Maze Solving Algorithms

Andrew Valesky, Kirsten Dawes

Questions

1. What are two useful application of and for mazes.

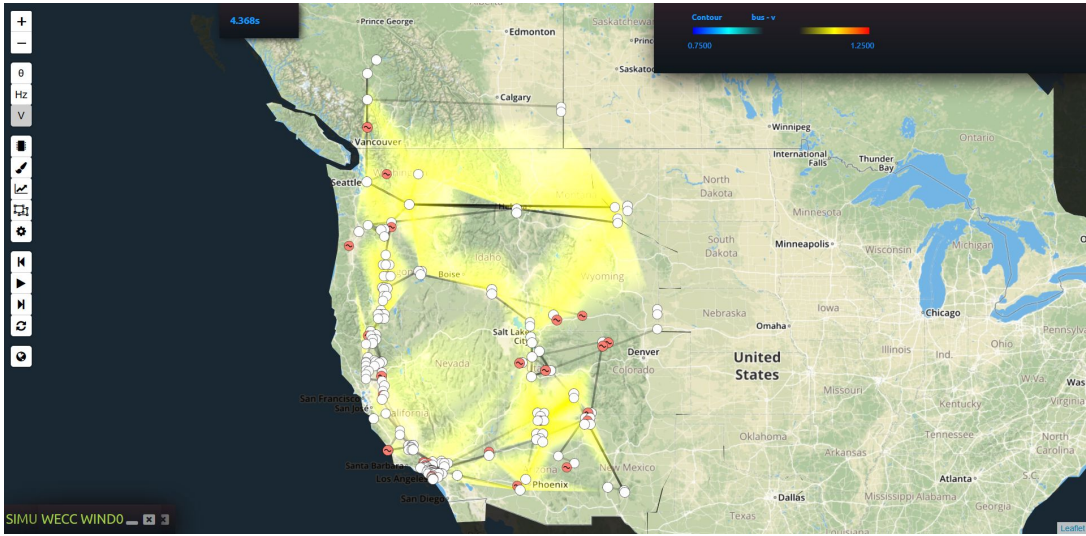
2. Can we use graph algorithms in mazes?

3. What maze solving algorithm led to DFS and pacman.

Outline

- About Us
- History
- Maze Representation
- Appliciations
- Algorithms
  - Tremaux
  - DFS
  - BFS
- Implementation and results
  - Serial
  - Parallel
- Open Issues
  - Quantum Maze Solution
- Discussion

Kirsten Dawes



Master of Science

Advisor: Jian Huang

Project: Spatial temporal data, ERC(power grid simulator)

TAing: Graphics

Other Professional Opportunities:

Lawrence Livermore National Lab - High Energy Density internship, 2015

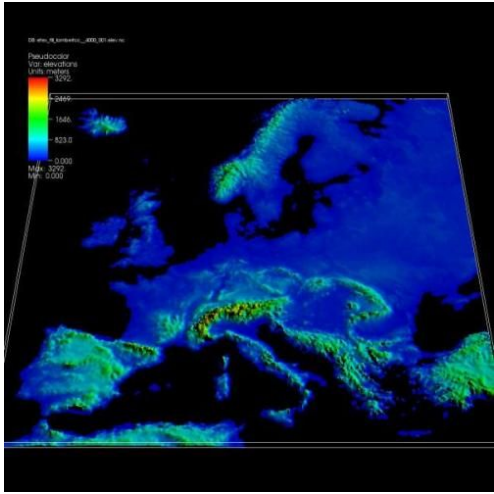
Undergraduate Research with CDUX research group (Hank Childs)

Undergraduate Teaching assistant

CDUX Staff Position

Interest:

Big data, scientific computing, HPC, visualization



From:

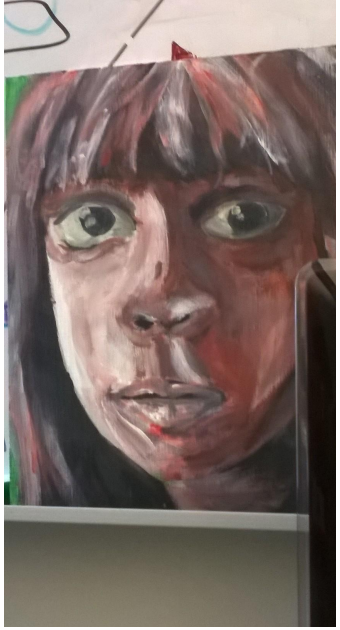
Las Vegas -> Schuylkill haven PA -> Las Vegas -> Klamath Falls ,OR -> Eugene, OR->Knoxville,TN

Interesting Fact:

Spent most time growing up in a gas station.

Interest:

Classical and Contemporary Ballet
Painting/Drawing
Teaching
Magic the Gathering
Watching Scary Movies



Andrew Valesky

Master of Science
- Advisor: Audris Mockus
- Project: ICPUTRD
- TA: cs340 Software Engineering

Other professional
- ORNL Internship

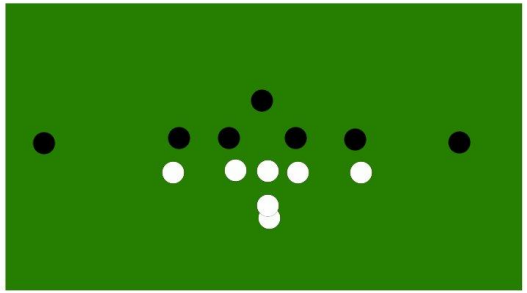
Interest:
- Machine Learning, Visual Processing

