6.858/18.428 Machine Learning **Lecture 26: December 14, 1994**

Lecturer: Mona Singh *Scribe: Charlie Kemp*

26.1 Outline

- Introduction to the piecemeal learning problem
- Linear time algorithm for piecemeal learning of city-block graphs
- Nearly linear time algorithm for piecemeal learning of general graphs

26.2 Introduction to the Piecemeal Learning Problem

26.2.1 Problem

- Learner must learn a complete map of its environment through exploration
- Learning must be done a *piece at a time*, with a return to start after each piece of exploration

26.2.2 Motivation

- A robot might want to refuel and drop off samples at a home location before continuing its exploration of an unknown environment. For example, a planetary exploration robot or a volcano exploration robot might have these requirements.
- Piecemeal learning can be used to model "learning on the job", in which phases of piecemeal learning help the learner improve its performance on other tasks it performs. For instance, a rookie taxi driver might explore the city between fares.

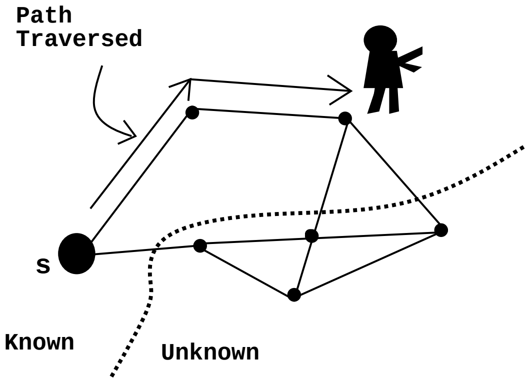

Figure 26.1: The learner knows the vertices it has visited and the edges it has traversed. The learner also knows the direction of each edge incident to the visited vertices.

26.2.3 Formal Model

The formal model of the problem consists of the specification of the environment and the learner. An example of a partially explored environment is shown in figure 26.1. The known and unknown regions represent what the robot knows about the environment after traversing the path marked by arrows. The following outline lists the significant attributes of the environment and learner in the formal model.

1. THE ENVIRONMENT

   (a) The environment is represented by a graph, $G = (V, E)$, which is finite, connected, and undirected.

   (b) The set of vertices V represents accessible locations.

   (c) The vertex $s \in V$ is the start vertex.

   (d) An edge, $(x, y) \in E$, indicates that the learner can move from x to y or back in one step.

   (e) The vertices are distinguishable.

2. THE LEARNER

   (a) What the learner knows at each step:

      i. The global position of seen vertices.

      ii. The direction of edges incident to seen vertices.

      iii. The edges it has traversed.

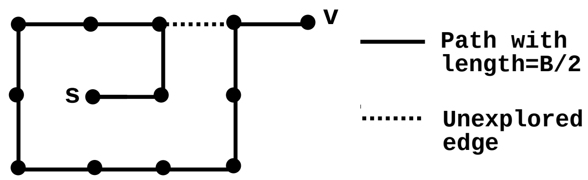

Figure 26.2: An example of a "stuck" DFS

(b) What the learner knows initially:

i. The start vertex, s.

ii. The upper bound, B, on the number of steps in an exploration phase. $B = (2 + \alpha)r$, where $\alpha > 0$ is some constant, and r is the radius of the graph. The radius of the graph is defined as the maximum of all shortest path distances between s and any vertex in G. B can be thought of as the size of the robot's gas tank. Intuitively, B is a reasonable size since it allows the learner to travel to any vertex in the graph, do some exploration, and still have enough steps to make it back to s.

(c) The learner's goal is to learn the graph efficiently in a piecemeal manner. The total cost is the number of edges traversed to learn the graph (Computational costs are not considered). Thus, the learner must traverse every edge in the graph with the least total cost possible, and must do so with the constraint that each of the its exploration phases has at most B steps, and between exploration phases it must return to the start vertex.

26.2.4 Standard Approaches

Two common methods for searching a graph are depth-first search and breadth-first search. As described below, both methods have unsatisfactory results when applied unaltered to the piecemeal learning problem.

1. Depth-first search with interrupts (DFS)

   In DFS, edges are always explored out of the most recently discovered vertex v that still has unexplored edges leaving it. As demonstrated in figure 26.2, this approach fails when the only path the robot knows to vertex v is $\frac{B}{2}$ steps away from the start vertex s. In this case the learner is unable to resume the DFS

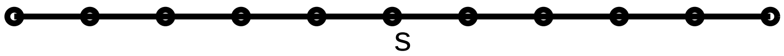

Figure 26.3: A simple graph for which the cost of BFS is quadratic in the number of edges.

exploration and still return to s without exceeding B. Note, however, that a shorter path to v exists in the graph, but the robot does not know it.

