Large Data Proc Final Project Report
Hengrui Xie & Yinzhe Ma

Result Table:

File name	Number of edges	Local Size of matching	Local Run Time	Local Rounds	GCP Size of matching	Run time on a 2x4 N1 core CPU in GCP	Rounds on a 2x4 N1 core CPU in GCP
com-orkut.ungraph.csv	117185083						
twitter_original_edges.csv	63555749						
soc-LiveJournal1.csv	42851237						
soc-pokec-relationships.csv	22301964						
musae_ENGB_edges.csv	35324	2389	10s	15	2415	277s	14
log_normal_100.csv	2671	48	7s	15	49	17s	14

Description of Theoretical Approach:

We first find maximal matching using Israeli-Itai algorithm:

Def Israeli-Itai_algorithm:

Input graph G = (V,E)

M = {}

***while** (there exists active vertices){*

every active vertex proposes to a random neighbor

every active vertex accept an arbitrary proposal, if there is any

every active vertex is randomly assigned value 0 or 1

if a proposed edge is from 0 and 1, then join M and deactivate the endpoints of those joined M}

Output M

Next, find augmenting path of length 3 based on the maximal matching we get from the first step and generate maximum matching

Def augmenting_path:

Input graph G = (V,E)

M=Israeli-Itai_algorithm(G)

***while** (there exists an augmenting path p relative to M){*

M ← M+p}

output M

The intuition behind such a strategy is that we want to use a relatively simple algorithm to find a certain number of matchings to ensure we have a set of benchmark results; then we would implement the more challenging *augmenting_path* function to help us discover more matching edges.

Discussion of Implementation:

Even though the idea behind Isareli-Itali algorithm seems simple and intuitive, we encountered a fair share of difficulties during implementation. First, in order to accomplish the randomness behind proposing to a neighboring edge and accepting a random proposal, we had to assign each edge with a random float; and by using two *aggregateMessage* methods, we were able to achieve just that. Secondly, to accommodate for limited heap space, we have not only increased the driver and executor storage space but also terminated the algorithm early so that we would not waste any unnecessary heap space.

> -**The intuition behind terminating the algorithm early**:
> Through testing, we've found that the algorithm becomes extremely inefficient towards the end of the algorithm (when the # of active edges becomes extremely small, typically 0.002*original active edges); therefore, we have decided to adopt early stoppings to avoid such memory waste. The screenshot below illustrates such scenario:

```
(base) yinzhes-mbp-2:finalproject yinzhema$ spark-submit --class final_project.main --master local[*] target/scala-2.12/project_3_2.12-1.0.jar data/log_norma
l_100.csv result.csv
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/usr/local/Cellar/apache-spark/3.1.1/libexec/jars/spark-unsafe_2.12-3.1.1.jar) t
o constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
21/05/03 18:18:33 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
21/05/03 18:18:34 WARN SparkContext: Using an existing SparkContext; some configuration may not take effect.
21/05/03 18:18:35 INFO FileInputFormat: Total input files to process : 1
21/05/03 18:18:35 WARN BlockManager: Block rdd_18_0 already exists on this machine; not re-adding it
1196
640
21/05/03 18:18:36 WARN BlockManager: Block rdd_210_1 already exists on this machine; not re-adding it
261
129
30
19
6
1
1
1
1
1
0
The alogrithms ran for 13 rounds.
=================================
compute function completed in 4s.
=================================
```

Discussion of Merits:

As we previously mentioned, our strategy in general ensured that we would have a set of benchmark results to fall back onto while giving us a reasonable number of matching edges. This is why we chose Isareli-Itali algorithm as our benchmark algorithm; it is relatively easier to implement, and the algorithm is intuitive to understand. In terms of runtime, it theoretically guarantees log(n) rounds with an approximately O(k*log(n)) rounds guarantee in practice. In terms of result, we are confident that the outputs for each dataset are matched edges (such claim is verified by the verifier program provided). Please see our results on the github repository.

Source Code: https://github.com/yinzhema/LargeDataProc_FinalProject