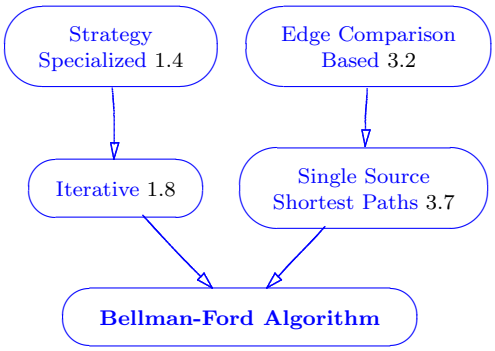
3.17 Bellman-Ford Algorithm

Section authors: Noboru Obata, Lei Zhang, and Huai Kai Lin.



Refinement of: Single Source Shortest Path (§3.7), Iterative Algorithm (§1.8).

Prototype:

```
template <class Graph, class Size,
          class P, class T, class R>
bool bellman_ford_shortest_paths
(Graph& g, Size N, const bgl_named_params<P, T, R>& params)
```

Input: A graph `g`, either directed and undirected, with `N` vertices, and a named parameter `params` which may contain the following property maps.

- An edge weight map w_map.
- A vertex distance map d_map. This must be initialized such that $d\_map[u] = \infty$ for all vertices u in the graph, except for the source vertex s, $d\_map[s] = 0$, by which the source vertex is identified.
- A predecessor map p_map (optional). This must be initialized to have $p\_map[u] = u$ for all vertices u in the graph.

Output: TRUE if the graph contains no negative-weight cycles that are reachable from the source; FALSE otherwise.

Effects: For every vertex u in the graph, $d\_map[u]$ is the shortest path weight from the source vertex s. (If a vertex u is not reachable from the source vertex, $d\_map[u] = \infty$.)

If a vertex u is reachable from the source vertex s, then $p\_map[u] = v$ and $u \neq v$, where v is the parent node of u in the minimum spanning tree rooted at s. If either $u = s$ or a vertex u is not reachable from s, then $p\_map[u] = u$.

Pseudocode: The implementation in BGL has an important change from the algorithm presented in CLRS. A flag *relaxed* is turned TRUE only if an edge is relaxed in the loop of lines 7–12. If no edges are relaxed, then the outer loop of lines 5–16 is terminated immediately. This improvement dramatically decreases the number of outer loop iterations actually executed.

BELLMAN-FORD-BGL(G, w, s)

```
 1: for each vertex v ∈ V[G] do
 2:    d[v] ← ∞
 3: end for
 4: d[s] ← 0
 5: for i ← 1 to |V[G]| do
 6:    relaxed ← FALSE
 7:    for each edge (u, v) ∈ E[G] do
 8:       if d[v] > d[u] + w(u, v) then     ▷ Relaxation call
 9:          d[v] ← d[u] + w(u, v)    ▷ Relaxation step
10:          relaxed ← TRUE
11:       end if
12:    end for
13:    if relaxed = FALSE then
14:       exit the loop
15:    end if
16: end for
17: for each edge (u, v) ∈ E[G] do
18:    if d[v] > d[u] + w(u, v) then
19:       return FALSE
20:    end if
21: end for
22: return TRUE
```

Asymptotic complexity:

- Average case (random data): $O(|V||E|)$
- Worst case: $O(|V||E|)$

Complexity in terms of operation counts: The complexity of the Bellman-Ford algorithm depends on the number of edge examinations, or relaxation calls (line 8). (Note this is different from relaxation steps which refer to the actual changes performed in line 9.) As mentioned, the number of relaxation calls can be smaller than $|V||E|$ with the BGL implementation. In fact, it is much smaller than $|V||E|$ in the average case.

The first table shows the number of relaxation calls for random directed graphs with non-negative random edge weights, allowing self-edges and parallel edges. Vertices, edges, and operation counts are shown in thousands. For example, the top-left cell indicates that operation counts are 20 with $V = 10$ and $E = 10$.

