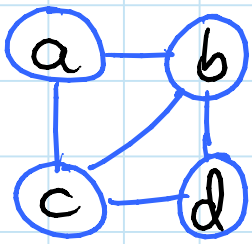
6.006	Lecture 10	Oct. 11, 2018

TODAY: BFS / Unweighted Shortest Paths
- unweighted shortest-paths problem
- graph representations
- breadth-first search algorithm

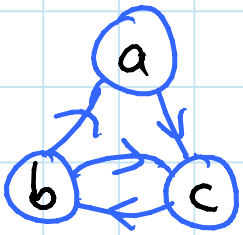
Recall: graph $G=(V,E)$
- V = set of vertices (arbitrary labels)
- E = set of edges i.e. vertex pairs (u,v)
  - ordered pair ⇒ directed edge & graph
  - unordered pair ⇒ undirected

e.g.



UNDIRECTED

$V=\{a,b,c,d\}$
$E=\{\{a,b\},\{a,c\},\{b,c\},\{b,d\},\{c,d\}\}$



DIRECTED

$V=\{a,b,c\}$
$E=\{(a,c),(b,c),(c,b),(b,a)\}$

- path $p = v_1 \to v_2 \to \cdots \to v_k$ from v_1 to v_k if $(v_i, v_{i+1}) \in E$ for $1 \le i < k$
  - can repeat vertices and/or edges
  - path length $l(p) = k-1$ = # edges

(Unweighted) shortest path from $u \in V$ to $v \in V$
= a path of minimum length from u to v
(among all paths from u to v)
- length of such a path is:
- shortest-path length $\delta(u,v)$

$$\begin{cases} \min\{\ell(p) \mid \text{path } p \text{ from } u \text{ to } v\} \\ \infty \quad \text{if no path from } u \text{ to } v \end{cases}$$

Shortest-paths problems:
- single-pair: $\delta(s,t)$, and such a path
- single-source: $\delta(s,v)$ for all $v \in V$, and shortest-path tree containing a shortest path from s to every $v \in V$
- all-pairs: $\delta(u,v)$ for all $u,v \in V$

- turns out single-pair & single-source problems seem to have equal difficulty so we focus on the latter
(all-pairs is harder but reduces to multiple single-sources)

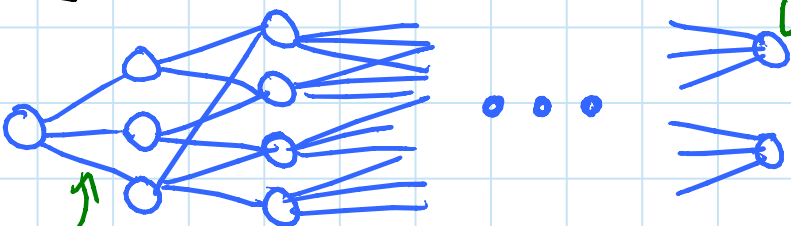
Applications: *many*

- web crawling (how Google finds pages)
- social networking (Facebook friend finder)
- Bacon/Erdős numbers
- simple network routing
- solving puzzles & games

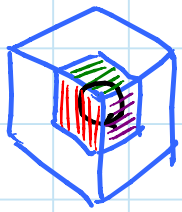
Pocket Cube: 2×2×2 Rubik's cube

- configuration graph:
  - vertex for each possible state
  - edge for each basic move (e.g., 90° turn) from one state to another
  - undirected: moves are reversible

11 for 2×2×2 ← diameter ("God's Number") → "hardest configs"
20 for 3×3×3
$\Theta(n^2/\lg n)$ for $n \times n \times n$
[Demaine, Demaine, Eisenstat, Lubiw, Winslow 2011]



- #vertices $= 8! \cdot 3^8 = 264{,}539{,}520$
  - $8!$: 8 cubelet in arbitrary positions
  - 3^8: each cubelet has 3 possible twists



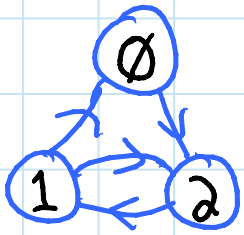
- $\times \frac{1}{24}$ if we remove cube symmetries
- $\times \frac{1}{3}$ actually reachable (3 conn. components)

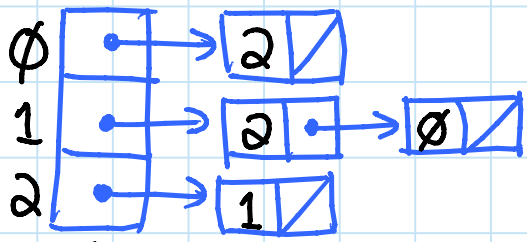
Graph representation (OS): adjacency sets
for each vertex $u \in V$, store u's (outgoing) neighbor set $Adj(u) = \{v \in V \mid (u,v) \in E\}$
(just outgoing edges if directed)

- which set data structure?
  - linked lists ("adjacency lists")
    generally suffice: mostly need iter()
  - hash table $\Rightarrow$ can test $(u,v) \stackrel{?}{\in} E$ in $O(1)$ expected time
  - generally $\Theta(V+E)$ space ("linear space")
  - implicit: compute on the fly (e.g. Rubik)

Array rep.: assume $V = \{0, 1, \ldots, |V|-1\}$
$\Rightarrow$ Adj is an array of $|V|$ sets e.g. linked lists

e.g.





