Declarative Abstractions and Scalable Platforms for Big Data Analytics

Amol Deshpande

Department of Computer Science and UMIACS
University of Maryland at College Park
Explosion of data, in pretty much every domain

- Sensing devices and sensor networks that can monitor everything from temperature to pollution to vital signs 24/7
- Increasingly sophisticated smart phones
- Internet, social networks making it very easy to publish data
- Scientific experiments and simulations
- Internet of Things
- Many aspects of life being turned into data ("dataification")

"Big Data" (= extracting knowledge and insights from data) becoming fundamental

- Science, business, politics -- largely driven by data and analytics
- Many others (Education, Social Good) are slowly being
Four V’s of Big Data

- **Big data not just about “Volume”**
  - Large scale of data certainly poses many problems
  - But most datasets are pretty small...

- **Variety** and heterogeneity in both data and applications
  - Text, networks, time series, nested/hierarchical, multimedia, ...
  - Increasingly complex and specialized analysis tasks

- **Velocity**
  - Data generated at very high rates and often needs to be processed in real time

- **Veracity**
  - What/who to trust? How to reason about data quality issues?
  - Easy to draw wrong statistical conclusions from large datasets
  - Issues becoming more important with increasing automation...
Building data management systems to address challenges in managing and analyzing big data by:

- Designing intuitive, formal, and **declarative abstractions** to empower users, and
- Developing **scalable platforms** and algorithms to support those abstractions over large volumes of data

Major research thrusts over the last 10 years

- Uncertain and probabilistic data management
- **Graph data management**
- Data management in the cloud
- **Collaborative data analytics**
- Query processing and optimization
Outline

- Graph Data Management
  - Declarative Graph Cleaning
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
  - Recreation/Storage Tradeoff in Version Management (or, why git/svn are not good at managing datasets)
Increasing interest in querying and reasoning about the underlying graph (network) structure in a variety of disciplines.
Different types of “queries”

*Subgraph pattern matching*: Given a “query” graph, find where it occurs in a given “data” graph

*Reachability; Shortest path; Keyword search; …*

*Historical or Temporal queries*: “Find most important nodes in a communication network in 2002?”
Wide Variety in Graph Queries/Analytics

Different types of “queries”
- Subgraph pattern matching;
- Reachability;
- Shortest path;
- Keyword search;
- Historical or Temporal queries…

Continuous “queries” and Real-time analytics
- Online prediction in response to new data
- Monitoring: “Tell me when a topic is suddenly trending in my friend circle”
- Anomaly/Event detection: “Alert me if the communication activity around a node changes drastically”

A protein-protein interaction network

Social networks
 Federal funds networks

Knowledge Graph
 World Wide Web

Citation networks
 Communication networks
 Disease transmission networks
 Financial transaction networks
 Stock Trading Networks
Wide Variety in Graph Queries/Analytics

Different types of “queries”
- Subgraph pattern matching;
- Reachability; Shortest path;
- Keyword search; Historical or Temporal queries…

Continuous “queries” and Real-time analytics
- Online prediction; Monitoring;
- Anomaly/Event detection

Batch analysis tasks/Network science
- Centrality analysis: Find the most central nodes in a network
- Community detection: Partition vertices into groups with dense interactions
- Network evolution: Build models for network formation and evolution
- Network measurements: Measure statistical properties
- Graph cleaning/inference: Remove noise in the observed network data

A protein-protein interaction network
Social networks
Citation networks
Communication networks
Disease transmission networks
Financial transaction networks

Working Paper Series No 986
Wide Variety in Graph Queries/Analytics

Different types of “queries”
- Subgraph pattern matching;
- Reachability; Shortest path;
- Keyword search; Historical or Temporal queries…

Continuous “queries” and Real-time analytics
- Online prediction; Monitoring;
- Anomaly/Event detection

Batch analysis tasks
- Centrality analysis; Community detection; Network evolution;
- Network measurements; Graph cleaning/inference

