Single-source Shortest Path Algorithms

CS 260 Design and analysis of algorithms

Wenting Wu, Yuanzhao Chen, Peihao Zhu, Guoqing Ma

outline

introduction

Introduction to Problems

Shortest path problem[1] is the problem of finding a path between two vertices in a graph, which is one with the smallest weight among all paths between these two vertices.

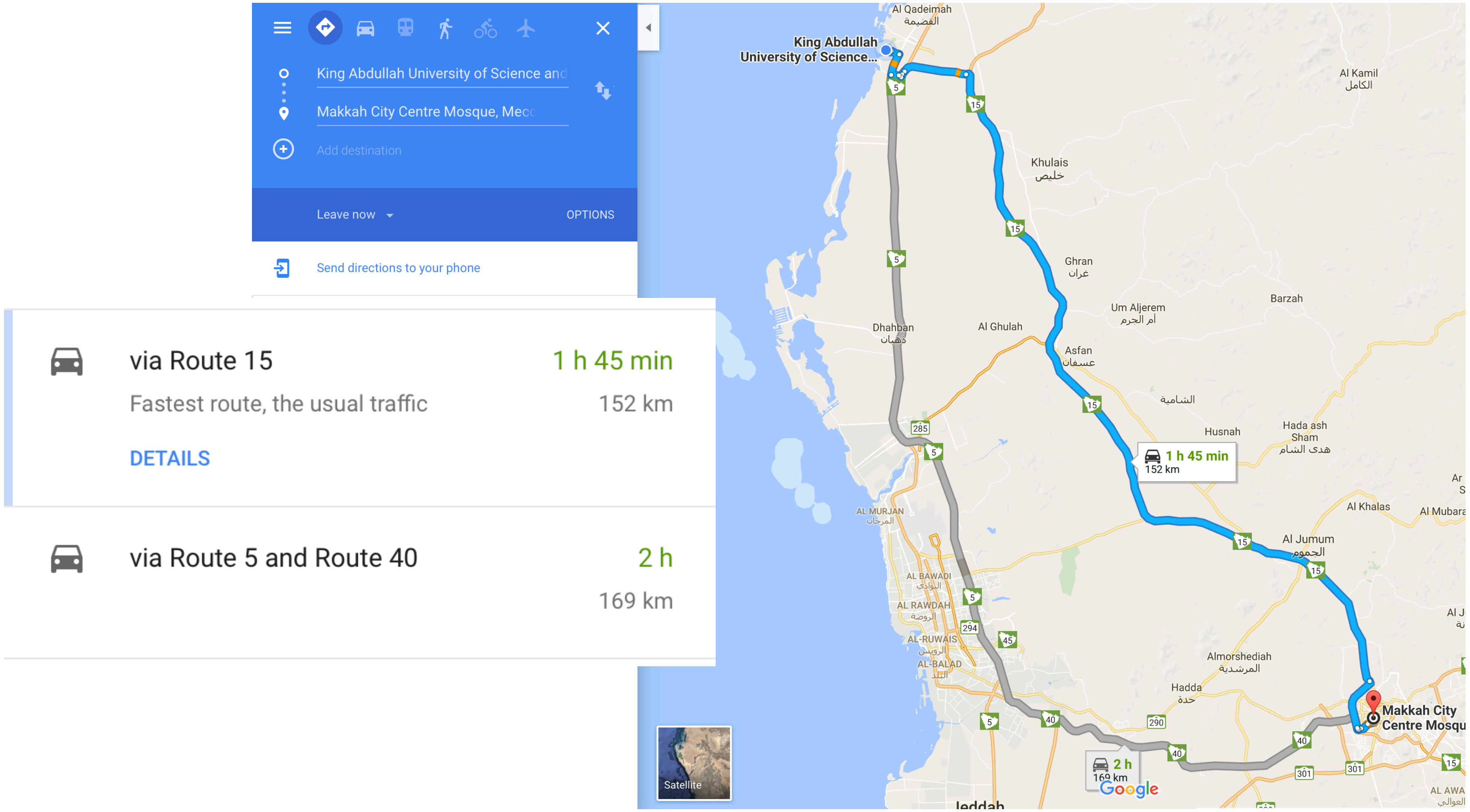
$\mathbf{G\ (V, E)}$: directed graph;
V : vertices set;
$\mathbf{E}$: edge set;
$\mathbf{e(v_i, v_{i+1})}$: edge between $\mathrm{v_i}$ and $\mathrm{v_{i+1}}$;
$\mathbf{w(e(v_i, v_{i+1}))}$: weight of directed edge $\mathrm{e(v_i, v_{i+1})}$

Formally, single-source shortest path problem from start node S to terminal node T can be formulated as following:

$$\mathbf{find: min} \sum \mathbf{w(e(v_i, v_{i+1}))}$$

$$\mathbf{St: v_i} \in \{\mathbf{v_0 = S, v_1, v_2, \dots v_n = T}\}$$

Shortest Path Application


King Abdullah University of Science and

Makkah City Centre Mosque, Mecc

Add destination

Leave now

OPTIONS

Send directions to your phone

via Route 15 — 1 h 45 min

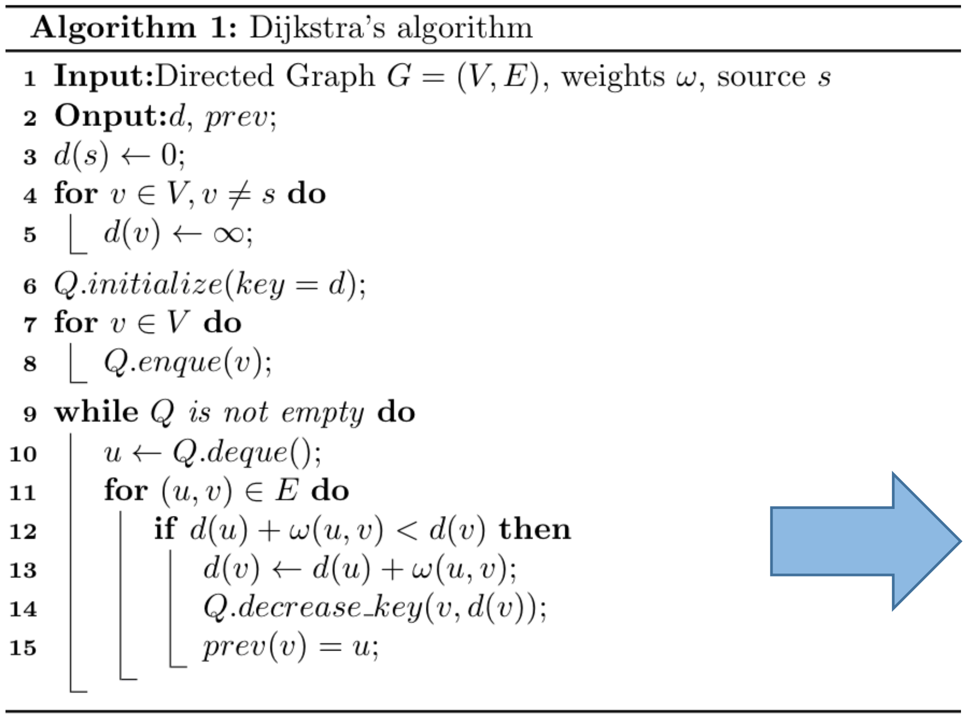
Fastest route, the usual traffic — 152 km

DETAILS

via Route 5 and Route 40 — 2 h

169 km

Introduction to Dijkstra's Algorithm[2]



Algorithm 1: Dijkstra's algorithm

```
Input: Directed Graph $G = (V, E)$, weights $\omega$, source $s$
Onput: $d$, $prev$;
$d(s) \leftarrow 0$;
for $v \in V, v \neq s$ do
    $d(v) \leftarrow \infty$;
$Q.initialize(key = d)$;
for $v \in V$ do
    $Q.enque(v)$;
while $Q$ is not empty do
    $u \leftarrow Q.deque()$;
    for $(u, v) \in E$ do
        if $d(u) + \omega(u, v) < d(v)$ then
            $d(v) \leftarrow d(u) + \omega(u, v)$;
            $Q.decrease\_key(v, d(v))$;
            $prev(v) = u$;
```

- Solve SSSP problem (**traditional**)
- **Non-negative** weight for every edge
- **Greedy** algorithm implemented using a **priority queue**
- The number of iterations is exactly the number of nodes

$d(v)$: the minimum cost from node S to node v;
$prev(v)$: the last node before v in this shortest path;
Q : priority queue with d as the key value.
decrease_key : decrease a key-value pair in the queue.