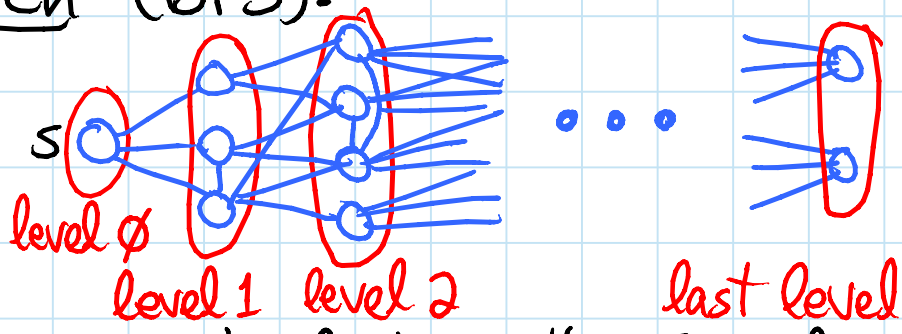
supports multiple graphs on same vertices

Dictionary rep.: assume vertices $\in V$ hashable
$\Rightarrow$ Adj is a dictionary mapping $u \in V$ to sets

Object rep.: object for each vertex $u \in V$
u.Adj stores $Adj(u)$
- can also represent edges by objects

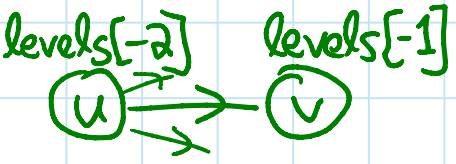
Breadth-first search (BFS):

explore graph level by level from s

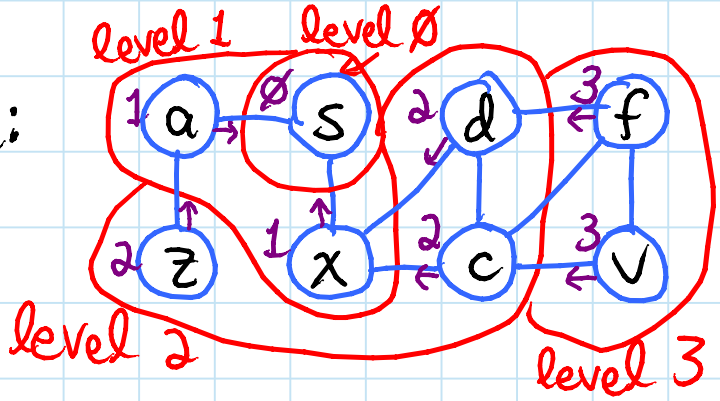


- level $\emptyset = \{s\}$
- level i = vertices reachable by path of i edges but not fewer
- build level $i > \emptyset$ from level $i-1$ by trying all outgoing edges, but ignoring vertices from previous levels

```
BFS(Adj, s):
    d = {s: 0, rest: ∞}
    parent = {s: s}                    (root has self as parent)
    levels = [[s]]
    while levels[-1]:
        levels.append([])
        for u in levels[-2]:
            for v in Adj[u]:
                if v not in parent:    (not yet seen)
                    d[v] = d[u] + 1 = δ(s,v) = len(levels) - 1
                    parent[v] = u
                    levels[-1].append(v)
```



Example:



$frontier_0 = \{s\}$
$frontier_1 = \{a, x\}$
$frontier_2 = \{z, d, c\}$
$frontier_3 = \{f, v\}$
(not x, c, d)

Analysis:

- vertex v enters levels[-1] (& then levels[-2]) only once (because parent[v] then set)
  - base case: $v = s$

$\Rightarrow$ Adj[v] looped through only once

- time $= \sum_{v \in V} |Adj[v]| = \begin{cases} |E| & \text{for directed graphs} \\ 2|E| & \text{for undirected graphs} \end{cases}$

$\Rightarrow O(E)$ time for actual search

- $O(V+E)$ to set $d[v] = \infty$ for unreachable v

↖ "LINEAR TIME"

Shortest paths:

- $d[v] = \delta(s, v)$ for all $v \in V$
- parent pointers form shortest-path tree

$\Rightarrow$ to find shortest path, take v, parent[v], parent[parent[v]], etc., until s (or None)