Machine learning tasks
- Many algorithms can be seen as message passing in specially constructed graphs

- A protein-protein interaction network
- Social networks
- Citation networks
- Communication networks
- Disease transmission networks
- Financial transaction networks
- Federal funds networks
- Knowledge Graph
- World Wide Web
- Stock Trading Networks
Graph Data Management: State of the Art

- Much prior and ongoing work – most of it outside, or on top of, general-purpose data management systems
  - Specialized indexes or algorithms for specific types of queries
  - Stand-alone prototypes for specific analysis tasks
- Emergence of specialized graph databases in recent years
  - Neo4j, Titan, OrientDB, DEX, AllegroGraph, ...
  - Rudimentary declarative interfaces/query languages
- Several “vertex-centric” frameworks in recent years
  - Pregel, Giraph, GraphLab, GRACE, GraphX, ...
  - Only work well for a very limited set of tasks
- Little work on continuous/real-time query processing, or on supporting evolutionary or temporal analytics
What we are doing

- **Goal: A graph data management system with unified declarative abstractions for graph queries and analytics**
- **Work so far**
  - Declarative graph cleaning [GDM’11, SIGMOD Demo’13]
  - NScale: a distributed programming framework [VLDB Demo’14]
  - Real-time continuous queries [SIGMOD’12, ESNAM’14, SIGMOD’14]
    - Techniques for continuous query processing over large dynamic graphs
    - Expressive query language for specifying anomaly detection queries
  - Historical graph data management [ICDE’13, SIGMOD Demo’13]
    - New indexing structure for retrieving historical snapshots
    - A high-level temporal/evolutionary analytics framework
  - Subgraph pattern matching and counting [ICDE’12, ICDE’14]
  - GraphGen: graph analytics over relational data [VLDB Demo’15]
Graph Data Management
  - Declarative Graph Cleaning
  - A Framework for Distributed Graph Analytics

DataHub: A platform for collaborative data science
  - Recreation/Storage Tradeoff in Version Management
Motivation

- The *observed, automatically-extracted information networks* are often noisy and incomplete

- Need to extract the underlying *true information network* through:
  - Attribute Prediction: *to predict values of missing attributes*
  - Link Prediction: *to infer missing links*
  - Entity Resolution: *to decide if two references refer to the same entity*
**Collective (relational) Inference**

**Attribute prediction:** Predict topic of the paper

- A Statistical Model for **Multilingual Entity Detection** and Tracking
- **Language** Model Based **Arabic** Word Segmentation
- Automatic **Rule** Refinement for **Information Extraction**
- **Why Not?**
- **Join Optimization** of Information Extraction Output: Quality Matters!
- An Annotation Management System for **Relational Databases**
- Tracing Lineage Beyond **Relational Operators**

**Link prediction**

- Divesh Srivastava
- Flip Korn
- Graham Cormode
- Lukasz Golab
- Avishek Saha
- Nick Koudas
- Theodore Johnson
- Vladislav Shkapenyuk

**Entity resolution**

- Petre Stoica
- Prabhu Babu
- Amol Deshpande
- Barna Saha
- Samir Khuller
- William Roberts
- Jian Li
- Jian Li
The *observed, automatically-extracted information networks* are often noisy and incomplete.

Need to extract the underlying *true information network* through:
- **Attribute Prediction**: to predict values of missing attributes
- **Link Prediction**: to infer missing links
- **Entity Resolution**: to decide if two references refer to the same entity

Typically iterative and interleaved application of the techniques
- Use results of one to improve the accuracy of other operations
- Significant benefits to using graph structure (“collective” inference)

Numerous techniques developed for the tasks in isolation
- No support from data management systems
- Hard to evaluate new techniques, especially for joint inference
Declara've Graph Cleaning

Enable declara've specifica'on of graph cleaning tasks

i.e., attribute predic'on, link predic'on, entry resolu'on.

