Core Maintenance

This repository contains C++ codes and datasets for the paper.

Introduction

We present efficient algorithms for maintaining (α, β)-core (bi-core) over bipartite graphs. Our approach introduces the concept of bi-core numbers and uses them to narrow down the scope of updates caused by edge insertions and deletions. Based on this concept, we propose efficient algorithms for maintaining (α, β)-core, for both edge insertion and edge deletion.

Environment

The algorithms are implemented in C++ and compiled with the g++ compiler at -O3 optimization level. The development environment used for implementing and testing is:

- Linux version: Ubuntu 20.04.3 LTS
- Kernel version: 5.4.0-139-generic
- g++ version: 9.4.0

Datasets

We use eight large bipartite graphs, including seven real graphs (IMDB, WC, AR, DBLP, DE, DTI, WT) obtained from KONECT and one synthetic graph (PL) generated according to the bipartite network model[6], following the power-law distribution. You can generate PL by running the following command in the terminal:

```
1) cd random
2) g++ bigraph.cpp gephi.cpp main.cpp random.cpp utility.cpp -O3 -o rand.o
3) ./rand.o output_location 5000000 5000000 1000000000
```

Our paper contains the detailed statistics and original sources of the above datasets.

Dataset format:

- `graph.meta` file containing the number of vertices on each side and the number of edges.
- `graph.e` file containing all the edges.

Example:

```
graph.meta:
3 //number of vertices in U-side
3 //number of vertices in V-side
6 //number of edges

graph.e:
//U V
1 1
1 3
2 1
2 2
2 3
3 1
```

How to Run the Codes

A. Code Compilation

Both the `insertion-maintenance` and `deletion-maintenance` folders contain a `Makefile`. To compile the corresponding code:

```
1) cd insertion-maintenance/ or cd deletion-maintenance/
2) make
```

After compilation, an executable file named `bicore` will be generated.

B. Experimentation

1. Algorithms

- Recompute: the state-of-the-art bi-core decomposition algorithm.
- BiCore-Index-Ins* (BII): the bi-core maintenance algorithm to handle an edge insertion [26].
- BiCore-Index-Rem* (BIR): the bi-core maintenance algorithm to handle an edge deletion [26].
- Edge-Insert (EI): our proposed bi-core maintenance algorithm for handling an edge insertion.
- Edge-Delete (ED): our proposed bi-core maintenance algorithm for handling an edge deletion.
- BI-batch: the batch algorithm in [26].
- Edge-batch: our batch algorithm.

Parameter	Description
`graph_path`	The path of the graph.
`vertex_1/vertex_2`	(vertex_1, vertex_2) is the inserted or deleted edge.
`insert_path/delete_path`	The inserted/deleted edges list file.
`edge_num`	The total number of inserted and deleted edges.
`U_num/V_num`	The number of left/right side vertices in the graph.
`α/β`	(α, β) is the given integer pair.
`vertex`	The given vertex.
`is_left`	Indicates whether the given vertex belongs to U or V. 1 (True) for U, 0 (False) for V.

2. Preparation

Parameter of Command Line	Description	Command
`BBI`	Build BiCore-Index	`sh BBI.sh graph_path`
`BBN`	Compute all bi-core numbers	`sh BBN.sh graph_path`

3. Edge Insertions

3.1 Insert a specific edge (u_1, v_1)

Parameter of Command Line	Description	Command
`RCI`	Recompute	`sh RCI.sh graph_path vertex_1 vertex_2`
`BII`	BiCore-Index-Ins* (BII)	`sh BII.sh graph_path vertex_1 vertex_2`
`EI`	Edge-Insert (EI)	`sh EI.sh graph_path vertex_1 vertex_2`

3.2 Insert edges from an edge list file

Parameter of Command Line	Description	Command
`RCIS`	Recompute	`sh RCIS.sh graph_path insert_path`
`BIIS`	BiCore-Index-Ins* (BII)	`sh BIIS.sh graph_path insert_path`
`EIS`	Edge-Insert (EI)	`sh EIS.sh graph_path insert_path`

4. Edge Deletion

4.1 Delete a specific edge (u_1,v_1)

Parameter of Command Line	Description	Command
`RCR`	Recompute	`sh RCR.sh graph_path vertex_1 vertex_2`
`BIR`	BiCore-Index-Rem* (BIR)	`sh BIR.sh graph_path vertex_1 vertex_2`
`ED`	Edge-Delete (ED)	`sh ED.sh graph_path vertex_1 vertex_2`

4.2 Delete edges from an edge list file.

Parameter of Command Line	Description	Command
`RCRS`	Recompute	`sh RCRS.sh graph_path delete_path`
`BIRS`	BiCore-Index-Rem* (BIR)	`sh BIRS.sh graph_path delete_path`
`EDS`	Edge-Delete (ED)	`sh EDS.sh graph_path delete_path`

5. Batch Update

Parameter of Command Line	Description	Command
`BBatch`	BII-batch	`sh BBatch.sh graph_path edge_num U_num V_num`
`EBatch`	EI-batch	`sh EBatch.sh graph_path edge_num U_num V_num`

6. Query

6.1 (α, β)-core query:

Parameter of Command Line	Description	Command
`QCBI`	BiCore-Index	`sh QCBI.sh graph_path α β`
`QCBN`	BiCore number	`sh QCBN.sh graph_path α β`

6.2 community search:

Parameter of Command Line	Description	Command
`QSBI`	BiCore-Index	`sh QSBI.sh graph_path α β vertex is_left`
`QSBN`	BiCore number	`sh QSBN.sh graph_path α β vertex is_left`

6.3 α / β-offsets query:

Parameter of Command Line	Description	Command
`QOBI`	BiCore-Index	`sh QOBI.sh graph_path vertex is_left`
`QOBN`	BiCore number	`sh QOBN.sh graph_path vertex is_left`

Running Toy Graph

1. Toy Graph

- Number of vertices: 6, number of edges: 6
- Edge list:
  - u_1 - v_1
  - u_1 - v_3
  - u_2 - v_1
  - u_2 - v_2
  - u_2 - v_3
  - u_3 - v_1
- We can obtain the original bi-core numbers of all vertices:

U	V
u_1: (1, 3), (2, 2)	v_1: (1, 3), (2, 2), (3, 1)
u_2: (1, 3), (2, 2), (3,1)	v_2: (3, 1)
u_3: (1, 3)	v_3: (2, 2), (3, 1)

2. Edge Insertion

Suppose we want to insert edge (u_1, v_2), we can run:

```
sh EI.sh ../data/toy-graph/ 1 2
```

The updated bi-core number of all vertices are stored in "bi-core number". The content is listed as follows:

U	V
u_1: (1, 3), (3, 2)	v_1:(1, 3), (3, 2)
u_2: (1, 3), (3, 2)	v_2: (3,2)
u_3: (1, 3)	v_3: (3, 2)

3. Edge Deletion

Suppose we want to delete edge (u_1, v_2) after the edge insertion, we can run:

```
sh ER.sh ../data/toy_graph_AfterIn/ 1 2
```

The updated bi-core number of all vertices are stored in "bi-core number". The content is listed as follows:

U	V
u_1: (1, 3), (2, 2)	v_1:(1, 3), (2, 2), (3,1)
u_2: (1, 3), (2, 2), (3,1)	v_2: (3,1)
u_3: (1, 3)	v_3: (2, 2), (3,1)