Polylogarithmic-Overhead Piecemeal Graph Exploration*

Baruch Awerbuch
Dept. of Computer Science
Johns Hopkins University
baruch@cs.jhu.edu

Stephen Kobourov
Dept. of Computer Science
Johns Hopkins University
kobourov@cs.jhu.edu

January 29, 1998

Abstract

We introduce a new traversal technique in the context of piecemeal exploration of unknown graphs. The problem of learning a graph via piecemeal exploration requires a robot to create a complete map of its environment, subject to two constraints. First, it cannot jump between non-adjacent vertices in one time step and second, it must return to a fixed starting point every so often. This paper presents the *recursive piecemeal search* (RPS) strategy together with an algorithm for the above problem. We are able to achieve $O(\log^2 n)$ overhead (where n is the number of vertices), improving on previous results of Awerbuch, Betke, Rivest, and Singh which require $O(n^\epsilon)$ overhead. The graph is discovered via the recursive piecemeal search, which can be viewed as a combination of breadth-first and depth-first passes. The construction of RPS trees relies on the concept of sparse neighborhood covers and captures nicely the nature of the graph exploration problem.

*This paper is eligible for student travel support and for the student paper prize.

1 Introduction

Formal approaches to robot navigation and environment learning have been extensively studied in theoretical computer science [1, 9, 13]. Papadimitriou and Yanakakis [15] addressed the exploration of unknown environments, and the problem was further investigated by Betke [8], and Blum *et al.* [10]. The geometric aspect of the navigation problem (in the context of room exploration with or without polygonal obstacles) has been examined by Deng *et al.* [12], and Hoffmann *et al.* [14]. On-line naviagaiton was studied by Blum *et al.* [10] and Bar-Eli *et al.* [4]. Recently, randomized robot navigation was addressed by Berman *et al.* [6] while Bender and Slonim [5] examined graph exploration by two robots. A survey of robot navigation problems and algorithms can be found in Rao *et al.* [16].

The graph exploration problem can be defined as follows. We are given a connected, undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$. We are also given a *start* vertex from which the search must begin. The goal is to explore the graph and create a complete map in an efficient way. Betke, Rivest, and Singh [7] refined the problem to learning a graph via a *piecemeal* search (the graph is discovered one piece at a time). They achieve linear time on special graphs such as grid-graphs with non-convex obstacles, triangular tesselations with triangular obstacles, and planar graphs in general. Further studies in this area resulted in algorithms for general graphs, for which Awerbuch, Betke, Rivest, and Singh [3] have achieved $O(n^{1+\epsilon} + m)$ running time.

In this paper we introduce a novel search technique along with a new algorithm for the piecemeal graph exploration problem. Our *recursive piecemeal search* (`RPS`) relies on the sparse neighborhood covers of Awerbuch, Berger, Cowen, and Peleg [2]. By combining breadth-first with depth-first searches we significantly reduce the overhead associated with previous algorithms. The `RPS` algorithm has a running time of $O(n \log^2 n + m)$. The rest of the paper is organized as follows.

In Section 2 we define the problem, looking at existing graph exploration algorithms. We also show why the attempts to solve the piecemeal learning problem using such algorithms are bound to either not work or be inefficient. Section 3 outlines the new search technique, `RPS`, and the algorithm which uses `RPS` to efficiently solve the piecemeal learning problem. Section 4 contains the analysis of the correctness and the running time of the algorithm. In Section 5 we present concluding remarks and discuss related open problems.

2 Exploration Model and Existing Algorithms

2.1 Exploration Model

Piecemeal exploration of an unknown graph introduces two natural constraints. The first constraint, *continuity*, does not allow jumping over from a vertex to another non-adjacent vertex in one time step. The second constraint, *interruptibility*, forces us to return to the start node every so often. Consequently, exploration of the graph is achieved in a piecemeal fashion.

Consider a robot which is given the task of exploring an unknown graph. If it needs to get from one vertex to another non-adjacent vertex, the robot cannot "teleport" there, but rather it has to traverse the edges on a path between these two vertices. Furthermore, suppose that the robot cannot traverse the whole graph all at once. In this case, after a fixed number of traversals, it must return to the start vertex (where it can recharge its batteries or fill its gas tank) before it can resume the exploration.

