Supplementary files of the article "Finding small feedback arc set on large graphs"

The supplementary files are organized as follows.

In the first section, we describe format of `Results.xlsx`, which contains detailed results of our experiments.

In the second section, we describe how to obtain the datasets involved in our experiments.

In the third section, we describe the usage of source codes provided with this document.

Experimental Results

There are 13 sheets in the `Results.xlsx`.

- Results: We list the data obtained for each experiment separately
- Stats_Unweighted: Basic statistics of the graphs we tested.
- Stats_Weighted: We use random seed fixed from 1 to 10 to generate random positive integral weights for each arc. This sheet lists the sum of arc weights of the generated weighted digraphs.
- SolutionQuality_Heuristics: The size of the solutions found on unweighted graphs by the heuristics we tested.
- SolutionQuality_Heuristics_W: The size of the solutions found on weighted graphs by the heuristics we tested, measured by the weight of the feedback arc set divided by the sum of arc weights in the random graph, listed in the sheet 'Stats_Weighted'.
- RunningTime_Heuristics: Running time of the heuristics on unweighted graphs.
- RunningTime_Heuristics_W: Running time of the heuristics on weighted graphs.
- NumArcsAfterReduction: Number of arcs after running the reduction algorithms we tested.
- ArcWeightsAfterReduction: Sum of arc weights after running the reduction algorithms we tested.
- RunningTime_Reduction: Running time of the reductions we tested.
- Summary: Tables that summarizes our results.
- ResultsHeuristicNormalized: Intermediate sheet for calculating the standard deviation.
- StandardDeviation: Intermediate sheet for calculating the standard deviation.

Datasets

There are four datasets included by our experiments, ISCAS, ISPD, LAW, and SNAP.

The format conversion of the first two datasets was done by Ali Dasdan, which are publicly available at https://github.com/alidasdan/graph-benchmarks.

For LAW and SNAP, you can find all graphs we tested in https://law.di.unimi.it/datasets.php and https://snap.stanford.edu/data/index.html respectively.

Converting graphs with WebGraph format

Our algorithm cannot read graphs with compressed WebGraph format directly.

Therefore, we provide converter scripts(`conv.bat` for Windows and `conv.sh` for Linux) and associated java libs under the folder `webgraphconv`.

We tested our scripts on a PC running the following OSs.

1. Windows 10 Pro 21H2 with openjdk version "11.0.15" 2022-04-19 LTS installed.
2. WSL Ubuntu 20.04 with openjdk version "11.0.16" installed.

You can follow these steps to convert a graph in WebGraph format into a graph that can be read by our algorithms.

1. Download `.graph` and `.properties` files from LAW. For example, you can find the graph `word-assoc` in https://law.di.unimi.it/webdata/wordassociation-2011/ .
2. Put `.graph` and `.properties` files directly with our script.
3. Run `conv.bat <GraphNameBeforeExtension>`. For example, execute `conv.bat wordassociation-2011` in windows cmd or `./conv.sh wordassociation-2011` in linux bash.
4. The program outputs `wordassociation-2011.txt` in the same folder.

Read the graphs

You need to provide `InputGraphType` parameter in command line to our programs in order to specify the file format of the input graph.

We provide the this parameter for each graph we tested in the third column of `datasets.csv`.

You can check `\src\include\graph\io.hpp` for more details.

Codes

We provide our code under the `src` folder.

We successfully compile our codes(`fas_alg.cpp`, `fas_red.cpp`, `generate_synthetic_graphs.cpp`) with the following compilers(flags `-O3 -I"include"`):

- `gcc version 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)`
- `gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.1)`
- `gcc version 7.3.0 (x86_64-posix-seh-rev0, Built by MinGW-W64 project)`
- `gcc version 12.1.0 (x86_64-posix-seh-rev3, Built by MinGW-W64 project)`

Heuristics

You can use the following command lines to test the heuristic algorithms we implemented.

```
fas_alg InputGraphPath InputGraphType AlgName  [-r Seed] [-p] [-w MaxWeight] [-o OutputSolutionPath]
```

Parameter `InputGraphPath` denotes the path of the input graph file.

Parameter `InputGraphType` is described above.