Interac've system for execu'ng them over large datasets.
Enable declarative specification of graph cleaning tasks. 

- Attribute prediction, link prediction, entity resolution

Interactive system for executing them over large datasets.
Overview of the Approach

- **Declarative specification of the cleaning task**
  - Datalog-based language for specifying --
    - Prediction features (including local and relational features)
    - The details of how to accomplish the cleaning task
    - Arbitrary interleaving or pipelining of different tasks
Task Specification Framework

Specify the domain

Compute features

Make Predictions, and Compute Confidence in the Predictions

Choose Which Predictions to Apply
Task Specification Framework

For attribute prediction, the domain is a subset of the graph nodes.

For link prediction and entity resolution, the domain is a subset of pairs of nodes.

Local: word frequency, income, etc.

Relational: degree, clustering coeff., no. of neighbors with each attribute value, common neighbors between pairs of nodes, etc.
Task Specification Framework

Specify the domain

Compute features

Make Predictions, and Compute Confidence in the Predictions

Choose Which Predictions to Apply

Attribute prediction: the missing attribute

Link prediction: add link or not?

Entity resolution: merge two nodes or not?

After predictions are made, the graph changes:
- Attribute prediction changes local attributes.
- Link prediction changes the graph links.
- Entity resolution changes both local attributes and graph links.
Overview of the Approach

- Declarative specification of the cleaning task
  - Datalog-based language for specifying --
    - Prediction features (including local and relational features)
    - The details of how to accomplish the cleaning task
    - Arbitrary interleaving or pipelining of different tasks

- A mix of declarative constructs and user-defined functions to specify complex prediction functions

- Prototype implementation using Java BerkeleyDB
  - Datalog rules converted into SQL
  - Optimized the execution through caching, incremental evaluation, and pre-computed data structures
Real-world PubMed graph
  - Set of publications from the medical domain, their abstracts, and citations
50,634 publications, 115,323 citation edges
Task: Attribute prediction
  - Predict if the paper is categorized as Cognition, Learning, Perception or Thinking
Choose top 10% predictions after each iteration, for 10 iterations

DOMAIN Uncommitted(X):-Node(X,Committed='no')
{ ThinkingNeighbors(X,Count<Y>):- Edge(X,Y), Node(Y,Label='Thinking')
  PerceptionNeighbors(X,Count<Y>):- Edge(X,Y), Node(Y,Label='Perception')
  CognitionNeighbors(X,Count<Y>):- Edge(X,Y), Node(Y,Label='Cognition')
  LearningNeighbors(X,Count<Y>):- Edge(X,Y), Node(Y,Label='Learning')
  Features-AP(X,A,B,C,D,Abstract):- ThinkingNeighbors(X,A), PerceptionNeighbors(X,B),
                                 CognitionNeighbors(X,C), LearningNeighbors(X,D), Node(X,Abstract, __)
}
ITERATE(10)
{ UPDATE Node(X,_,P,'yes'):- Features-AP(X,A,B,C,D,Text), P = predict-AP(X,A,B,C,D,Text),
                        confidence-AP(X,A,B,C,D,Text) IN TOP 10%
Outline

- Graph Data Management
  - Declarative Graph Cleaning
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
  - Recreation/Storage Tradeoff in Version Management
Graph analytics/network science tasks too varied
- Centrality analysis; evolution models; community detection
- Link prediction; belief propagation; recommendations
- Motif counting; frequent subgraph mining; influence analysis
- Outlier detection; graph algorithms like matching, max-flow
- An active area of research in itself...

Scaling Graph Analysis Tasks

Counting network motifs

Identify Social circles in a user’s ego network
Scaling Graph Analysis Tasks

- Graph analytics/network science tasks too varied
  - Centrality analysis; evolution models; community detection
  - Link prediction; belief propagation; recommendations
  - Motif counting; frequent subgraph mining; influence analysis
  - Outlier detection; graph algorithms like matching, max-flow
  - An active area of research in itself...