Continuity implies that the search proceeds along the edges of the graph only. This is an actual constraint in contexts where traversal corresponds to physical exploration of a terrain, or in distributed environments where exploration is performed by a token carried by a message. As we are going to show, this property is possessed by DFS but not by BFS.

Interruptibility requires that at any point the sub-graph induced by the vertices and edges traversed must contain a path to the origin of the search. The path must also be at most *constant* times longer than the diameter of the graph. More precisely, we are given an upper bound Max on the number of edges we can explore. To successfully explore the graph we must have $Max \geq (2+c)Rad$, where $c \geq 2$ is a constant and Rad is the radius of the graph. We will show that this property is possessed by BFS but not by DFS.

In this model, the robot always recognizes already visited vertices. At any given vertex, the robot can "see" only the incident edges. We assume that it can distinguish between the edges and knows if any of them have been traversed. The measure of efficiency of the exploration algorithm is the total number of edge traversals necessary to produce a complete map of the environment. It follows that "thinking," path planning, and combining different maps are for free.

2.2 BFS and DFS

Classic search algorithms, such as BFS and DFS [11], can be viewed as natural algorithms for exploration of unknown graphs. If we allow discontinuities and if interruptibility is not a constraint, using either of them we can successfully search a graph. DFS is an efficient search algorithm since it will visit no more than twice the number of edges in the graph. Adding interruptibility to the algorithm, however, results in the altogether failure of the algorithm to perform (Fig. 1 a).

BFS, on the other hand, can handle interruptions. Let a *frontier vertex* be a vertex which has unexplored edges, and let an *expansion* of a frontier vertex be the traversal of all its unexplored edges. Traversal in breadth-first fashion consists of consecutive expansions of frontier vertices, layer after layer. As long as $Max \geq (2+c)Rad$, BFS will successfully explore the graph. In the context of physical exploration, where continuity is necessary, BFS is inefficient as it has to traverse $O(m^2)$ edges in the worst case (Fig. 1 b).

2.3 Recursive Exploration

To deal with the above problems Awerbuch, Betke, Rivest, and Singh [3] use recursion. A BFS tree of small depth r is constructed directly with small overhead cost. For a large depth

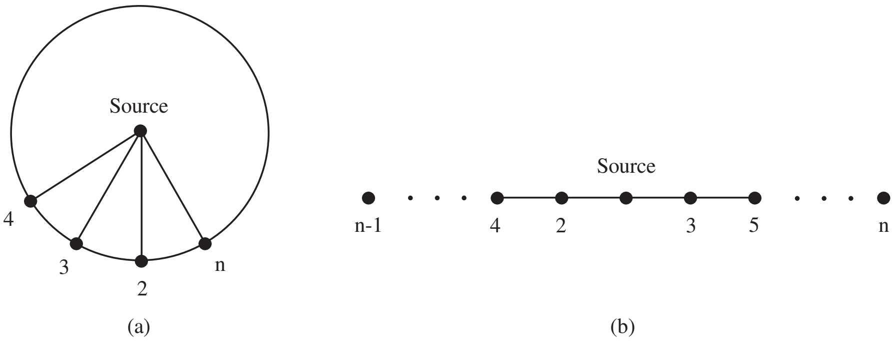

Figure 1: (a) The radius of the graph is 1 but for any $Max < 2(n-1)$ DFS fails to explore the graph. (b) All nodes lie on a path and the source is in the middle. BFS is inefficient as it traverses $O(m^2)$ edges.

r, the algorithm divides the exploration area into two *strips*, each strip containing $r/2$ BFS layers. It next tries to process each one of these two strips using local exploration procedures for distance $r/2$.

A strip may contain collections of *source* nodes (all the nodes in the last layer of the previous strip), rather than just one source node. The classic BFS, in which continuity is not a constraint, has no problem combining all the source nodes into one, and thus combining searches from many different nodes into a single exploration. As we already observed above, we cannot do this in the case of physical exploration.

One way to address this problem is to interrupt the exploration from each source node so as to synchronize it with other explorations. This introduces into the algorithm multiplicative overhead of the graph's radius. Another alternative is to run explorations from different source nodes and then combine the results. Unfortunately, this also introduces overhead which grows linearly with the number of source nodes.

