Table of contents

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

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

Dijkstras Algorithm

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

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

Delta Step

Experiment - Number of relations with changing delta

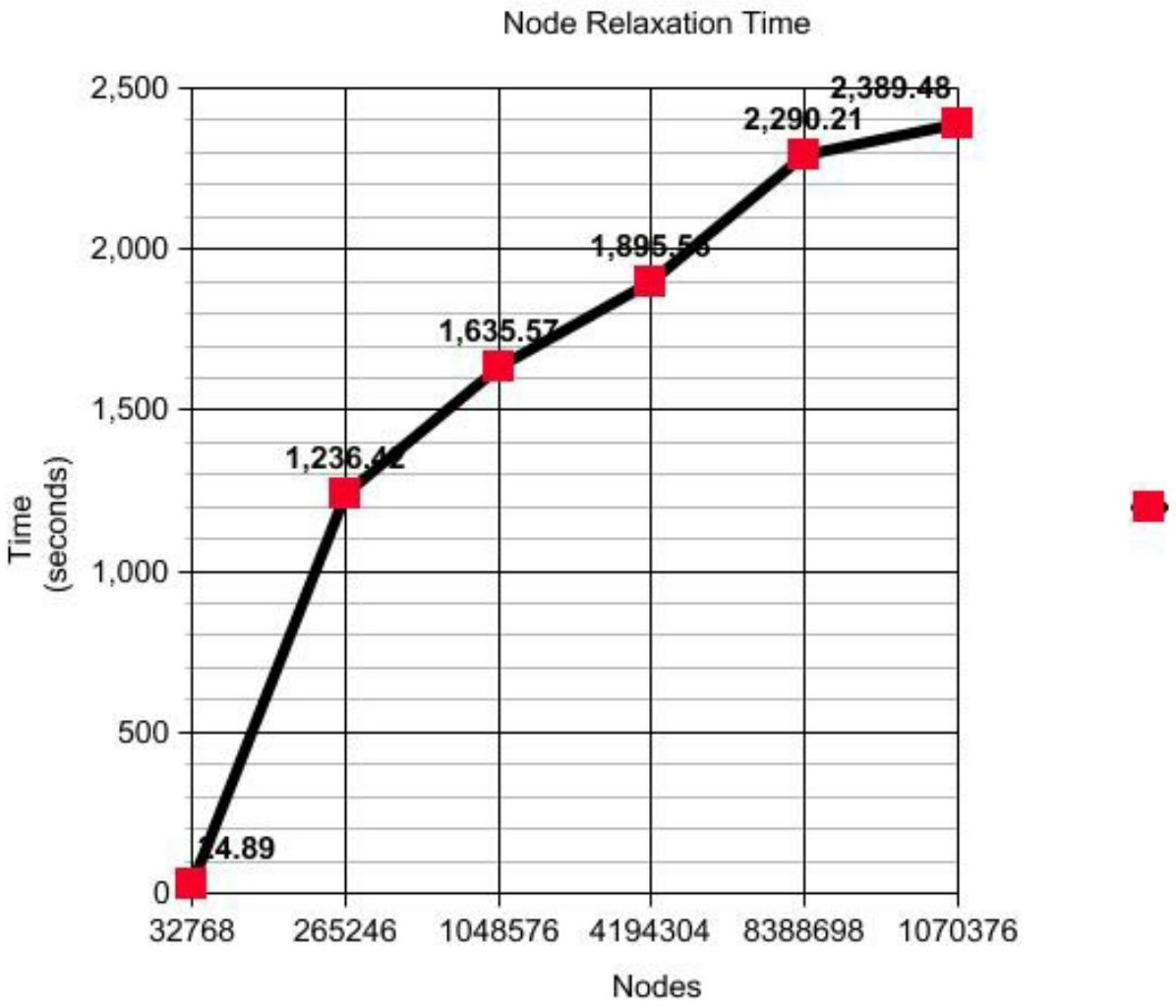
Delta	rmat15 relaxations	road-NY relaxations