- Hard to build general platforms like MapReduce
  - What is a good programming abstraction to provide?
    - Needs to cover a large fraction of use cases, and be easy to use
    - MR not a good fit for graph analytics
  - No clear winner yet, so little progress on systems
    - Especially on distributed or parallel systems
  - Application developers largely doing their own thing
“Vertex-centric” Frameworks

- Introduced by Google in a system called “Pregel”
  - Inspired by BSP (Bulk Synchronous Protocol)
- Adopted by many other systems
  - GraphLab, Apache Giraph, GraphX, Xstream, ...
  - Most of the research, especially in databases, focuses on it
- “Think like a vertex” paradigm
  - User provides a single `compute()` function that operates on a vertex
  - Executed in parallel on all vertices in an iterative fashion
  - Exchange information at the end of each iteration through message passing
Example: PageRank

Compute() at Node n:

\[ PR(n) = \text{sum up all the incoming weights} \]

Let the outDegree be \( D \)

Send \( PR(n)/D \) over each outgoing edge

PageRank values computed in iteration 10

Messages sent after iteration 10
Programming Frameworks

- Vertex-centric framework
  - Works well for some applications
    - Pagerank, Connected Components, ...
    - Some machine learning algorithms can be mapped to it
  - However, the framework is very restrictive
    - Most analysis tasks or algorithms cannot be written easily
    - Simple tasks like counting neighborhood properties infeasible
    - Fundamentally: Not easy to decompose analysis tasks into vertex-level, independent local computations

- Alternatives?
  - Galois, Ligra, GreenMarl: Not sufficiently high-level
  - Some others (e.g., Socialite) restrictive for different reasons
**Example: Local Clustering Coefficient**

\[ \text{LCC}(n) = \text{neighborhood density around } n \]

Compute() at Node n:

- Need to count the no. of edges between neighbors
- But does not have access to that information

**Option 1:** Each node transmits its list of neighbors to its neighbors
- Huge memory consumption

**Option 2:** Allow access to neighbors’ state
- Neighbors may not be local
- What about computations that require 2-hop information?
• An end-to-end distributed graph programming framework

• Users/application programs specify:
  • Neighborhoods or subgraphs of interest
  • A kernel computation to operate upon those subgraphs

• Framework:
  • Extracts the relevant subgraphs from underlying data and loads in memory
  • Execution engine: Executes user computation on materialized subgraphs
  • Communication: Shared state/message passing
NScale: LCC Computation Walkthrough

NScale programming model

Underlying graph data on HDFS

Subgraph extraction query:

**Compute** (LCC) on

**Extract** ({Node.color=orange} {k=1} {Node.color=white} {Edge.type=solid})

- Query-vertex predicate
- Neighborhood Size
- Neighborhood vertex predicate
- Neighborhood edge predicate
NScale: LCC Computation Walkthrough

NScale programming model

Specifying Computation: BluePrints API

```
ArrayList<RVertex> n_arr = new ArrayList<RVertex>();
for(Edge e: this.getQueryVertex().getOutEdges)
    n_arr.add(e.getVertex(Direction.IN));

int possibleLinks = n_arr.size() * (n_arr.size()-1)/2;

// compute #actual edges among the neighbors
for(int i=0; i < n_arr.size()-1; i++)
    for(int j=i+1; j < n_arr.size(); j++)
        if(edgeExists(n_arr.get(i), n_arr.get(j)))
            numEdges++;
```

Program cannot be executed as is in vertex-centric programming frameworks.
NScale: LCC Computation Walkthrough

GEP: Graph extraction and packing

Underlying graph data on HDFS

MapReduce
- Subgraph Extraction
- Cost based optimizer
- Set Bin Packing

Node to Bin mapping

MR2: Map Tasks
- MR2: Reducer 1
  - Exec Engine
- MR2: Reducer N
  - Exec Engine
NScale: LCC Computation Walkthrough