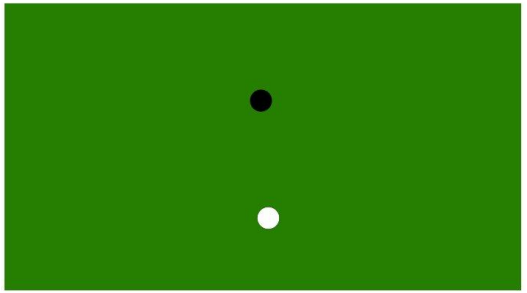
Born: Atlanta

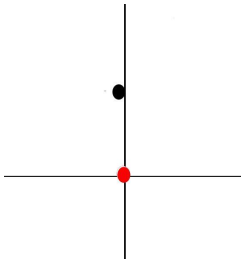
Interests:

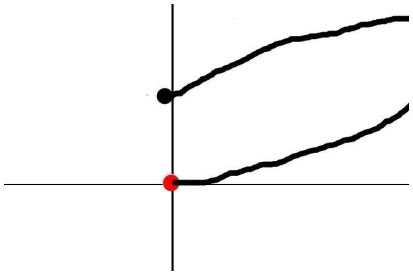
Art & Soccer

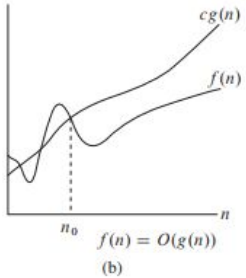
Time Complexity and the American Football “Option”

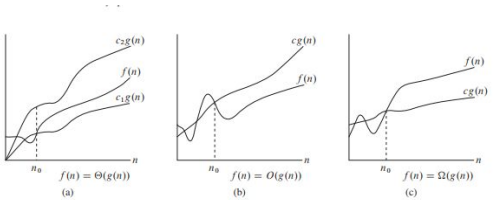






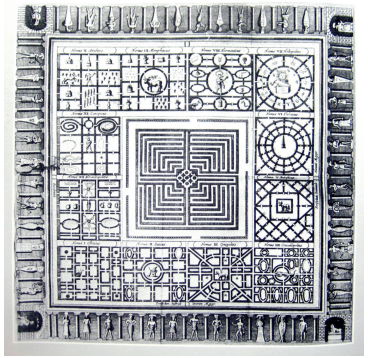






History

- Egyptian Labyrinth.
  - Built by pharaoh Amenemhet III
  - 19 BCE



- The myth of Theseus and the Minotaur
  - Created by Daedalus and his son Icarus
  - The palace of Knossos

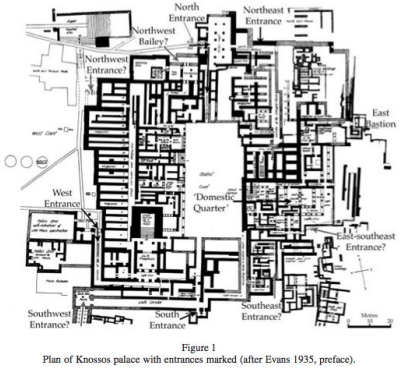


Figure 1
Plan of Knossos palace with entrances marked (after Evans 1935, preface).

History

- The Man in the maze
  - The Pima Indians (Aztec)
  - The maze of life

- Hedge mazes
  - Corn Mazes

Joseph Bernard Kruskal, Jr.

- January 29, 1928 – September 19, 2010
- PHD at Princeton University
- Worked at Bell Labs from 1959 to 1993
- Biggest contribution: formulation of multidimensional scaling.
- Most known:Kruskal's algorithm for computing the minimal spanning tree (MST) of a weighted graph
- Applied linguistics

Robert Clay Prim

- Ph.D. in Electrical Engineering from Princeton University
- Worked at United States Naval Ordnance Lab
  - Engineer
  - Mathematician
- Most known for Prim’s algorithm
- Worked along sided Joseph Kruskal

Charles Pierre Trémaux

- 19th-century French mathematician
- Solved maze with depth first search modification
- Not much else is none about him

John Pledge

- 12 year-old
- Exeter, England
- Most known for Pledge algorithm

Leonhard Euler

- 15 April 1707 – 18 September 1783
- Major contributing to the fields:
  - geometry
  - trigonometry
  - Calculus
- first to analyze plane mazes mathematically
  - Lead to the field of topology

What is a Maze? How can it be defined?

- A structure with well defined walls and open space areas.
- Must have at least one space that is open.
- Other types of maze off base has more rules

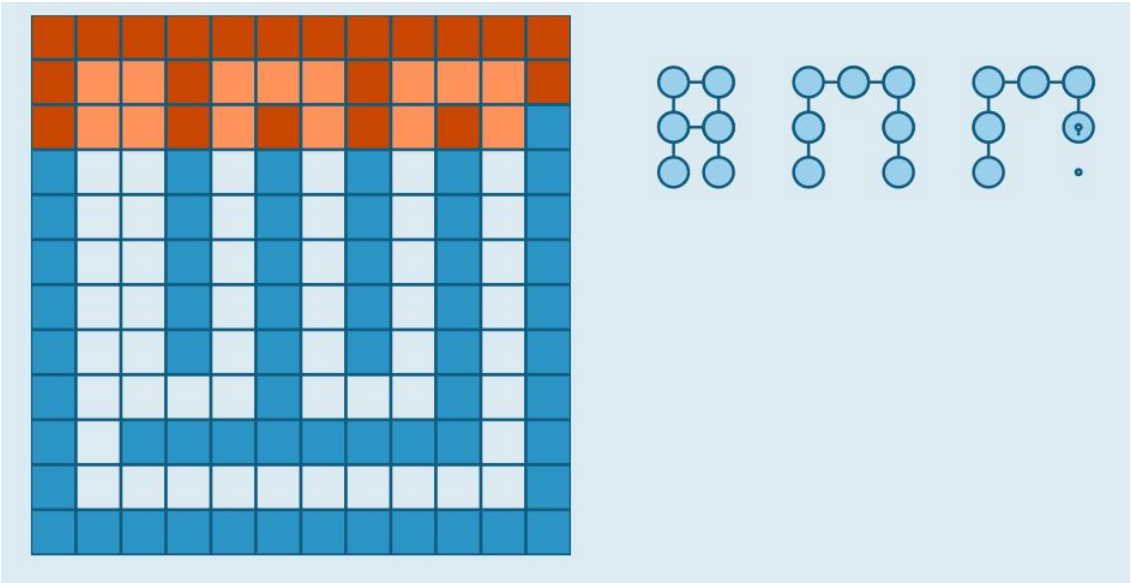
Usually defined as a 2d matrix where black or 1’s are walls and white or 0’s are open space

Can be defined as 1d array as well.

