9th DIMACS Implementation Challenge*
Core Problem Families

Camil Demetrescu Andrew V. Goldberg David S. Johnson

October 29, 2006

1 General Comments

Below we describe core problem families. These should be adjusted to a specific study. For example, some of the problems may be too big for available hardware, and will be omitted. For other studies, such as a parallel algorithm running on a large cluster of processors, only very large problems are of interest. In this case synthetic problem sizes should be modified as appropriate.

2 Graph Families

Some graph families are used in several problems, so we describe graph families first.

Unless otherwise mentioned, all graphs we discuss are directed. When we talk about undirected graphs, we mean symmetric directed graphs, i.e., the length of arcs (v, w) and (w, v) are the same.

2.1 Random Graphs

These graphs are specified by the number of vertices n and the number of arcs $m \geq n$. To make sure that the graph is strongly connected, we use n arcs to create a Hamiltonian cycle. The we add arcs by repeatedly choosing a random vertex v and a random vertex $w \neq v$ and adding the arc (v, w). Note that one can have parallel arcs. Arc lengths are generated uniformly and independently at random from a specified interval of integers.

Random graphs have two important properties. First, they are expanders (if the average degree is high enough), and as a result the average

*`http://www.dis.uniroma1.it/~challenge9`

number of labeled vertices at a point of a shortest path computation is large. Second, they have poor locality.

2.2 Grid Graphs

Vertices of these graphs form a two-dimensional $x \times y$ grid. Each vertex is connected by and arc to the neighbors above, below, to the right, and to the left, except on the boundaries where these neighbors are not present. Arc lengths are generated uniformly and independently at random from a specified interval of integers.

Two grid shapes are of particular interest. *Long grids* have y fixed to a small constant and x growing with the number of vertices. On these grids, the number of labeled vertices at any time during an execution of Dijkstra's algorithm is small. *Square grids* have $x = y$. On these grids, the number of labeled vertices is moderate.

Note that we number vertices at random. This eliminates dependency of the running time on vertex layout in memory.

2.3 Tor Graphs

Tor graphs are similar to grid graphs, but the grid "wraps around", i.e., is embedded on the torus instead of the plane. These graphs have an easy to think about cycle structure and we use them mostly for negative cycle detection problem. The negative cycles always go in x-dimension (e.g., involve vertices with the same y coordinate).

The generator parameters are x, y, M, C, and a random seed. The first two parameters determine the grid size and shape. Parameter M determines the range of arc lengths, and C is the number of negative cycles in the graph. Except for the arcs whose length we define in the next paragraph, all arc lengths are selected uniformly at random from $[0, M]$.

To generate a graph, we first select C random y coordinates that determine the negative cycles. Consider a selected x coordinate. We select a y coordinate for it. All arcs corresponding to vertices with this x coordinate (going up or down) have length zero except for the arc going from the vertex corresponding to (x, y) to $(x, y + 1 \mod Y)$, which has the length of $-(1 + M/2)$. To "contain" the resulting negative cycle, we define the length of the arcs going left and right of the vertex corresponding to $(x, y + 1 \mod Y)$ to be $(1 + M/2)$.

2.4 Road Graphs

Road graphs are graphs of road networks. These graphs have two natural lengths functions, both of which are of interest: distance and transit times.

NAME	DESCRIPTION	VERTICES	ARCS	LATITUDE (N)	LONGITUDE (W)
USA	Full USA	23 947 347	58 333 344	-	-
CTR	Central USA	14 081 816	34 292 496	[25.0; 50.0]	[79.0; 100.0]
W	Western USA	6 262 104	15 248 146	[27.0; 50.0]	[100.0; 130.0]
E	Eastern USA	3 598 623	8 778 114	[24.0; 50.0]	$[-\infty; 79.0]$
LKS	Great Lakes	2 758 119	6 885 658	[41.0; 50.0]	[74.0; 93.0]
CAL	California and Nevada	1 890 815	4 657 742	[32.5; 42.0]	[114.0; 125.0]
NE	Northeast USA	1 524 453	3 897 636	[39.5, 43.0]	$[-\infty; 76.0]$
NW	Northwest USA	1 207 945	2 840 208	[42.0; 50.0]	[116.0; 126.0]
FLA	Florida	1 070 376	2 712 798	[24.0; 31.0]	[79; 87.5]
COL	Colorado	435 666	1 057 066	[37.0; 41.0]	[102.0; 109.0]
BAY	Bay Area	321 270	800 172	[37.0; 39.0]	[121; 123]
NY	New York City	264 346	733 846	[40.3; 41.3]	[73.5; 74.5]

Table 1: USA Road Networks derived from the TIGER/Line collection.