Google maps

from: 32 Vassar Street, Cambridge, MA to:Times Square, New York, NY

Search Maps

Get Directions | My Maps

Print | Send | Link

32 Vassar St, Cambridge, Middlesex, Massach

Times Square, New York, NY

Add Destination - Show options

Get Directions

Driving directions to Theater District - Times Square, New York, NY

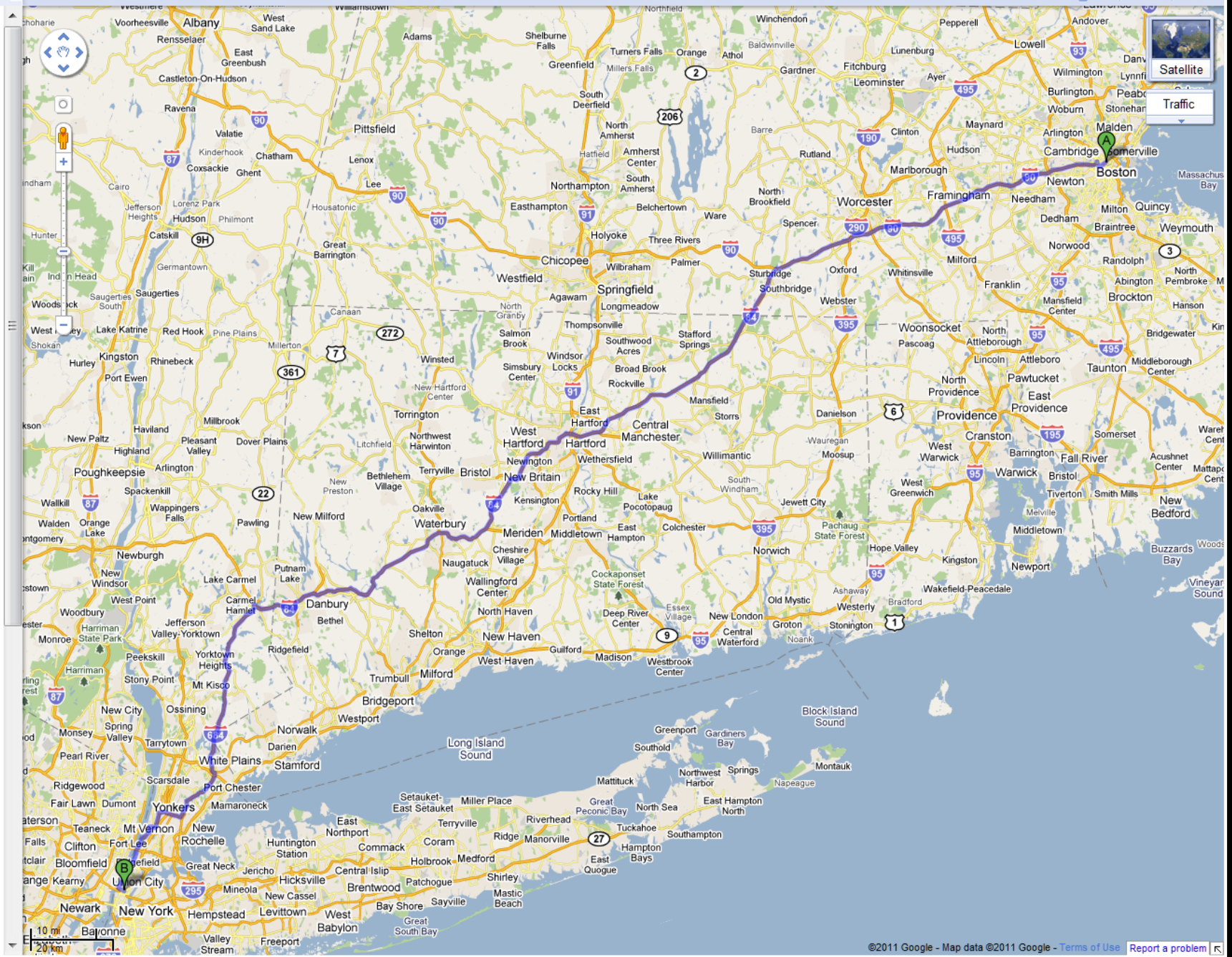
Suggested routes

1.	**I-84 W** **221 mi**	**3 hours 53 mins**
2.	I-90 W 209 mi	3 hours 58 mins
3.	I-395 S and I-95 S 227 mi	4 hours 13 mins

This route has tolls.

32 Vassar St
Cambridge, MA 02139

	1. Head **southwest** on **Vassar St**	1.0 mi
	2. Turn right at **Memorial Dr**	1.0 mi
	3. Turn left at **Western Ave**	0.1 mi
	4. Turn left at **Soldiers Field Rd**	0.2 mi
	5. Take the **I-90** ramp Toll road	0.3 mi
	6. Keep right at the fork and merge onto **I-90 W** Partial toll road	52.1 mi
	7. Take exit **9** to merge onto **I-84 W** toward **US-20/Hartford/New York City** Partial toll road	109 mi
	8. Take exit **20** for **I-684** toward **NY-22/White Plains/Pawling**	0.1 mi
	9. Keep left at the fork and merge onto **I-684 S**	29.1 mi
	10. Merge onto **Hutchinson River Pkwy S**	7.7 mi
	11. Continue onto **Cross County Pkwy** (signs for **George Washington Bridge**)	4.7 mi