Finally, we can search from different source nodes in a certain order so that new explorations do not invade into previously explored territory. This avoids multiplicative overhead, but fails to process the strip. Algorithm `TREASURE_SEARCH` [3] applies a similar procedure a logarithmic number of times and manages to correctly process the strip, but incurs a multiplicative logarithmic overhead inside the recursion. This method is doomed to $\Omega(m+n^{1+o(1)})$ complexity.

3 `RPS` Exploration Strategy

3.1 The Main Idea

Our algorithm follows the recursive framework outlined above. The novelty of our exploration procedure is that, in contrast to direct recursion [3], traversal is *not* confined to the strip and in fact may "invade" a logarithmic number of subsequent strips (Fig. 2). To control the complexity, each exploration procedure must designate at least *half* of its explored territory as *private*. Private territory cannot be touched by subsequent explorations. These innovations together with the notion of sparse neighborhood covers of Awerbuch, Berger, Cowen, and Peleg [2] allow us to eliminate the multiplicative factor associated with standard recursive BFS procedures in [3].

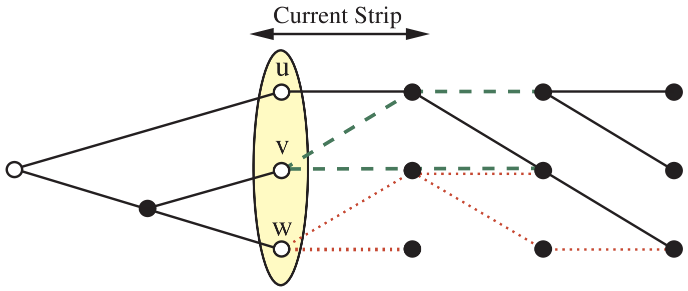

Figure 2: The forest constructed while processing a strip of size 1. Observe invasion into subsequent strips as the trees rooted at v and w extend past the current strip.

We use a local exploration routine which, given a source vertex v and a distance r, explores the r-neighborhood of v. We do this recursively by making two calls for explorations at distance $r/2$. If each of those calls covers territory at distance $r/2$, then we are reducing the original problem of processing a strip of distance r into two problems half the size. The above described invading effect, however, may cause this procedure to run for distance $r \log n$.

In fact, some of the local explorations may not even yield a forest of BFS trees. All that matters is that they *contain* this forest. In order to actually compute the BFS forest, we only need to combine the maps generated by these local explorations. Recall that in our model combining different maps is for free, i.e., their computational complexity is $O(1)$.

3.2 The Procedures

The main data structure is the two sets $\mathcal{Z}^r$ and $\mathcal{Y}^r$. Set $\mathcal{Z}^r$ is made of trees Z^r and set $\mathcal{Y}^r$ is made of trees Y^r. Given a source node v and a distance r the local exploration procedure `EXPLORE` produces a pair of trees (Y^r, Z^r). Y^r can be viewed as the "kernel" of the tree Z^r and at any time $|Z^r| \leq 2|Y^r|$. It is these trees that make up the sets $\mathcal{Z}^r$ and $\mathcal{Y}^r$ for the different values of r. More details on how they are created and maintained are given below when we define procedure `EXPLORE`.

The algorithm starts by initializing sets $\mathcal{Z}^r \leftarrow \emptyset$ and $\mathcal{Y}^r \leftarrow \emptyset$, for $r = 1, 2, 4, \ldots, Max$. Next we call the search routine `RPS` $(r = Max, S = \{start\}, T = start)$. This call will return a BFS tree rooted at *start* in the graph G.

RPS (r, S, T)**:** The heart of our algorithm is the recursive piecemeal search routine. It constructs a forest of BFS trees for distance r around a set of source vertices S, which are spanned by a common tree T. To avoid re-invading the same territory from multiple directions, `RPS` (r, S, T) runs in a subgraph $G' \subset G$. The subgraph G' is induced by deleting from the original graph G the nodes spanned by the collection of trees Y^r (Fig. 3). At the highest level $\mathcal{Y}^r = \emptyset$, and consequently, at the top level of recursion `RPS` (r, S, T) is the plain BFS tree that spans the entire graph.