Type of Mazes

- Hamilton Maze
  - A maze in which the goal is to find the unique Hamiltonian cycle
- Number Maze
  - A maze in which numbers are used to determine jumps that form a pathway, allowing the maze to criss-cross itself many times.
- Orthogonal Maze
  - rectangular grid of only paths that have right turns
- Perfect Maze
  - No cycles, no caverns(inaccessible areas)
- Braid Maze
  - No dead ends
- Normal Maze
  - In Euclidean Space
- Fractal Maze
  - A maze that is composed of smaller mae of the same maze

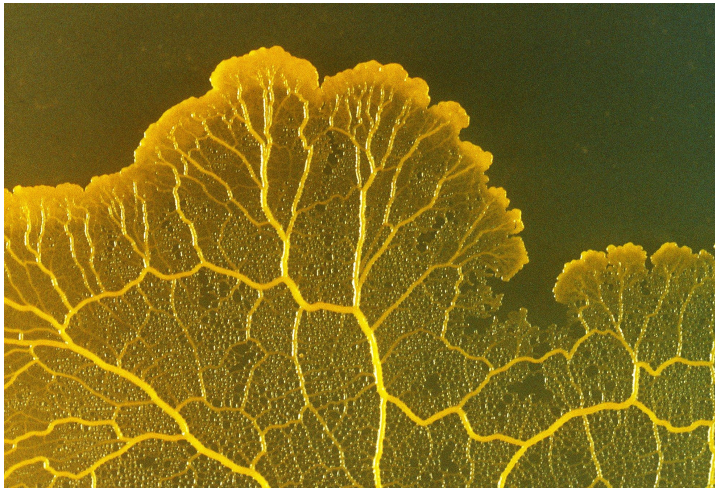
Mazes to Graphs



- Conversion
  - Each open space is a node
  - Each line is the connection to neighbor open cell that can be walked to
  - Can have cycles
  - Disjoin graphs if there a caverns
- Goals is to find a connection between to point in the subgraph of maze

Application Of Mazes

- Navigation of cities
  - City planning
  - Traffic congestion
- Microfluidic Networks
  - Fatty Acid Chemistry
  - Cellular Automata
- Brain communication and stimulus
  - Spatial and memory in sheep
  - Rodents and memory
- Slime mold
  - Is a living organism that solves mazes

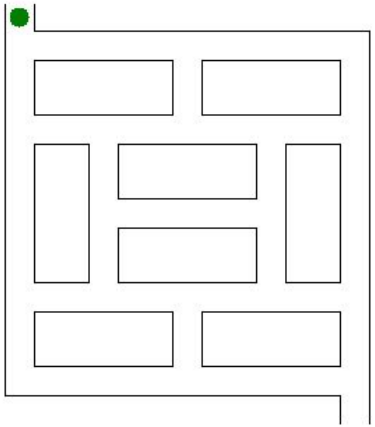


Algorithms

- Tremaux’s Algorithm
- Depth First Search (DFS)
- Breadth First Search (BFS)
  - Serial Top down and bottom up
  - Parallel Shared memory
  - Parallel Distributed

Tremaux’s

- Mark path as the path is traveled
- Paths with two marks are closed
- Take path with no marks,
  - if more than one, take arbitrary unmarked path
- Otherwise:
  - If all paths have one mark, return along that same path, marking it again.
  - If not, choose arbitrarily one of the remaining paths with the fewest marks (zero if possible, else one), follow that path, and mark it.
- When solution is reached, paths marked exactly once indicate solution
- If there is no exit, this method will take you back to the start where all paths are marked twice.



Tremaux’s

- Used in Robots
- Other Robot Mazes
  - Wall-Following
  - The pledge algorithm
- Tremaux’s is considered a precursor to DFS and PacMan
- Bee Routes between Flowers
  - https://web.eecs.utk.edu/~mclennan/Classes/420-527/NetLogoWeb/Ant-Foraging.html
  - They found that focussing simply on sequences of visits to feeder stations, rather than the actual movements between stations or the way that routes develop, is insufficient to understand how animals solve route optimisation problems.

DFS

O(|V| + |E|). This covers two cases:

- If there are more vertices than edges, then the running time is *O(|V|)*.
- If there are more edges than vertices (recall that the number of edges can be as big as *|V|(|V|-1)//2*), then the running time is *O(|E|)*.

Each node has a boolean value called *visited*. "*Visited"* is set to **false**, for all nodes.

Different Data structures Create a different search

i. Stack creates a DFS
ii. Queue Creates a BFS
iii. (c++) MultiMap creates Dykstra's

DFS

Call DFS on the root.

- Check *n's visited* field. If **true**, then return.
- Set *n's visited* field to **true**.
- Optionally do some activity on *n*.
- Then, for all edges of the form *(n,u)* call DFS on *u*.
- Optionally Do some final activity on *n*.
- Return.

When you call DFS on a node *n*, the DFS will visit every node connected to *n*.