2. Breadth-first search (BFS)

   BFS explores all vertices at the same distance from s before exploring any vertices that are further away from s. This approach does not fail. However, standard BFS is inefficient as it requires $\Omega(|E|^2)$ steps. Figure 26.3 gives an environment which causes this many edge traversals. We would like a more efficient algorithm.

26.2.5 New Techniques

In order to achieve better algorithms some definitions and theorems will be helpful.

Definition 1 *An on-line search is* **optimally interruptible** *if it always knows a shortest path back to s.*

For example, BFS is optimally interruptable.

Definition 2 *A search is* **efficiently interruptible** *if it always knows a path back to s via explored edges of length at most r, the radius of the graph. (Optimally interruptable implies efficiently interruptible.)*

Theorem 1 *An efficiently interruptible algorithm for searching an undirected graph G in $T(n,m)$ running time, where n is the number of vertices and m is the number of edges in G, can be transformed into a piecemeal learning algorithm which takes time $O(T(n,m))$.*

Sketch of Proof: An efficiently interruptible algorithm is converted into a piecemeal learning algorithm by breaking its exploration into segments. Each exploration phase has at most B steps, where $B = (2+\alpha)r$. At the beginning of each phase, besides the first, the robot moves from the start vertex s to the vertex at which the original algorithm was interrupted. In the first phase the robot simply stays at s instead of moving. The robot then explores αr edges of the original algorithm, after which it ends the phase by returning to s. By the definition of an efficiently interruptable algorithm, the beginning and end of the phase take at most $2r$ steps. Thus, each phase takes $\leq 2r + \alpha r = B$ steps. For convenience, assume $\alpha = 2$. Then, the total running time is at most $(\frac{T(n,m)}{2r} + 1)4r = 2T(n,m) + 4r < 6T(n,m)$. ■

With this theorem we can "ignore" the piecemeal constraint and focus on efficiently interruptable searches.

26.3 Linear Time Algorithm for Piecemeal Learning of City-Block Graphs

The general problem of efficiently interruptible searches is difficult so we will first focus on city-block graphs.

City-Block Graphs: As shown in the figure 26.4, a city-block graph is a grid graph containing some nontouching axis-parallel rectangular obstacles. In other words, city-block graphs are rectangular planar graphs in which all edges are either vertical or horizontal, and in which all faces are axis-parallel rectangles whose opposing sides have the same number of edges.

As can be seen in the figure 26.4, BFS produces patterns of search which look like waves from a pebble dropped at the start vertex s. Each wavefront consists of all the vertices a certain number of steps from s. By considering these wavefronts we will try to efficiently mimic BFS.

Notation:

- Let $\delta(v, v')$ denote the length of the shortest path between v and v'. Let $d[v] = \delta(v, s)$ which is the length of the shortest path from v to s or vice versa.
- A *wavefront* w is an ordered list of explored vertices $\langle v_1, v_2, ... v_m \rangle$, $m \geq 1$ such that $\forall i (d[v_i] = d[v_1]$ and $\delta(v_i, v_{i+1}) \leq 2)$

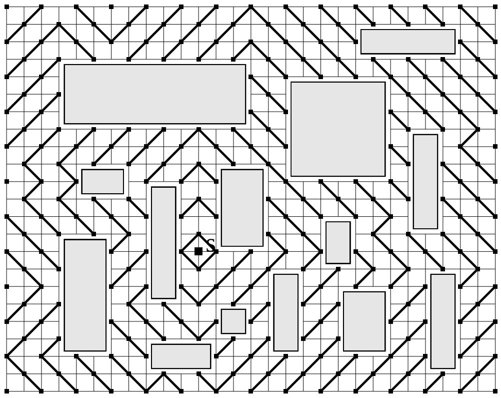

Figure 26.4: An example of a city-block environment explored by BFS, showing only "wavefronts" at odd distance to s.

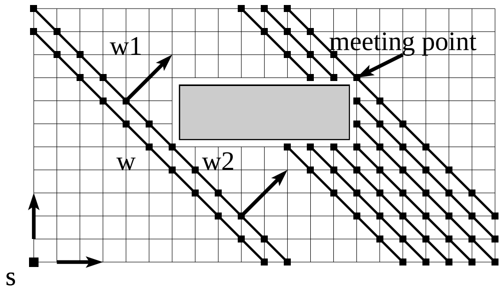

Figure 26.5: Wave w is shown splitting into two waves $w1$ and $w2$ which merge on the other side of the obstacle at the meeting point. The wavefronts in wave $w1$ are siblings of the wavefronts in wave $w2$.