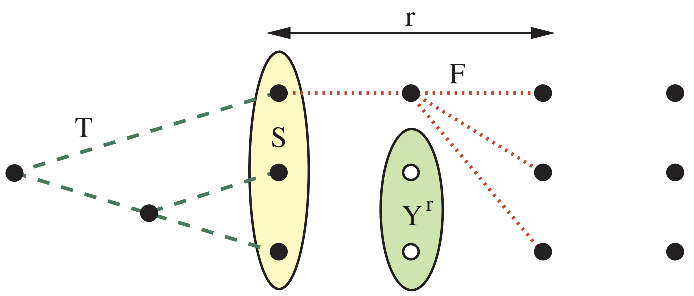

Figure 3: Given r, S, and T, `RPS` constructs forest F which covers all the unexplored nodes in the strip.

For clarity, we have split the pseudocode of the algorithm into two pieces (Fig. 4). The base case of the recursion is $r = 1$, in which we need to explore all the unvisited neighbors of the vertices in S. We relocate to each unexplored vertex in S using the DFS route in T (`NDFS`) and add the newly discovered edges and vertices to a temporary forest *Map*. This forest is then passed to procedure `MBFS` which removes overlaps and returns the BFS forest for the explored area.

`RPS` $(r = 1, S, T)$
  $Map \leftarrow \emptyset$
  $src \leftarrow NULL$
  while $(src \leftarrow$ `NDFS` $(src, T)) \neq NULL$
    if $src \in S$
      $Map \leftarrow Map \cup$ `PBFS` (src)
  return `MBFS` $(Map, S, 1, T)$

`RPS` $(r > 1, S, T)$
  $src \leftarrow NULL$
  for $i = 1$ **to** 2 **do**
    while $(src \leftarrow$ `NDFS` $(src, T)) \neq NULL$
      if $src \in S$ **and** $dist\,(src, \mathcal{Y}^{\frac{r}{2}}) > \frac{r}{2}$
        `EXPLORE` $(src, \frac{r}{2})$
    $F_i \leftarrow$ `MBFS` $(\mathcal{Z}^{\frac{r}{2}}, S, \frac{r}{2}, \mathcal{Y}^r \cup T)$
    $T \leftarrow T \cup F_i$
    $S \leftarrow$ `FRONT` (F_i)
  return $F_1 \cup F_2$

Figure 4: `RPS`.

If $r > 1$ then we split the current strip in two parts. We recursively process them by calling the local exploration routine `EXPLORE`, with radius $r/2$. Here F_1 and F_2 are the forests constructed for the first and second strips, respectively. Notice that when we want to process the second strip, we actually need to use the BFS forest of the first one, together with the new set of source nodes (Fig. 5). Procedure `MBFS` gives us the forest from the first strip and procedure `FRONT` returns the source nodes for the second strip.

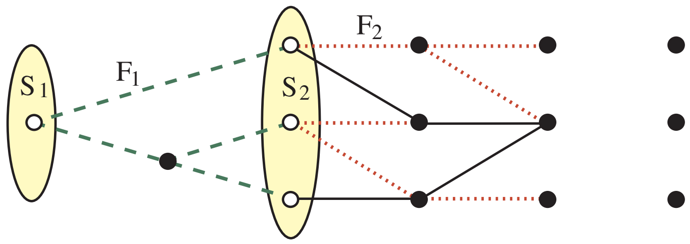

Figure 5: The concept of recursion: concatenation of two BFS forests for each one of the strips.

As in the base case, here we relocate between the source vertices using `NDFS`. We also need to make sure that the same parts of the graph are not explored over and over again. The condition $dist\,(src, \mathcal{Y}^{\frac{r}{2}}) > \frac{r}{2}$ specifies that the kernels of the trees we produce are not overlapping, where $dist\,(u, F)$ is defined as $min\{\, d\,(u,v) \,|\, v$ is a node in forest $F\,\}$. Notice that we are not going to construct trees from vertices which are at distance less than $r/2$ from an already discovered area. The sets $\mathcal{Y}^r$ and $\mathcal{Z}^r$ keep track of the explored territory are are updated in `EXPLORE` each time we make a new tree.