Introduction to Shortest Path Faster Algorithm[3]

Algorithm 2: SPFA

```
Input:Directed Graph $G = (V, E)$, weights $\omega$, source $s$
Onput:$d$, $prev$;
$d(s) \leftarrow 0$;
for $v \in V, v \neq s$ do
    $d(v) \leftarrow \infty$;
$Q.initialize()$;
for $v \in V$ do
    $Q.enque(v)$;
while $Q$ is not empty do
    $u \leftarrow Q.deque()$;
    for $(u, v) \in E$ do
        if $d(u) + \omega(u, v) < d(v)$ then
            $d(v) \leftarrow d(u) + \omega(u, v)$;
            $prev(v) = u$;
            if $v \notin Q$ then
                $Q.enque(v)$;
```

- An **improvement** of the Bellman–Ford algorithm
- Deal with edges with **negative weights**
- **Cannot work** on graphs with **cycles** of negative weights
- **FIFO queue** (simpler than Dijkstra)
- Work well on **sparse** graphs

$\mathbf{d(v)}$: minimum cost from node $\mathbf{S}$ to node $\mathbf{v}$
$\mathbf{prev(v)}$: the last node before $\mathbf{v}$ in this shortest path.
$\mathbf{Q}$: priority queue with $\mathbf{d}$ as the key value.

○
●
○
○
○

Theoretic Time Complexity

Theoretic Complexity of Dijkstra's

- Unordered array :
  - cost $= v + e + (v \cdot (1 + e)) + (e \cdot (1 + e)) + v \in O((v \cdot e) + e^2) = O(e^2)$
- Binary heap ：
  - cost$=v+(2\lceil^{\log 2(e)\rceil \cdot 2)+(} v\cdot(1+\lceil \log 2(e)\rceil))+(e\cdot(1+\lceil \log 2(e)\rceil))+v \in O((v+e)+(e\cdot \log(e)))+(v\cdot \log(e)) = O(e \cdot \log(e))$
- Fibonacci heap :
  - cost$=v+e+(v\cdot(1+\log 2(e)))+(e\cdot(1+1))+v \in O((v+e)+(v\cdot \log(e)))=O(e + (v \cdot \log(e)))$

Theoretic Complexity of SPFA

Worst case:

$$\text{cost} = v + e + v * \left(\frac{v}{2}\right) + e * \left(\frac{v}{2}\right) + v \in O(e * v)$$

Average case:

$$\text{cost} = v + e + v * (\text{enq}) + e * (\text{enq} + 1) + v$$
$$\in O(v * (\text{enq}) + e * (\text{enq}))$$

Theoretic Complexity of SPFA

$$\sum_{v\in V} enq(v) \le \epsilon * (1+\epsilon+\epsilon^2+\ldots+\epsilon^{\lceil \frac{|V|^2}{|E|} \rceil - 1}) = \epsilon * \frac{\epsilon^{\lceil \frac{|V|^2}{|E|} \rceil} - 1}{\epsilon - 1}$$

$$averagecost = O(\frac{\epsilon}{|V|} * \frac{\epsilon^{\lceil \frac{|V|^2}{|E|} \rceil} - 1}{\epsilon - 1} * (|E|+|V|))$$

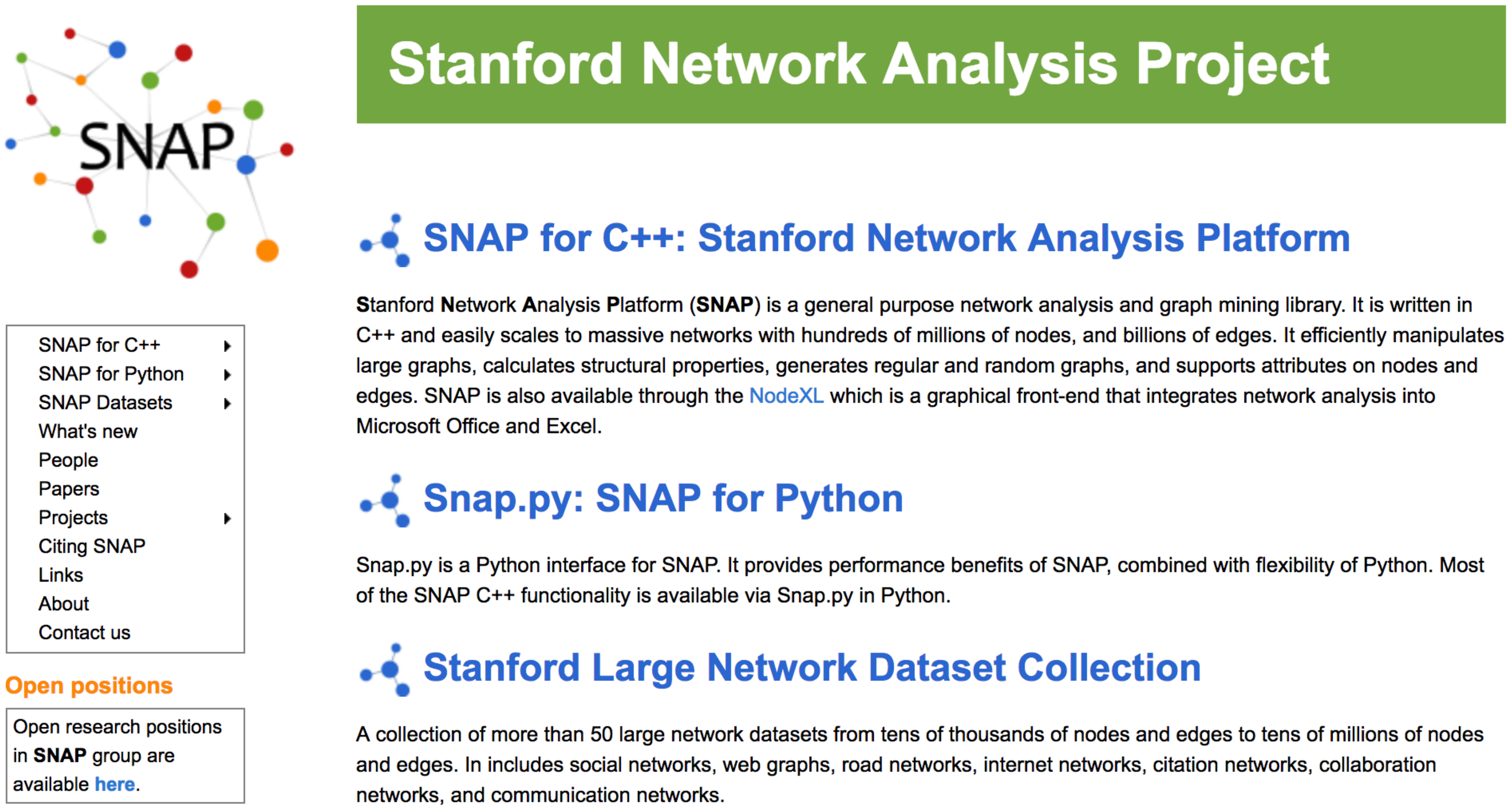
If |E|>>|V|, $averagecost = O(|V| * (|E|+|V|))$
If |E|$\approx$ $|V|$, $averagecost = O(|E|+|V|)$

○
○
●
○
○

Data analysis

Data Collection

- Directed Graph
- With weight
- One vertex can connect to all the other vertices



Stanford Network Analysis Project

Open positions

Open research positions in **SNAP** group are available **here**.