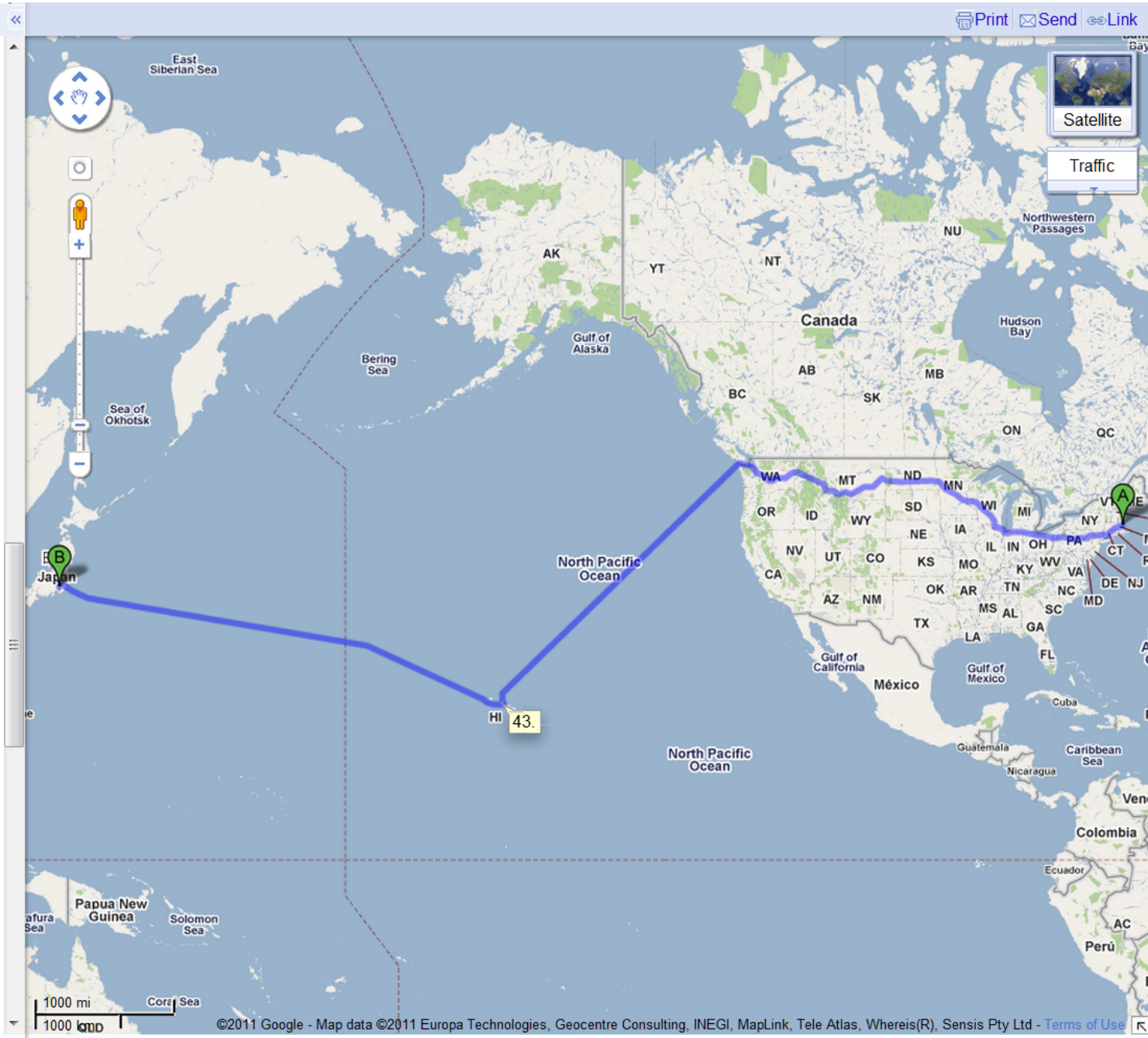
You Can't Get There From Here

Google maps | from 32 Vassar Street, Cambridge, MA to:tokyo japan | Search Maps

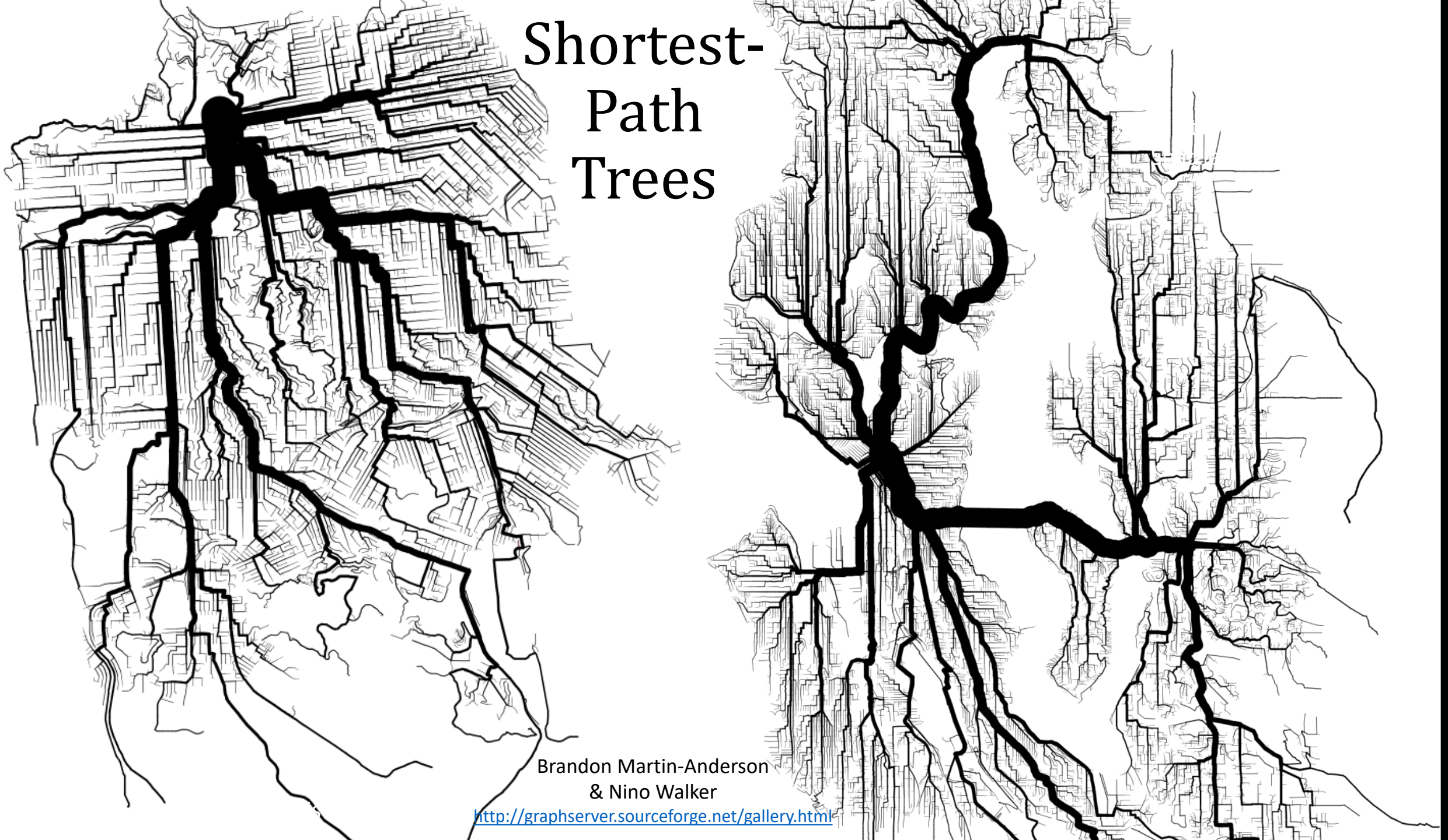
Get Directions | My Maps

Print | Send | Link

36. Take exit **13B** toward **Halawa Hts. Stadium** — 0.3 mi
37. Merge onto **I-H-201 E** — 4.1 mi
38. Merge onto **I-H-1 E** — 4.1 mi
39. Take exit **23** for **Punahou St** toward **Waikiki/Manoa** — 0.2 mi
40. Turn right at **Punahou St** — 0.1 mi
41. Take the 1st right onto **S Beretania St** — 0.1 mi
42. Take the 1st left onto **Kalakaua Ave** — 1.9 mi
43. Kayak across **the Pacific Ocean**
Entering Japan — 3,879 mi
44. Turn left toward **県道275号線** — 0.4 mi
