $G_{\exp}$ in Fig. 2). Weights of these edges grow without bound, including the edge that is considered for traversal infinitely many times but is never picked after time t (edge e_{ij} in Fig. 2). By definition, LRV will be forced to traverse this edge after time t, which is a contradiction.

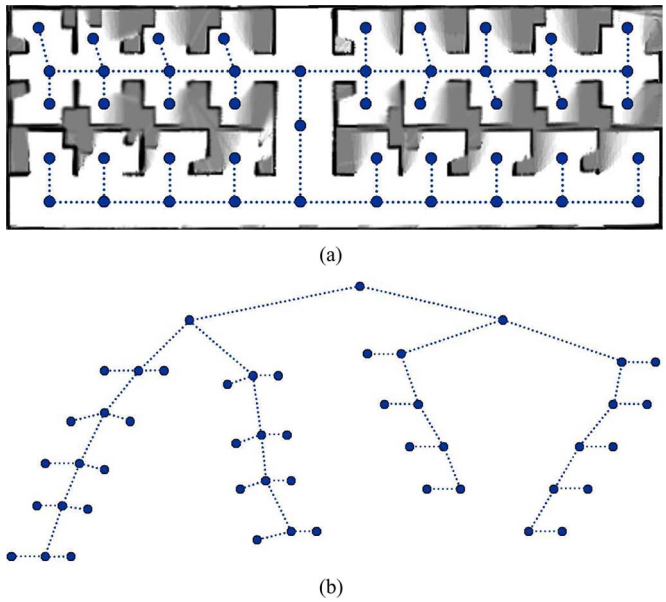


Fig. 3. Graph as a tree. (a) A map of the environment with an embedded sensor network with a tree-like topology. (b) Tree isomorphic to embedded sensor network topology of (a).

Definition (Degree of a Vertex) 5: The number of edges incident to a vertex is called degree and is denoted as $\deg(v)$.

Theorem 2: For a graph $G = (V,E)$ with maximum degree $d = \max_{v \in V}(\deg(v))$, if Cover Time $= O(f(V))$, then Exploration Time $= O(d * f(V))$.

Proof: Suppose LRV executes on a graph G until every vertex is visited at least once. It is obvious that at least one edge per vertex is traversed. Thus, after the first execution of the algorithm, the number of untraversed edges at every vertex is at most $d - 1$. Note, that at a given vertex, while there are untraversed edges, LRV will choose one arbitrarily. Hence, after at most d executions of the algorithm every vertex would be covered and every edge would be traversed. Thus, if Cover Time $= O(f(V))$, then Exploration Time $= d * O(f(V))$.

B. LRV on a Tree

In this section, we study the performance of LRV on trees. Fig. 3(a) shows a map of the environment with an embedded sensor network. The sensor network has a tree-like topology. Fig. 3(b) shows the tree which represents the embedding. A tree differs from other graphs in two major ways: 1) the vertex degree is not bounded and 2) a tree does not contain cycles. The next two Lemmas establish local properties of LRV needed for the main result of this section: performance of LRV on trees is linear or asymptotically optimal.

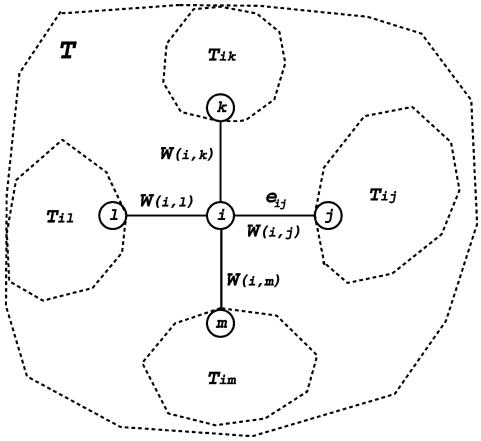


Fig. 4. Illustration for Lemma 1. A tree T at a point of time when a robot enters vertex v_i through an incoming edge e_{ij}.

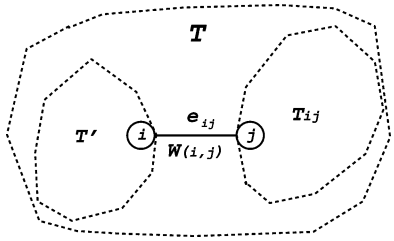


Fig. 5. Illustration for Lemma 2. A tree T at a point of time when a robot enters vertex v_i of an unexplored subtree T' through an incoming edge e_{ij}.

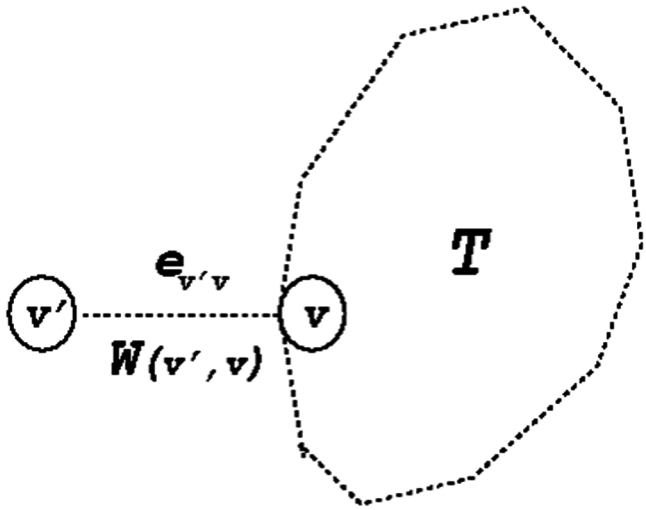


Fig. 6. Illustration for Theorem 3. A tree T augmented with a vertex v' and an edge $e_{v'v}$ connecting v' to vertex $v \in T$.

Definition (Incoming Edge) 6: For a vertex v_i of a graph G, the incident to v_i edge e is called an incoming edge if it is the first edge traversed to enter vertex v_i.

Lemma 1: In a tree T, an incoming edge e_{ij} is traversed twice iff every other edge incident to v_i is traversed twice.

Proof: Consider a tree T. Initially, the weights of all edges of the tree are zero. Suppose a robot enters vertex v_i through an incoming edge e_{ij} (refer to Fig. 4). The weight of e_{ij} is incremented and equal 1, whereas the weights of other edges incident to v_i are 0.

Next, LRV picks one of the 0-weighted edges, say e_{ik}, and traverses it. The weight of e_{ik} is incremented and equal to 1, the weight of e_{ij} is equal to 1 and the weights of other edges incident to v_i are 0. Due to the completeness theorem, eventually the robot returns back to v_i by traversing an edge e_{ik}. The weight of e_{ik} is incremented and equals 2, the weight of e_{ij} is equal to 1, and the weights of other edges incident to v_i are 0.

Apply the same reasoning to every other 0-weighted edge incident to v_i. The weight of e_{ij} is equal to 1, whereas the weights of other edges incident to v_i are 2. At this point, LRV is forced to pick e_{ij} as the only edge of minimum weight incident to v_i. Hence, an incoming edge e_{ij} is traversed twice iff every other edge incident to v_i is traversed twice.

It follows from Lemma 1 that if before traversing an edge e_{ij}, the weights of all edges incident to v_i are equal (initially all 0), then after an incoming edge e_{ij} is traversed twice the weights of all edges incident to v_i are equal and incremented by two.

Lemma 2: An incoming edge e_{ij} is traversed twice iff in a subtree $T' = T - (T_{ij} + e_{ij})$ every edge is traversed twice.

Proof: Consider a subtree T' (refer to Fig. 5). LRV starts at vertex v_i. Applying Lemma 1 to v_i results in every edge incident to v_i traversed twice. Applying Lemma 1 recursively to every vertex of T' results in every incident to every vertex edge traversed twice. Hence, an incoming edge e_{ij} is traversed twice iff in a subtree $T' = T - (T_{ij} + e_{ij})$ every edge is traversed twice.

Using the results of Lemmas 1 and 2, we can prove the main result of this section, stated as Theorem 3.

Theorem 3: The exploration time of LRV on a tree is no more than $2|E|$.

Proof: Consider a tree T (refer to Fig. 6). Augment T with a vertex v' and an edge $e_{v'v}$ connecting v' to vertex $v \in T$. Consider LRV on the augmented tree starting at v'. It follows from Lemma 2 that the robot executing LRV would traverse $e_{v'v}$ twice when in tree T every edge is traversed twice. Hence, the exploration time of LRV on a tree is no more than $2|E|$.

Theorem 3 asserts that the performance of LRV on trees is linear or asymptotically optimal. Please note that the proof of Theorem 3 is general for any vertex of a tree as a starting vertex. An addition of a vertex v' and an edge $e_{v'v}$ is used purely for the purposes of a proof. This new vertex v' is attached to a vertex v which can be any vertex of a tree T.

C. LRV on a Square Lattice: Empirical Results From Simulation

In this section, we consider the performance of LRV on the following special graph G.