SNAP for C++: Stanford Network Analysis Platform

Stanford **N**etwork **A**nalysis **P**latform (**SNAP**) is a general purpose network analysis and graph mining library. It is written in C++ and easily scales to massive networks with hundreds of millions of nodes, and billions of edges. It efficiently manipulates large graphs, calculates structural properties, generates regular and random graphs, and supports attributes on nodes and edges. SNAP is also available through the NodeXL which is a graphical front-end that integrates network analysis into Microsoft Office and Excel.

Snap.py: SNAP for Python

Snap.py is a Python interface for SNAP. It provides performance benefits of SNAP, combined with flexibility of Python. Most of the SNAP C++ functionality is available via Snap.py in Python.

Stanford Large Network Dataset Collection

A collection of more than 50 large network datasets from tens of thousands of nodes and edges to tens of millions of nodes and edges. In includes social networks, web graphs, road networks, internet networks, citation networks, collaboration networks, and communication networks.

Analyzing Tool

tinyEWD.txt tinyEWD_qa-info.txt

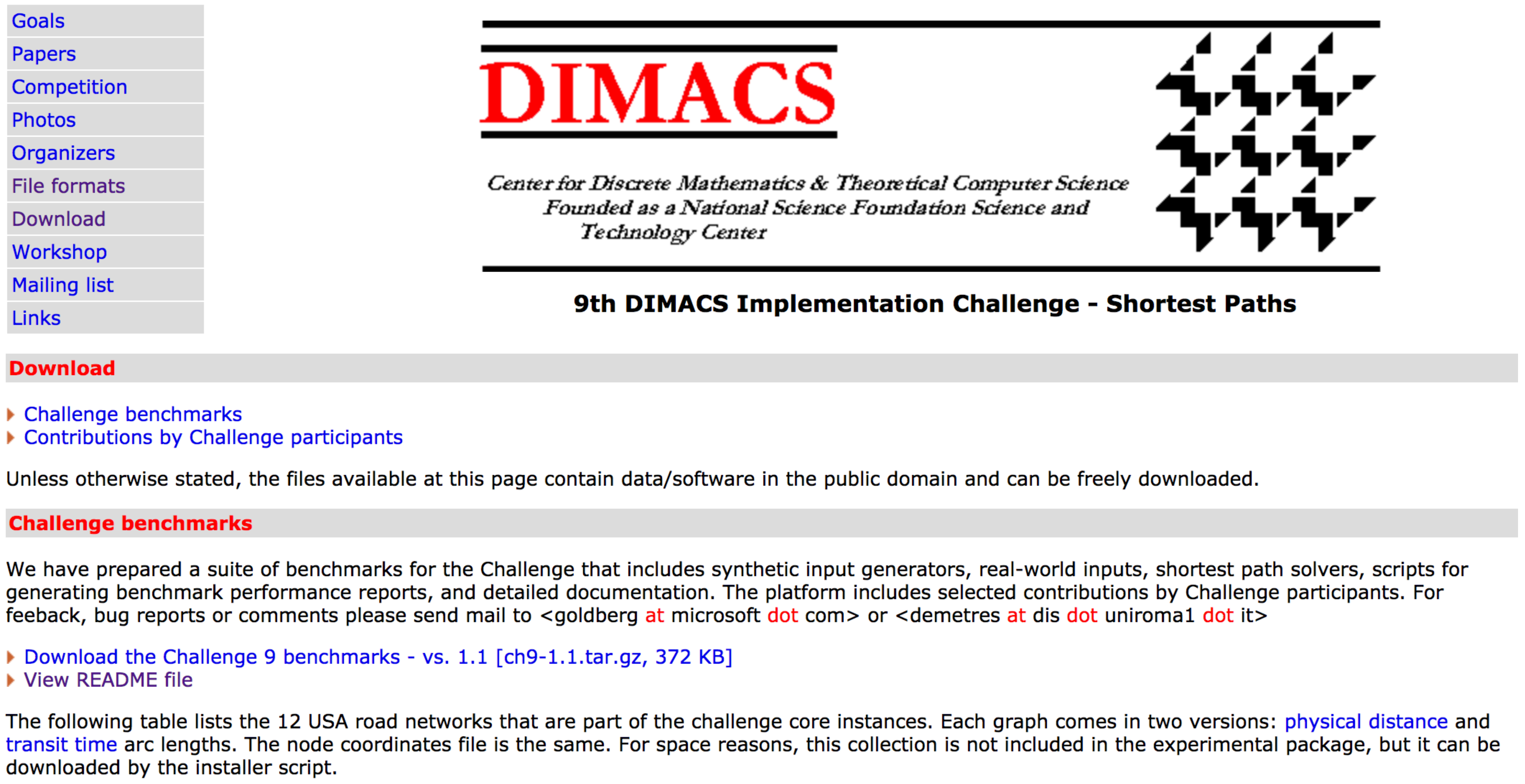
mediumEWD.txt mediumEWD_qa-info.txt

10000EWD.txt 10000EWD_qa-info.txt

1000EWD.txt 1000EWD_qa-info.txt

```
8
15
4 5 0.35
5 4 0.35
4 7 0.37
5 7 0.28
7 5 0.28
5 1 0.32
0 4 0.38
0 2 0.26
7 3 0.39
1 3 0.29
2 7 0.34
6 2 0.40
3 6 0.52
6 0 0.58
6 4 0.93
```

```
QA Stats: Directed
  Nodes:                     8
  Edges:                     15
  Zero Deg Nodes:            0
  Zero InDeg Nodes:          0
  Zero OutDeg Nodes:         0
  NonZero In-Out Deg Nodes:  8
  Unique directed edges:     15
  Unique undirected edges:   13
  Self Edges:                0
  BiDir Edges:               4
  Closed triangles:          3
  Open triangles:            22
  Frac. of closed triads:    0.120000
  Connected component size:  1.000000
  Strong conn. comp. size:   1.000000
  Approx. full diameter:     3
  90% effective diameter:   1.907692
```



- Goals
- Papers
- Competition
- Photos
- Organizers
- File formats
- Download
- Workshop
- Mailing list
- Links

DIMACS

Center for Discrete Mathematics & Theoretical Computer Science
Founded as a National Science Foundation Science and Technology Center

9th DIMACS Implementation Challenge - Shortest Paths

Download

- Challenge benchmarks
- Contributions by Challenge participants

Unless otherwise stated, the files available at this page contain data/software in the public domain and can be freely downloaded.