45. Turn left toward **県道275号線** — 358 ft
46. Turn left toward **県道275号線** — 0.2 mi
47. Turn right at **県道275号線** — 0.1 mi
48. Turn left at **国道125号線** — 499 ft
49. Turn right at **県道24号線** — 0.6 mi
50. Turn left at **千束町（交差点）** onto **国道354号線** — 2.0 mi
51. Turn right at **中村陸橋下（交差点）** to stay on **国道354号線** — 1.0 mi
52. Take the ramp to **常磐自動車道**
Toll road — 0.3 mi
53. Keep left at the fork, follow signs for **東京** and merge onto **常磐自動車道**
Toll road — 23.8 mi
54. Take exit **三郷ＪＣＴ** on the right toward **首都高・銀座・湾岸線**



Shortest-Path Trees



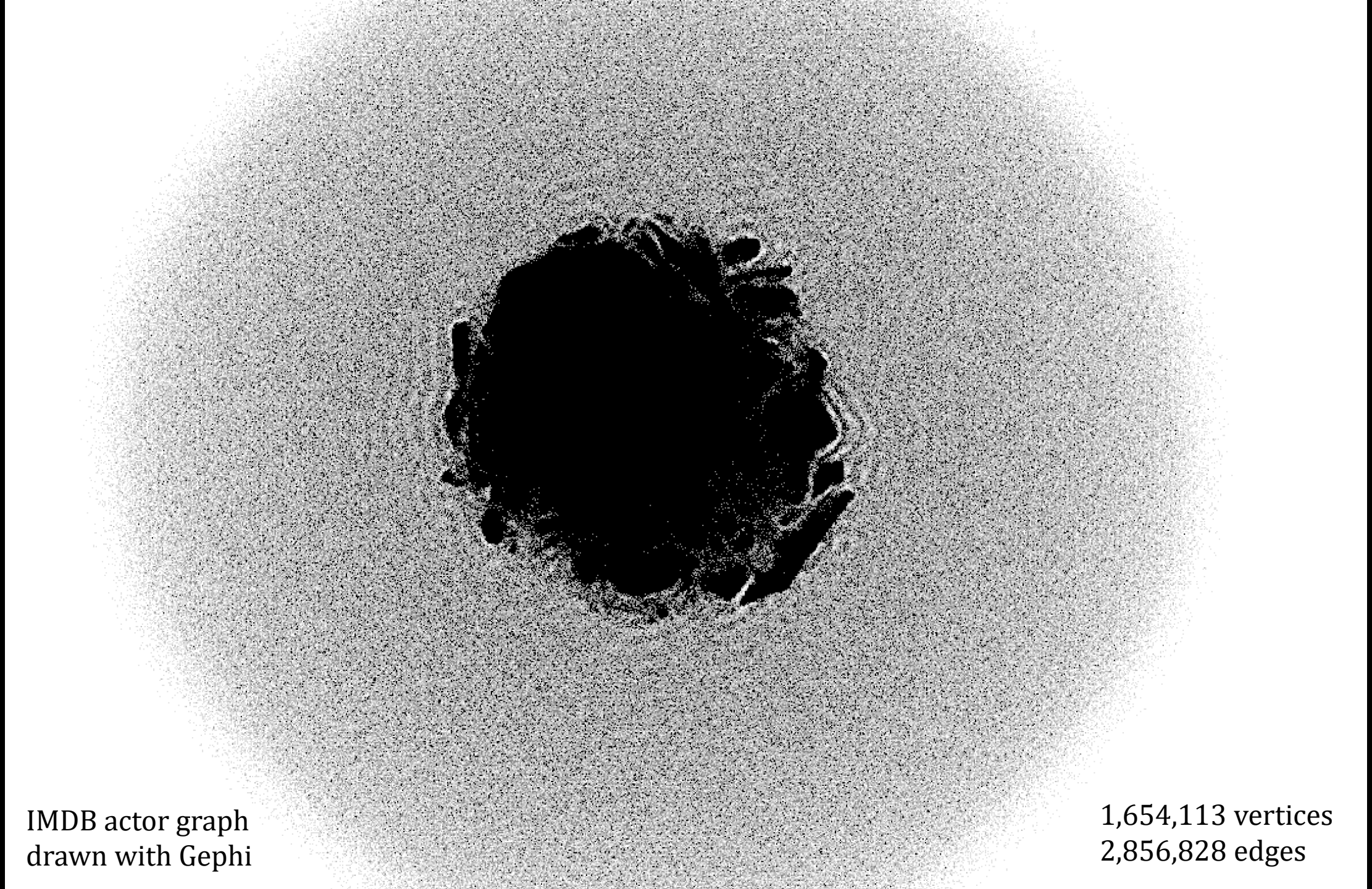
Brandon Martin-Anderson
& Nino Walker
http://graphserver.sourceforge.net/gallery.html

