1. Table of contents

- Finding Highest Outdegree
- Chaotic Relaxation
- Dijkstra's Algorithm
- Delta Step Experiments
- Optimal Delta Step
- How to run
- Documentation

1.0.1. Finding Highest Outdegree

General Approach The graph is represented by a `NxN` matrix, where the rows represent `from` vertices and the columns represent the `to` vertices. Therefore, if `A[u][v] = w`, there is edge from `u` to `v` with edge weight `w`.

With this definition, calculating the highest outdegree is equivalent to finding the row with the most non-zero entries.

Finding most non-zero entries with Compressed Sparse Row The number of non-zero entries can be computed with the `IA` vector of Compressed Sparse Row.

The `IA` vector has the following definition: - IA[0] = 0 - IA[i] = IA[i - 1] + number of non-zero entries for row i - 1

Code
```
for(int i = 0; i < numRows){
    int currDegree = IA[i + 1] - IA[i];
    if(currDegree > oldDegree){
        row = i;
        oldDegree = currDegree;
    }
}
return row;
```

Confirming correctness with test data:

Graph	Expected	Output
rmat15	0	0
rmat20	0	0
rmat22	0	0
rmat23	0	0
road-FLA	140960	140960
road-NY	316606	316606

2. Chaotic relaxation

Experiment - Relaxations for different random seeds

Seed	Edge Relaxations	Node Relaxations	Total relaxations
10	175499	2518	178017
20	175499	2518	178017
50	175499	2518	178017

3. Dijkstras Algorithm

Experiment - rmat15 & roadNY on dijkstras (delta = 1)

Graph	Edge Relaxations	Node Relaxations	Total relaxations
rmat15	176678	2498	179176
roadNY	2910283	1464269	4374552

4. Delta Step

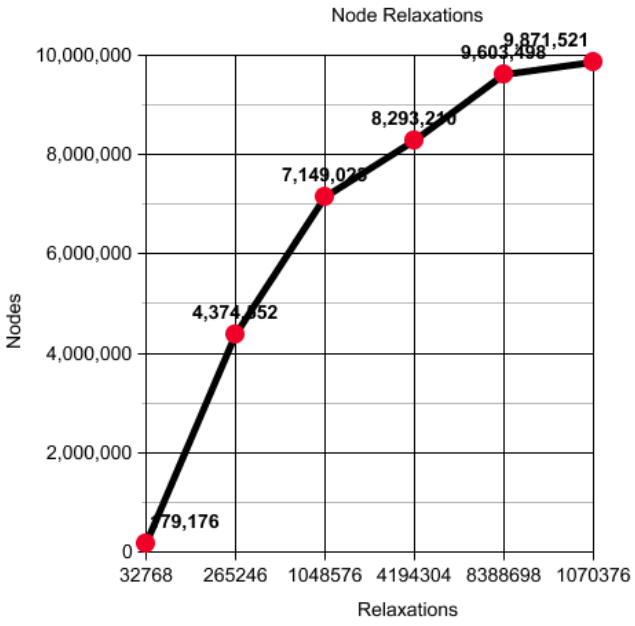
Experiment - Number of relations with changing delta

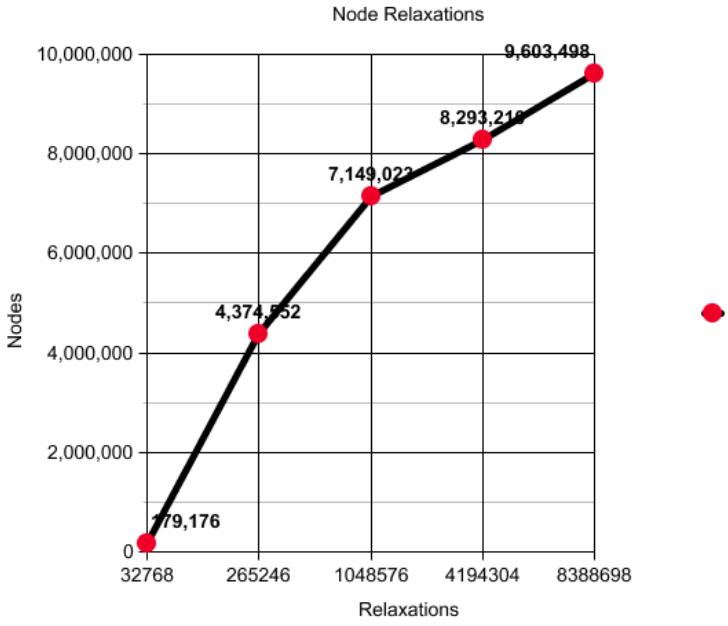
Delta	rmat15 relaxations	road-NY relaxations
1	179176	4374552
11	179133	4389122
21	178004	439788
31	178037	439763
41	178012	439797
51	178016	440818