PBFS (v)**:** Procedure `PBFS` performs a *physical* breadth-first search of depth 1 around a given vertex v. In effect, it visits the immediate neighborhood of v, where v is a source vertex. Thus, `PBFS` is responsible for the physical learning of the graph. Recall that the robot recognizes vertices that have been already explored. Hence, when performing `PBFS` the robot need not visit already explored vertices. This procedure returns the newly explored tree.

NDFS (src, T)**:** Procedure `NDFS` is responsible for moving the robot between source vertices. It takes as input a source node src and a tree T. If $src = NULL$ the procedure returns a frontier vertex from T and marks it as visited. Otherwise, we relocate from src to another unvisited leaf, mark it as visited and return it. If there are no more unvisited nodes it T we return $NULL$. As defined above, a frontier vertex is one that has unexplored edges. Recall that in our exploration model we cannot jump from a vertex to another non-adjacent one, but instead have to traverse a path between those two vertices. `NDFS` relocates the robot between different sources via the DFS path between them in the given tree T.

MBFS $(\mathcal{A}, S, r, \mathcal{B})$**:** Procedure `MBFS` receives as input a forest of trees $\mathcal{A}$ and some already explored territory $\mathcal{B}$. It then creates a BFS forest of depth r with respect to S, the source vertices in the current strip. Note that there is no traversal associated with this procedure, all we need to do is extract the information we have already gathered from the sets $\mathcal{A}$, $\mathcal{B}$ (Fig. 6).

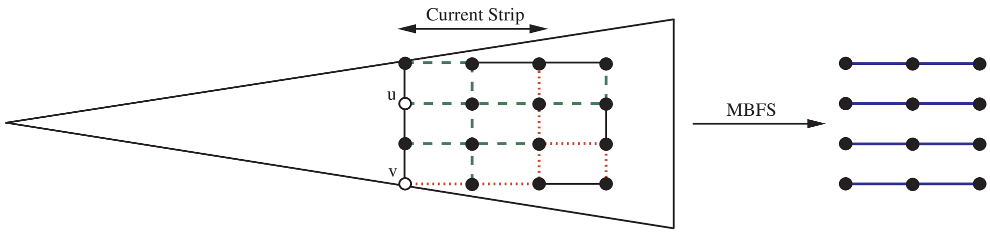

Figure 6: Suppose an underlying grid-graph. There are four source nodes and two of them (one rooted at u and the other rooted at v) become the roots of two exploration trees. Notice that the other two sources did not grow trees. Also observe that the grown trees are not BFS trees, but MBFS produces the BFS forest for that strip.

EXPLORE (src, r)**:** `EXPLORE`, where src is a vertex and r is the distance, returns a tree Z such that the BFS tree of depth r around src is entirely contained in Z (Fig. 7). The construction of these trees is similar in concept to the construction of the sparse neighborhood covers of Awerbuch, Berger, Cowen, and Peleg [2]. It proceeds in steps, maintaining the invariant that tree Y at any given step is identical to tree Z of the previous step. We view tree Y as the private territory, or kernel, of tree Z. At each step, tree Z is increased by adding to it the vertices at distance $2r$ from it. When we looking for new vertices to add to Z we are looking in the graph G' induced from G by removing all vertices in $\mathcal{Y}^r$. In this way, we prevent the invasion in other kernels of the same level). This process continues until $|Z| < 2|Y|$.

```
EXPLORE (src, r)
   Z ← RPS (r, src, ∅)
   repeat
      Y ← Z
      do twice: Z ← Z ∪ RPS (r, FRONT (Z), Z)
   until |Z| ≤ 2 · |Y|
   𝒴^r ← 𝒴^r ∪ {Y}
   𝒵^r ← 𝒵^r ∪ {Z}
   𝒴^{2r} ← 𝒴^{2r} ∪ {Z}
```

Figure 7: Local Exploration Procedure.

FRONT (F)**:** Procedure `FRONT` takes as an input a forest F and produces the set of frontier vertices, namely, vertices which have unexplored incident edges.

4 Analysis

Lemma 1 A call to $\texttt{EXPLORE}(u, r)$ results in the complete exploration of the area around a source vertex u at distance r.

