Restless reachability in temporal graphs : algebraic methods and applications

In this supplementary we briefly describe the implementation, compilation and using the tool to compute restless reachability in temporal graphs.

1 IMPLEMENTATION

We use the source code from [3] as a starting point for our implementation. The finite-field arithmetic implementation is borrowed from [1].

Our implementation is written in `C` programming language with OpenMP constructs to achieve thread-level parallelism. Vector parallelism is achieved by enabling parallel executions of the same arithmetic operations, which make use of advanced vector extensions (AVX-2). Additionally, we make use of carry-less multiplication of one quadword (PCLMULQDQ) instruction set to enable fast finite-field arithmetic.

Our engineering effort boils down to implementing the recursions in Equation (1), (3) and evaluating the polynomial using 2^ℓ random substitutions for the x-variables. Recall that from the construction of the generating function that the y-variables are unique, as such we generate the values of y-variables using a pseudorandom number generator. The values of x-variables are computed using Equation (3). Our implementation loops over four variables: the outer most loop is over $[\ell]$, the second loop is over $[\tau]$, the third loop is over V and final loop over $\{0, \ldots, \delta(u)\}$ for $u \in V$.

In Equation (3), computing the polynomial $\chi_{u,\ell,i}(\mathbf{x}, \mathbf{y})$ is independent for each $u \in V$ if we fix ℓ and i, so the algorithm can be thread parallelised up to $|V| = n$ threads. We make use of openMP API using `omp parallel for` with default scheduling over vertices in V to achieve thread parallelism. Additionally, performing 2^ℓ random substitutions of x-variables is independent of each other so each of the 2^ℓ evaluations can be vector-parallelised. We achieve this by grouping the arithmetic operations on 2^ℓ random substitutions of variables in x and enabling the vector extensions from AVX-2. Remember that our inner-most loop is over $\{0, \ldots, \delta(u)\}$, so we arrange the memory layout as $n \times \tau$ to saturate the memory bandwidth. We also employ hardware prefetching by making the processor to fetch data for the subsequent computations while we are still performing the computations on the data which is already in the memory. For more technical details related to implementation engineering we refer the reader to previous work [2, 4].

Note that our current implementation takes $\mathcal{O}(n\tau + m)$ memory, which comes from the adjacency list representation of the temporal graphs.

1.1 Build

Our current build is configured for Intel architectures which support AVX-2 and PCLMULQDQ instruction sets. Other builds are possible but might require manual changes in the 'makefile'. We have executed our experiments using a workstation with a Haswell processor (for hardware details see experiments section in the paper). Many Intel microarchitectures including Broadwell, Skylake and Kabylake support AVX-2 instruction set and most likely the build should compile without changes to makefile in these architectures.

1.2 Graph generator

The graph generator is borrowed from [3], with minor changes to accommodate resting time. In the `regular-const`, `powlaw-const` graphs the resting time of all the vertices is constant specified by the input parameter `<rt>`. However, `regular`, `powlaw` graphs will have resting times assigned uniformly at random in range [`<rt>`]. The sample usage of graph generator is shown in Table 1 and 2.

1.3 Graph file format

The first line of the graph is a parameter line of the form `p motif <n> <m> <t> <rt> <is-dir>`. The parameter `<is-dir>` is 0 for undirected graphs and 1 for directed graphs. An edge (u, v, i) is described in a line `e u v i`. The resting time of each vertex $u \in V$ is reported in line `r` u $\delta(u)$. The source vertices are included using the line `s <num-sources> ...`, where `num-sources` is the