1) G is undirected.
2) G has degree $\deg_G \leq 4$. If all nodes have degree 4, then G is a square lattice, i.e., a regular graph of degree 4.
3) $|V| = \Theta(|E|)$.

We consider this special graph because in practical implementations of LRV, a physical compass on the sensor node determines direction. If this compass has k bits of resolution, then each node is capable of identifying 2^k directions resulting in a graph of degree $\leq 2^k$. As we shall see in Section VII, our experiments were all done with $k = 2$, resulting in square lattice-like

deployments. Hence, we analyze LRV on a square lattice. It has been shown [25] that the *cover time* of a random walk (RW) on a regular graph with V vertices is bounded below by $V \ln V$ and above by $2V^2$. If we assume that passive nodes can be used, and the graph $G = (V, E)$ is known (a topological map is available) and the robot can drop nodes of three independent colors, then the problem of coverage can be solved optimally by applying depth-first search (DFS) which is linear in V. DFS assumes that all resources are available—nodes, map, localization, and perfect navigation.

In [12], the problem of coverage is considered in the context of mapping a graph-like environment with V vertices. Their algorithm explores the environment and constructs a topological map on the fly. The assumptions of the algorithm are that the robot has $k(k < V)$ nodes, and perfect localization and navigation within the graph. The *cover time* of their algorithm is bounded by $O(V^2)$. It is important to note that the problem addressed in [12] is more complex than simple coverage, since they build a map while exploring. We have conducted simulation experiments running RW, DFS, and LRV on graphs with $V \in [100^2, 1000^2]$ nodes. For every experiment, the steady state (or recurrent[1]) cover time is reported. For all of the algorithms, the tie breaks are resolved randomly. In the case of LRV or 1-LRTA*, it implies the implementation of $\mathrm{ANY\_OF}(T)$ as a random function. The results of the experiments are shown in Fig. 7. Fig. 7 also shows the $n \log n$ curve for reference. These experiments lead us to the following.

1) Conjecture 1: The asymptotic cover time of LRV is $O(V^{1+\epsilon})$.

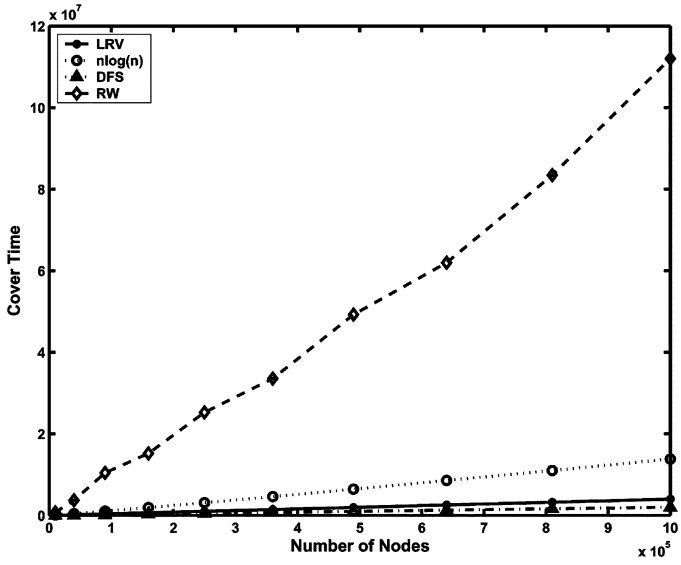
From Fig. 7, it is clear that cover time of LRV is less than $V \ln V$, however, the bound is not tight. In order to establish a tighter bound and validate Conjecture 1, we use the following method.

2) Method 1 (Establishing an Empirical Asymptotic Bound): Assuming that the function representing LRV is monotonic we can analyze the sequence $(\mathrm{LRV}(i)/f(i))$ on the subject of increase (asymptotically $\mathrm{LRV}(i) = \Omega(f(i))$), decrease (asymptotically $\mathrm{LRV}(i) = O(f(i))$), or does not change (asymptotically $\mathrm{LRV}(i) = \Theta(f(i))$).

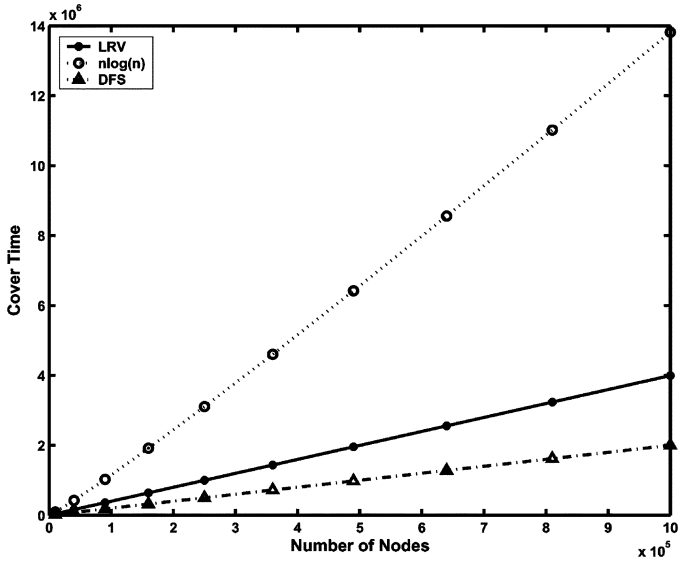
Using Method 1, we can establish an upper bound for LRV. The number set at the bottom of the page shows the sequence for $i \in [100^2, 1500^2]$ nodes and $f(i) = i^{1.0005}$, $(\mathrm{LRV}(i)/i^{1.0005})$. Note that the sequence increases initially, then stabilizes at value 3.9685 for $i \in [900^2, 1000^2]$, and finally decreases. It follows that in this example $\epsilon \leq 5 * 10^{-4}$ and asymptotic cover time of LRV is $O(V^{1.0005})$. Hence, this supports Conjecture 1 that the asymptotic cover time of LRV is $O(V^{1+\epsilon})$.

Using Theorem 2 and Conjecture 1, we get the following result for a square lattice.

[1]Steady state or recurrent cover time is taken when the cover time does not change significantly from coverage to coverage.



(a)



(b)

Fig. 7. Comparison of graph coverage algorithms. The $n \ln n$ curve is shown for reference. (a) Comparison of *Cover Time* for DFS, RW, and LRV. The $n \ln n$ curve is shown for reference. (b) A comparison between DFS and LRV. This graph is a magnified view of (a).

3) Conjecture 2: LRV explores the environment in asymptotic time $O(V^{1+\epsilon})$.

Let us consider different tradeoffs between the previously mentioned techniques. As mentioned earlier, the clear performance boundaries for the coverage task are given by RW (upper) and DFS (lower). The more interesting comparisons are between LRV and DFS and our algorithm and an algorithm with a limited number of passive node markers [12].

$$[3.9406 \quad 3.9587 \quad 3.9639 \quad 3.9661 \quad 3.9672 \quad 3.9678 \quad 3.9682 \quad 3.9684 \quad 3.9685 \quad 3.9685 \quad \ldots \quad 3.9682]$$

Fig. 7(b) shows that the asymptotic performance of our algorithm is similar to DFS. Note that in order to determine the identity of neighboring vertices and to navigate perfectly from node to node, DFS assumes that a map of the environment is available and that the robot is perfectly localized. Our algorithm, on the other hand, does not have access to global information and the robot does not localize itself. The nodes used in our algorithm are more complicated than those used in DFS and the cover times are asymptotically somewhat larger than the cover times of DFS.

In [12], the algorithm builds a topological map of the environment and assumes perfect navigation (and thus, localization) on the graph. The node *markers* are very simple (the only function is to mark the vertex) and the robot cannot differentiate between them. In addition, the algorithm assumes that there exists a local enumeration of edges. The cover time of this algorithm, however, is bounded by $O(V^2)$. Our algorithm, on the other hand, does not have a map and the robot does not localize itself. Another important difference is that we assume that the number of nodes available to us is equal to the number of vertices. In addition, the nodes used in our algorithm are more complex, since they keep a certain state per direction and are uniquely identifiable. The cover time of our algorithm, however, is conjectured to be less than $V^{1+\epsilon}$. Thus, the apparent tradeoff is using a large number of "smart" nodes (and no global information or localization) versus a limited number of simple nodes (with mapping and partial localization within the graph). The cover time achieved by our algorithm is clearly better. However, if the nodes are a precious resource, the algorithm described in [12] would be preferred.

Another algorithm we compare LRV to is 1-LRTA* [26]. 1-LRTA* is a well known graph search algorithm that can be applied to graph coverage. Algorithm 4 shows the details of 1-LRTA*. In 1-LRTA*, a weight is associated with a node. The edge to traverse is chosen based on weights of neighboring nodes. The weight of a node is incremented with the weight of a node the robot transitions to. Hence, 1-LRTA* requires nodes to communicate.