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0    192.168.3.1    192.168.3.41     281
          0.0.0.0          0.0.0.0       10.8.0.1        10.8.0.2     286
         10.8.0.0  255.255.255.252         On-link        10.8.0.2     286
         10.8.0.2  255.255.255.255         On-link        10.8.0.2     286
         10.8.0.3  255.255.255.255         On-link        10.8.0.2     286
        127.0.0.0        255.0.0.0         On-link         127.0.0.1     306
        127.0.0.1  255.255.255.255         On-link         127.0.0.1     306
  127.255.255.255  255.255.255.255         On-link         127.0.0.1     306
      169.254.0.0        255.255.0.0         On-link    169.254.112.65     276
   169.254.112.65  255.255.255.255         On-link    169.254.112.65     276
  169.254.255.255  255.255.255.255         On-link    169.254.112.65     276
      192.168.1.0    255.255.255.0       10.8.0.1        10.8.0.2      30
      192.168.3.0    255.255.255.0         On-link      192.168.3.41     281
    192.168.3.41  255.255.255.255         On-link      192.168.3.41     281
   192.168.3.255  255.255.255.255         On-link      192.168.3.41     281
        224.0.0.0        240.0.0.0         On-link         127.0.0.1     306
        224.0.0.0        240.0.0.0         On-link    169.254.112.65     276
        224.0.0.0        240.0.0.0         On-link        10.8.0.2     286
        224.0.0.0        240.0.0.0         On-link      192.168.3.41     281
  255.255.255.255  255.255.255.255         On-link         127.0.0.1     306
  255.255.255.255  255.255.255.255         On-link    169.254.112.65     276
  255.255.255.255  255.255.255.255         On-link        10.8.0.2     286
  255.255.255.255  255.255.255.255         On-link      192.168.3.41     281
===========================================================================
Persistent Routes:
  Network Address          Netmask  Gateway Address  Metric
          0.0.0.0          0.0.0.0       10.8.0.1  Default
```

The Oracle of Bacon

Erik D. Demaine has a Bacon number of 3. Find a different link

Erik D. Demaine
was in
The Man Who Saved Geometry (2009) (TV)
with
Benoît B. Mandelbrot
was in
The Revenge of the Dead Indians (1993)
with
Yoko Ono (I)
was in
Imagine New York (2003)
with
Kevin Bacon

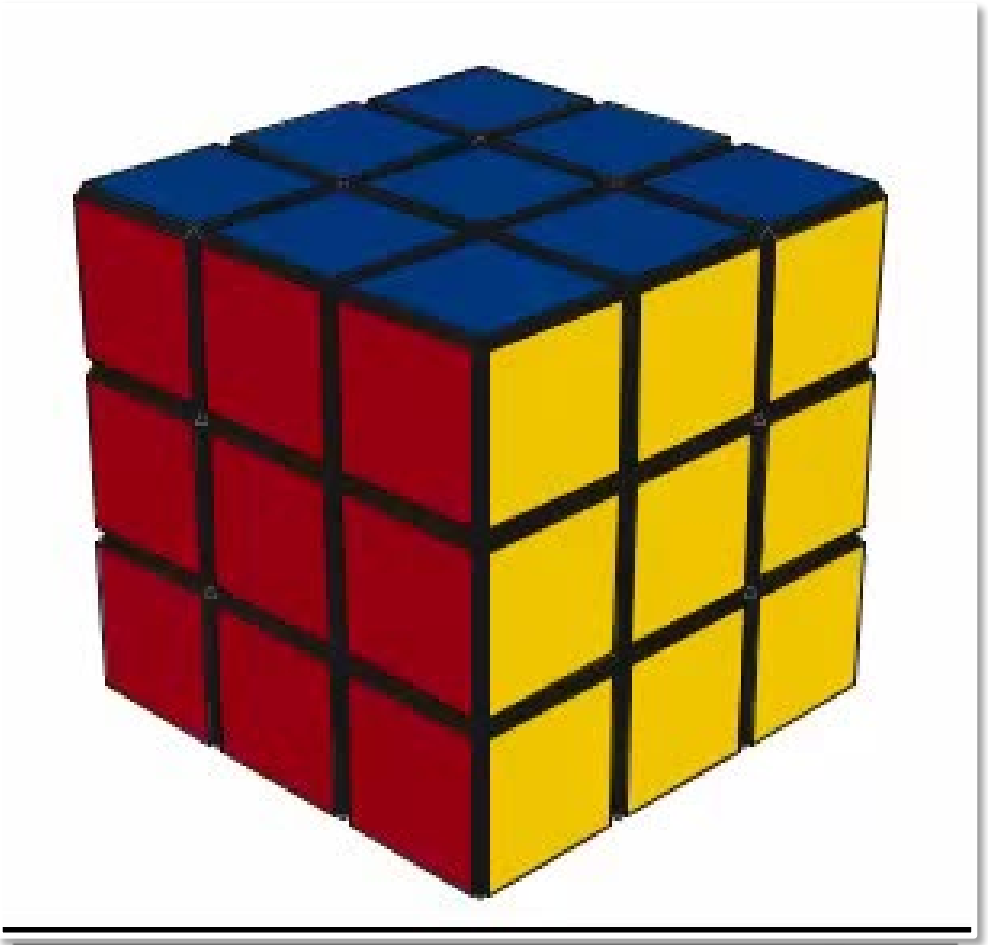


IMDB actor graph
drawn with Gephi

1,654,113 vertices
2,856,828 edges

God's Number is 20

http://www.cube20.org/



RLU2FU-DF2R2B2LU2F-B-UR2DF2UR2U

Superflip, the first position proven to require 20 moves.

Every position of Rubik's Cube™ can be solved in twenty moves or less.

With about 35 CPU-years of idle computer time donated by Google, a team of researchers has essentially solved every position of the Rubik's Cube™, and shown that no position requires more than twenty moves. We consider any twist of any face to be one move (this is known as the half-turn metric.)