Table 1. **Using graph generator**

```
usage: ./graph-gen <type> <arguments>
available types (all parameters positive integers unless indicated otherwise):

  regular         <n> <d> <t> <rt> <ns> <nt> <seed>              (with 1 <= k <= n and n*d even)
  regular-const   <n> <d> <t> <rt> <ns> <nt> <seed>              (with 1 <= k <= n and n*d even)
  powlaw          <n> <d> <al> <w> <t> <rt> <ns> <nt> <seed>     (with al < 0.0, 2 <= w <= n, 1 <= k <= n)
  powlaw-const    <n> <d> <al> <w> <t> <rt> <ns> <nt> <seed>     (with al < 0.0, 2 <= w <= n, 1 <= k <= n)

--------------------------------------------------------------------------------
Arguments
<n>    : number of vertices, integer value 1 <= n <= 2^63
<d>    : degree, n*d even
<t>    : maximum timestamp, integer value 1 <= n <= 2^63
<rt>   : maximum resting time, integer value 1 <= rt <= t
<ns>   : number of sources, integer value in range 1 to <n>
<nt>   : number of separators, integer value in range 1 to <n>
<al>   : alpha, float value < 0.0
<w>    : weight, integer value 2 <= w <= n
<seed> : seed for random number generator, integer value 1 <= seed <= 2^63
```

Table 2. **Example: generating graphs using the graph generator**

```
/* d-regular graphs */
./graph-gen regular 1000 5 10 5 1 0 1234
./graph-gen regular-const 1000 5 10 5 1 0 1234

/* Power-law graphs */
./graph-gen powlaw 1000 10 -0.5 100 10 5 1 0 1234
./graph-gen powlaw-const 1000 10 -0.5 100 10 5 1 0 1234
```

number of source vertices followed by the list of source vertices. The separators are included using the line `s <num-separators> ...`, where `num-separators` is the number of separators followed by the list of separators.

1.4 Solving restless reachability

Our current implementation only supports single source vertex. The sample usage and outputs are reported in Table 4, 5 and 6. The runtime of execution is reported in the output line `command done [ <command time>ms <total time>ms]`. `command time` is the execution time of the oracle, where as `total time` include the preprocessing time such as reading the graph input and creating datastructures. The peak memory usage is reported in line `grand total [<grand total time ms>] peak: <peak memory>GiB`.

The restless reachability output is updated in the `<output-file>` specified in the command line argument. The first column is the `vertex-id` and the columns $\{2, \ldots, k\}$ in the output file is the minimum reachability time for $\ell = \{2, \ldots, k\}$, respectively. The last column is the minimum reachability time of each row. If a vertex is not reachable from source, it is indicated with -1 and a positive value indicates the vertex is reachable.

REFERENCES

[1] Andreas Björklund, Petteri Kaski, Łukasz Kowalik, and Juho Lauri. 2015. Engineering Motif Search for Large Graphs. In *ALENEX*. 104–118.

[2] Petteri Kaski, Juho Lauri, and Suhas Thejaswi. 2018. Engineering Motif Search for Large Motifs. In *SEA*. 1–19.

[3] Suhas Thejaswi and Aristides Gionis. 2020. Pattern detection in large temporal graphs using algebraic fingerprints. In *SDM*. 1–10.

[4] Suhas Thejaswi, Aristides Gionis, and Juho Lauri. 2020. Finding path motifs in large temporal graphs using algebraic fingerprints. arXiv:2001.07158 [cs.DS]

Table 3. **Sample graph input**

```
p motif 10 25 10 5 0
e 4 6 9
e 8 1 3
e 5 7 6
e 10 8 7
e 9 1 6
e 6 9 4
e 10 4 4
e 1 2 9
e 8 3 10
e 9 3 4
e 5 8 3
e 5 10 2
e 3 4 10
e 2 2 5
e 9 6 3
e 7 3 4
e 7 7 2
e 4 6 4
e 6 3 7
e 2 4 10
e 10 10 5
e 5 5 9
e 2 9 10
e 7 1 8
e 1 8 10
r 1 3
r 2 3
r 3 2
r 4 4
r 5 3
r 6 3
r 7 2
r 8 1
r 9 5
r 10 2
s 1 1
t 1 10
```

Table 4. **Using lister to compute restless reachability**

```
usage: ./LISTER_DEFAULT -pre <value> -optimal -<command-type> -seed <value> -in <input-file> -<file-type>
       ./LISTER_DEFAULT -h/help

  -<command-type>    : oracle             - decide existence of a solution
                       vloc               - single run of vertex localisation
                       vloc-finegrain     - fine-grained evaluation of the oracle
                       baseline           - exhaustive-search algorithm
 -k <value>          : integer value in range 1 to n-1
 -seed <value>       : integer value in range 1 to 2^32 -1, default value `123456789`
 -in <input-file>    : path to input file, `stdin` by default
 -out <output-file>  : path to output file, `reachability.out` by default
 -min                : reports minimum reachable time for each vertex to `output-file`
 -h or -help         : help
```