Algorithm 4 1-LRTA* Algorithm on a Graph

n—current node; the node the robot is at
n'—next node; the node the robot transitions to

while Covered/Explored the graph = FALSE **do**
  $n' = \text{ANY\_OF}(\arg\min_{\forall\, j \in E(n)} W(j))$
  $W(n) := W(n') + 1$
  $n := n'$

Fig. 8 shows that generally 1-LRTA* outperforms LRV. However, it should be noted that LRV is a deployment and exploration algorithm, whereas 1-LRTA* is a graph exploration algorithm which assumes the graph is given. Moreover, 1-LRTA* uses information obtained from the neighbors via communication, whereas LRV is purely local.

Finally, lets examine how fast LRV, 1-LTRA*, and RW converge to a full coverage. Fig. 9 shows a comparison of the convergence speed for LRV, 1-LRTA*, and RW on a 1000^2 regular

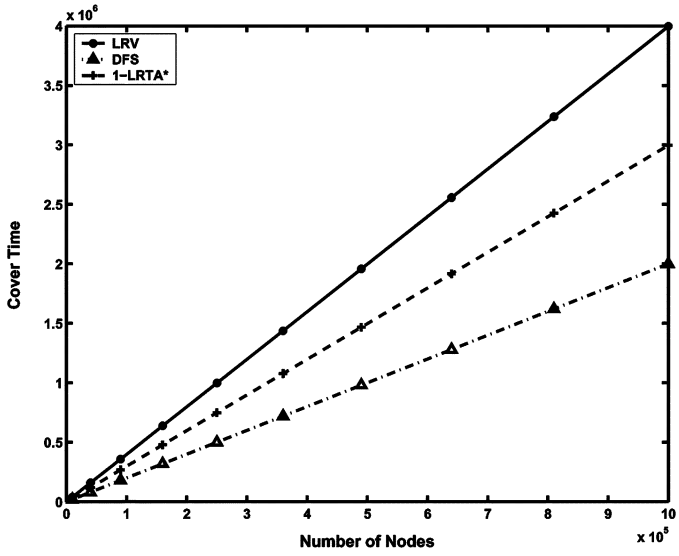


Fig. 8. Comparison of *Cover time* of DFS, 1-LRTA*, and LRV.

square lattice. In other words, Fig. 9 shows the percentage of the whole graph covered as the algorithms progress (i.e., visit more and more nodes or as more actions are taken). Fig. 9 (top) shows a linear convergence of LRV and 1-LRTA*, whereas, RW on the interval of 50%–100% of complete coverage [see Fig. 9 (bottom)] exhibits a nonlinear increase in running time. The curve for DFS is not included, since the convergence factor is optimal and the cover time is at most twice the number of nodes.

D. LRV on a Cube Lattice or LRV in 3-D

In Section IV-C, we considered an application of LRV to the problem of coverage and exploration on a plane. In the planar case, the graph representing the network embedding is a square lattice. In this section, we extend discussion of LRV performance to a general 3-D case. In 3-D case, the graph representation of the deployed sensor network is a graph G.

1) G is undirected.
2) G has degree $\deg_G \leq 6$. If all nodes have degree 6, then G is a cube lattice, i.e., a regular graph of degree 6.
3) $|V| = \Theta(|E|)$.

Fig. 10 shows an example of a cube lattice which represents possible network embedding in a real 3-D environment. Note that in comparison with a square lattice representation in the planar case, a general cube lattice has two more edges incident to every node. At the same time, we will show that the performance of LRV scales similarly with the size of the graph.

We have conducted simulation experiments running LRV and 1-LRTA* on graphs with $V \in [25^3, 115^3]$ nodes. For every experiment the steady state (or recurrent) cover time is reported. The tie breaks are resolved randomly (e.g., $\text{ANY\_OF}(T)$ is a random function). The results of the experiments are shown in Fig. 11. As shown on Fig. 11, 1-LRTA* outperforms LRV (as is the case in the planar case), however, it is also more sensitive to the initial conditions (various starting points).

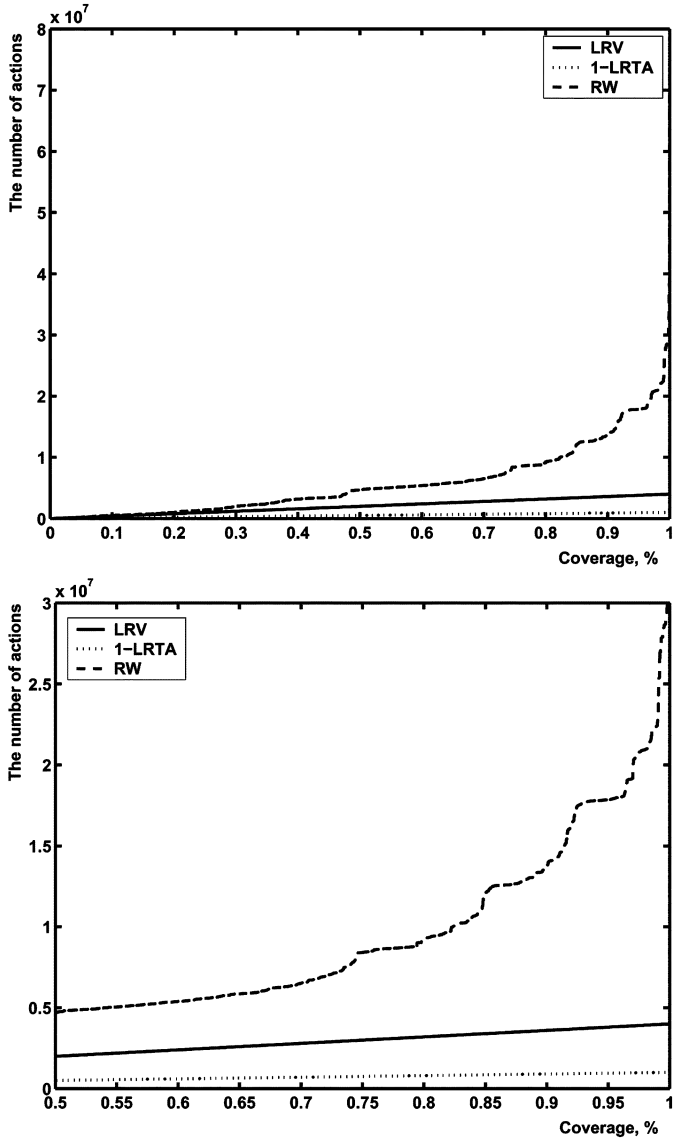


Fig. 9. Comparison of the convergence speed for LRV, 1-LRTA*, and RW. (a) On the interval 0%–100%. (b) Magnified view of (a), interval 50%–100%.

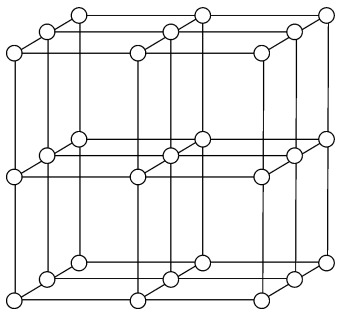


Fig. 10. Cube lattice representing possible embedding of a sensor network into 3-D environment.

Using Method 1, we can establish an upper bound for LRV on a cube lattice. The following is the sequence for $i \in [25^3, 115^3]$ nodes and $f(i) = i^{1.005}$, $(\mathrm{LRV}(i)/i^{1.005})$:

$$[5.3757 \quad 5.5034 \quad 5.5304 \quad 5.5330 \quad 5.5321\,].$$

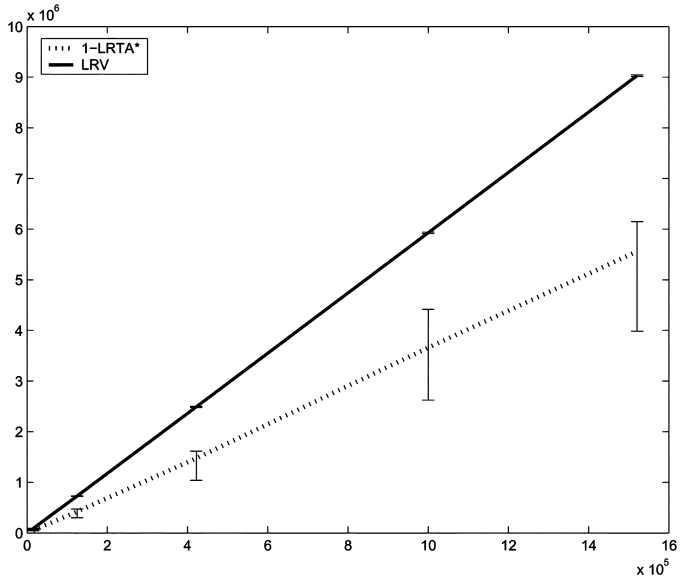


Fig. 11. Comparison of *Cover time* of 1-LRTA* and LRV on a cube lattice.