Challenge benchmarks

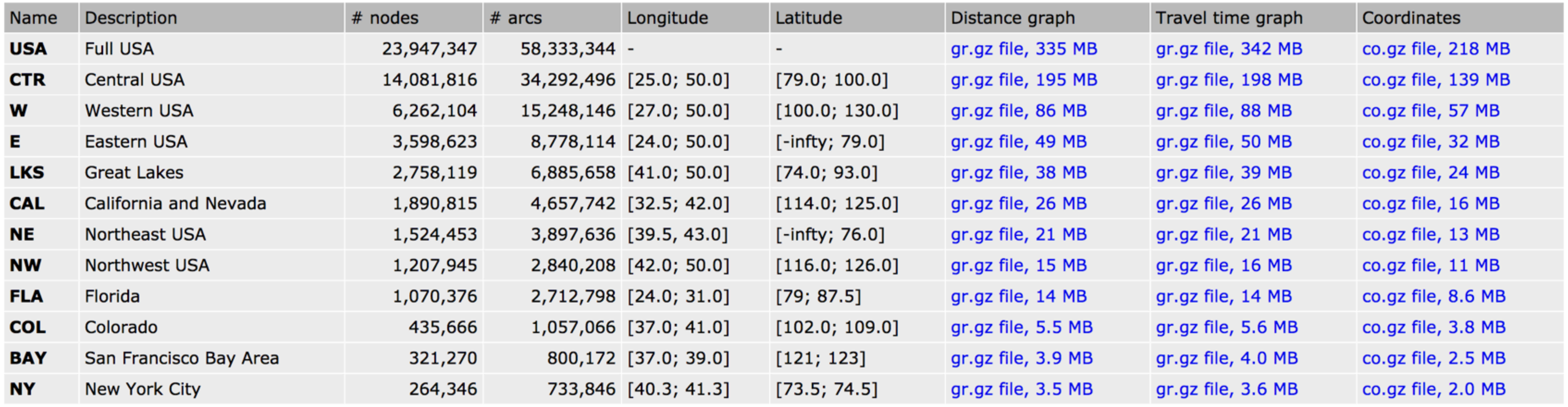
We have prepared a suite of benchmarks for the Challenge that includes synthetic input generators, real-world inputs, shortest path solvers, scripts for generating benchmark performance reports, and detailed documentation. The platform includes selected contributions by Challenge participants. For feeback, bug reports or comments please send mail to <goldberg at microsoft dot com> or <demetres at dis dot uniroma1 dot it>

- Download the Challenge 9 benchmarks - vs. 1.1 [ch9-1.1.tar.gz, 372 KB]
- View README file

The following table lists the 12 USA road networks that are part of the challenge core instances. Each graph comes in two versions: physical distance and transit time arc lengths. The node coordinates file is the same. For space reasons, this collection is not included in the experimental package, but it can be downloaded by the installer script.