5. Optimal Delta Step

From the previous experiment, the optimal value for delta is `1`

5.0.1. Graphs


Node Relaxations
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
Nodes
9,871,521
9,603,498
8,293,210
7,149,022
4,374,952
179,176
32768
265246
1048576
4194304
8388698
1070376
Relaxations


Node Relaxations
10,000,000
8,000,000
6,000,000
4,000,000
2,000,000
0
Nodes
9,603,498
8,293,210
7,149,022
4,374,952
179,176
32768
265246
1048576
4194304
8388698
Relaxations

5.0.2. Data

Graph	# of nodes	Relaxations
rmat15	32768	179176
road-NY	265246	4374552
rmat20	1048576	7149023
rmat22	4194304	8293210
rmat23	8388698	9603498
road-FLA	1070376	9871521

Graph	# of nodes	Running time (seconds)
rmat15	32768	24.89
rmat-NY	264246	1236.42
rmat20	1048576	1635.57
rmat22	4194304	13084.56
rmat23	8388608	1895.56
road-FLA	1070376	2389.48

6. How to run

The makefile separates by experiment. To run the experiments, run the following commands, respectively

```
$ make chaotic
$ make delta
$ make deltaNY
$ make dijkstra
$ make dijkstraNY
$ make deltaOptimal
$ make deltaFLA
$ make degree
```

To run, put the input into the experiment like so

```
$ chaotic.o < rmat15.dimacs
```

7. Documentation

7.1. Files

- CSR.cpp/h
  - Compressed Sparse Row class
- Parser.cpp/h
  - Parsing input class, returns an instance of CSR
- SSSP.cpp/h

  - Contains delta step algorithm

- Worklist.cpp/h
  - Worklist class, abstracts a map of 'buckets'
- *Experiment.cpp
  - Different runners for the various experiments

7.2. CSR.cpp - Compressed Sparse Row implementation

void put (int32_t x, int32_t y, int32_t val)

- Sets x,y in the adjaceny matrix to val
- x is from edge, y is to edge

int32_t get (int32_t x, int32_t y)

- Returns the weight for edge x->y

vector> iterate()

- Returns a vector of vectors
- Each vector will have the format <u, v, weight>

void printNodeLabels()

- print all the labels for each of given nodes

long getTent (int32_t u)

- returns the tentative cost of node u

void setTent (int32_t u, long val)

- set the tentative cost of node u to cost val

void debugInfo()

- print out the inner workings of the CSR
- IA, JA, and the values

bool nodeFullyRelaxed (int32_t node)

- returns true if all the nodes out of `node` have been relaxed

void relaxNode (int32_t src, int32_t dest)

- sets the edge as relaxed

7.3. Worklist.cpp - Worklist implementation

bool hasElements()

- returns true if there are still items in a bucket

long getIndex()

- returns the index of the first non-empty bucket

set get(long i)

- returns the bucket stored at i

void put(long i, set nodes)

- puts a set of nodes at bucket i

void relaxNodes(set req, int seed)

- relaxes the set of nodes in `req`, shuffles with seed `seed`

void printRelaxCount()

- prints the number of edge and node relaxations

set> getLight()

- returns the light edges

void setLight(set> s)

- sets the light edges to set `s`

set> getHeavy()

- returns the heavy edges

void setHeavy(set> s)

- sets the heavy edges to `s`

set match(set bucket, set> s)

- returns a set to be relaxed, the nodes in both `bucket` and `s`

7.4. SSSP.cpp - DeltaStep implementaton

- Constructor
- takes in a CSR graph, step size, and a seed for shuffling

run(bool printNLabels, bool printRelaxCount)‘]

- Runs the delta step algorithm
- If `printNLabels` is true, the node labels are printed
- If `printRelaxCount` is true, the number of relaxations are printed

7.5. Parser.cpp - Input parser

CSR parseInput()

- returns a created CSR from the input file

7.6. Runner Files - *Experiment.cpp

chaoticExperiment.cpp

- Measures clock time and node relaxations across different seeds
- Seeds: 10, 20, 50

deltaStepExperiment.cpp

- Measures node relaxations across different step sizes
- Prints out node labels and number of steps

dijkstraExperiment.cpp

- Runs dijkstra by setting delta to 1
- prints out node labels and node relaxations

deltaOptimalExperiment.cpp

- runs the optimal delta across all the different graphs
- prints node relaxations and clock time