Note that the sequence increases to the value 5.5330 $(i = 100^3)$, and then decreases. It follows that in this example $\epsilon \leq 5 * 10^{-3}$ and asymptotic cover time of LRV is $O(V^{1.005})$. Hence, Conjectures 1 and 2 seem to be valid for LRV on cube lattices as well.

V. Exploring in Order

As was shown in previous sections, LRV is different from existing techniques like 1-LRTA*, RW, DFS, etc. A common feature of all these algorithms is that in each case when the algorithm has to pick one of several equally good edges to traverse, traditionally, this edge is determined randomly. In the case of LRV, this choice is dictated by the function $\mathrm{ANY\_OF}(T)$, which returns a single element of a set T according to some arbitrary rule (i.e., in order, random, etc). In Sections IV-C and IV-D we compared LRV with 1-LRTA* and used a random order in $\mathrm{ANY\_OF}(T)$ function.

In this section, we consider the benefits of using ordered rules in the $\mathrm{ANY\_OF}(T)$ function. We constrain our discussion to a square lattice, however, the results can be generalized to the cube lattice as well. Fig. 12(a) shows a node k of a general square lattice. Node k has four incident edges which can be considered for traversal. We label these edges numerically 0–3 as shown in Fig. 12(a). We use this labeling to define the *order rule*.

Definition (Order Rule) 7: An order rule is a sequence of labels identifying the order in which a set of equally good edges should be traversed.

For example, an *order rule* $\{0213\}$ means that if several edges are equally good for traversal (say edges 2 and 3), then an edge appearing first in the *order rule* would be preferred (i.e., edge 2 in this case).

Without loss of generality, assume that the edges of the square lattice are aligned with the XY plane [as in Fig. 12(a)]. Following the labeling of Fig. 12(a), edges 0 and 2 span the Y dimension, whereas dimension X is spanned by edges 1 and

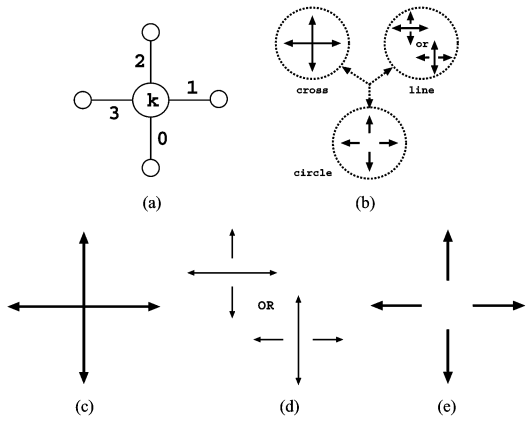


Fig. 12. Edge labeling and four possible traversal orders. (a) A typical node k of a square lattice with four incident edges marked with labels 0 through 3. (b) Random order. An algorithm breaks ties randomly. (c) Cross order. An order where dimensions traversed in order. (d) Line order. An order where one dimension is set to be traversed, then interrupted for complete traversal of another dimension, followed by completion of the traversal of the first dimension. (e) Circle order. An order where half of the two dimensions are traversed in clockwise or counter-clockwise order.

3. Lets define an edge e_k, where k is a dimension drawn from the set $\{X, Y\}$, e_k then is an edge drawn from the set $\{1, 3\}$ if $k = X$ and $\{0, 2\}$ if $k = Y$. Further, define an operator $'$ that takes the remaining value of the set. For example, suppose e_k draws X dimension $(k = X)$ and an edge 1, then k' draws Y dimension and the value of $e_{k'}$ is drawn from the set $\{0, 2\}$. Likewise, e'_k draws X dimension and an edge 3.

In the case of a square lattice, each node can have up to four edges to traverse. Hence, there are 24 *order rules* possible. We subdivide these rules into three classes: cross, line, and circle. Each class consists of eight symmetric *order rules*.

Definition (Cross Order) 8: An order of the form $\{e_k e'_k e_{k'} e'_{k'}\}$ is a cross order.

Fig. 12(c) shows a sketch of a cross order. For example, an *order rule* $\{0213\} \in$ Cross Order class. There are eight symmetrical *order rules* in cross order class, some other examples are $\{2031\}$, $\{1302\}$, $\{3120\}$, etc.

Definition (Line Order) 9: An order of the form $\{e_k e_{k'} e'_{k'} e'_k\}$ is a line order.

Fig. 12(d) shows a sketch of a line order. For example, *order rules* $\{0132\}, \{2130\}, \{1023\}, \{3201\} \in$ Line Order class. There are eight symmetrical *order rules* in line order class altogether.

Definition (Circle Order) 10: An order of the form $\{e_k e_{k'} e'_k e'_{k'}\}$ is a circle order.

Fig. 12(e) shows a sketch of a circle order. There are eight symmetrical *order rules* in circle order class, some examples are $\{1230\}$, $\{0321\}$, $\{3210\}$, etc.

The three introduced classes of *order rules* define 24 possible orders. Fig. 12(b) shows a sketch of the last kind of *order rules* possible—the traditionally used random order. Random order breaks ties in equally good edges randomly. In case of a square lattice, random order essentially takes form of one of the three previously defined classes every time the choice has to be made. Hence, there are 25 *order rules* total.

We performed an extensive set of experiments running LRV with four classes of *order rules* on a square lattice with 100^2 nodes. For each *order rule*, every node of a graph was tried as a starting point and the corresponding cover time recorded. As a result, Fig. 13(a)–(d) show *cover time maps* for four possible classes of *order rules*: random, cross, line, and circle. The darker shade indicates lower cover time and lighter shade—higher cover time. Note that the correspondence of the shade range to the cover times range is different for each graph. Fig. 13(e) shows a plot of another circle order (1230), which is a variation of a circle order 0123 [see Fig. 13(d)]. Note symmetrical change in the plot.

Fig. 13(f) and (g) show a comparison of cover time statistics for the four possible order classes. As shown in Fig. 13(f) and (g), the worst performance is demonstrated by the circle order (0123), the best by cross order (0213). Random order demonstrates a slightly better performance than the line order (0132).

The main result is that the order in which ties are resolved has tremendous impact on the cover time of LRV. In addition, if we have extra knowledge of where the robot is in the graph (or in the environment, in general), then an appropriate order rule can be chosen to achieve the optimal performance. To better illustrate this idea, consider a square lattice with 100^2 nodes. Suppose LRV starts at the leftmost top corner node (0,0). In this case, LRV should use cross order 0213. Then the resulting cover time is equal to 9999 traversed edges, which is an optimal result for this graph. In general, for the square lattices it is enough to be able to recognize corner nodes in order to apply one of the symmetric cross orders to obtain optimal results with LRV. Note that if we would simply use random order, traditionally used in literature, the optimal cover time is highly unlikely to be obtained.

VI. LRV Implementation Details

In this section, we describe implementation details associated with LRV to obtain a system that functions on real hardware and in simulation. Specifically, we consider a planar coverage and exploration problem with one robot in a bounded environment. Loosely put, in LRV, the task of each node is to recommend a locally preferred direction of movement for the robot within its communication range. Thus, each node acts as a local signpost telling the robot which direction to explore next. In practice, the robot treats this information as a recommendation and combines it with local range sensing (to avoid obstacles) to make a decision about which direction to actually pursue.

As shown in Fig. 14, each node has a state associated with four cardinal directions (South, East, North, West). The choice of four directions is arbitrary. It implies that each node is equipped with a 2-bit compass. For each direction, the node maintains a binary state (T), a counter (C), and block E which might be used for additional information. The state T can be either `OPEN` or `EXPLORED`, signifying whether the particular direction was explored by the robot previously. The counter C associated with each direction stores the time since that particular direction was last explored.