GEP: Graph extraction and packing

Underlying graph data on HDFS

Extracted Subgraphs

MapReduce (Apache Yarn)
Subgraph extraction
Goal:
• Group graphs with high similarity
• Minimizes memory consumption

Techniques explored
• Set bin packing, graph partitioning, clustering

Shingle based set bin packing
• Min-hash signatures based sorting
• Grouping based on Jaccard similarity

Bin Packing
• Set union operation
• Bin Capacity: Elastic resource allocation
• Max # Subgraphs: Handles Skew
NScale: LCC Computation Walkthrough

GEP: Graph extraction and packing

Underlying graph data on HDFS

Graph Extraction and Loading

Sample bin packing using Shingles

MapReduce (Apache Yarn)

Subgraph extraction

Cost Based Optimizer

Data Rep & Placement

Bin 1: SG-1, SG-4

Bin 2: SG-2, SG-3
NScale: LCC Computation Walkthrough

GEP: Graph extraction and packing

Underlying graph data on HDFS

Graph Extraction and Loading

Subgraphs in Distributed Memory

- Underlying graph data on HDFS
- Graph Extraction and Loading
- Subgraphs in Distributed Memory

MapReduce (Apache Yarn)

Subgraph extraction

Cost Based Optimizer

Data Rep & Placement

Underlying graph data on HDFS

Graph Extraction and Loading

Subgraphs in Distributed Memory
NScale: LCC Computation Walkthrough

Underlying graph data on HDFS

Graph Extraction and Loading

MapReduce (Apache Yarn)
Subgraph extraction
Cost Based Optimizer
Data Rep & Placement

Subgraphs in Distributed Memory

Distributed Execution Engine

Node Master

Distributed execution of user computation
NScale: Summary

• Users write programs at the abstraction of a graph
  • More intuitive for graph analytics
  • Captures mechanics of common graph analysis/cleaning tasks

• Generalization: Flexibility in subgraph definition
  • Subgraph = vertex and associated edges: vertex-centric programs
  • Subgraph = an entire graph: global programs

• Scalability
  • Only relevant portions of the graph data loaded into memory
    • User can specify subgraphs of interest, and select nodes or edges based on properties
  • Carefully partition (pack) nodes across machines so that:
    • Every subgraph is entirely in memory on a machine, while using very few machines
Experimental Evaluation

• **Datasets**
  - Web graphs
  - Communication/interaction graphs
  - Social networks

• **Graph applications**
  - Local Clustering Coefficient
  - Motif counting
  - Identifying weak ties
  - Triangle Counting
  - Personalized Page Rank

• **Baselines**
  - Apache Giraph
  - GraphLab
  - GraphX

• **Evaluation Metrics**
  - Computational Effort
  - Execution Time
  - Cluster Memory

• **Cluster Setup**
  - 16 Node Cluster
  - Apache YARN (MRv2)
  - Each Node:
    - 2 x 4-core Intel Xeon
    - 24GB RAM, 3 x 2 TB disks
## Experimental Evaluation

### Local Clustering Coefficient

<table>
<thead>
<tr>
<th>Dataset</th>
<th>NScale</th>
<th>Giraph</th>
<th>GraphLab</th>
<th>GraphX</th>
</tr>
</thead>
<tbody>
<tr>
<td>EU Email</td>
<td>377</td>
<td>9.00</td>
<td>1150</td>
<td>26.17</td>
</tr>
<tr>
<td>Google Web</td>
<td>658</td>
<td>25.82</td>
<td>2024</td>
<td>35.35</td>
</tr>
<tr>
<td>WikiTalk</td>
<td>726</td>
<td>24.16</td>
<td>DNC</td>
<td>OOM</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>1800</td>
<td>50.00</td>
<td>DNC</td>
<td>OOM</td>
</tr>
<tr>
<td>Orkut</td>
<td>2000</td>
<td>62.00</td>
<td>DNC</td>
<td>OOM</td>
</tr>
</tbody>
</table>