Relaxation calls:

	Vertices				
Edges	**0.01**	**0.1**	**1**	**10**	**100**
0.01	0.02	0.01	0.01	0.01	0.01
0.1	0.4	0.3	0.1	0.1	0.1
1	3.3	6.3	3.6	1.1	1.0
10	33.7	64.7	91.0	31.0	10.7
100	225.7	794.3	900.0	1238.6	371.4

† Directed graph, edge weights $[0 : 1000]$, average of seven attempts.

It is hard to create large random graphs with negative edge weights because the they tend to have negative-weight cycles. The second table shows the results with random graphs with some negative weight edges.

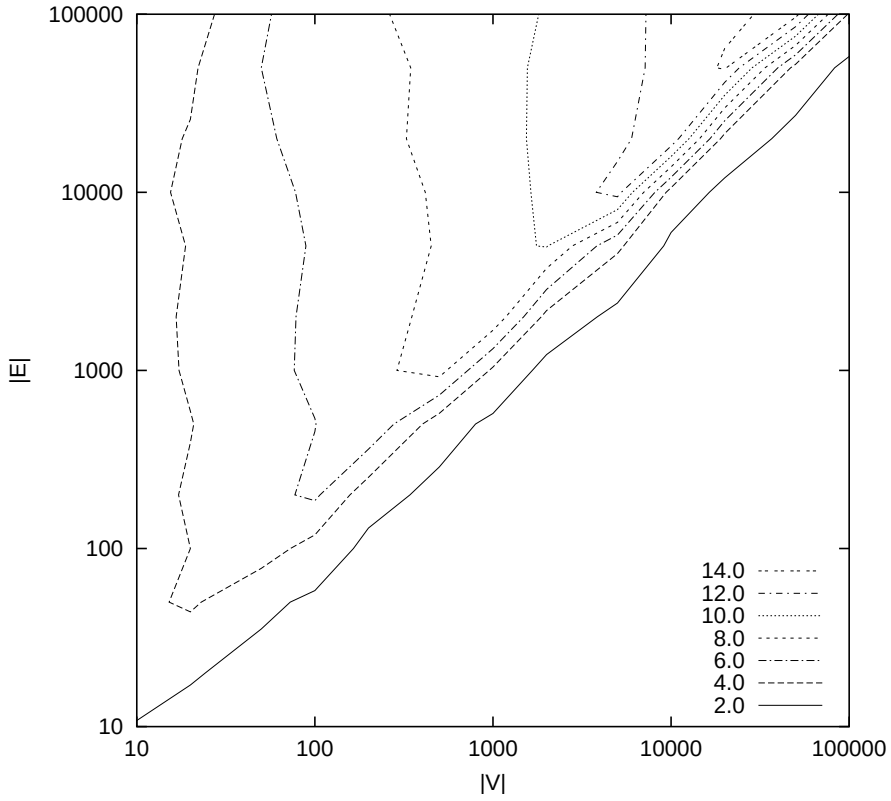
	Vertices				
Edges	**0.01**	**0.1**	**1**	**10**	**100**
0.01	0.03	0.01	0.01	0.01	0.01
0.1	0.3	0.2	0.1	0.1	0.1
1	4.1	6.0	2.4	1.0	1.0
10	—	—	92.9	28.6	11.4
100	—	—	—	1357.1	357.1

† Directed graph, edge weights $[-10 : 1000]$, average of successful seven attempts.

Formulas for average case:

$$\text{Relaxation calls: } \begin{cases} 1.13|E| & \text{if } |E| < |V|, \\ 0.95|E| \lg |V| & \text{if } |E| > |V|. \end{cases}$$

Let L be the number of the outer loop (lines 5–16) executed. Then, the number of relaxation calls is represented exactly by $L|E|$. So the analysis is done in terms of L, which shows interesting variations as shown in the following contour graph. In the region $|E| < |V|$, L is very close to 1 because the average size of each connected component is less than 1 and so few edges are relaxed. In the region $|E| > |V|$, however, L grows in proportion to $\lg |V|$.