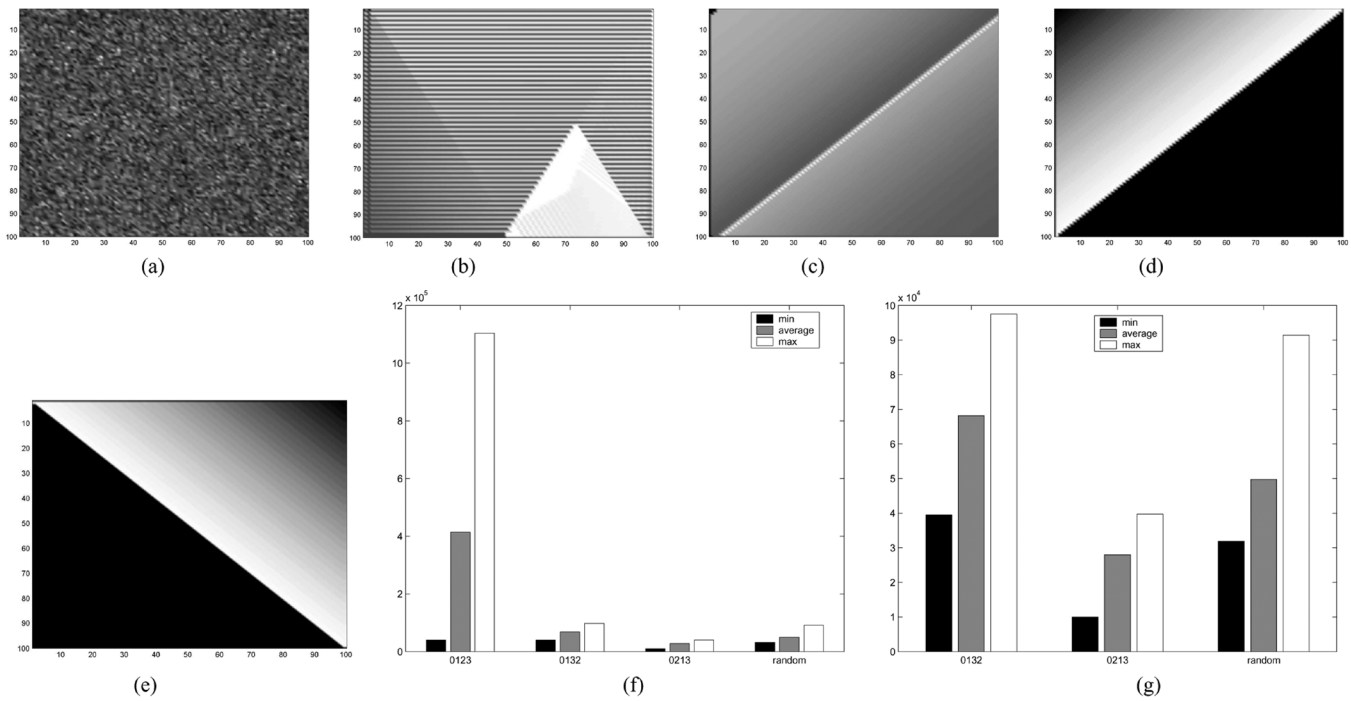


Fig. 13. Possible orders and their effect on cover time. (a)–(d) Four distinctive types of orders possible in the case of a square lattice. Plots show the cover times of running LRV with representative order on a square lattice with 100^2 nodes. Every node tried as a starting point. Grayscale pixels in each plot represent a cover time of LRV using a node at corresponding XY location as a starting node. The darker shade indicates lower cover time and the lighter shade indicates higher cover time. Note that the correspondence of the color range to the cover times range is different for each graph. (e) Circle order 1230 is a variation of a circle order 0123. Note symmetrical change in the plot. (f)–(g) Comparison of cover times statistic for the four different possible orders presented in (a)–(d). (a) Random order. (b) Cross (0213) order. (c) Line (0132) order. (d) Circle (0123) order. (e) Circle (1230) order. (f) Comparison of cover times. (g) Magnified view of f).

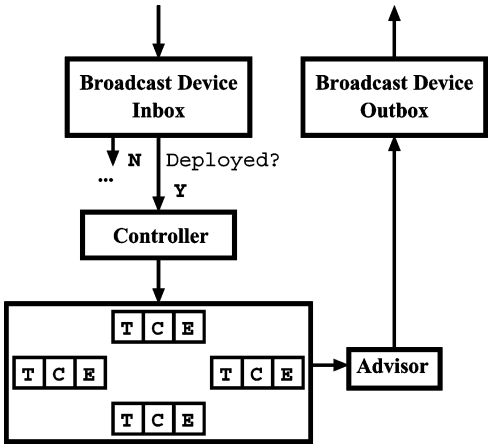


Fig. 14. Node architecture.

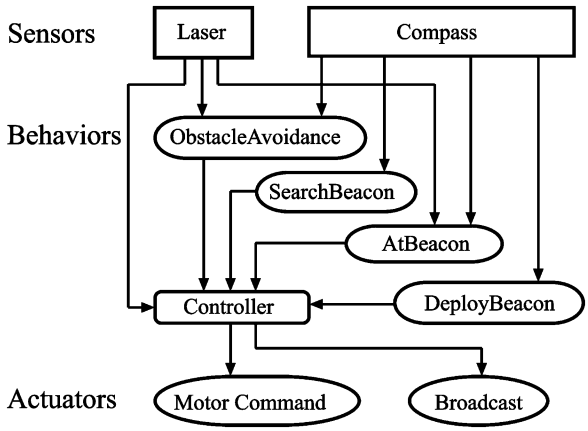


Fig. 15. System architecture showing robot behaviors.

When deployed, a node emits two data packets with different signal strengths. The packet with the lower signal strength is called the *MIN*-packet and the one with the higher signal strength is called the *MAX*-packet. The *MAX*-packet is used for data propagation within the deployed network. The *MIN*-packet contains information about the suggested direction the robot should take for coverage/exploration. This implies that the robot's compass and the node's compass agree locally on their measurement of direction. Given the coarse coding of direction we have chosen, this is not a problem in realistic settings. The policy used by the nodes to compute the suggested direction for exploration/coverage to recommend the least recently visited directions preferentially. All `OPEN` directions are recommended first (in order from South to West), followed by the `EXPLORED` directions with least last update value (least value of C). Note that this algorithm does not use inter-node communication.

The robot is programmed using a behavior-based approach [27] with arbitration [28] for behavior coordination. Priorities are assigned to every behavior *a priori*. As shown in Fig. 15, the robot executes four behaviors: *ObstacleAvoidance*, *AtBeacon*, *DeployBeacon*, and *SearchBeacon*. In addition to priority, every behavior has an activation level, which decides, given the sensory input, whether the behavior should be in an active or passive state (1 or 0, respectively). Each behavior computes the product of its activation level and corresponding priority and

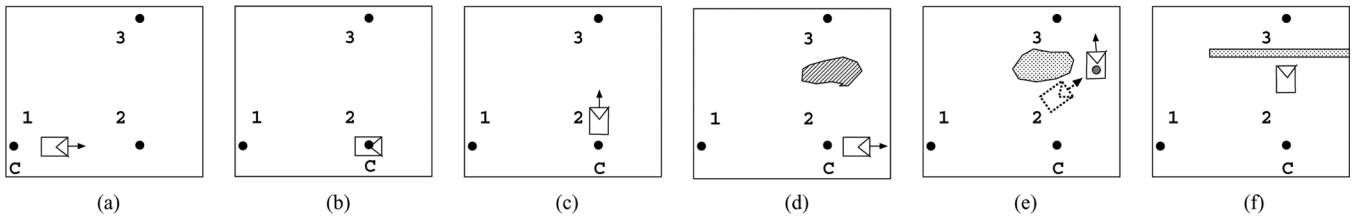


Fig. 16. Behavior Switching. (a) The robot is executing *SearchBeacon* behavior traversing suggested direction (Current Node is 1). (b) The robot is executing *AtBeacon* behavior, analyzing sensor readings (Current node is 2). (c) The robot is executing *SearchBeacon* behavior, supposing the node suggests direction *UP* and there are no obstacles detected in the sensor data (Choice of direction at the node (no obstacles)). (d) The robot is executing *SearchBeacon* behavior traversing direction, not originally suggested by the node (Choice of actions while in motion (with small obstacles)). (e) The robot is executing *SearchBeacon* and on its way encounters a small obstacle. At this point in time *ObstacleAvoidance* behavior is activated and the robot avoids an obstacle. Finally, if the robot leaves the range of its current beacon (beacon 2) and there are no nodes in the vicinity, *DeployBeacon* is executed and a new node is deployed. (f) The robot is executing a *SearchBeacon* behavior and encounters a large obstacle on its way. *DeployBeacon* behavior forces the robot to deploy a new node, which becomes a current node for the robot (Choice of actions while in motion (with large obstacles)).

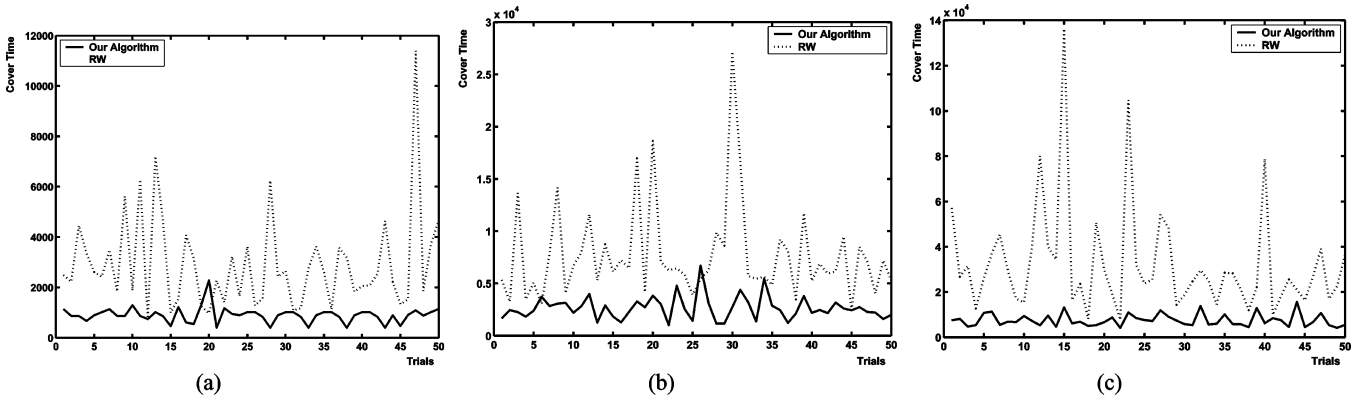


Fig. 17. Simulation results for different environment sizes across 50 trials. Our algorithm consistently outperforms random walk by an order of magnitude and is significantly more stable. (a) 25 m^2. (b) 49 m^2. (c) 100 m^2.

sends the result to the Controller, which picks the maximum value and assigns the corresponding behavior to command the Motor Controller for the next command cycle.