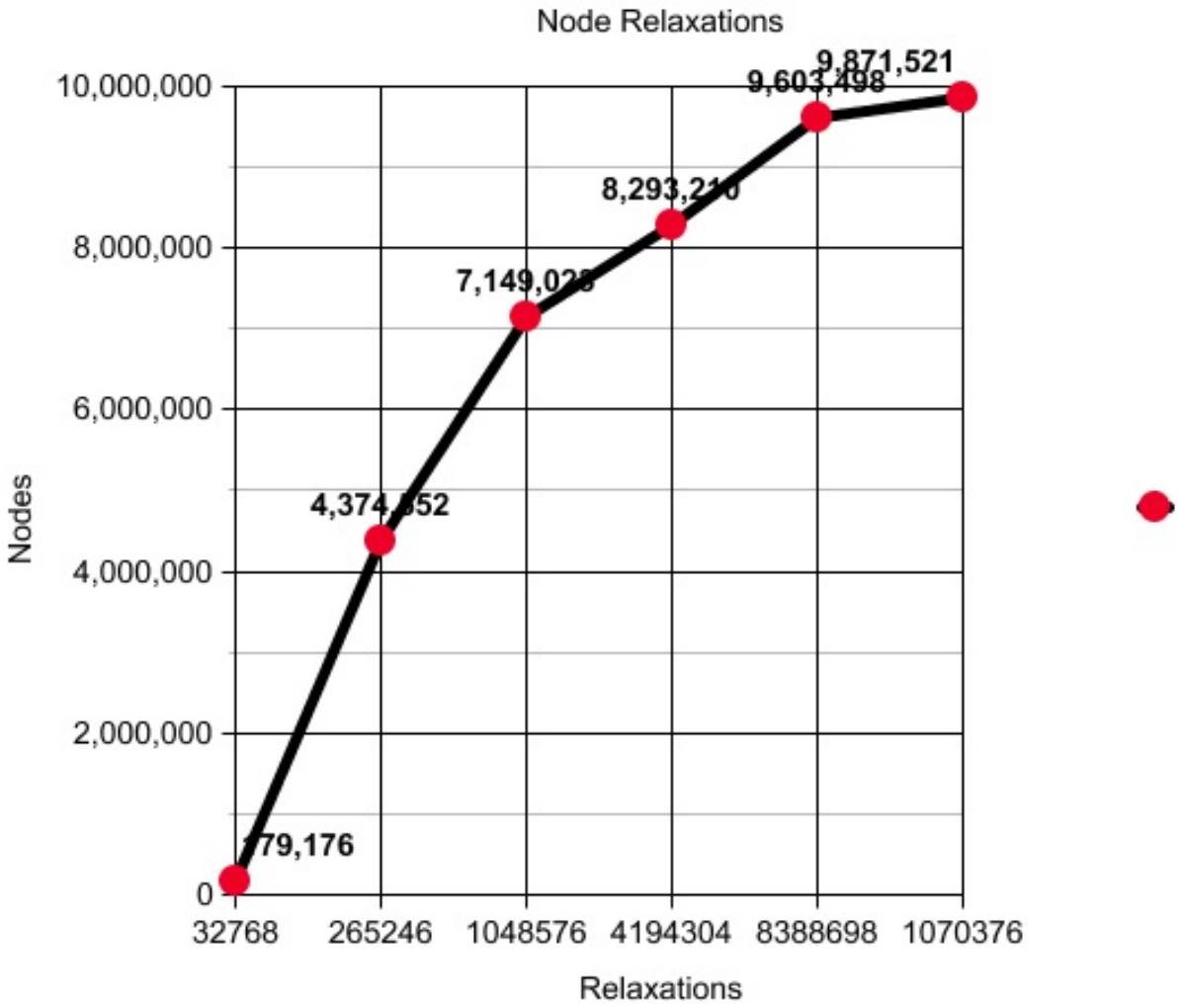
1	179176	4374552
11	179133	4389122
21	178004	439788
31	178037	439763
41	178012	439797
51	178016	440818

Optimal Delta Step

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

Graphs


Node Relaxation Time
2,500
2,000
1,500
1,000
500
0
Time (seconds)
14.89
1,236.42
1,635.57
1,895.56
2,290.21
2,389.48
32768
265246
1048576
4194304
8388698
1070376
Nodes



Data

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

Graph	Number 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

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
```

Documentation

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

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<vector<int32_t>> 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

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<int32_t> get(long i)

- returns the bucket stored at i

void put(long i, set<int32_t> nodes)

- puts a set of nodes at bucket i

void relaxNodes(set<csrTuple> 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<vector<int32_t>> getLight()

- returns the light edges

void setLight(set<vector<int32_t>> s)

- sets the light edges to set `s`

set<vector<int32_t>> getHeavy()

- returns the heavy edges

void setHeavy(set<vector<int32_t>> s)

- sets the heavy edges to `s`

set<csrTuple> match(set<int32_t> bucket, set<vector<int32_t>> s)

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

SSSP.cpp - DeltaStep implementaton

DeltaStep(CSR csr, int32_t step, int seed)

- 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

Parser.cpp - Input parser

CSR parseInput()

- returns a created CSR from the input file

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