Klee

Unassisted and Automatic Generation of High-Complex Systems Programs

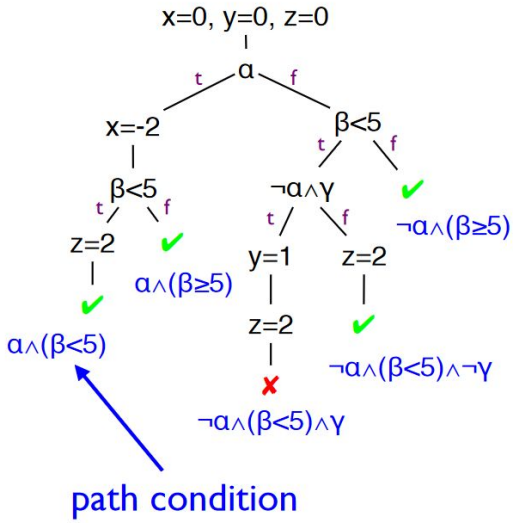
A debugging tool

Symbolic execution refers to execution of program with symbols as argument

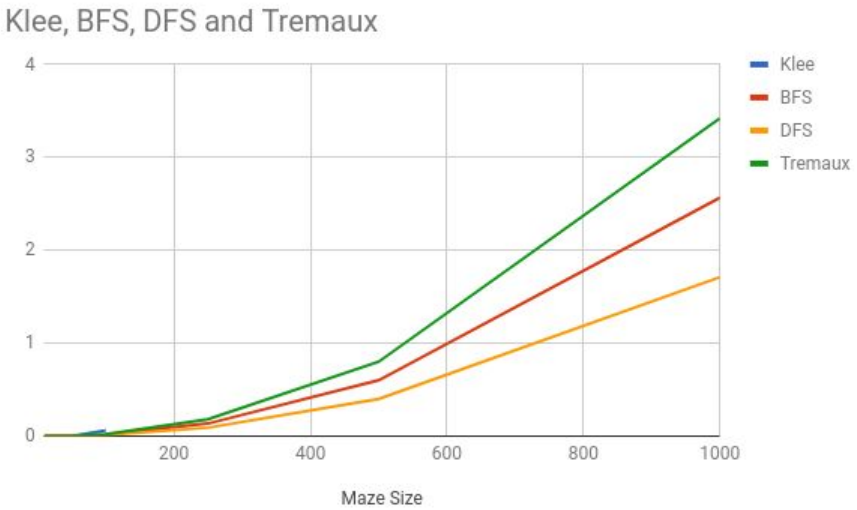
Symbolic execution allows the program can take any feasible path

Does not scale when the number of paths are large.

```
1. int a = α, b = β, c = γ;
2.                  // symbolic
3. int x = 0, y = 0, z = 0;
4. if (a) {
5.   x = -2;
6. }
7. if (b < 5) {
8.   if (!a && c)  { y = 1; }
9.   z = 2;
10.}
11.assert(x+y+z!=3)
```



Running Times of Maze Algorithms



Breadth First Search

Top down- traditional

Runtime : O(|v| +|E|)

Algorithm 1 Sequential top-down BFS algorithm

Input: $G(V, E)$, source vertex s

Output: $parent[1..n]$, where $parent[v]$ gives the parent of $v \in V$ in the BFS tree or -1 if it is unreachable from s

```
1: parent[:] ← −1, parent[s] ← s
2: frontier ← {s}, next ← φ
3: while frontier ≠ φ do
4:     for each u in frontier do
5:         for each neighbor v of u do
6:             if parent[v] = −1 then
7:                 next ← next ∪{v}
8:                 parent[v] ← u
9:     frontier ← next, next ← φ
```

Bottom up

Runtime: O(|v| +|E)

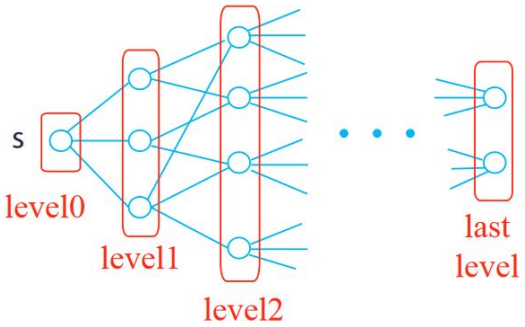
Algorithm 2 Sequential bottom-up BFS algorithm

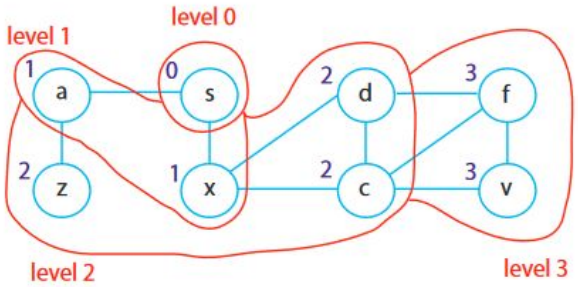
Input: $G(V, E)$, source vertex s

Output: $parent[1..n]$, where $parent[v]$ gives the parent of $v \in V$ in the BFS tree or -1 if it is unreachable from s

```
1: parent[:] ← −1, parent[s] ← s
2: frontier ← {s}, next ← φ
3: while frontier ≠ φ do
4:     for each u in V do
5:         if parent[u] = −1 then
6:             for each neighbor v of u do
7:                 if v in frontier then
8:                     next ← next ∪{u}
9:                     parent[u] ← v
10:                    break
11:    frontier ← next, next ← φ
```