Table 5. **Example of using lister to solve restless reachability (reading graph from 'stdin').**

```
/* Reading graph from `stdin` */
./graph-gen/graph-gen powlaw 100000 10 -0.5 10 10 5 1 0 1234 | ./LISTER_DEFAULT -vloc-finegrain -k 3 -out reachability.out -seed 12345

/* Sample output */
invoked as: ../graph-gen/graph-gen powlaw 100000 10 -0.5 10 10 5 1 0 1234
powlaw: n = 100000, d = 10, alpha = -0.500000, w = 10, beta = 3.460163, norm = 7.460467
invoked as: ./LISTER_DEFAULT -vloc-finegrain -k 3 -out reachability.out -seed 12345
random seed = 12345
path length specified in command line, changing to `k = 3`
gen-count [powlaw, undirected]: n = 100000, m = 517209, t = 10, rt = 5
input: n = 100000, m = 517209, t = 10, rt = 5 [622.05 ms] {peak: 0.03GiB} {curr: 0.02GiB}
sources [1]:  66549
separators [0]:
build query: [zero: 0.47 ms] [pos: 7.97 ms] [adj: 25.78 ms] [adjsort: 3.45 ms] [rtime: 0.12 ms] done. [37.86 ms] {peak: 0.05GiB} {curr: 0.05GiB
finegrained-oracle [2, shade:0.15ms]: 0x5847799695814646 124.22 ms 1 {peak: 0.17GiB} {curr: 0.03GiB} [124.28 ms]-- true [125.14ms, 0.70ms]
finegrained-oracle [3, shade:0.34ms]: 0x473CE36ED9F1B50A 181.72 ms 1 {peak: 0.17GiB} {curr: 0.03GiB} [181.75 ms]-- true [182.68ms, 0.59ms]
command done [ 594.98 ms 1257.08 ms]
grand total [1257.08 ms] {peak: 0.17GiB}
host: maagha
build: multithreaded, prefetch, restless_path_genf, 2 x 256-bit AVX2 [4 x GF(2^{64}) with four 64-bit words]
compiler: gcc 9.1.0
```

Table 6. **Example of using lister to solve restless reachability (reading graph from input file).**

```
/* Reading graph from input file */
./LISTER_DEFAULT -vloc-finegrain -k 3 -in input-graph.txt -out reachability.out -seed 12345

/* Sample output */
invoked as: ./LISTER_DEFAULT -vloc-finegrain -k 3 -in input-graph.txt -out reachability.out -seed 12345
random seed = 12345
no input file specified, defaulting to stdin
path length specified in command line, changing to `k = 3`
input: n = 100000, m = 517209, t = 10, rt = 5 [348.60 ms] {peak: 0.03GiB} {curr: 0.02GiB}
sources [1]:  66549
separators [0]:
build query: [zero: 0.46 ms] [pos: 7.33 ms] [adj: 25.63 ms] [adjsort: 3.73 ms] [rtime: 0.11 ms] done. [37.36 ms] {peak: 0.05GiB} {curr: 0.05GiB
finegrained-oracle [2, shade:0.17ms]: 0x5847799695814646 125.37 ms 1 {peak: 0.17GiB} {curr: 0.03GiB} [125.43 ms]-- true [126.26ms, 0.66ms]
finegrained-oracle [3, shade:0.21ms]: 0x473CE36ED9F1B50A 160.13 ms 1 {peak: 0.17GiB} {curr: 0.03GiB} [160.16 ms]-- true [160.84ms, 0.47ms]
command done [ 559.97 ms 948.79 ms]
grand total [948.79 ms] {peak: 0.17GiB}
host: maagha
build: multithreaded, prefetch, restless_path_genf, 2 x 256-bit AVX2 [4 x GF(2^{64}) with four 64-bit words]
compiler: gcc 9.1.0
```