During motion, the robot maintains the notion of a current node [see Fig. 16(a)]. This is the node whose *MIN*-packets are received by the robot most frequently. When the robot moves to the vicinity of a new node, the *AtBeacon* behavior is triggered and the robot's current node is updated [see Fig. 16(b)]. *AtBeacon* analyzes the *MIN*-packets received from the current node and orients the robot along the suggested direction contained in those packets. In addition, the robot sends an update message to the node telling it to mark the direction from which the robot approached it as EXPLORED. This ensures that the direction of the recent approach will not be recommended soon. We term this the *last-neighbor-update*. After the robot has been oriented in a new direction, it checks its range sensor for obstacles. If the scan does not return any obstacles, the robot proceeds in the suggested direction [see Fig. 16(c)], while sending a message to its current node updating the state of the suggested direction to EXPLORED (the node also resets the corresponding C value). If, however, the suggested direction is obstructed, the *AtBeacon* behavior updates the node with this information and requests a new suggested direction [see Fig. 16(d)]. The *Obstacle Avoidance* behavior is triggered if an obstacle is detected in front of the robot, in which case an avoidance maneuver takes place. This situation is described on Fig. 16(e). If the obstacle is too large, the *DeployBeacon* behavior is triggered and a new node is deployed [see Fig. 16(f)].

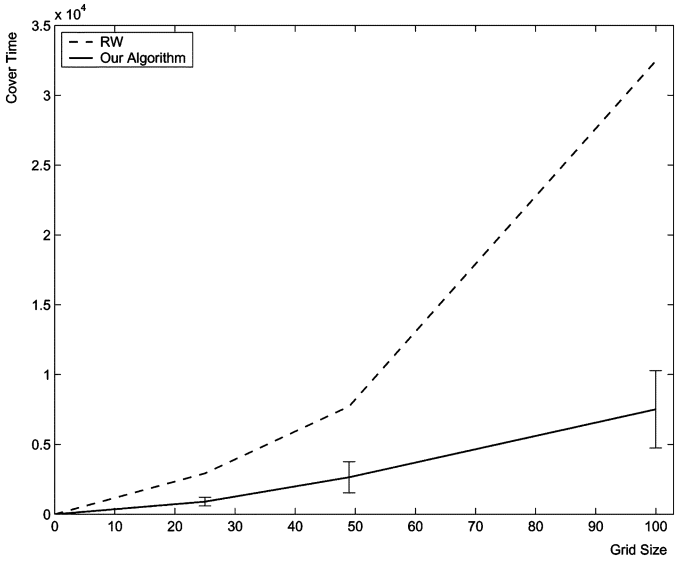


Fig. 18. Average cover times for three different grid sizes in simulation. Environment sizes are 25, 49 and 100 m^2.

Once the robot is oriented in a new direction (whether as a result of taking the advice of the node or as a result of avoiding

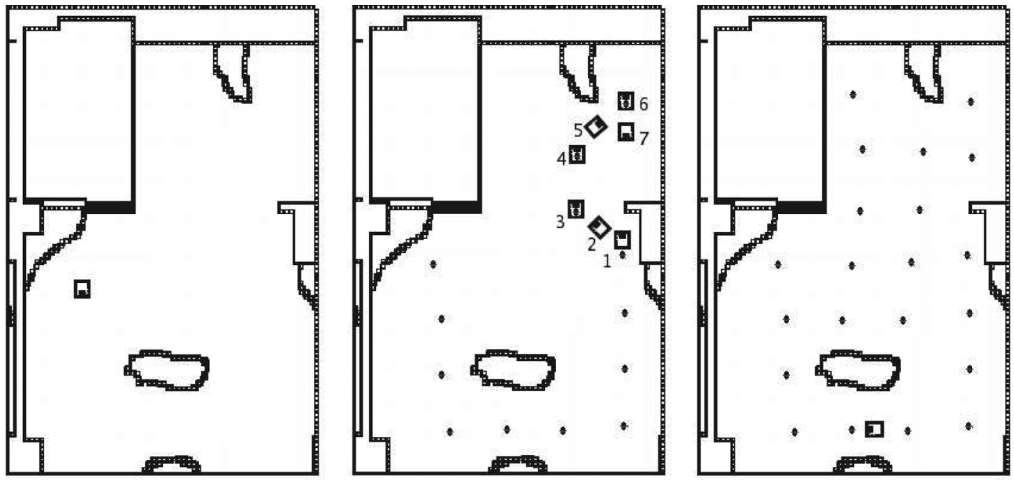


Fig. 19. Snapshots of the simulation trial of LRV on an environment with obstacles. (a) Environment before the experiment. (b) A robot explores the environment in the directions suggested by the nodes. Stages that robot goes through are marked on the snapshot. For example, look at transition from stages 1–3. In stage 1, a robot deploys a node. A node recommends the robot to traverse direction UP. While traversing this direction, the robot encounters a small obstacle and avoids it (stage marked 2). In stage marked 3, the robot is out of MIN-packet reach of node marked 1 and a new node is deployed. (c) Simulation snapshot after the deployment. The robot deployed the network into the environment and uses it for continuous coverage in a patrolling fashion.

an obstacle), the *SearchBeacon* behavior is triggered. *SearchBeacon* causes the robot to travel a predetermined distance without a change in heading (assuming there are no obstacles in the way). The *DeployBeacon* behavior is triggered if the robot does not receive a *MIN*-packet from any node after a certain timeout value. In this case, the robot deploys a new node into the environment.

Two interesting cases are shown on Fig. 16(e) and (f). If the robot executes a *SearchBeacon* behavior and on its way encounters a small obstacle, *ObstacleAvoidance* behavior is activated and the robot avoids an obstacle. Finally, if the robot leaves the range of its current beacon (beacon 2) and there are no nodes in the vicinity, *DeployBeacon* is executed and a new node is deployed [see Fig. 16(e)]. Fig 16(f) shows another situation when the robot is executing a *SearchBeacon* behavior and encounters a large obstacle on its way. *DeployBeacon* behavior forces the robot to deploy a new node, which becomes a current node for the robot.

VII. Simulation Experiments

In our experiments, we used the Player/Stage [29], [30] simulation engine populated with a simulated Pioneer 2DX mobile robot equipped with two 180° field-of-view planar laser range finders positioned back-to-back (equivalent to a 2-D omnidirectional laser range finder), wireless communication, and a set of nodes.

A. Experiments in an Open Bounded Environment

Values for communication radius and range of the laser were set to 1500 mm in the simulations. Fig. 17 shows the *cover times* for the random walk algorithm and LRV on environments of different sizes: 25, 49, and 100 m^2. Note that the *cover time* on a simulated or a real system is the amount of simulation/real time in seconds it takes the robot to cover every point of the environment under its sensor shadow (laser range finder). For every grid size 50 experiments were conducted for both algorithms. The experiments show that our algorithm outperforms random walk and is more stable. In addition, Fig. 18 shows the average *cover times* for three different grid sizes. In comparing the results of the simulation with those obtained on the graph model please keep in mind the following.

1) The notion of cover time in the Player/Stage simulations used in this section are the actual time in seconds it takes the robot to cover the environment completely with the sensor (laser range finder), whereas in the graph world, the cover time is the number of edges the robot traverses to visit every node at least once.
2) The noise inherently present in the simulations forces the robot to visit the same nodes several times to make sure that every point of the environment is covered by the robot's sensor, whereas in the graph model the noise is absent.

Note the direct correspondence between the trends in the results obtained in the graph world and the results of the simulation.

B. Experiments in a Bounded Environment With Obstacles

We also performed a set of experiments in a bounded environment with obstacles to highlight the robustness of the approach. Fig. 19 shows three snapshots of the experiment. This experiment demonstrates the robustness of LRV to obstacles in the environment. In particular, Fig. 19(b) demonstrates simulation examples of LRV when an obstacle avoidance is used in the deployment cycle. An environment used in this section was also used in a more elaborate series of experiments with LRV reported in [31] that modify the environment in the real time, while the robot is successfully executing coverage and exploration tasks.