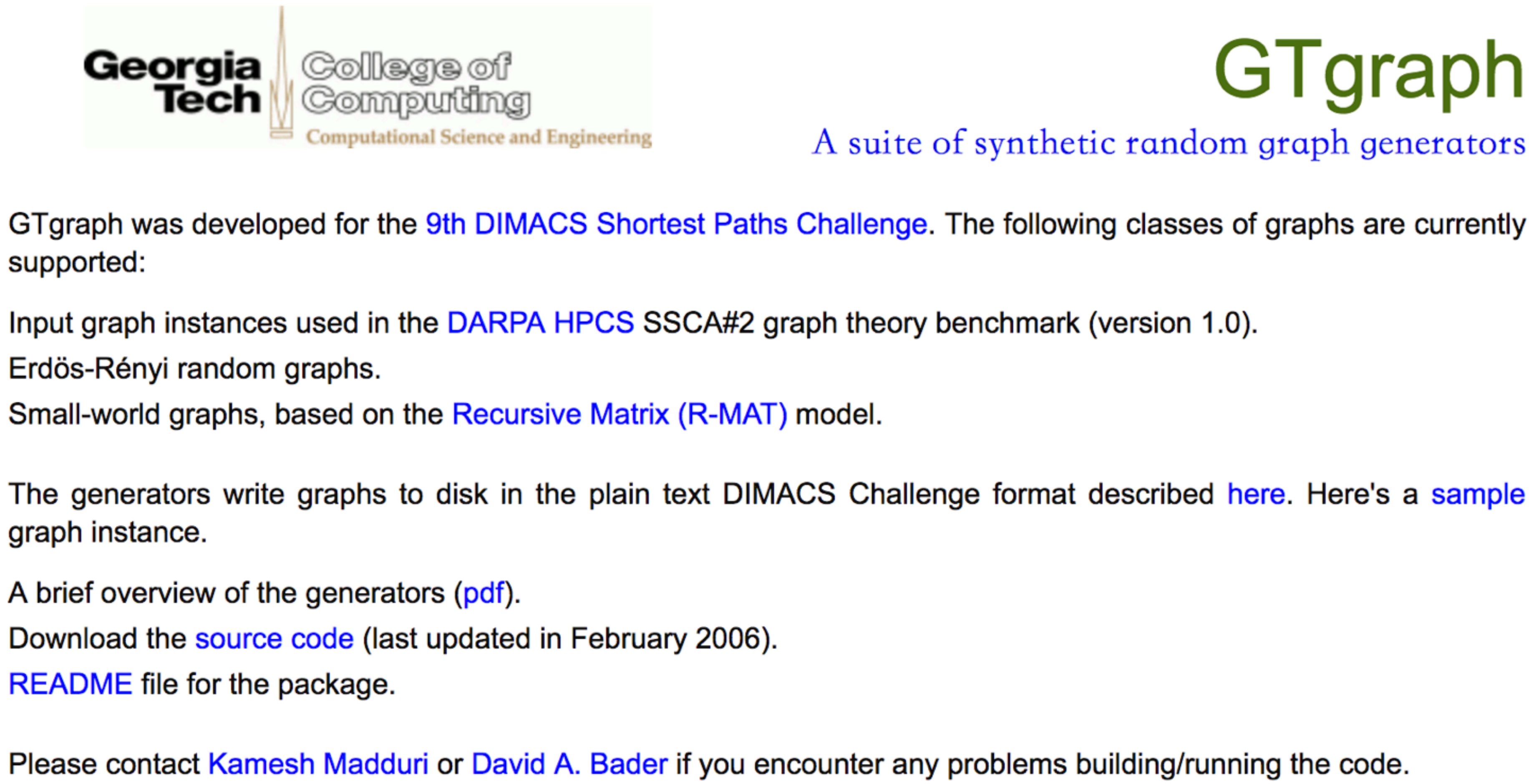
Real Data

- 12 USA road networks
- Physical distance and transit time arc lengths



Name	Description	# nodes	# arcs	Longitude	Latitude	Distance graph	Travel time graph	Coordinates
USA	Full USA	23,947,347	58,333,344	-	-	gr.gz file, 335 MB	gr.gz file, 342 MB	co.gz file, 218 MB
CTR	Central USA	14,081,816	34,292,496	[25.0; 50.0]	[79.0; 100.0]	gr.gz file, 195 MB	gr.gz file, 198 MB	co.gz file, 139 MB
W	Western USA	6,262,104	15,248,146	[27.0; 50.0]	[100.0; 130.0]	gr.gz file, 86 MB	gr.gz file, 88 MB	co.gz file, 57 MB
E	Eastern USA	3,598,623	8,778,114	[24.0; 50.0]	[-infty; 79.0]	gr.gz file, 49 MB	gr.gz file, 50 MB	co.gz file, 32 MB
LKS	Great Lakes	2,758,119	6,885,658	[41.0; 50.0]	[74.0; 93.0]	gr.gz file, 38 MB	gr.gz file, 39 MB	co.gz file, 24 MB
CAL	California and Nevada	1,890,815	4,657,742	[32.5; 42.0]	[114.0; 125.0]	gr.gz file, 26 MB	gr.gz file, 26 MB	co.gz file, 16 MB
NE	Northeast USA	1,524,453	3,897,636	[39.5, 43.0]	[-infty; 76.0]	gr.gz file, 21 MB	gr.gz file, 21 MB	co.gz file, 13 MB
NW	Northwest USA	1,207,945	2,840,208	[42.0; 50.0]	[116.0; 126.0]	gr.gz file, 15 MB	gr.gz file, 16 MB	co.gz file, 11 MB
FLA	Florida	1,070,376	2,712,798	[24.0; 31.0]	[79; 87.5]	gr.gz file, 14 MB	gr.gz file, 14 MB	co.gz file, 8.6 MB
COL	Colorado	435,666	1,057,066	[37.0; 41.0]	[102.0; 109.0]	gr.gz file, 5.5 MB	gr.gz file, 5.6 MB	co.gz file, 3.8 MB
BAY	San Francisco Bay Area	321,270	800,172	[37.0; 39.0]	[121; 123]	gr.gz file, 3.9 MB	gr.gz file, 4.0 MB	co.gz file, 2.5 MB
NY	New York City	264,346	733,846	[40.3; 41.3]	[73.5; 74.5]	gr.gz file, 3.5 MB	gr.gz file, 3.6 MB	co.gz file, 2.0 MB

Synthesis Data



Georgia Tech | College of Computing
Computational Science and Engineering

GTgraph

A suite of synthetic random graph generators

GTgraph was developed for the 9th DIMACS Shortest Paths Challenge. The following classes of graphs are currently supported:

Input graph instances used in the DARPA HPCS SSCA#2 graph theory benchmark (version 1.0).

Erdös-Rényi random graphs.

Small-world graphs, based on the Recursive Matrix (R-MAT) model.

The generators write graphs to disk in the plain text DIMACS Challenge format described here. Here's a sample graph instance.

A brief overview of the generators (pdf).

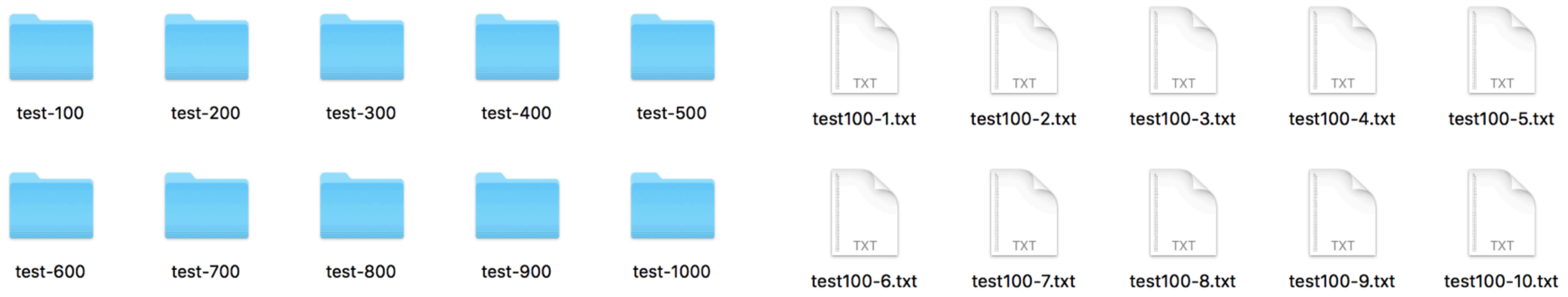
Download the source code (last updated in February 2006).

README file for the package.

Please contact Kamesh Madduri or David A. Bader if you encounter any problems building/running the code.

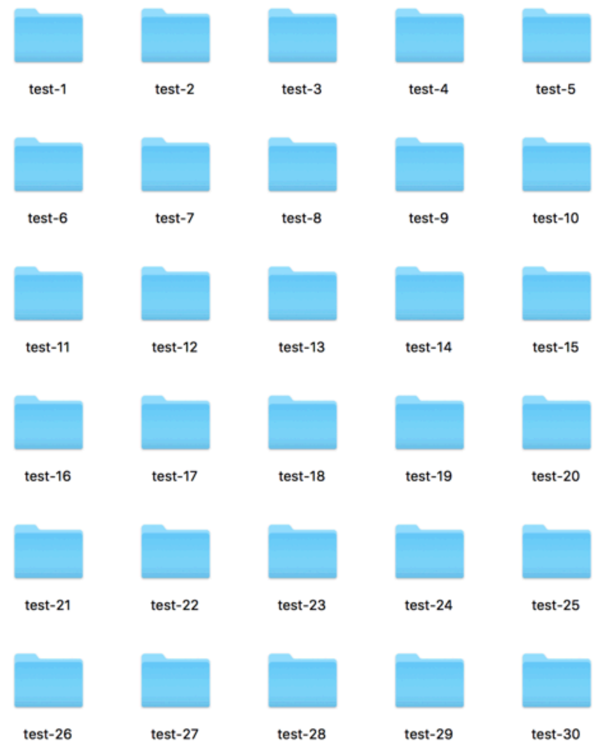
Synthesis Data

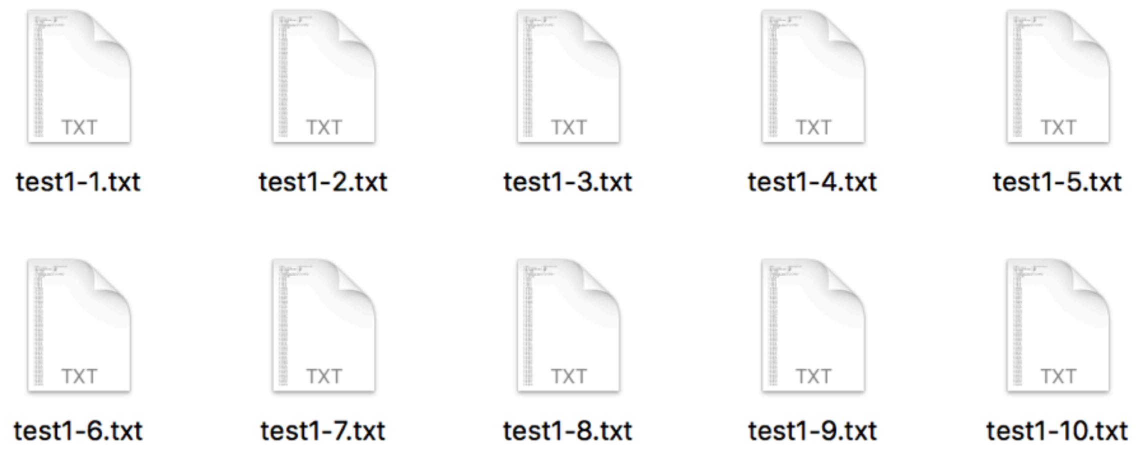
- Vertices from 100 to 1000 (interval 100)
- Edges from 1000 to 10000 (interval 1000)