### Personalized Page Rank on 2-Hop Neighborhood

<table>
<thead>
<tr>
<th>Dataset</th>
<th>NScale</th>
<th>Giraph</th>
<th>GraphLab</th>
<th>GraphX</th>
</tr>
</thead>
<tbody>
<tr>
<td>EU Email</td>
<td>3200</td>
<td>52</td>
<td>3.35</td>
<td>782</td>
</tr>
<tr>
<td>NotreDame</td>
<td>3500</td>
<td>119</td>
<td>9.56</td>
<td>1058</td>
</tr>
<tr>
<td>Google Web</td>
<td>4150</td>
<td>464</td>
<td>21.52</td>
<td>10482</td>
</tr>
<tr>
<td>WikiTalk</td>
<td>12000</td>
<td>3343</td>
<td>79.43</td>
<td>DNC</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>20000</td>
<td>4286</td>
<td>84.94</td>
<td>DNC</td>
</tr>
<tr>
<td>Orkut</td>
<td>20000</td>
<td>4691</td>
<td>93.07</td>
<td>DNC</td>
</tr>
</tbody>
</table>
Graph Data Management
- Declarative Graph Cleaning
- A Framework for Distributed Graph Analytics

DataHub: A platform for collaborative data science
- Recreation/Storage Tradeoff in Version Management
Collaborative Data Science

- Widespread use of “data science” in many many domains

A typical data analysis workflow

1. CSV from data.gov
2. EDIT: Append Column
3. NEW: Add file
4. EDIT: Correct “addresses”
5. EDIT: Project columns
6. EDIT: Partition rows

1000s of versions
Collaborative Data Science

- Widespread use of "data science" in many domains
- Increasingly, the "pain point" is managing the process, especially during collaborative analysis
  - Many private copies of datasets
  - Massive redundancy
  - No easy way to keep track of dependencies between datasets
  - Manual intervention needed for resolving conflicts
  - No efficient organization or management of datasets
  - No way to analyze/compare/query versions of a dataset
- Ad hoc data management systems (e.g., Dropbox) used
  - Much of the data is unstructured, so typically can’t use DBs
  - The process of data science itself is quite ad hoc and exploratory
  - Scientists/researchers/analysts rely much on their own
DataHub: A Collaborative Data Science Platform

The one-stop solution for collaborative data science and dataset version management

http://data-hub.org
The one-stop solution for collaborative data science and dataset version management

- a dataset management system – import, search, query, analyze a large number of (public) datasets
- a dataset version control system – branch, update, merge, transform large structured or unstructured datasets
- an app ecosystem and hooks for external applications (Matlab, R, iPython Notebook, etc)
Can we use Version Control Systems (e.g., Git)?

No, because they typically use fairly simple algorithms and are optimized to work for code-like data.

LF Dataset (Real World)
#Versions = 100
Avg. version size = 423 MB

- gzip = 10.2 GB
- svn = 8.5 GB
- git = 202 MB
- *this = 159 MB
Can we use Version Control Systems (e.g., Git)?

- **NO!** Because they typically use **fairly simple algorithms** and are optimized to work for code-like data.
- **NO!** Git ends up using **large amounts of RAM** for large files.

*Use extensions*

**DON’T!**

GitHub Help:

**Working with large files**

A Git repository contains every version of every file. But for revisions of large files increase the clone and fetch times. A $1$GB Git requires $10$GB or more free space (for the .shadow file). With $1$GB, Git requires $10$GB or more free space for the .shadow file.