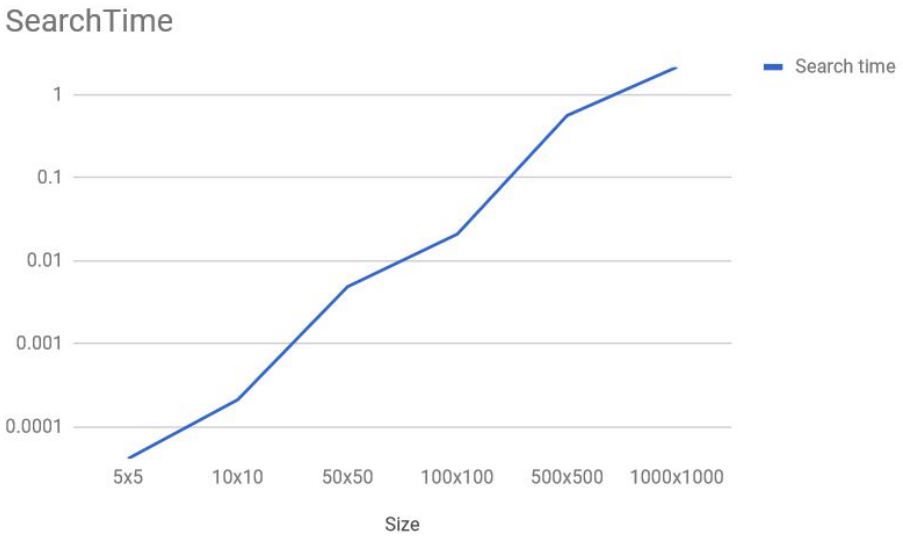
Breadth First Search Bottom Up Structure

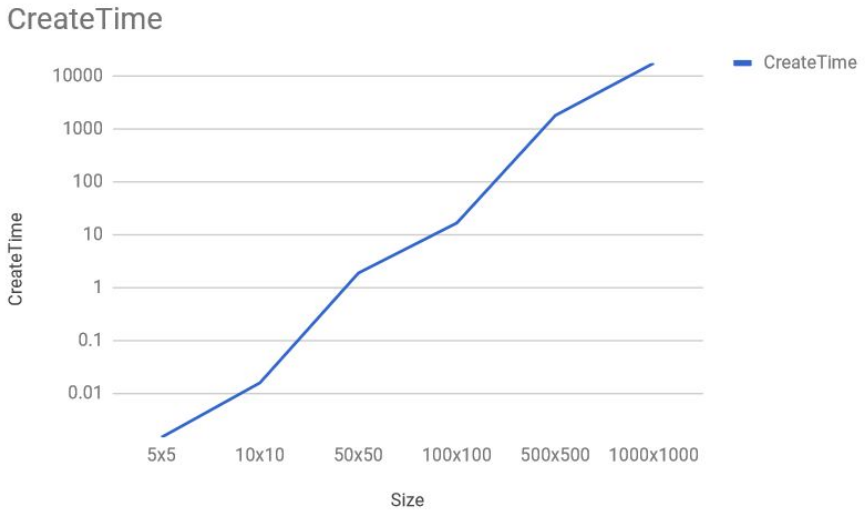




$frontier_0$ = {s}
$frontier_1$ = {a, x}
$frontier_2$ = {z, d, c}
$frontier_3$ = {f, v}
(not x, c, d)

Results for BFS Top Down Serial





Shared vs Distributed memory

Shared Memory

- Each thread/process has access to same memory
- Communication is implicit
- Sequential code hurts
- Effective for data decomposition
- Explicit synchronization
    - Data races
    - Cache consistency
    - Deadlocks

Distributed Memory

- Memory is disjoint between nodes
- Must use communication access information from other nodes
    - Create over head if data is small
    - Message passing
- Communication is explicit
- Synchronization is implicit

Parallel Breadth First Search Shared memory

- Api’s that can be used are OpenMP, Cilk, pthreads
- Limitations for Top Down
  - Load balancing
  - Synchronization overhead
  - Small number of neighbours
- Limitations for Bottom Up
  - Partitions affect speed up and communication
  - Double memory footprint if graph is directed
  - Issues with small graphs

Parallel Breadth First Search Shared memory

```
#pragma omp parallel private(local_count)
{
    local_count = 0;
    int* local_frontier = (int*)malloc(sizeof(int) * (g->num_nodes/NUM_THREADS));
    #pragma omp for
    for (int i=0; i<frontier->count; i++) {

        int node = frontier->present[i];

        int start_edge = g->outgoing_starts[node];
        int end_edge = (node == g->num_nodes-1) ? g->num_edges : g->outgoing_starts[node+1];

        // attempt to add all neighbors to the new frontier
        for (int neighbor=start_edge; neighbor<end_edge; neighbor++) {
            int outgoing = g->outgoing_edges[neighbor];

            if( __sync_bool_compare_and_swap(&distances[outgoing], NOT_VISITED_MARKER, distances[node] + 1)) {
                local_frontier[local_count] = outgoing;
                local_count++;
            }
        }
    }

    #pragma omp critical
    {
        memcpy(new_frontier->present+new_frontier->count, local_frontier, local_count*sizeof(int));
        new_frontier->count += local_count;
    }
```

1: **function** TOP-DOWN-SENDER-NAIVE($A_{i,j}, f_i$)
2: **for** $u \in f_i$ **in parallel do**
3: **for** $v \in A_{i,j}(:, u)$ **do**
4: $k \leftarrow \text{owner}(v)$
5: $t_{i,j,k} \leftarrow t_{i,j,k} \cup \{(u, v)\}$
6: **end for**
7: **end for**
8: **end function**

1: **function** TOP-DOWN-SENDER-LOAD-BALANCED($A_{i,j}, f_i$)
2: **for** $u \in f_i$ **in parallel do**
3: **for** $k \in P(i, :)$ **do**
4: $(v_0, v_1) \leftarrow \text{edge-range}(A_{i,j}(:, u), k)$
5: $r_{i,j,k} \leftarrow r_{i,j,k} \cup \{(u, v_0, v_1)\}$
6: **end for**
7: **end for**
8: **for** $k \in P(i, :)$ **in parallel do**
9: **for** $(u, v_0, v_1) \in r_{i,j,k}$ **do**
10: **for** $v \in A_{i,j}(v_0 : v_1, u)$ **do**
11: $t_{i,j,k} \leftarrow t_{i,j,k} \cup \{(u, v)\}$
12: **end for**
13: **end for**
14: **end for**
15: **end function**

