

## 1. Table of contents

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

## 2. Finding Highest Outdegree

**General Approach** The graph is represented by a  $N \times N$  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;
```

	Graph	Expected	Output
Confirming correctness with test data:	rmat15	0	0
	rmat20	0	0
	rmat22	0	0
	rmat23	0	0
	road-FLA	140960	140960
	road-NY	316606	316606

## 3. Chaotic relaxation

	Seed	Edge Relaxations	Node Relaxations
Experiment - Relaxations for different random seeds	10	175499	2518
	20	175499	2518
	50	175499	2518

## 4. Dijkstras Algorithm

	Graph	Edge Relaxations	Node Relaxations
Experiment - rmat15 & roadNY on dijkstras (delta = 1)	rmat15	176678	2498
	roadNY	2910283	1464269

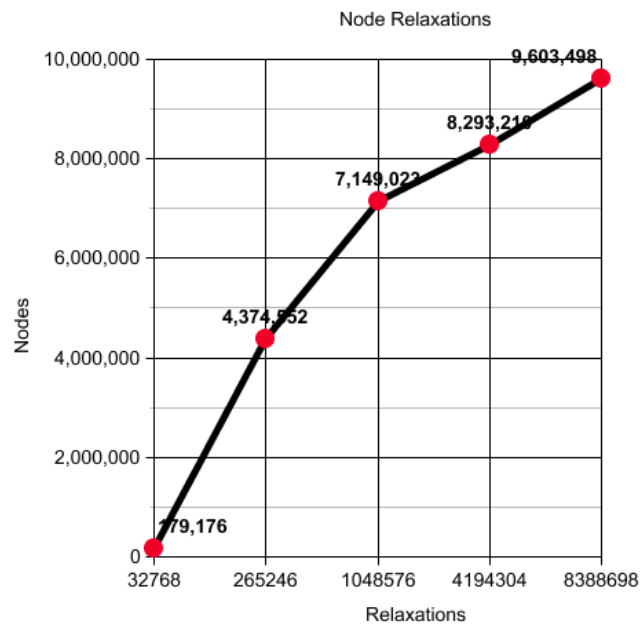
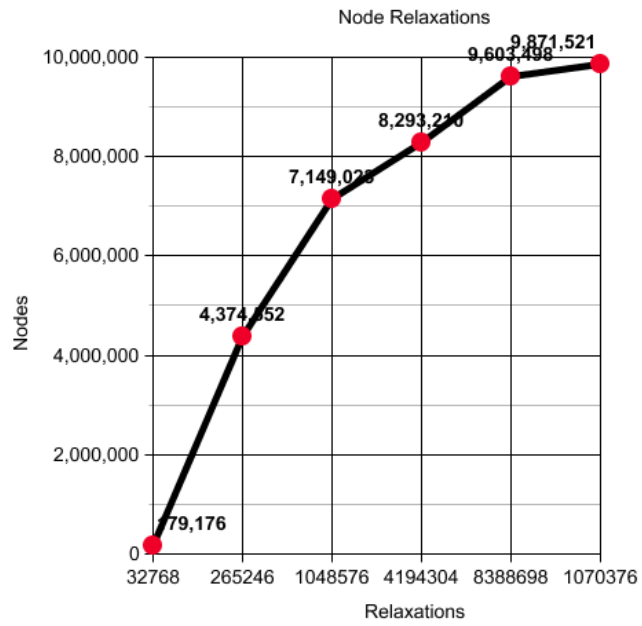
## 5. Delta Step

	Delta	rmat15 relaxations	road-NY relaxations
Experiment - Number of relations with changing delta	1	179176	4374552
	11	179133	4389122
	21	178004	439788
	31	178037	439763
	41	178012	439797
	51	178016	440818

## 6. Optimal Delta Step

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

### 6.0.1. Graphs



### 6.0.2. 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

## 7. 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
```

## 8. Documentation

### 8.1. 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

## 8.2. CSR.cpp - Compressed Sparse Row implementation

**void put (int32\_t x, int32\_t y, int32\_t val)**

- Sets x,y in the adjacency 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< 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

### 8.3. 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`

### 8.4. 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

## **8.5. Parser.cpp - Input parser**

**CSR parseInput()**

- returns a created CSR from the input file

## **8.6. 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