We need the following terminology to explain the algorithm. The wavefront terms below are illustrated in figure 26.5. A more formal description of this algorithm can be found in [1].

Wavefront Terms:

- **successor**
  A wavefront at a distance t generates a *successor* wavefront at distance $t + 1$
- **wave**
  A sequence of successor wavefronts
- **split**
  A wave *splits* when it hits an obstacle. In other words, a wavefront has two

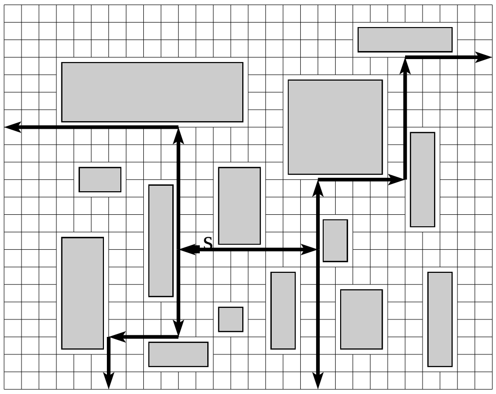

Figure 26.6: An example of the four monopaths

successor wavefronts when the successors are divided by an obstacle. Refer to the figure 26.5 and wavefront w.

- **merge**
  On the far side of an obstacle two waves may *merge* into one wave. Wavefronts of both waves have connected successors, so there is only one successor wavefront for the two wavefronts.

- **sibling wavefronts**
  Two wavefronts are *sibling* wavefronts if they each have exactly one endpoint on the same obstacle and if the waves to which they belong merge on the far side of that obstacle.

- **meeting point**
  The point on an obstacle where the waves first meet is called the *meeting point* of the obstacle (see Figure 26.5).

Monotone Paths: By sending out four monotone paths the algorithm divides the map into four regions which can be considered separately. The monotone paths are east-north, east-south, west-north, and west-south. The east-north path starts by moving to the east from the source. The path then moves north after it meets the first obstacle. At the next obstacle the path moves east again and continues to alternate in this way for all obstacles encountered. The three other monotone paths are created in much the same way (see figure 26.6). The paths form a northern region, eastern region, southern region and western region. For the rest of the notes, discussion focuses on the northern region, although the results are analagous for the three other regions.

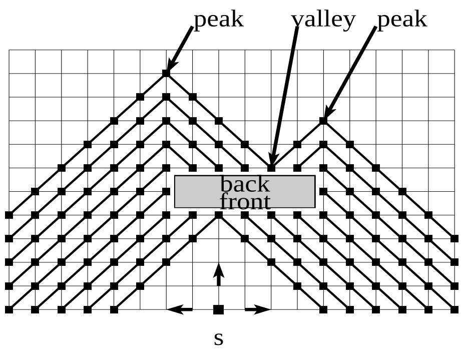

Figure 26.7: Shapes of wavefronts in northern region. Illustration of peaks, valleys, and front and back of an obstacle. The meeting point is the lowest point in the valley.

The following properties of wavefronts, illustrated in figure 26.7, can be shown by induction.

Properties of Wavefronts:

1. A wavefront can only consist of diagonal segments.
2. Shapes of wavefronts stay the same except around obstacles.
3. Wavefronts hit at the front of an obstacle.
4. Wavefronts meet at the back of an obstacle at the *meeting point*, which is the bottom of a valley.
5. Peaks form above corners of obstacles. In the northern region, all peaks point north. There are no sideways peaks or flat wavefronts.

26.3.1 The Wavefront Algorithm and its Properties

The wavefront algorithm attempts to mimic BFS in an efficient way. A wavefront and its successor are close to one another. So when possible the robot chooses to explore the successor of the wavefront that it just finished exploring. In this way, the robot avoids spending a lot of time relocating between wavefronts. However, the robot explores so that it still knows the relevant shortest paths. The challenge is to create correct wavefront exploration patterns while constructing them in a different order from standard BFS. For instance, if the robot naively pushes wavefronts ahead,

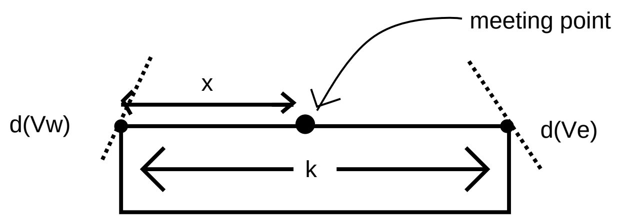

Figure 26.8: Illustration of meeting point calculation where v_w is the northwest vertex, v_e is the northeast vertex, k is the length of the obstacle, and x is the distance from v_w to the meeting point.

always following one wave, incorrect wavefronts will be formed at the obstacles. This problem arises because siblings interact with each other in forming correct wavefronts. In particular, sibling wavefronts should merge at their meeting point on an obstacle. If a wavefront is pushed beyond the meeting point, then it is no longer a correct wavefront.