You need to specify the heuristic algorithm you want to run according to the following table:

Algorithm	**InputGraphType**
The Greedy algorithm proposed by Eades, et al.	`Greedy`
Preprocess the input with SCC Decomposition, then solve with Greedy.	`scc+Greedy`
Preprocess the input with two-cycle elimination first, then SCC Decomposition, then solve with Greedy.	`scc+2cyc+Greedy`
Preprocess the input with SCC Decomposition, two-cycle elimination, the chain contraction, then solve with Greedy.	`scc+2cyc+chain+Greedy`
Preprocess the input with the non-recursive version of our Reduce algorithm, then solve with Greedy.	`RED+Greedy`
Preprocess the input with our Reduce algorithm, then solve with Greedy.	`REDstar+Greedy`
Preprocess the input with HCS reduction algorithm, then solve with Greedy.	`HCS+Greedy`
SimFASDC algorithm with parameter $r = 1/q$.	`RASq`
The FASDC algorithm with parameter $r = 1/3$.	`RASstar`

P.S. The implementation of our FASDC algorithm are templated, therefore you may only chose $q = 2, 3, 4, 6, 8, 12, 16, 24, 32, 48$. For more options, you will need to modify our source code.

The following list describes the usage of the flags.

- The flag `-r Seed` assigns a random seed for the following flags. The seed is fixed to zero by default. In our experiments, we fixed the seeds from 1 to 10 for each graph, each algorithm.
- The flag `-p` instructs the program to permute the label of vertices and the order of arcs in the input. This will not affect the vertex labels in the output.
- The flag `-w MaxWeight` instructs the program to assign random positive integral weights that uniformly distributes in `[1, MaxWeight]`. The maximum weight is set to one by default.
- The flag `-o OutputSolutionPath` instructs the program to output the linear ordering found by the algorithm into `OutputSolutionPath`.

Example:

```
fas_alg ../Datasets/ISCAS/s27.d DU Greedy -r 2023 -p -w 5 -o out.txt
```

The expected output is(the last column might vary with different machines):

```
s27.d_Greedy_5_2023,4,0
```

The first column denotes the task that has been executed, in the format of `[GraphName]_[AlgName]_[MaxWeight]_[Seed]`.

The second column denotes the weight of the minimum feedback arc set found by the algorithm.

The last column denotes the time consumed by the algorithm, in milliseconds.

In the `out.txt`, you may expect the following content:

```
9 2 0 18 4 54 10 3 1 19 53 5 50 32 31 37 41 30 12 13 21 39 14 22 26 6 43 42 40 15
25 35 29 28 27 17 16 34 48 38 33 20 52 51 45 44 36 47 24 23 8 7 46 11 49
```

The i-th number of the output denotes to the position of the vertex with number $i-1$ in the corresponding linear ordering. i.e., vertex 0 is ranked at the 10-th place in the result linear ordering, since the first integer in the file equals to 9.

Reductions

You can use the following command lines to test the reduction algorithms we implemented.

```
fas_red InputGraphPath InputGraphType AlgName  [-r Seed] [-p] [-w MaxWeight]
```

Parameter `InputGraphPath` denotes the path of the input graph file.

Parameter `InputGraphType` is described above.

You need to specify the heuristic algorithm you want to run according to the following table:

Algorithm	InputGraphType
Reduce the input with SCC Decomposition.	`scc`
Reduce the input with two-cycle elimination first, then SCC Decomposition.	`scc+2cyc`
Reduce the input with SCC Decomposition, two-cycle elimination, the chain contraction.	`scc+2cyc+chain`
Reduce the input with the non-recursive version of our Reduce algorithm.	`RED`
Reduce the input with our Reduce algorithm.	`REDstar`
Reduce the input with HCS reduction algorithm.	`HCS`

Example:

```
./fas_red ../Datasets/ISPD/ibm18.d DU REDstar -r 2023 -p -w 1
```

The expected output is(running time in the last column might vary with different machines):

```
ibm18.d_REDstar_1_2023,85227,105451,1343
```

The first column denotes the task that has been executed, in the format of `[GraphName]_[AlgName]_[MaxWeight]_[Seed]`.

The second column denotes the number of arcs in the reduced digraph.

The third column denotes the sum of arc weights in the reduced digraph.

The last column denotes the time consumed by the algorithm, in milliseconds.

Generating synthetic graphs

Source code: `gen_pa.cpp` and `gen_er.cpp`.

You may get the synthetic graphs under Erdos-Renyi(`gen_er.cpp`) model and the Preferential Attachment(`gen_pa.cpp`) model directly by compile and run.