9th DIMACS Implementation Challenge*
Benchmark Solvers

Camil Demetrescu Andrew V. Goldberg David S. Johnson

February 21, 2006

1 General Comments

We provide solvers for some, but not all, problems. These solvers are intended to serve as benchmarks and to facilitate correctness checking. The solvers are not necessarily state of the art. All solvers in the core package assume arc weights in the input graph to be 32-bit signed integers. However, they use 64-bit words to calculate distances.

For many problems, good codes are very efficient and initialization can make the difference for timing. Unless stated otherwise, for such problems we solve a sequence of problems on the same graph and ignore parsing and the initial setup for timing purposes, but include re-initialization between problem instances.

2 NSSP Problem

The solver for the non-negative single-source shortest path problem, `sq`, is based on the smart queue data structure that combines the multi-level buckets and the caliber heuristic. The solver takes two arguments, the graph and the auxiliary file names. Recall that the auxiliary file specifies a set of sources. The solver runs on the sequence of problems specified by the input and prints out data in human-readable form in comment lines starting with "c". At the end it prints a line starting with "t" followed by the number of nodes and arcs, the min and max arc weight, and by two floating point numbers, first the average running time (in milliseconds, which can be changed by editing `main.c`) and the average number of vertices scanned. Note that "c" lines are written to `stderr`, while "t" lines are written to `stdout`: this makes it possible to redirect performance measures to a report file, while still getting the results in human-readable form on your terminal.

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

The `sqC` version of the solver is aimed at correctness testing. It outputs comment lines followed by checksum lines starting with "d". For a problem, the checksum is a long long integer that contains a sum of the distances of all vertices reachable from the source modulo 2^{62}. As computing the sum is expensive, the timing does not make sense and is not output.

3 P2P Problem

The solver for the point-to-point shortest path problem, `mbp`, is an implementation of the multilevel bucket algorithm that does not use the caliber heuristic (and therefore implements strict Dijkstra's algorithm) and stops when it is about to scan t. The correctness-checking version of the algorithm is `mbpC`; it outputs the s-t distance instead of the checksum.

Note that for most problems, `mbp` is be very slow compared to state of the arc codes.

Like the `sq` NSSP solver, `mbp` writes "c" lines to `stderr` and "t" lines to `stdout`.

4 SSP and NCD Problems

The single-source shortest path (SSP) problem is the general version of the NSSP problem where arc lengths may be negative. Note that if the graph can have negative cycles, and the goal is either to find distances from the source to all vertices reachable from it, or a negative cycle reachable from the source. In the negative cycle detection (NCD) problem is to find a negative cycle in the graph if one exists. The two problems are closely related. One can convert the latter into the former by creating a source vertex and connecting it to all vertices in the graph by zero length arcs. This is the approach we take with our solvers. Note, however, that this may not be the most efficient approach, especially if the graph has many small negative cycles.

The format for the SSP problem is the same as for the NSSP problem. The NCD problem format is simpler, as only the graph description (and no auxiliary file) is needed.

We provide two solvers for these problems, `bfct` and `gorc`. The former is based on Tarjan's version of the Bellman-Ford algorithm and the latter on an algorithm by Goldberg and Radzik. When supplied with two file arguments (the graph and the auxiliary files), the solvers assume that the input is an SSP problem. Given a single argument (the graph file), they assume the input is an NCD problem and use the reduction described above.

Like in the NSSP case, the `bfctC` and `gorcC` solver variants compute checksums that can be used for correctness checking. If the graph has no

negative cycle, the checksum is the sum of vertex distances. In the negative cycle case, the checksum if the length and the number of arcs on the negative cycle found. Note, however, that different algorithms may find different negative cycles.

The output format is similar to the NSSP case. It has a human-readable form and an abbreviated form.