To solve the problem of pushing a wavefront beyond a meeting point we can compute the location of meeting points. In the northern region, the meeting point on an obstacle is a point on the back of an obstacle such that there is a shortest path if you go either east or west. Once we know the shortest path to the back corners we can calculate the meeting point. Thus, as shown in figure 26.8, $x + d[v_w] = (k - x) + d[v_e]$ where v_w is the northwest vertex, v_e is the northeast vertex, k is the length of the obstacle, and x is the distance from v_w to the meeting point. So, $x = \frac{k+d[v_e]-d[v_w]}{2}$.

Operationally this means:

1. We do not expand a wavefront if it is at a back corner and its sibling has not reached the back corner. Since we do not know the meeting point, we do not want to expand this wavefront anymore (See figure 26.9).

2. We do not expand a wavefront if it has reached a meeting point and its sibling has not reached the meeting point, since we want the wavefronts to merge properly (See figure 26.10).

3. Each wavefront should be pushed forward until it is blocked and then another wavefront should be pushed forward.

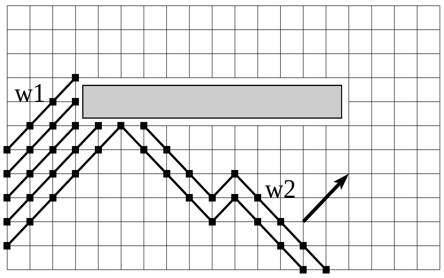

Figure 26.9: An example of the first type of blocking

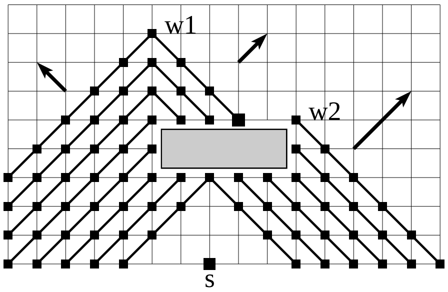

Figure 26.10: An example of the second type of blocking

We now give a sketch of the MAIN-ALGORITHM:

1. Create monotone paths
2. EXPLORE each region (North, South, East, and West).

To EXPLORE a region:

1. Maintain an ordered list L of wavefronts.
2. $L = \langle s \rangle$
3. $current\_wavefront = s$
4. REPEAT the following UNTIL L is empty
   (a) Expand $current\_wavefront$ to successor wavefront w and update L
   (b) Calculate any new meeting points
   (c) If w can be merged with another wavefront then merge and update L

(d) If w hits obstacles split w and update L

(e) Set $current\_wavefront$ to be the unblocked wavefront nearest to the learner

Theorem 2 *The wavefront algorithm is an optimally interruptible search algorithm for city-block graphs.*

Sketch of Proof: We can show that there is always a wavefront that is not blocked, so the algorithm explores the entire region. We can also show that by constructing wavefronts the algorithm maintains optimal interruptibility. ■

Theorem 3 *The wavefront algorithm is linear in the number of edges in the city-block graph.*

Sketch of Proof: By theorem 2, the algorithm is optimally interruptible. We can show that it runs in linear time. So, by theorem 1 it can be converted to a linear time piecemeal learning algorithm for grid graphs. ■

26.4 Nearly Linear Time Algorithm for Piecemeal Learning of General Graphs

We now turn to the problem of piecemeal learning of *general* undirected graphs. The best algorithm known for this problem runs in time $O(E + V^{1+o(1)})$. The rest of the notes will describe a simpler version of the algorithm, called the STRIP-EXPLORE algorithm, which is $O(E + V^{1.5})$. The STRIP-EXPLORE algorithm gives an *efficiently interruptable* algorithm based on BFS.

Terminology:

- **layer**
  A *layer* in a BFS tree consists of vertices that have the same shortest path distance to the start vertex.

- **frontier vertex**
  A *frontier vertex* is a vertex that is incident to unexplored edges.
- **expanded**
  A frontier vertex is *expanded* when the learner has traversed all unexplored edges incident to it.