Every solver of the Cube uses an algorithm, which is a sequence of steps for solving the Cube. One algorithm might use a sequence of moves to solve the top face, then another sequence of moves to position the middle edges, and so on. There are many different algorithms, varying in complexity and number of moves required, but those that can be

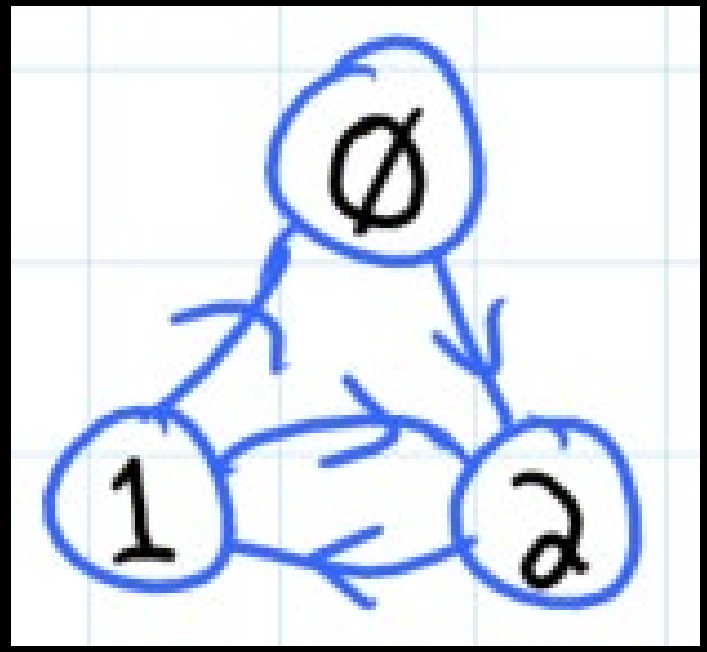
[Rokicki, Kociemba, Davidson, Dethridge 2010]

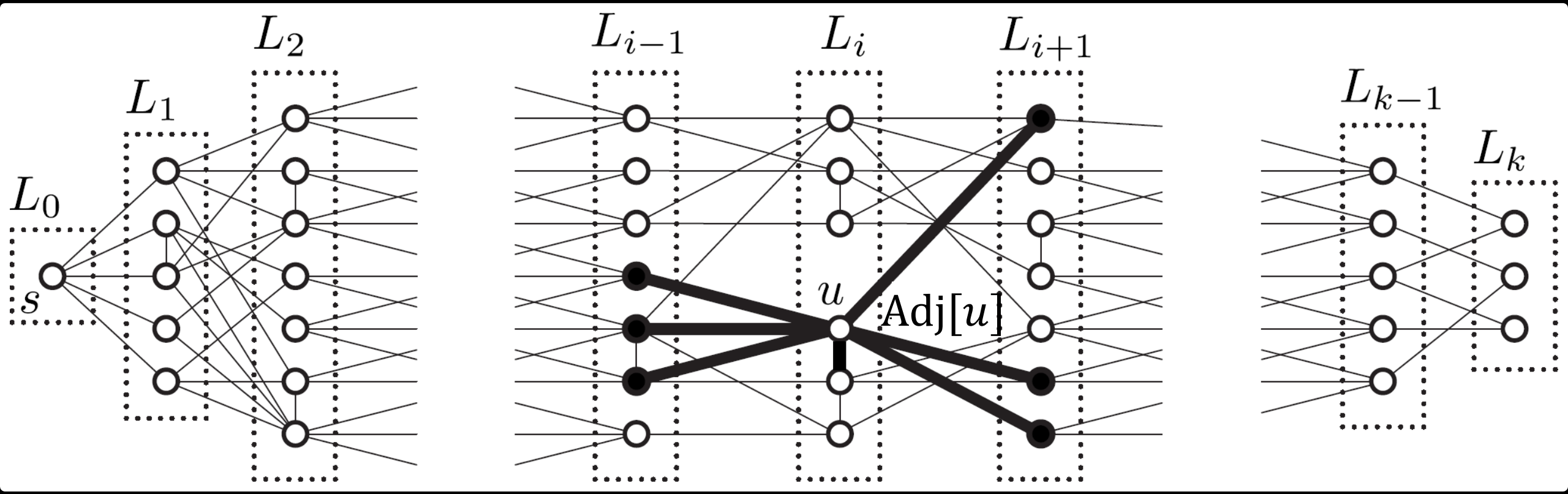
Date	Lower bound	Upper bound	Gap	Notes and Links
July, 1981	18	52	34	Morwen Thistlethwaite proves 52 moves suffice.
December, 1990	18	42	24	Hans Kloosterman improves this to 42 moves.
May, 1992	18	39	21	Michael Reid shows 39 moves is always sufficient.
May, 1992	18	37	19	Dik Winter lowers this to 37 moves just one day later!
January, 1995	18	29	11	Michael Reid cuts the upper bound to 29 moves by analyzing Kociemba's
January, 1995	20	29	9	Michael Reid proves that the "superflip" position (corners correct, edges moves.
December, 2005	20	28	8	Silviu Radu shows that 28 moves is always enough.
April, 2006	20	27	7	Silviu Radu improves his bound to 27 moves.
May, 2007	20	26	6	Dan Kunkle and Gene Cooperman prove 26 moves suffice.
March, 2008	20	25	5	Tomas Rokicki cuts the upper bound to 25 moves.
April, 2008	20	23	3	Tomas Rokicki and John Welborn reduce it to only 23 moves.
August, 2008	20	22	2	Tomas Rokicki and John Welborn continue down to 22 moves.
July, 2010	20	20	0	Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge Cube is exactly 20.

Distance	Count of Positions
0	1
1	18
2	243
3	3,240
4	43,239
5	574,908
6	7,618,438
7	100,803,036
8	1,332,343,288
9	17,596,479,795
10	232,248,063,316
11	3,063,288,809,012
12	40,374,425,656,248
13	531,653,418,284,628
14	6,989,320,578,825,358
15	91,365,146,187,124,313
16	about 1,100,000,000,000,000,000
17	about 12,000,000,000,000,000,000
18	about 29,000,000,000,000,000,000
19	about 1,500,000,000,000,000,000
20	about 300,000,000

Data Structure	Static Set		Dynamic Set		D.O.S.	Ordered Set			Space ~
	find-key(k)	iter()	insert(x)	delete-key(k)	delete-min/ max()	find-next/ prev(k)	find-min/ max()	order-iter()	
static array	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg n)$	$1 \cdot n$
linked list	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg n)$	$3 \cdot n$
dynam. array	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$ a.	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg n)$	$[n, 4n]$
sorted array	$\Theta(\lg n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(1)$	$\Theta(n)$	$1 \cdot n$
direct-access array	$\Theta(1)$	$\Theta(u)$	$\Theta(1)$	$\Theta(1)$	$\Theta(u)$	$\Theta(u)$	$\Theta(u)$	$\Theta(u)$	u
binary heap	$\Theta(n)$	$\Theta(n)$	$\Theta(\lg n)$ a.	$\Theta(n)$	one in $\Theta(\lg n)$	$\Theta(n)$	one in $\Theta(1)$	$\Theta(n \lg n)$	$1 \cdot n$
AVL tree	$\Theta(\lg n)$	$\Theta(n)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(n)$	$5 \cdot n$
hash table	$\Theta(1)$ e.	$\Theta(n)$	$\Theta(1)$ e. a.	$\Theta(1)$ e. a.	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n \lg n)$	$4 \cdot n$

