1. **Graph Data Management**

**Graph Analysis Tasks Vary Widely**
- Different types of Graph Queries
- Continuous/Real-Time Analysis
- Batch
- Machine Learning

**Many different ways to deal with graph data**
- Graph Databases (neo4j, orientDB, RDF stores)
- Distributed Batch Analysis Frameworks (Giraph, GraphX, GraphLab)
- In-Memory Systems (Ligra, Green-Marl, X-Stream)
- Many research prototypes / custom indexes

2. But first…Where is your data?

- Users’ data typically in RDBMSs or Key-Value Stores with some sort of schema
- Graph systems require lists of nodes & edges
- Extraction step often overlooked but can be quite involved
  - User needs to write custom SQL queries for ETL
  - Can be unintuitive & time consuming
  - Large selectivity estimation errors due to complex joins
  - Need to repeat every time database is updated

3. **GraphGen**

- A software layer over relational/structured databases (implemented as a library)
- User specifies graph extraction queries in a **Dataflow-based DSL**
- Can serialize the graph and load it into other frameworks/libraries
- Exposes vertex-centric API or direct graph access through Java API
  - WIP: Supporting a Dataflow Based DSL for Querying/Analytics

4. Condensed Representation

**Key Challenge #1:** Graphs often orders-of-magnitude larger than input. **May not fit in-memory!**

**Solution:** Instead extract a **Condensed Representation**

1. Translate Nodes to SQL and execute them.
2. Edges statements (cyclic, aggregation-free) are split by join.
3. For each join between R, R’, retrieve number of distinct values for the join condition attribute(s).
4. Every join where |R∪R’| ≤ 2 → (R∪R’)-marked large-output
5. Create virtual nodes for every large-output join. Execute rest of joins in database

5. **Duplicate Elimination**

**C-DUP**

- On-the-fly de-duplication caching every getNeighbors() call
- Great for graph queries that touch small portions of the graph
- Most storage-efficient solution

**DEDUP-1**

- Works on Multi-layered Graphs
- Apply algorithm at every layer
- **Main idea:** Use bitmaps at every virtual node to avoid duplicate paths

6. Structural De-duplication

**Bitmaps**

- Add a bitmap at every virtual node
- Structural de-duplication of C-DUP
- Single-path per pair of neighbors
- Most portable solution

**Solution:** Override the getNeighbors() iterator to enable any algorithm over the Condensed Representation

**C-DUP**

- **Noise Virtual Nodes:** First, choose which real node to remove randomly
- **Noise Real Nodes:** Same, remove all duplication for each real node u before moving on the next one

**Greedy Virtual Nodes:**

- Heuristic: Compute global benefit/cost ratio of disconnecting real node u from virtual node p1 vs p2

**Greedy Real Nodes:**

- Heuristic: Computes benefit based on reduction in edges resulting from using virtual node p1 vs p2

**DEDUP-1: Algorithms**

- Noise Virtual Nodes: First, choose which real node to remove randomly
- Noise Real Nodes: Same, remove all duplication for each real node u before moving on to the next one
- Greedy Virtual Nodes: Heuristic: Compute global benefit/cost ratio of disconnecting real node u from virtual node p1 vs p2
- Greedy Real Nodes: Heuristic: Computes benefit based on reduction in edges resulting from using virtual node p1 vs p2

**De-duplication:** Given a condensed graph remove edges until there is one path between each pair of neighbors

**Bi-clique Compression:** Partition edges into minimum set of bipartite cliques (NP-Complete) [Feder, Motwani ’94]

**Some complexity, same output, different input**

7. De-duplication using Bitmaps

**Optimization Problem**

- Let O(V) the set of real nodes connected to virtual node Vn.
- Given a real node v, its virtual neighbors {v1,v2,...,vn} find the smallest subset of O(V1),O(V2),...O(Vn) that covers their union
- Heuristic based on standard greedy set cover

**Main idea:** Use bitmaps at every virtual node to avoid duplicate paths

**Bad Bitmap placement**

**Good Bitmap placement**

8. Trade-offs and Benefits

**Small Datasets**

**Iteration Performance on Condensed Graphs**

**Integration with Apache GraphLab**

**Large Datasets**

**GraphGen:** Efficient in-memory extraction and analysis of larger-than-memory graphs hidden within relational databases