Parallel Breadth First Search Distributed Memory

There are four parts to algorithm I followed:

- Expand
- Local Discovery
- Fold
- Local Update

Focus on using bottom up:

- More complication interior loops
- Allows for early termination on each frontier
- Issues with small or non connected graphs

Parallel Breadth First Search Distributed Memory

Algorithm 3 Parallel 2D top-down BFS algorithm (adapted from the linear algebraic algorithm [6])

Input: A: graph represented by a boolean sparse adjacency matrix, s: source vertex id

Output: π: dense vector, where $\pi[v]$ is the predecessor vertex on the shortest path from s to v, or -1 if v is unreachable

1: $\pi(:) \leftarrow -1$, $\pi(s) \leftarrow s$
2: $f(s) \leftarrow s$ ▷ f is the current frontier
3: **for** all processors $P(i, j)$ **in parallel do**
4: **while** $f \neq \emptyset$ **do**
5: TRANSPOSEVECTOR(f_{ij})
6: $f_i \leftarrow$ ALLGATHERV$(f_{ij}, P(:, j))$
7: $t_{ij}(:) \leftarrow 0$ ▷ t is candidate parents
8: **for** each $f_i(u) \neq 0$ **do** ▷ u is in the frontier
9: **for** each neighbor v of u in A_{ij} **do**
10: $t_{ij}(v) \leftarrow u$
11: $t_{ij} \leftarrow$ ALLTOALLV$(t_i, P(i, :))$
12: $f_{ij}(:) \leftarrow 0$
13: **for** each $t_{ij}(v) \neq 0$ **do**
14: **if** $\pi_{ij}(v) \neq -1$ **then** ▷ Set parent if new
15: $\pi_{ij}(v) \leftarrow t_{ij}(v)$
16: $f_{ij}(v) \leftarrow v$

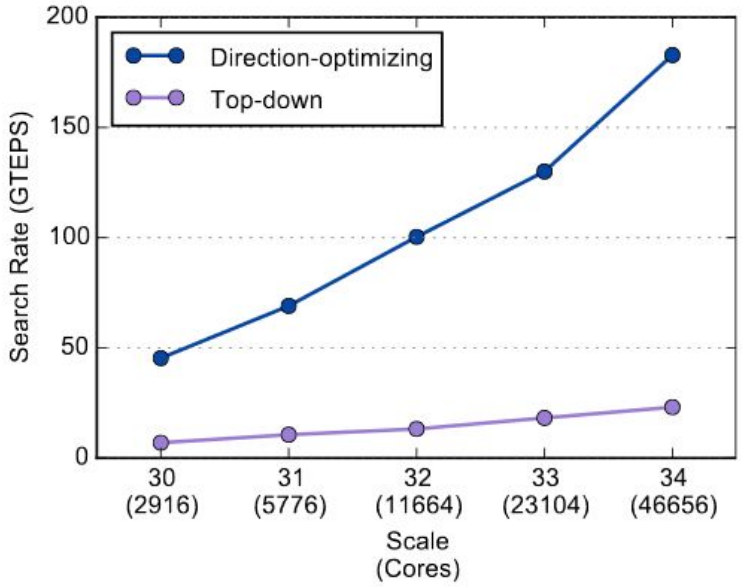
Parallel Breadth First Search Distributed Memory

Operation	Type	Communications / Step	64-bit Words / Search
Transpose (Expand)	p2p	$O(1)$	n
Gather Frontier (Expand)	ag	$O(1)$	np_r
Sending Edges (Fold)	a2a	$O(1)$	$4m$
Total		$O(1)$	$4m + n(p_r + 1)$

TABLE I
TOP-DOWN COMMUNICATION COSTS

Operation	Type	Communications / Step	64-bit Words / Search
Transpose	p2p	$O(1)$	$s_b n/64$
Frontier Gather	ag	$O(1)$	$s_b np_r/64$
Parent Updates	p2p	$O(p_c)$	$2n$
Rotate Completed	p2p	$O(p_c)$	$s_b np_c/64$
Total		$O(p_c)$	$n(\frac{s_b(p_r+p_c+1)}{64} + 2)$