Proof: Intuitively, it should be clear that if this claim is true then the correctness of the algorithm follows directly from it. We want to show now that, given a vertex u and radius r, we have indeed explored u's entire r-neighborhood after the call to EXPLORE. Suppose that the claim is not true for some call to $\texttt{EXPLORE}(u, r)$. Suppose there exists a node w at distance r or less from u which did not get explored. Since w is unvisited, the exploration from node u did not reach node w. There are two possible reasons why we may have stopped our exploration from node u before we explored all vertices at distance r or less from u. The first possibility is that w belongs to some private territory of a previous exploration. In this case, w has been explored already. The second possibility is that there was an obstacle in our path, some private territory Y, which prevented us from reaching node w and also $w \notin Y$. This implies that there exists a node v, such that $dist\,(v, w) \leq r$ and $v \in Y$, since Y is on the way from u to w and $dist\,(u, w) \leq r$ (Fig. 8). But then, consider the tree Z associated with this Y: It is always true that all the vertices on the border of Y have their r-neighborhoods contained in Z (by definition of the Y and Z trees). In this case, $w \in Z$. But in this case we have also explored node w (since we have visited it when we were building the Z). Thus, all the nodes in the r-neighborhood of u must have been explored.

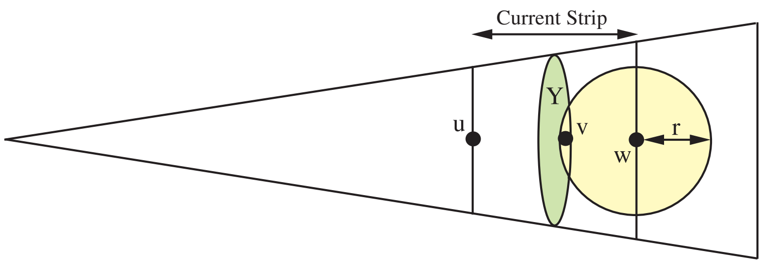

Figure 8: Given a strip for some r and a set of source vertices, let $u \in S$, and $dist\,(u, w) \leq r$

Lemma 2 A call to $\texttt{RPS}(r, S, T)$ results in the complete exploration of the area around the source nodes in the set S at distance r.

Proof: By the definition of the algorithm a call to RPS splits the current strip in half and calls procedure EXPLORE from each of the source vertices in the two new strips. In the lemma above we showed that EXPLORE indeed covers the entire r-neighborhoods around the source vertices. It becomes a straightforward observation that the entire strip is also explored. From these two lemmas we obtain:

Corollary 1 After the completion of the algorithm, the graph $G = (V, E)$ is completely explored.

Theorem 1 Algorithm RPS is a piecemeal search algorithm which completely explores graph $G = (V, E)$, given a start vertex $start$ and provided that $Max \geq 4Rad$.

Proof Sketch: From Corollary 1, it follows that the graph is explored in its entirety. To satisfy the piecemeal constraint, at any stage of the algorithm the robot needs to know a path of length no more than $2Rad$ back to $start$. By construction, the robot updates its global map every time a new piece of the graph is explored. Hence, the longest distance that we need to be able to traverse is the sum of the max lengths of the recursive exploration depths: $r + r/2 + \ldots + 1 = 2r$. Initially, $r = Rad$ and it follows that given $Max \geq 4Rad$ the robot is never farther away from the start node than $2Rad$.

Theorem 2 The overall running time of the algorithm is $O(m + n \log^2 n)$.

Proof Sketch: Recall that the running time of the algorithm is measured by the total number of edges traversed. To prove an upper bound on that number we use several straightforward properties of the sets constructed by the algorithm. Recall that when we explore the territory around a source node we build and maintain the trees Y and Z, where Y is the private territory of Z.

- $|Z| \leq 2|Y|$
- $Y_i \bigcap Y_j = \emptyset$, for trees $Y_i, Y_j \in \mathcal{Y}^r$.

