CS 744: BIG DATA SYSTEMS

Shivaram Venkataraman
Fall 2018
ADMINISTRIVIA

- Midterm grades up today
- Pick up papers office hours today or Tuesday class
- Course Projects: round 2 meetings
GRAPH MINING
WHATS DIFFERENT?

Graph Analytics

Examples
PageRank
Shortest path
Connected components
...

Graph Mining

Examples
Counting motifs
Frequent sub-graph mining
Finding cliques
...

GRAPH MINING: DEFINITIONS

Graph $G = (V, E)$ Vertices and edges have unique ids.

Embedding sub-graph of $G$, i.e., subset of vertices and edges
- Vertex-induced – start from vertices, include all edges for vertices
- Edge-induced – start from edges, include all endpoint vertices

Pattern any arbitrary graph

Pattern is a template, embedding is an instance
**AUTOMORPHISM, ISOMORPHISM**

Embedding is isomorphic to pattern iff
one-to-one mapping between vertices, edges
vertex mapped has same label
edges connect matching vertices

Embedding $e$ is **automorphic** to $e'$ iff
contain same edges and vertices
Motifs: Connected patterns that are non-isomorphically

- $k=3$ – two patterns
- $k=4$ – six patterns

Goal:
Find counts of each pattern in graph
FILTER PROCESS MODEL

Two UDFs: Filter embedding $\Phi$ and Process embedding $\pi$

Algorithm

Initial embedding set $I$
For each embedding in set
  $C \leftarrow$ generate embeddings (add one vertex)
  For each embedding $e$ in $C$
    If $\Phi(e)$:
      $F \leftarrow F \cup \pi(e)$
  Terminate if $F$ is empty
else loop with $I \leftarrow F$

BSP Execution by parallelizing this loop
AGGREGATION FUNCTIONS

Aggregation functions:
Filter function $\alpha$, Aggregate function $\beta$
Similar to `groupByWindowAndApply`?

Consistency properties
If embeddings are automorphic, all UDFs return same value
Anti-monotonicity – filter return same values for extensions
OTHER APPROACHES

Think like Vertex
- Vertex has local embedding
- “Push” message to border vertex

Cons
- Highly connected vertex → hotspot
- Duplicate messages, one per border

Think like Pattern
- Don’t materialize embeddings
- Store patterns, recompute embeddings on the fly

Cons
- Partition by pattern (fewer ?)
- Popular pattern, load imbalance
ARABESQUE API: EXAMPLE

boolean filter(Embedding e){
    return (numVertices(e) <= MAX_SIZE);
}

void process(Embedding e){
    mapOutput (pattern(e),1);
}

Pair<Pattern,Integer> reduceOutput ( Pattern p, List<Integer> counts){
    return Pair (p, sum(counts));
}
DISTRIBUTED EXECUTION

Apache Giraph based distributed implementation

Synchronous super-steps (BSP)
- Workers receive messages sent previously [Embeddings]
- Process messages [Filter Process]
- Send new messages to be delivered [Aggregate output?]

Can be implemented in any BSP system? (e.g., Spark)
EXPLORATION STRATEGY

Goals
- Prune embeddings that are “identical” (i.e. automorphic)
- Need to do this without coordination (why ?)

Approach
- Determine a “canonical” embedding (unique and extensible)
- Canonical property
  - Start with smallest id
  - Add the neighbor with smallest id not visited yet
- Incremental check while adding vertex to embedding
EFFICIENT STORAGE: ODAG

Storage model: Ordered lists of vertex / edge ids (integers)

ODAG format
- Store all first elements of embeddings in one array (and so on)
- Links between array indices if embedding has a such link
- Could lead to spurious embeddings
EFFICIENT STORAGE: ODAG

ODAG benefits

- N vertices can have up to $N^k$ embeddings of size $k$
- ODAG upper bound $O(k \cdot N^2)$ ($k \ll N$)

Using ODAGs

- Avoid spurious embeddings using filter, aggregateFilter

Merging ODAGs

- Every worker creates ODAG outputs
- Use map-reduce to do the merge!
- Map each entry based on position to worker
OTHER OPTIMIZATIONS

Partitioning Embeddings
- Performed at start of every iteration
- Round-robin scheme with block size $b$
- Estimate embeddings that start from a vertex

Two-level aggregation
- Need to aggregate by pattern. Equality requires isomorphism check
- Quick pattern calculated locally and aggregated
- Use canonical pattern to do second level aggregation
Graph Mining: new workload that is compute and data intensive

First system to do distributed graph mining

Challenges: Lots of intermediate state (trillions of embeddings)

Key ideas:
  - Filter / prune embeddings using canonical definition
  - Efficient storage using ODAGs