TABLE II
BOTTOM-UP COMMUNICATION COSTS



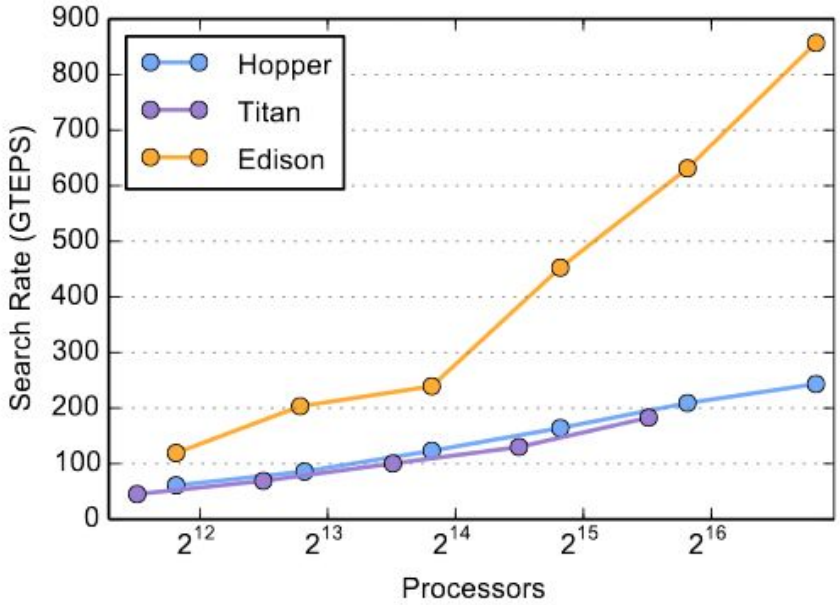
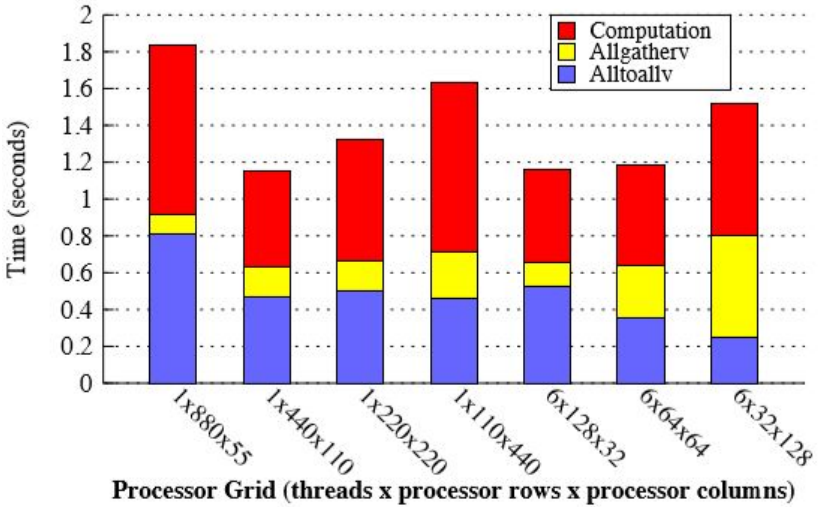
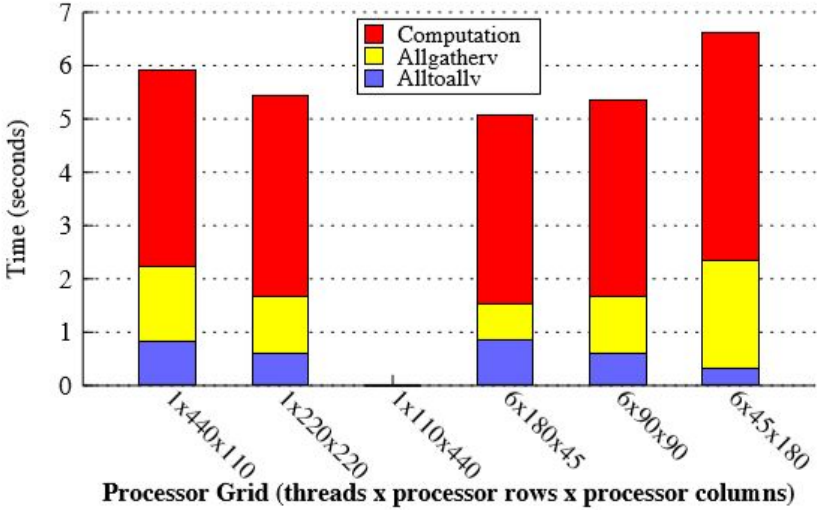


Figure 3: R-MAT weak scaling on Titan

Results for BFS MPI with Graph partitioning



(a) scale 32



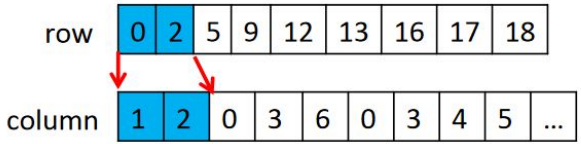
(b) scale 33

Compression BFS

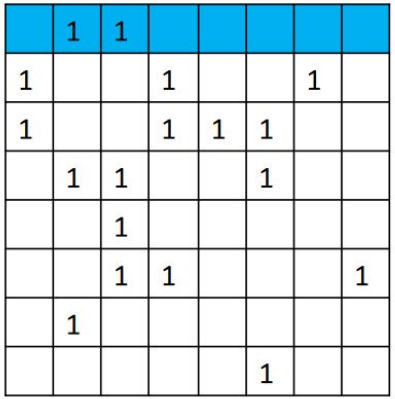
Compression of data affects how efficient MPI is for BFS due to communication