USA graphs. The USA graph has been assembled by Dominik Schultes[1] based on TIGER/Line data[2]. Only undirected (symmetric) version of the graph is available. We have cut the graph to obtain subgraphs of different sizes listed in Table 1. For each of them, we have extracted the largest connected component.

3 NSSP Problem

Two main classes of algorithms for this problem are label-setting (e.g., Dijkstra's) algorithms that scan each vertex at most once, and label-correcting (e.g., Pape's) algorithms which may scan a vertex more than once. The latter algorithm have bad worst-case performance but work well on some natural graph classes, both real-life and synthetic.

Best time bounds for this problem are of the form $O(m+nD)$, where D is the data structures overhead, which is small for state of the art algorithms. Therefore only sparse graphs are of interest, as for other graphs arc scans dominate the running time.

For some label-setting algorithms, the typical number of labeled vertices at a point during the execution of the algorithm is a key performance factor. Thus we have problem families where this number is large and families where it is small. For others, the range of the arc length values is a factor. We have problem families where the range varies.

The Square0 problems are relatively hard for some label-correcting algorithms, such as Pape's and Pallottino's.

We recommend using 10 different sources for each graph.

[1] `http://i10www.ira.uka.de/index-e.html`

[2] `http://www.census.gov/geo/www/tiger/tigerua/ua_tgr2k.html`

3.1 Random Graphs

We use random graphs with $m = 4n$ and consider two problem families. Let the range from which arc lengths are selected be $[0, \ldots, C]$.

Random4-n family: For this family, n grows and $C = n$. Suggested values of n are 2^i for $i = 10, \ldots, 21$.

Random4-C family: For this family, n is fixed and C grows. Suggested values are $n = 2^{20}$ and $C = 4^i$ for $i = 0, \ldots, 15$.

3.2 Grid Graphs

We use combinations of long and square grids. The long grid family has $y = 16$. For the square grid family of a target size n', we use $x = \lfloor \sqrt{n'} \rfloor$ and $y = \lfloor n'/x \rfloor$. The actual number of vertices, $n = xy$, may be slightly less than n'.

Long-n family: For this family of long grids, n grows and $C = n$. Suggested values of n are 2^i for $i = 10, \ldots, 21$.

Square-n family: For this family of square grids, n grows and $C = n$. Suggested values of n' (the target number of vertices) are 2^i for $i = 10, \ldots, 21$.

Square0-n family: These problems are the same as the Square-n problems, but the range of arc lengths for x-direction arcs is $[0, 0]$. As the result, the graph is partitioned into horizontal layers, with distances between vertices in adjacent layers determined by the minimum length of an inter-layer arc. The resulting problems are fairly natural, but difficult for incremental graph algorithms, such as Pape's and Pallottino's.

Long-C family: For this family of long grids, n is fixed and C grows. Suggested values are $n = 2^{20}$ and $C = 4^i$ for $i = 0, \ldots, 15$.

Square-C family: For this family of square grids, n is fixed and C grows. Suggested values are $n' = 2^{20}$ and $C = 4^i$ for $i = 0, \ldots, 15$.

3.3 Road Graphs

We use both road graphs with distance lengths and road graphs with travel time lengths.

USA-road-t family: This family includes road graphs with transit time lengths.

USA-road-d family: This family includes road graphs with distance lengths.

3.4 Discussion

The goal of the core problem families is to test algorithm dependence on the graph structure and arc length range.

Structure of the graphs in the core families leads to different shortest path tree structure and different behavior of label-setting algorithms. Random graphs have shallow shortest path trees (expected depth $\Theta(\log n)$), and have the property that during execution of a label-setting algorithm, the expected number of labeled vertices at a random point of the computation is large. In contrast, long grid graphs have deep ($\Theta(n)$) shortest path trees, and the expected number of labeled vertices is small. For square grids, the tree depth is moderate ($\Theta(\sqrt{\log n})$), and so is the expected number of labeled vertices.

The range of the problem arc lengths is important because algorithm performance may depend on the range, even if the range limit is not explicitly in the algorithm's worst-case time bound. The "C" problem families, for a fixed graph, vary the range of the length values to test how it affects performance.

Computing shortest paths on road networks is an important and well-studied problem. An interesting question is if, for the algorithms being studied, their performance on syntheric problems allows one to predict the relative performance and estimate the absolute performance on the road network problems.

4 Point-to-Point Problem

Remark: We allow preprocessing for the point-to-point problem. As a result, one can assume that input lengths are non-negative.