VIII. Implicit Sensor Network Repair and Maintenance

An emergent property of LRV is the ability to perform network repair and maintenance. Since the algorithm is shown to be complete, it is guaranteed to visit the same node over and

over again. Suppose that one of the nodes, say node k, ran out of power or was damaged. Further consider a moment in time just before the robot traverses direction d towards the damaged node. Now, the robot is moving along direction d towards node k. According to the deployment function that is used, there should be a communication/sensing gap in the deployed sensor network (unless the network was over deployed and does not require repair). Hence, while facing the same deployment situation and using the same deployment function at the location where node k was deployed, the robot simply deploys a new node, thereby solving the problem of sensor network repair and maintenance implicitly. Note that if the robot can recognize the nodes then it can attempt repairing the node first (or retrieving for later repair at the base) before deploying the new node.

IX. Remarks on Generalization

In this paper, we have presented an algorithm based on the policy of choosing the least recently visited directions preferentially. At the same time, depending on the application requirements, other policies might be better. It is possible to extend other search/coverage algorithms using physical network embedding to function in unknown, unstructured, dynamic environments. The basic idea is to augment a coverage algorithm with an ability to deploy and maintain physical network infrastructure, while simultaneously using the network to aid in coverage and exploration. Thus, what we want to highlight is the underlying philosophy of network deployment as an integral part of coverage and exploration.

Sensor network deployment is one of the building blocks of this paper. A robot decides when to deploy a node based on a deployment condition, or more generally, deployment function. A deployment function should be designed based on the application. However, there are two main characteristics that every deployment function should have: 1) the deployed sensor network is connected (i.e., between any two nodes there should be a communication path) and 2) deployed static network configuration should be such that static coverage is maximized. In the proposed implementation of LRV, both characteristics are captured. The deployment is based on communication range thresholding. Hence, the two consecutive nodes are guaranteed to be connected (thus, the deployed sensor network is connected) and we adjusted communication range threshold so that if every sensor node is equipped with high fidelity sensor, static coverage is maximized. Note that in a general case when the dynamic coverage problem is considered, a complete coverage is achieved by LRV when the deployment distance is less than or equal to twice the sensing range, which is trivial to visualize geometrically.

Other deployment functions could also encode such parameters as desired topology (i.e., by specifying the number of nodes deployed in different directions, etc.), desired boundary, phenomenon density (deploy more nodes in places with high phenomenon density), deploy where landmarks detected, etc.

X. Conclusion

In this paper, we have presented an efficient, robust, and scalable algorithm to embed an active infrastructure (sensing, communication, and computation) into the environment while simultaneously using this infrastructure for coverage and exploration. The algorithm (LRV) is based on visiting the LRV directions preferentially and is decentralized, scalable, robust, fault tolerant, and can be used on simple robots. LRV is based on the idea of deploying sensor nodes from time to time. Once deployed, every node acts like a signpost recording which directions the robot has explored recently. When a robot is in the vicinity of a node, it recommends to the robot a direction that has been least recently visited (hence, the name LRV).

We analyzed the characteristics of LRV theoretically, modeling the static steady state of the deployed sensor network as a finite graph G. We proved that LRV is complete on G (i.e., the exploration time of LRV on a finite graph is finite). For a graph $G = (V, E)$ with maximum degree d, if Cover Time $= O(f(V))$, then Exploration Time $= d * O(f(V))$. We proved that exploration time is $\leq 2|E|$ (twice the number of edges, or asymptotically optimal) for the special case, when G is a tree. For another special case, when G is a square lattice, we empirically conjectured that both cover and exploration times are asymptotically $O(V^{1+\epsilon})$. The special case of a square lattice is also interesting from practical perspective, because in our LRV implementation and experiments we chose to maintain at most four directions, which results in a static steady state of the deployed sensor network resembling a square lattice.

We examined the tradeoffs that should be considered in choosing one exploration algorithm over another to solve this problem. The bounds for the coverage task are given by random walk (the robot has no information and explores randomly) and depth first search (a map of the environment is available in the form of a graph) which solves the problem optimally.

The data shown in Fig. 7, suggests strongly that our algorithm asymptotically outperforms the k node algorithm presented in [12].

In addition, it is shown in [12] that if the number k of available nodes reduces, the *cover time* increases rapidly. Therefore, in dynamic environments the performance of the algorithm decreases drastically even if one node is destroyed. Whereas in our algorithm, such a problem does not exist, since a new node will be deployed in place of the destroyed one automatically.

We compared LRV to 1-LRTA* [26]. 1-LRTA* is a well known graph search algorithm that can be applied to graph coverage. In 1-LRTA*, a weight is associated with a node. The edge to traverse is chosen based on weights of neighboring nodes. The weight of a node is incremented with the weight of a node the robot transitions to. Hence, 1-LRTA* requires nodes to communicate. Fig. 8 shows that generally 1-LRTA* outperforms LRV. However, in reality LRV deploys the network in addition to exploring, whereas 1-LRTA* requires the graph to operate on. An important result of this chapter is that it is possible to extend other search/coverage algorithms using physical network embedding to function in unknown, unstructured, dynamic environments.

We have extended an analysis of LRV to a cube lattice graph that represents network embedding in 3-D case. For this special case, when G is a cube lattice, we empirically conjectured that both cover and exploration times are asymptotically as in square lattice case and is bounded by $O(V^{1+\epsilon})$.

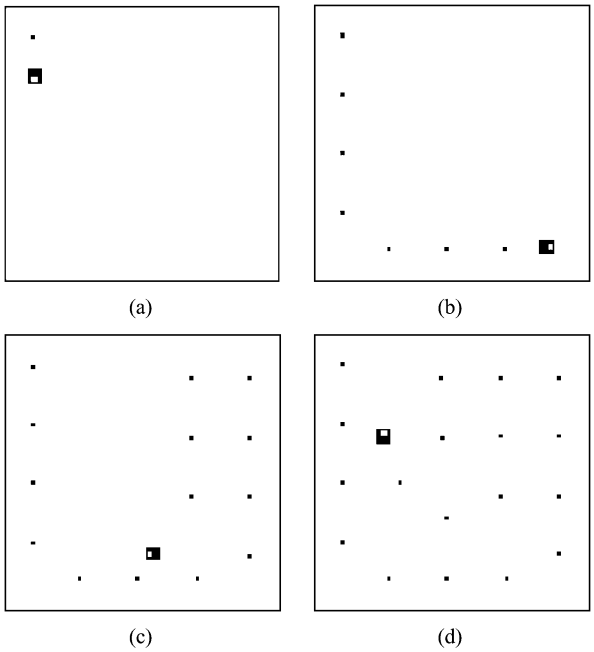


Fig. 20. Deployment of nodes in a representative simulation trial. Note that due to noise added in simulation, the deployed nodes do not form a perfectly square lattice.

We also studied the benefits of using ordered rules in tie break resolution (choosing one of equally good edges). In the case of a square lattice, each node can have up to four edges to traverse. Hence, there are 24 *order rules* possible. We proposed classification of these rules into three classes: cross, line, and circle. Each class consists of eight symmetric *order rules*. We have performed extensive experimental trials that showed cross orders perform better than traditionally used random order. Furthermore, in general, for square lattices it is enough to be able to recognize corner nodes in order to apply one of the symmetric cross orders to obtain **optimal** results with LRV. Note that if we would simply use random order, traditionally used in literature, the optimal cover time is highly unlikely to obtain.

We verified the performance of LRV and its asymptotic behavior in simulation. There exists a direct correspondence between the results obtained from the theoretical analysis (coverage on the graph) and the data from simulation experiments. Note also, that even though the lattice grid was considered as a graph environment for the theoretical analysis, in practice, the network of deployed nodes is not required to be a perfect grid. Fig. 20 shows a series of screen shots taken from one of the trials of the simulation in the 49 m^2 environment. Note also that the performance of our algorithm is not affected, since it does not rely on localization or mapping.

The theoretical analysis on graphs and verification in simulation shows that tradeoffs in the assumptions can affect *cover time* significantly. Simple algorithms like RW or DFS can be used for coverage, but only in the extreme cases as described previously. In the case where mapping and localization are not available but the number of available nodes is unlimited, our algorithm appears to outperform others.

The algorithm that we propose and analyze in this paper (LRV) allows us to deploy and maintain a sensor network, while covering and exploring the environment. In our paper, we use LRV as a fundamental system [32] for enabling network-mediated robot navigation [33], network-mediated multi-robot task allocation [34], and the general problem of spatiotemporal monitoring [35].