Algorithms:

BFS on general graphs gives an $O(rV + E)$ algorithm. Expansion of all the vertices takes time $O(E)$. Visiting all the vertices at layer i takes time $O(V)$. A basic BFS algorithm is shown below.

BASIC-BFS(s, r)

1. for $i = 0$ to $r - 1$
   (a) for each vertex u at a distance i from s, relocate to u and then expand u

LOCAL-BFS(s, L) is a slightly modified BFS algorithm which only goes to distance L from the source vertex s and only relocates to vertices that have unexplored neighbors:

LOCAL-BFS(s, L)

1. for $i = 0$ to $L - 1$
   (a) for each vertex u at a distance i from s
      i. if u has any unexplored neighbors then relocate to u and expand u

STRIP-EXPLORE uses LOCAL-BFS to explore strips of width L. This is more efficient than using BASIC-BFS because there is some overlap between the BFS trees. STRIP-EXPLORE only passes through an overlap region more than once when it is necessary to get to the frontier vertices. Figure 26.11 depicts the STRIP-EXPLORE algorithm.

STRIP-EXPLORE(s, L, r)

1. *sources*= $\langle s \rangle$
2. *numstrips*= $\lceil r/L \rceil$

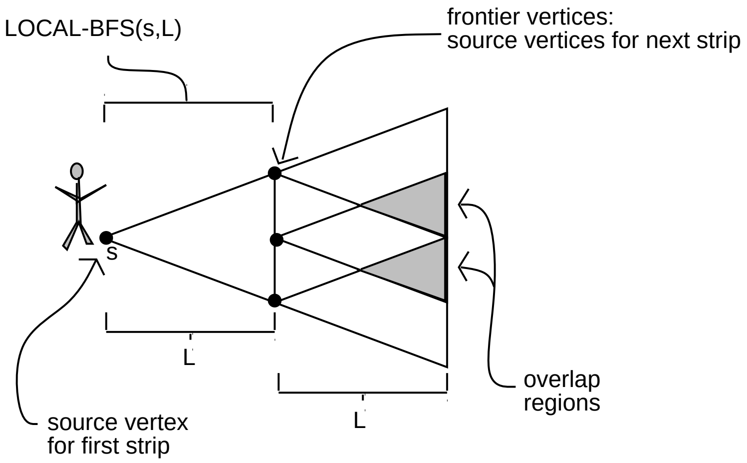

Figure 26.11: Visual representation of the STRIP-EXPLORE algorithm

3. For $i = 1$ to *numstrips* do
   (a) For each $u \in$ *sources* **do**
      i. Relocate to u
      ii. LOCAL-BFS(u, L)
   (b) *sources* = all frontier vertices

STRIP-EXPLORE analysis:

1. For any relocation between source vertices in the same strip there are at most $2V$ edge traversals. Since there are at most $\frac{r}{L}$ strips, there can be no more than $\frac{2rV}{L}$ edge traversals for relocating between source vertices.
2. To expand all vertices and explore all edges is $2E$ edge traversals.
3. Within a call of LOCAL-BFS, the learner only relocates to a vertex if the learner is about to expand it. Thus, the learner traverses at most $2L$ edges to relocate during a call to LOCAL-BFS. Futhermore, since LOCAL-BFS only relocates to vertices with unexplored neighbors, each vertex in the graph is relocated to at most once for expansion. Consequently, there are at most a total of $2LV$ traversals due to relocations within calls to LOCAL-BFS.

Based on the above analysis we see that the total number of edge traversals is at most $\frac{2rV}{L} + 2LV + 2E$. r is less than or equal to V. Therefore, this expression is minimized when $L = \sqrt{r}$, which shows that the STRIP-EXPLORE algorithm runs in time $O(V^{1.5} + E)$. The algorithm is also efficiently interruptible which means it can be easily converted to a piecemeal algorithm.

A more complicated recursive version [2] of this algorithm gives an $O(E + V^{1+o(1)})$ piecemeal learning algorithm.

References

[1] Margrit Betke, Ronald Rivest and Mona Singh. Piecemeal Learning of an Unknown Environment. *Proceedings of the 1993 Conference on Computational Learning Theory* (pp.277-286). Santa Cruz, CA: 1993. (Published as MIT AI-Memo 1474, CBCL-Memo 93; to be published in Machine Learning, vol. 28, 2–3, February 1995.)

[2] Baruch Awerbuch, Margrit Betke, Ronald Rivest, and Mona Singh. Piecemeal Graph Exploration by a Mobile Robot. Published as MIT/LCS/TM-516, 1995. To appear in *Proceedings of the 1995 Conference on Computational Learning Theory*.