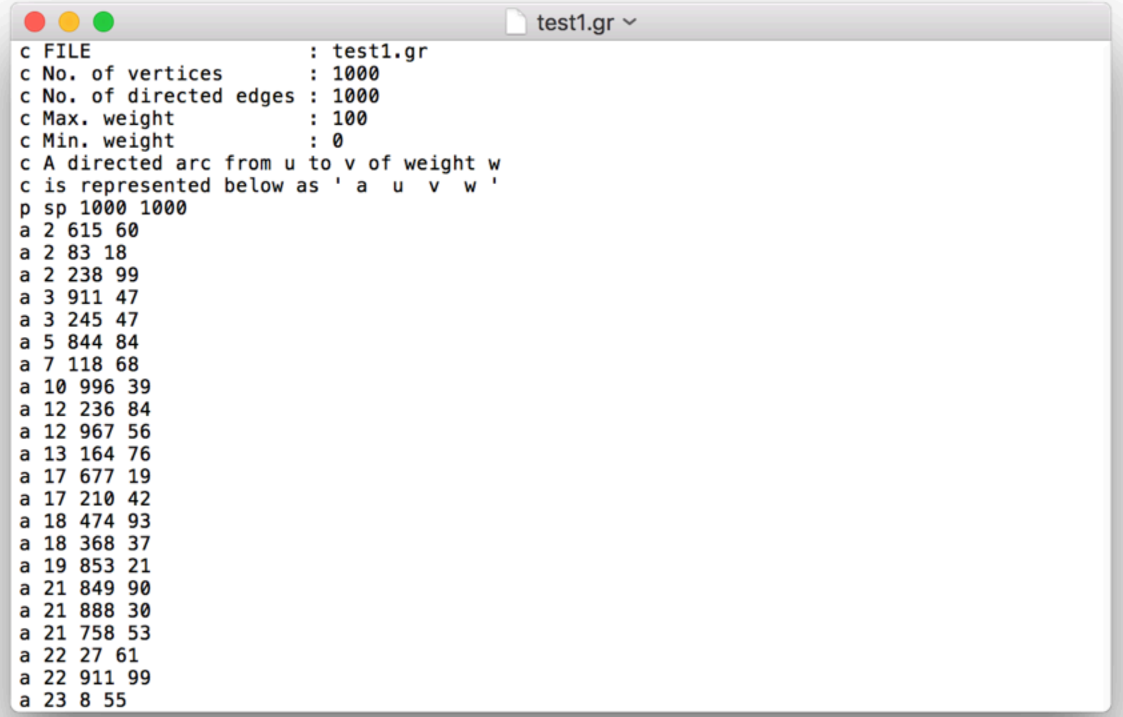
Synthesis Data

- Vertices 1000
- Edges from 1000 to 30000 (interval 1000)

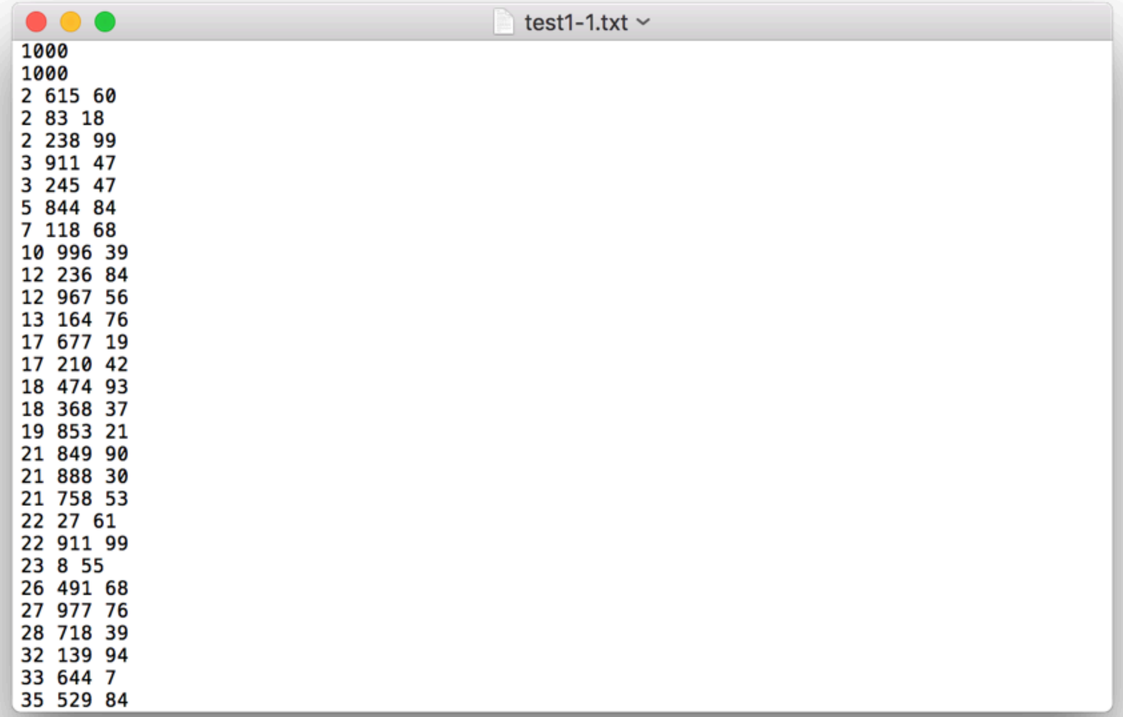




Data Preprocessing


test1.gr

```
c FILE                   : test1.gr
c No. of vertices        : 1000
c No. of directed edges  : 1000
c Max. weight            : 100
c Min. weight            : 0
c A directed arc from u to v of weight w
c is represented below as ' a  u  v  w '
p sp 1000 1000
a 2 615 60
a 2 83 18
a 2 238 99
a 3 911 47
a 3 245 47
a 5 844 84
a 7 118 68
a 10 996 39
a 12 236 84
a 12 967 56
a 13 164 76
a 17 677 19
a 17 210 42
a 18 474 93
a 18 368 37
a 19 853 21
a 21 849 90
a 21 888 30
a 21 758 53
a 22 27 61
a 22 911 99
a 23 8 55
```

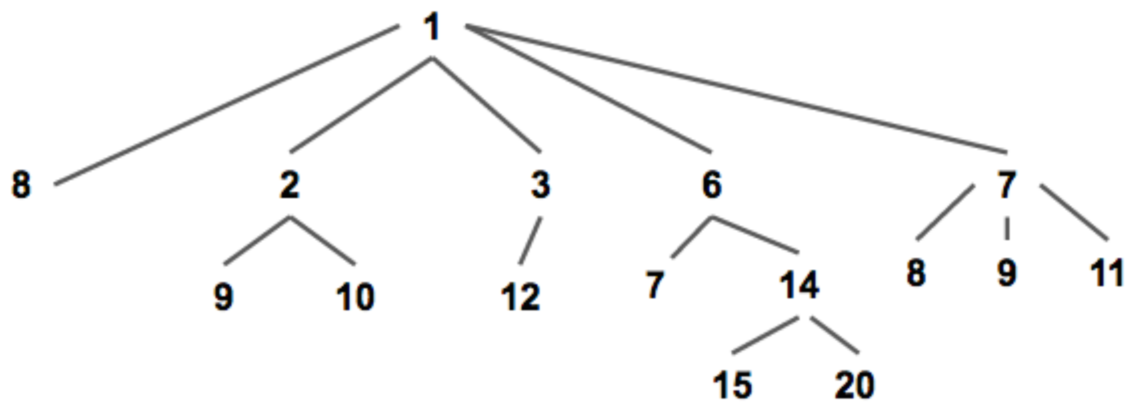

test1-1.txt

```
1000
1000
2 615 60
2 83 18
2 238 99
3 911 47
3 245 47
5 844 84
7 118 68
10 996 39
12 236 84
12 967 56
13 164 76
17 677 19
17 210 42
18 474 93
18 368 37
19 853 21
21 849 90
21 888 30
21 758 53
22 27 61
22 911 99
23 8 55
26 491 68
27 977 76
28 718 39
32 139 94
33 644 7
35 529 84
```