For a given graph, we generate two kinds of queries. For *random queries*, we select an s, t pair uniformly at random. We generate random queries for all graphs.

We also generate *local queries* for large graphs. These queries are generated as follows. We select s and random and run Dijkstra's algorithm from s to compute *rank* for all vertices, where a vertex has a rank of k is it is the k-th vertex scanned by the algorithm. Then for a set of values of i, we select a random vertex t_i from vertices with ranks $[2^i, 2^{i+1})$ and output an

s, t_i pair. Suggested values of i are from 2^9 [3] to $\lfloor \log n \rfloor$.

We use random queries in combination with all problem families for the single-source problem with non-negative arc lengths: Random4-n, Random4-C, Long-n, Long-C, Square-n, Square-C, USA-Road-d, and USA-Road-t. As these queries are much faster, we recommend using $1,000$ random pair queries for each graph and sufficiently large graphs (e.g., with $n \geq 2^{14}$).

For local queries, use all graph families, and for each family, use the biggest graph size in every family. The only exception are the road graphs, you should use maximal graphs (i.e., graphs that are not subgraphs of other graphs).

4.1 Discussion

All comments made in Section 3.4 about NSSP problem families apply to the P2P problem families. Another dimension of the P2P problem is the distribution of the source-sink pairs. Chosing those independently at random is natural and has been done in most previous studies, and we recommend this distributions as well.

In many applications, however, the source and the destination choises are related. One aspect of this relationship, query locality, is captured in our local query families. (This way of genetating local queries has been introduced by Sanders and Schultes.)

5 The SSP Problem

For problems when negative arcs are possible, we use the potential transformation that in some cases prevents obvious solutions. The potential transformation uses a parameter M that specifies the transformation magnitude. For each vertex v, we pick a potential $p(v)$ uniformly from the interval $[0, M]$. Then we obtain an equivalent problem by replacing the length of every arc, $\ell(v, w)$, by $\ell(v, w) - p(v) + p(w)$. Note that even if the original problem did not have negative arcs, the transformed problem may have them.

5.1 Random Graphs

We use the same generator as for the NSSP problem.

From NSSP Families to SSP Families The SSP problem is more general than the NSSP problem, and problem families for the former are also interesting for the latter. However, we suggest using the NSSP problem families with a potential transformation for $M = 2^{28}$ (are large value that

[3] We need to experiment with this value.

is unlikely to cause arc length overflows), to get SSP instances from NSSP instances.

Random4-neg family: For this family, we use a single graph size, $2,000,000$ vertices and $8,000,000$ arcs. Arc lengths are selected from the range $[-L, 32,000]$, with L changing. L is always nonnegative. For small L, the graph is unlikely to have cycles. For large L, the graph almost certainly has cycles. The transition to negative cycles is interesting, and we use denser test values for L around the transition point. We recommend the following values: $\{0, 2,000, 3,004, 3,005, 3,006, 3,007, 3,008, 3,009, 4,000, 8,000, 16,000, 32,000, 64,000\}$. For each size, we recommend one source and 10 graphs with a different seed. We apply the potential transformation with $M = 256,000$.

Tor family. For this family, we use the `tor` generator to produce "square" tors and vary the number of negative cycles. All graphs have $1,600 \times 1,600 = 265,000$ vertices. The number of cycles is 0 for TS-0 family, 1 for TS-1 family, 40 (moderate) for TS-m family, and 400 (high) for TS-h family. We use the `tor` generator parameter $M = 1$ so that cycles in the graph have a relatively small length. We apply the potential transformation with $M = 10,000$.

6 Discussion

An important fact is that some algorithms, such as the Bellman-Ford algorithm, are *potential invariant:* they visit the same sequence of vertices on the original and on the transformed graph. Other algorithms, such as Dijkstra's algorithm, are not potential-invariant. The problem structure (such as the shortest path tree depth) is still important, so we recommend running on the NSSP problems with a potential transformation. This tests algorithms on graphs with no negative cycles.

When a graph has negative cycles, the number of such cycles, their size, and their distribution in the graph may affect algorithm performance. Random graphs are a natural graph class, and one can adjust the probability that a given cycle in the graph is negative by adjusting the range from which the arc lengths are drawn.

The tor generator produces graphs with different (non-random) structure and allows tight control over the number and the size of negative cycles.

7 Negative Cycle Detection

Inputs for the negative cycle detection problem are those for the SSP problem minus the auxiliary file. As the problems have no sources, each graph

gives a single instance, and more graphs than in the SSP case may be used, however.

The discussion of Section 6 also applies to this case.