Since we are going to count edge traversals, let us divide the edges of the graph in two groups: edges that are in the final BFS tree, and other edges. Observe now that non-tree edges are explored by PBFS only once. Thus, we only need to count the number of traversals made over edges which belong to the final BFS tree. We cannot bound the number of traversals on individual edges, but we can amortize over the total number of traversals. Let us associate edge costs with each level of recursion. Since we halve the radius each time we recurse, there are $\log n$ such levels. For each level of recursion we need to use DFS $\log n$ times, once for each of the $\log n$ strips created. This introduces $\log n$ charge for each of the vertices in the Z trees. This charge can be amortized against the vertices in the Y trees, so that each vertex in Y is charged by at most 2 vertices in Z sets. The vertices in Y are disjoint, and thus the total charge per level adds up to $O(n \log n)$. Over $\log n$ levels, it sums up to $O(n \log^2 n)$ total charge.

5 Conclusion and Open Problems

We have presented a near-linear algorithm for exploration of general unknown graphs. The complexity of our algorithm is $O(m + n \log^2 n)$, and thus improves on previously known results. The natural lower bound for the problem, however, is $O(m + n)$ and it is still an open question whether there exists such a linear algorithm. It is also unclear whether $O(n + m)$ is indeed a tight lower bound for the problem.

Another open problem is whether the above presented algorithm, or a modification of it, can be implemented in a distributed setting. Currently, `RPS` requires "global knowledge" to combine the results from each local exploration to finally produce the entire map of the graph. Is it possible to achieve comparable complexity with only "local knowledge?"

Acknowledgments

We thank Lenore Cowen for stimulating discussions.

References

[1] Suzanne Albers and Monika Henzinger. Exploring unknown environments. In *29th Symposium on Theory of Computing*, pages 416–425, May 1997.

[2] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear cost constructions of neighborhood covers in sequential and distributed environments and their applications. In *34th Symposium on Foundations of Computer Science*, pages 638–647, November 1993.

[3] Baruch Awerbuch, Margit Betke, Ronald Rivest, and Mona Singh. Piecemeal graph exploration by a mobile robot. In *8'th Conference on Computational Learning Theory*, pages 321–328, July 1995.

[4] E. Bar-Eli, P. Berman, A. Fiat, and P. Yan. On-line navigation in a room. In *Journal of Algorithms*, 17(3), pages 319–341, November 1994.

[5] M. Bender and D. Slonim. The power of team exploration: Two robots can learn unlabeled directed graphs. In *35th Symposium on Foundations of Computer Science*, pages 75–87, November 1994.

[6] P. Berman, A. Blum, A. Fiat, H. Karloff, A. Rosen, and M. Saks. Randomized robot navigation algorithms. In *7th Symposium on Discrete Algorithms*, pages 74–84, January 1996.

[7] Margit Betke, Ronald Rivest, and Mona Singh. Piecemeal learning of an unknown environment. In *6th Conference on Computational Learning Theory*, pages 277-286, July 1993. (Also Published in *Machine Learning*, vol.18, 2-3, pages 231-254, February 1995).

[8] Margrit Betke. Algorithms for exploring an unknown graph. Master's thesis, MIT Department of Electrical Engineering and Computer Science, February 1992. (Published as MIT Laboratory for Computer Science Technical Report MIT/LCS/TR-536, March 1992).

[9] Avrim Blum and P. Chalasani. An online algorithm for improving performance in navigation. In *34th Symposium on Foundations of Computer Science*, pages 2–11, Nov 1994.

[10] Avrim Blum, Prabhakar Raghavan, and Baruch Schieber. Navigating in unfamiliar geometric terrain. In *SIAM Journal on Computing*, 26(1), pages 110-137, February 1997.

[11] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[12] Xiaotie Deng, Tiko Kameda, and Christos H. Papadimitriou. How to learn an unknown environment. In *32nd Symposium on Foundations of Computer Science*, pages 298–303, October 1991.

[13] Xiaotie Deng and Christos H.Papadimitriou. Exploring an unknown graph. In *31st Symposium on Foundations of Computer Science*, vol.1, pages 355–361, October 1990.

[14] F. Hoffmann, C. Icking, R. Klein and K. Kriegel. A competitive strategy for learning a polygon. In *8th Symposium on Discrete Algorithms*, pages 166-174, January 1997.

[15] C. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, vol.84, pages 127–150, 1991.

[16] Nagewara S. V. Rao, Srikumar Kareti, Weimin Shi, and S. Sitharama Iyengar. Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms. Technical report, Oak Ridge National Laboratory, July 1993. ORNL/TM-12410.