Links to: 3d graph animation and curve fitting animation.

Worst case operation counts: If the graph contains a negative-weight cycle that is reachable from the source vertex, the algorithm shows the worst case behavior. Again, vertices, edges, and operation counts are shown in thousands.

Relaxation calls:

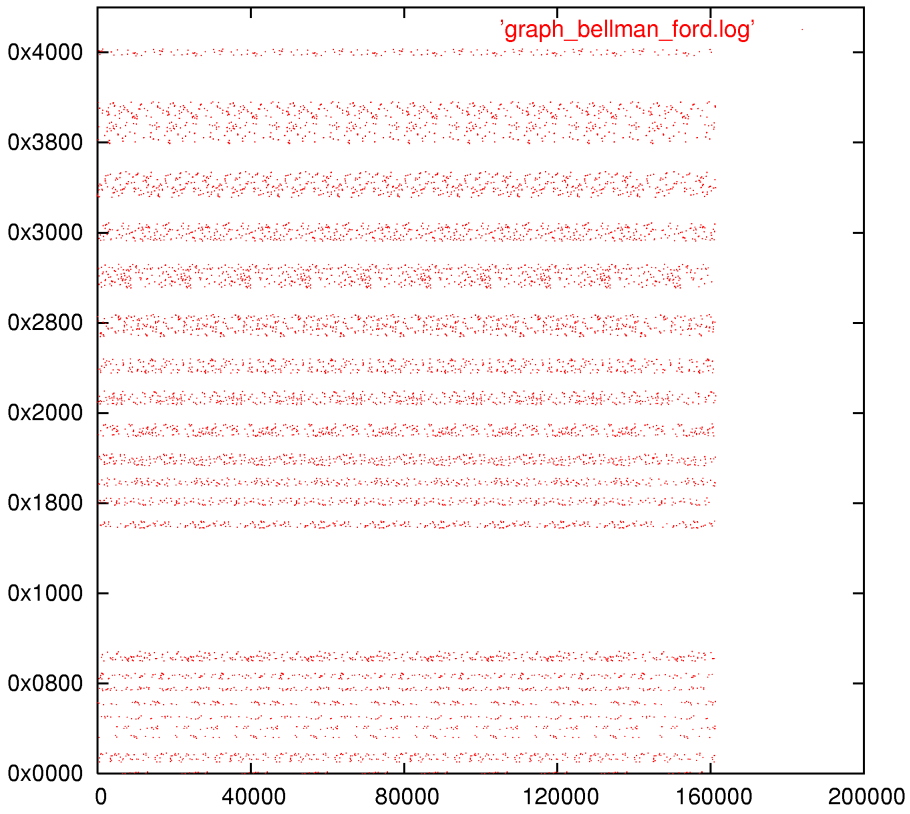
	Vertices			
Edges	**0.01**	**0.1**	**1**	**10**
0.01	0.1	1.0	10.0	100.0
0.1	1.0	10.0	100.0	1000.0
1	10.0	100.0	1000.0	10000.0
10	100.0	1000.0	10000.0	100000.0

† Directed graph, edge weights $[-1000 : -10]$, maximum of seven attempts.

Formulas for worst case:

Relaxation calls: $1.00|V||E|$

Iterator trace plot:



The plot shows the memory access pattern of the Bellman-Ford algorithm processing a directed graph with 1000 vertices and 4000 edges in the adjacency list representation (`vecS`, `vecS`). Only memory accesses to the graph data structure are drawn, and the addresses are shown relative to the smallest one. Red dots are distributed irregularly because edge vectors are allocated dynamically. The Bellman-Ford algorithm makes references to all edges at every loop of lines 7–12, which is repeated 9 times in this graph. Since the last loop (lines 17–21) makes a similar memory access, 10 repetitions of the same access patterns may be found if the plot is examined carefully. In the worst case, the same access patterns are repeated $|V|$ times.