REFERENCES

[1] M. A. Batalin and G. S. Sukhatme, "Efficient exploration without localization," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2003, pp. 2714–2719.
[2] M. A. Batalin and G. S. Sukhatme, "The analysis of an efficient algorithm for robot coverage and exploration based on sensor network deployment," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2005, pp. 3489–3496.
[3] D. W. Gage, "Command control for many-robot systems," in *Proc. 19th Annu. AUVS Techn. Symp.*, 1992, pp. 22–24.
[4] J. O'Rourke, *Art Gallery Theorems and Algorithms*. New York: Oxford Univ. Press, 1987.
[5] A. Howard, M. J. Mataric, and G. S. Sukhatme, "Mobile sensor network deployment using potential fields: A distributed, scalable solution to the area coverage problem," in *Proc. 6th Int. Symp. Distrib. Autonomous Robot. Syst.*, 2002, pp. 299–308.
[6] M. A. Batalin and G. S. Sukhatme, "Spreading out: A local approach to multi-robot coverage," in *Proc. 6th Int. Symp. Distrib. Autonomous Robot. Syst.*, 2002, pp. 373–382.
[7] M. A. Batalin and G. S. Sukhatme, "Sensor coverage using mobile robots and stationary nodes," in *Proc. SPIE*, 2002, pp. 269–276.
[8] H. Choset, "Coverage for robotics—A survey of recent results," *Annals Math. Artif. Intell.*, vol. 31, pp. 113–126, 2001.
[9] B. Yamauchi, "Frontier-based approach for autonomous exploration," in *Proc. IEEE Int. Symp. Comput. Intell., Robot. Autom.*, 1997, pp. 146–151.
[10] A. Zelinsky, "A mobile robot exploration algorithm," *IEEE Trans. Robot. Autom.*, vol. 8, no. 6, pp. 707–717, Dec. 1992.
[11] W. Burgard, D. Fox, M. Moors, R. Simmons, and S. Thrun, "Collaborative multirobot exploration," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2000, pp. 476–481.
[12] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes, "Robotic exploration as graph construction," *IEEE Trans. Robot. Autom.*, vol. 7, no. 6, pp. 859–865, Dec. 1991.
[13] M. A. Bender, A. Fernandez, D. Ron, A. Sahai, and S. Vadhan, "The power of a pebble: Exploring and mapping directed graphs," in *Proc. Annu. ACM Symp. Theory Comput. (STOC)*, 1998, pp. 269–278.
[14] R. T. Vaughan, K. Stoy, G. S. Sukhatme, and M. J. Matarić, "Whistling in the dark: Cooperative trail following in uncertain localization space," in *Proc. Autonomous Agents Multi-Agent Syst.*, 2000, pp. 187–194.
[15] R. T. Vaughan, K. Stoy, G. S. Sukhatme, and M. J. Matarić, "Lost: Localization-space trails for robot teams," *IEEE Trans. Robot. Autom., Special Issue Multi-Robot Syst.*, vol. 18, no. 5, pp. 796–812, Oct. 2002.
[16] S. Koenig, B. Szymanski, and Y. Liu, "Efficient and inefficient ant coverage methods," *Annals Math. Artif. Intell.*, vol. 41, no. 76, pp. 31–31, 2001.
[17] J. Svennebring and S. Koenig, "Trail-laying robots for robust terrain coverage," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2003, pp. 75–82.
[18] I. Wagner, M. Lindenbaum, and A. Bruckstein, "Distributed covering by ant-robots using evaporating traces," *IEEE Trans. Robot. Autom.*, vol. 15, no. 5, pp. 918–933, Oct. 1999.
[19] I. Wagner, M. Lindenbaum, and A. Bruckstein, "Efficiently searching a dynamic graph by a smell-oriented vertex process," *Annals Math. Artif. Intell.*, vol. 24, pp. 211–223, 1998.
[20] I. Wagner, M. Lindenbaum, and A. Bruckstein, "Mac vs. pc: Determinism and randomness as complementary approaches to robotic exploration of continuous unknown domains," *Int. J. Robot. Res.*, vol. 19, no. 1, pp. 12–31, 2000.
[21] S. Meguerdichian, F. Koushanfar, M. Potkonjak, and M. B. Srivastava, "Coverage problems in wireless ad-hoc sensor networks," in *Proc. IEEE INFOCOM*, 2001, pp. 1380–1387.
[22] M. McGlynn and S. Borbash, "Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks," in *Proc. MobiHoc*, 2001, pp. 137–145.
[23] T. Clouqueur, V. Phipatanasuphorn, P. Ramanathan, and K. K. Saluja, "Sensor deployment strategy for target detection," in *Proc. 1st ACM Int. Workshop Wireless Sensor Netw. Appl.*, 2002, pp. 42–48.

[24] S. S. Dhillon and K. Chakrabarty, "Sensor placement for effective coverage and surveillance in distributed sensor networks," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2003, pp. 1609–1614.
[25] L. Lovasz, "Random Walks on Graphs: A Survey," *Bolyai Soc. Math. Study, Combinatorics*, vol. 2, pp. 1–46, 1993.
[26] S. Koenig and R. Simmons, "Easy and hard testbeds for real-time search algorithms," in *Proc. Nat. Conf. Artif. Intell.*, 1996, pp. 279–285.
[27] M. J. Mataric, "Behavior-based control: Examples from navigation, learning, and group behavior," *J. Experimental Theoretical Artif. Intell., Special Issue Softw. Arch. Phys. Agents*, vol. 9, no. 2-3, pp. 323–336, 1997.
[28] P. Pirjanian, "Behavior coordination mechanisms—state-of-the-art," Inst. Robot. Intell. Syst., Univ. Southern California, Los Angeles, Tech. Rep. IRIS-99-375, 1999.
[29] B. P. Gerkey, R. Vaughan, K. Stoy, A. Howard, G. Sukhatme, and M. Mataric, "Most valuable player: A robot device server for distributed control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2001, pp. 1226–1231.
[30] R. Vaughan, "Stage: A multiple robot simulator," Inst. Robot. Intell. Syst., Univ. Southern California, Los Angeles, Tech. Rep. IRIS-00-393, 2000.
[31] M. A. Batalin and G. Sukhatme, "Coverage, exploration and deployment by a mobile robot and communication network," *Telecommun. Syst. J., Special Issue Wireless Sensor Netw.*, vol. 26, no. 2, pp. 181–196, 2004.
[32] M. Batalin, "Symbiosis: Cooperative algorithms for mobile robots and a sensor network," Ph.D. dissertation, Comput. Sci. Dept., Univ. Southern California, Los Angeles, 2005.
[33] M. Batalin, G. Sukhatme, and M. Hattig, "Mobile robot navigation using a sensor network," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2004, pp. 636–642.
[34] M. Batalin and G. Sukhatme, "Sensor network-mediated multi-robot task allocation," in *Proc. 3rd Int. Naval Res. Lab. Multi-Robot Syst. Workshop*, 2005, pp. 27–42.
[35] M. Batalin, M. Rahimi, Y. Yu, S. Liu, A. Kansal, G. Sukhatme, W. Kaiser, M. Hansen, G. Pottie, M. Srivastava, and D. Estrin, "Call and response: Experiments in sampling the environment," in *Proc. ACM SENSYS*, 2004, pp. 25–38.

Maxim A. Batalin (M'05) received the M.S. and Ph.D. degrees in computer science from the University of Southern California (USC), Los Angeles, and the B.S. degree in mathematics, computer science from the University of Oregon, Eugene, OR, in 2001, and the M.S. degree in management from Tavriya National University, Simferopol, Ukraine, in 2002.

He is a Researcher with the Center for Embedded Networked Sensing, University of California, Los Angeles (UCLA). His primary research interests are in mobile robotics, sensor networking, and biomedical systems. He has published over 20 technical papers.

Dr. Batalin is a member of Sigma Xi.

Gaurav S. Sukhatme (M'05) received the M.S. and Ph.D. degrees in computer science from University of Southern California (USC), Los Angeles.

He is an Associate Professor of Computer Science (joint appointment in Electrical Engineering) with USC. He is the Codirector of the USC Robotics Research Laboratory and the Director of the USC Robotic Embedded Systems Laboratory, which he founded in 2000. His research interests include multirobot systems, sensor/actuator networks, and robotic sensor networks. He has published extensively in these and related areas.

Prof. Sukhatme was a recipient of the NSF CAREER Award and the Okawa Foundation Research Award. He has served as PI on several NSF, DARPA, and NASA grants. He is a Co-PI on the Center for Embedded Networked Sensing (CENS), an NSF Science and Technology Center. He is a member of AAAI and the ACM. and is one of the founders of the Robotics: Science and Systems conference. He is program chair of the 2008 IEEE International Conference on Robotics and Automation and the Editor-in-Chief of *Autonomous Robots*. He has served as an Associate Editor of the IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, the IEEE TRANSACTIONS ON MOBILE COMPUTING, and on the editorial board of *IEEE Pervasive Computing*.