```
Adj = [
      [2],      # 0
      [2, 0],   # 1
      [1],      # 2
]
```




L_0
L_1
L_2
L_{i-1}
L_i
L_{i+1}
L_{k-1}
L_k
s
u
Adj[u]

```
def BFS(Adj, s): # BFS from s in graph given by Adj
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) = 0
    level = [[s]]     # level 0 = {s}
    while level[-1]: # stop at empty level
        level.append([]) # start next level
        for u in level[-2]:
            for v in Adj[u]: # edge (u,v)
                if parent[v] is None: # v new
                    parent[v] = u
                    d[v] = d[u] + 1 #= len(level)-1
                    level[-1].append(v)
```

```
def BFS(Adj, s): # BFS from s in graph given by Adj
    parent = [None] * len(Adj) # shortest-path tree
    parent[s] = s              # s is root
    d = [math.inf] * len(Adj)  # d[v] = δ(s,v)
    d[s] = 0                   # δ(s,s) = 0
    level0 = [s]  # level 0 = {s}
    while level0: # stop at empty level
        level1 = [] # start next level
        for u in level0:
            for v in Adj[u]: # edge (u,v)
                if parent[v] is None: # v new
                    parent[v] = u
                    d[v] = d[u] + 1
                    level1.append(v)
        level0 = level1
```

```
def shortest_path(parent, t):
    # Find unweighted shortest path from s to t,
    # given parent map returned from BFS(Adj, s).
    if parent[t] is None: return None
    path = [t]
    while parent[t] != t:  # stop at root s
        t = parent[t]
        path.append(t)
    path.reverse()
    return path
```