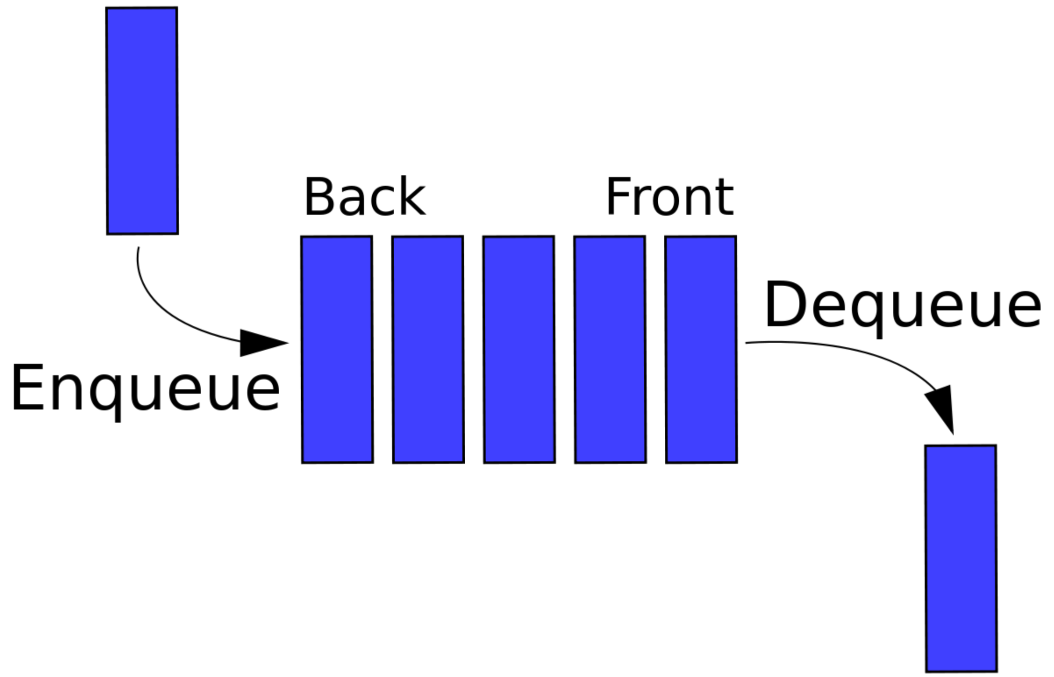
Results and discussion

Implementation Issues

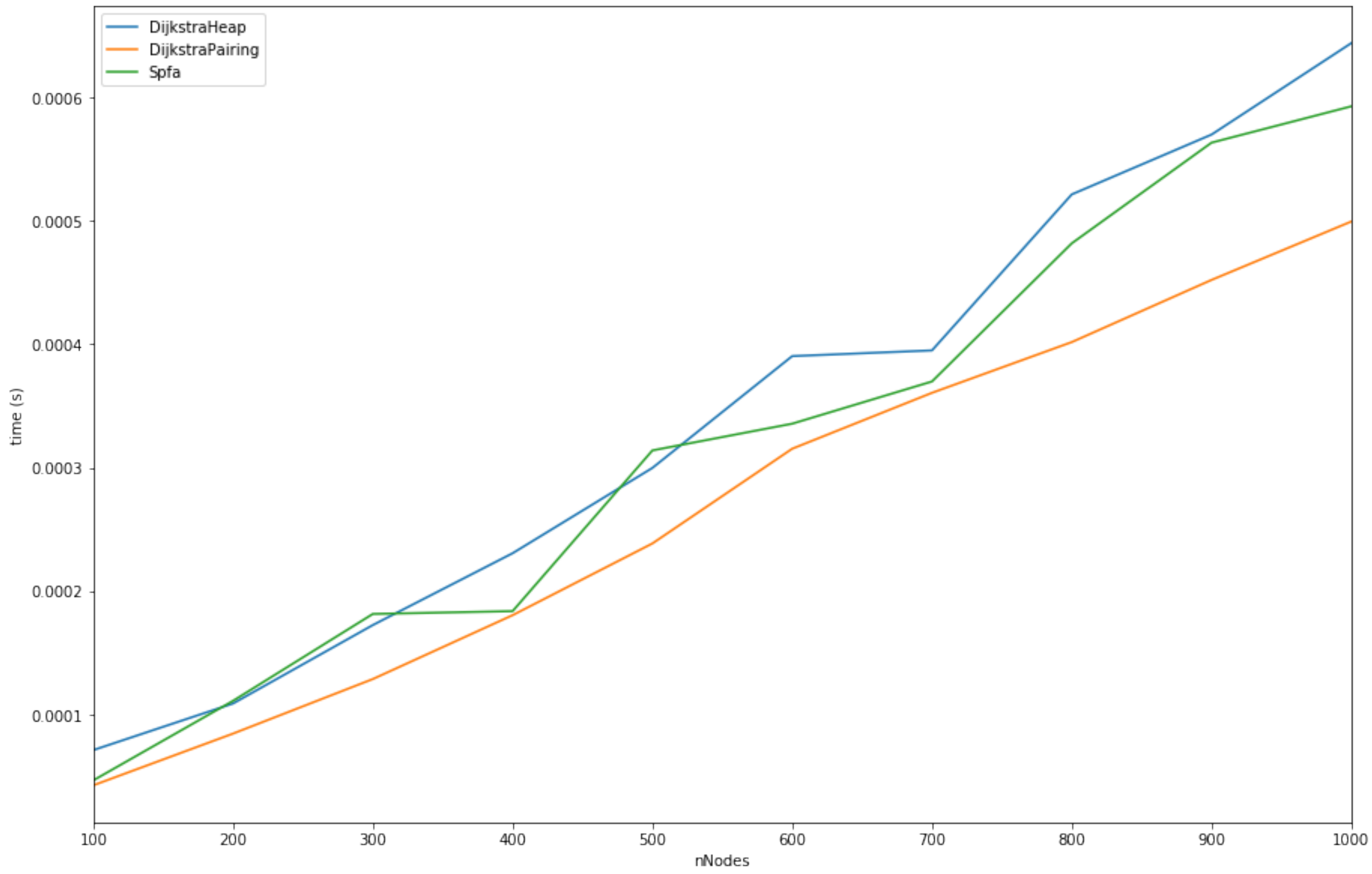
- Dijkstra
  - Priority Queue(Pairing Heap)