- Compressed Row Format
- Bitmap
- Varint-based encoder
- Word Aligned Hybrid

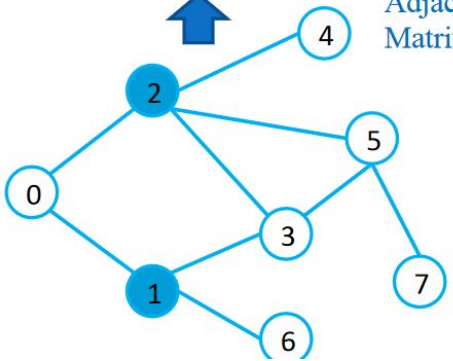
Compressed Row Format



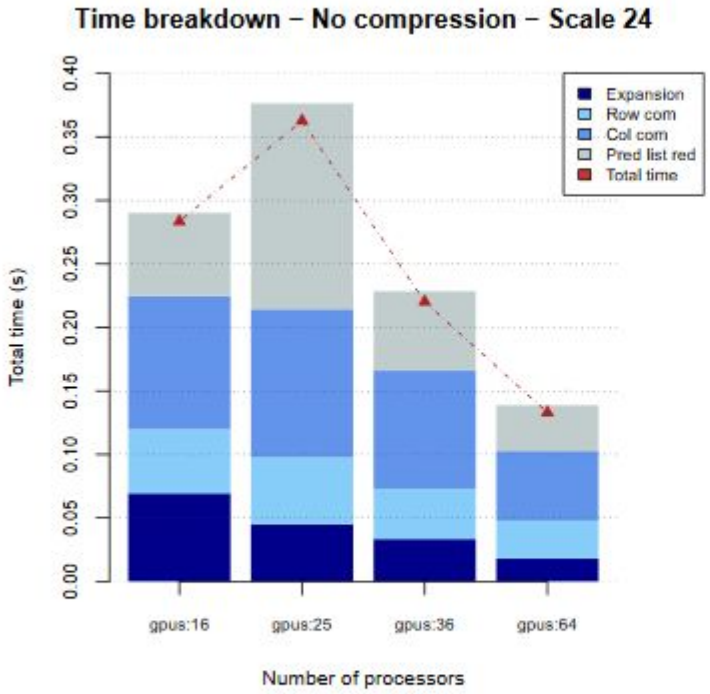
Compressed Sparse Row



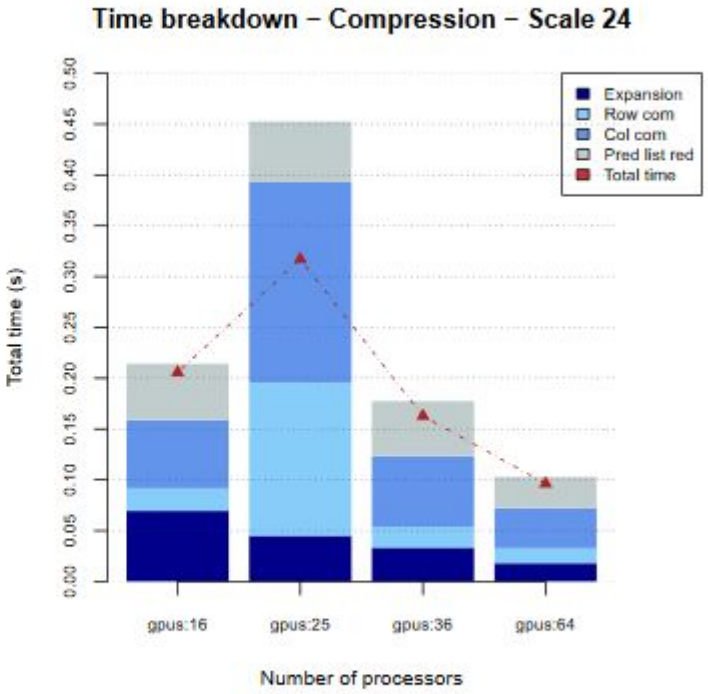
Adjacent Matrix



- A compressed form of adjacent matrix; store non-zero edges only
- Row: index, start & end position
- Column: vertex number
- E.g. v_0's neighbor locates in column[0, 2)



(a) Time breakdown. Compression disabled



(b) Time breakdown. Compression enabled

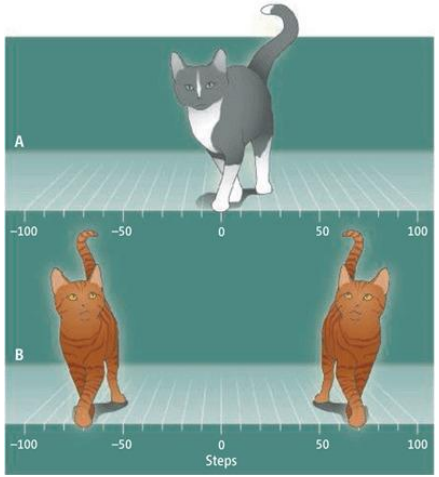
Quantum Maze Solution

Quantum Light Beam Solves Mazes, with a Little Help from Classical Noise

Photosynthesis—the capture of sunlight by plants and its conversion into stored energy

When the quantum and classical states are mixed in an optimal way, the transfer efficiency is some five orders of magnitude larger than the purely quantum example.

The quantum processes allow several routes to be explored at the same time while the classical processes allow random jumps that prevent the system seizing up due to interference.



Question

1. What are two useful application of and for mazes.

2. Can we use graph algorithms in mazes?

3. What maze solving algorithm led to DFS and pacman.

References

http://www.cems.uvm.edu/~rsnapp/teaching/cs32/lectures/mazes
http://coursesite.uhcl.edu/HSH/Whitec/terms/M/mazelab.htm
http://www-groups.dcs.st-and.ac.uk/history/Biographies/Kruskal_Joseph.html
https://www.biography.com/people/leonhard-euler-21342391
https://publik.tuwien.ac.at/files/PubDat_140308.pdf
http://bryukh.com/labyrinth-algorithms/
https://parlab.eecs.berkeley.edu/sites/all/parlab/files/main.pdf
http://cs.hadassah.ac.il/staff/martin/Adv_Architecture/slide08.pdf
https://arxiv.org/pdf/1705.04590.pdf

https://engineering.ucsb.edu/~hpscicom/p1.pdf
http://oa.upm.es/40842/1/PFC_JULIAN_ROMERA.pdf

http://www.mcs.anl.gov/~huiweilu/pdf/ANL13-Lu.pdf

http://courses.csail.mit.edu/6.006/fall09/lecture_notes/lecture12.pdf

References

https://www.technologyreview.com/s/534876/quantum-light-beam-solves-mazes-with-a-little-help-from-classical-noise/

https://www.sciencedaily.com/releases/2017/12/171211090739.html

https://web.eecs.utk.edu/~mclennan/Classes/420-527/NetLogoWeb/Ant-Foraging.html

http://science.sciencemag.org/content/329/5998/1477.full?rss=1

https://en.wikipedia.org/wiki/Maze_solving_algorithm (The Tremaux GIF)

http://blog.jamisbuck.org/2014/05/12/tremauxs-algorithm.html

http://web.eecs.utk.edu/~plank/plank/classes/cs302/Notes/DFS/

http://klee.github.io/docker/