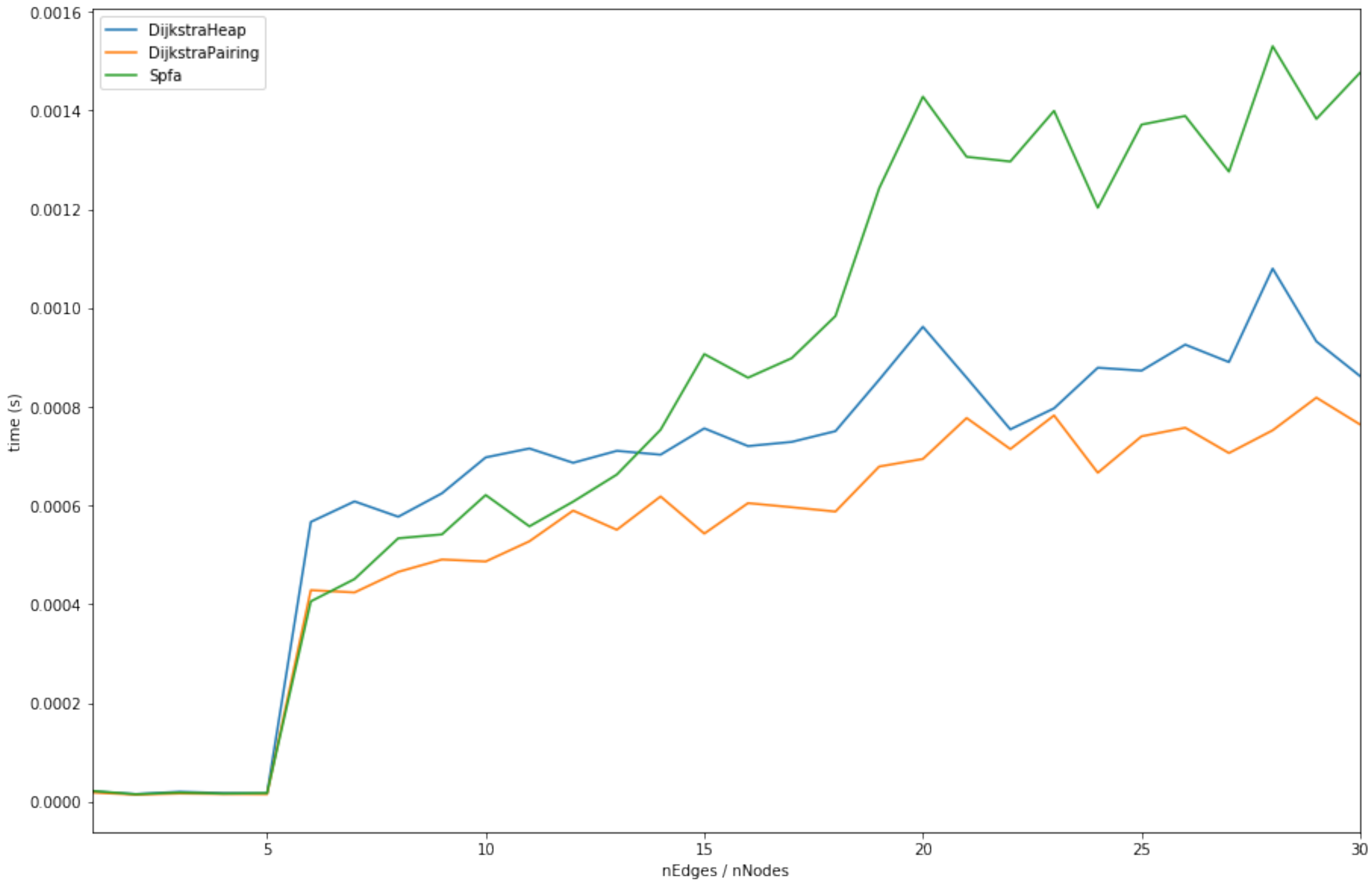
- SPFA
  - Normal Queue



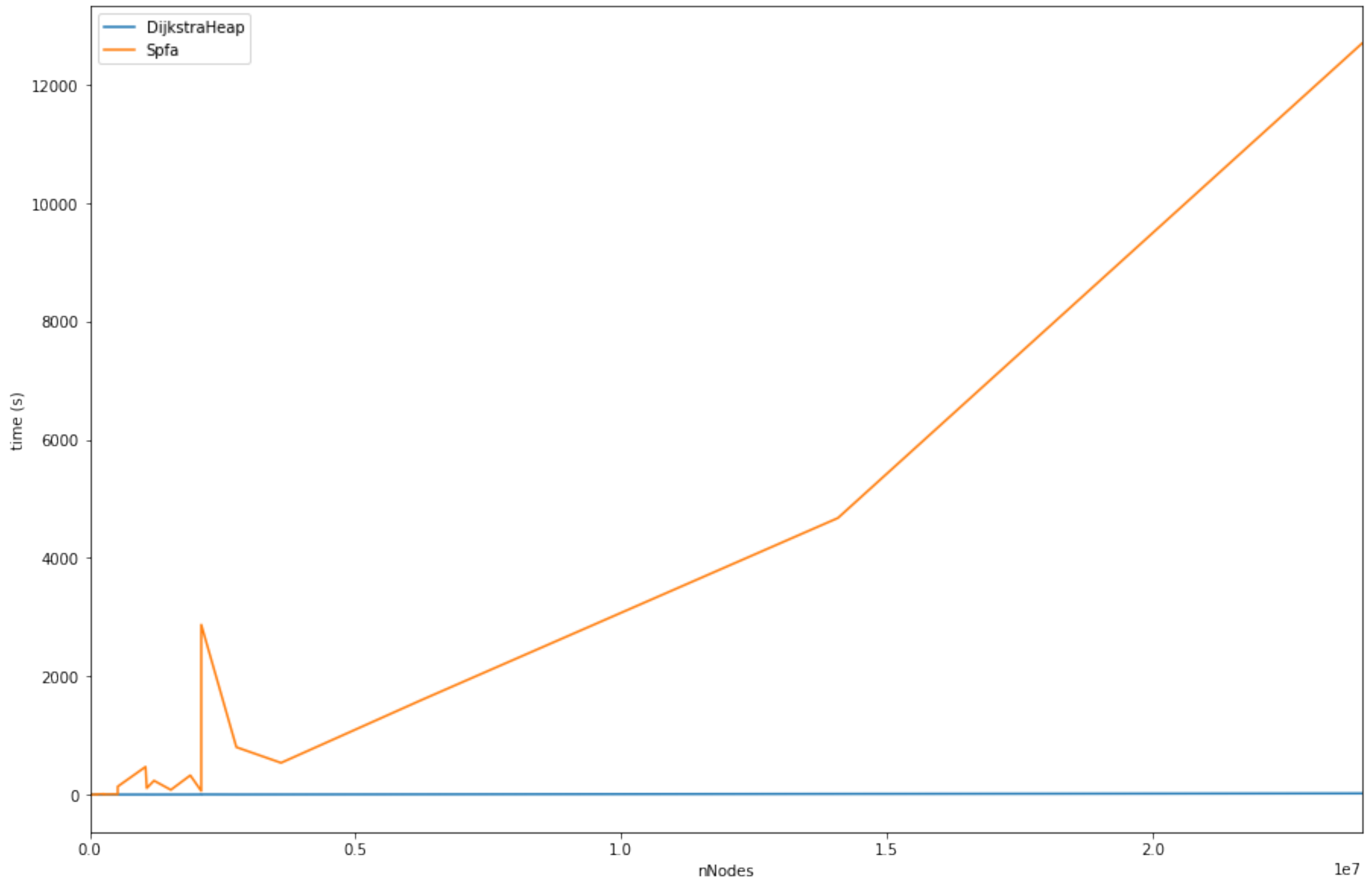
Result For different Nodes



Result For different Degrees



Result For Real Datasets



○
○
○
○
●

Conclusions

Conclusions From Experiments

- Dijkstra(with Fibonacci Heap) is actually the best algorithm for Single Source Shortest Path
- Spfa sometimes preforms well for small nodes, not too bad compared with Dijkstra(with normal priority queue)
- Spfa is easy to implement, you need to implement Fibonacci Heap for Dijkstra if you want good time performance and that is very hard

Thanks For Listening!

Any questions?