We suggest removing the following types of files:

- Code files
- Versioned assets, such as graphics
- Large configuration files

**Tip:** If you regularly push large files to GitHub, consider introducing a staging area.

Stack overflow:

**Why can't Git handle large files and large repos?**

Dozens of questions and answers on SO and elsewhere emphasize that Git can't handle large files or large repos. A handful of workarounds are suggested such as `git-fat` and `git-annex`, but ideally Git would handle large files/repos natively.

If this limitation has been around for years, is there reason the limitation has not yet been removed? I assume that there’s some technical or design challenge baked into Git that makes large file and large repo support extremely difficult.

Lots of related questions, but none seem to explain why this is such a big hurdle:

- `git with large files`
- What are the file limits?
- `Git - repository and file size`
- `Versioning large text files`
- `How to handle a large git repository`?
- `Managing large binary files with git`
- What is the practical maximum size of a Git repository full of text-based data? [Quora]
Can we use Version Control Systems (e.g., Git)?

- No, because they typically use fairly simple algorithms and are optimized to work for code-like data.
- Git ends up using large amounts of RAM for large files.
- Querying and retrieval functionalities are primitive, and revolve around single version and metadata retrieval.
- No way to specify queries like:
  - identify all datasets derived of dataset A that satisfy property P
  - identify all predecessor versions of version A that differ from it by a large number of records
  - rank a set of versions according to a scoring function
  - find the version where the result of an aggregate query is above a threshold
  - find parent records of all records in version A that satisfy certain property
Can we use Version Control Systems (e.g., Git)?

No, because they typically use **fairly simple algorithms** and are optimized to work for code-like data

Git ends up using **large amounts of RAM** for large files

**VQuel: A Unified Query Language for querying versioning and derivation information [USENIX TAPP’15]**

**Example:** What commits did Alice make after January 01, 2015?

```vquel
range of V is Version
retrieve V.all
where V.author.name = "Alice" and
    V.creation_ts >= "01/01/2015"
```

- find the version where the result of an aggregate query is above a threshold
- find parent records of all records in version A that satisfy certain property
Outline

- Graph Data Management
  - Declarative Graph Cleaning
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
  - Recreation/Storage Tradeoff in Version Management
**Storage cost** is the space required to store a set of versions:

\[
\begin{align*}
100 \text{ MB} & \quad 101 \text{ MB} & \quad 102 \text{ MB} \\
\Rightarrow & \quad (100 + 101 + 102) = 303 \text{ MB}
\end{align*}
\]

**Recreation cost** is the time* required to access a version:

\[
\begin{align*}
(100 + 101 + 102) & = 303 \text{ MB}
\end{align*}
\]

A **delta** between versions is a file which allows constructing one version given the other:

- **Directed delta**: [Diagram](#)
  - Example: Unix diff, xdelta, XOR, etc.
- **Undirected delta**: [Diagram](#)

A delta has its own **storage cost** and **recreation cost**, which, in general, are **independent** of each other.
Storage-Recreation Tradeoff

**Scenario 1**

- 100 MB
- 30 MB → 10 MB → Storage cost = (100+30+10) = 140 MB

```
Total Access Cost = 370 MB
```

**Scenario 2**

- 100 MB
- 30 MB → 11 MB → Storage cost = (100+30+11) = 141 MB

```
Total Access Cost = 341 MB
```

**Scenario 3**

- 110 MB
- 5 MB → 10 MB → Storage cost = (110+5+10) = 125 MB

```
Total Access Cost = 345 MB
```
Storage-Recreation Tradeoff

**Given**
1) a set of versions
2) partial information about deltas between versions

**Find a Storage Solution that:**
- minimizes total recreation cost given a storage budget, or
- minimizes max recreation cost given a storage budget

<table>
<thead>
<tr>
<th>Storage Cost</th>
<th>Recreation Cost</th>
<th>Undirected Case, $\Delta = \Phi$</th>
<th>Directed Case, $\Delta = \Phi$</th>
<th>Directed Case, $\Delta \neq \Phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>min $C$</td>
<td>$R_i &lt; \infty$, $\forall i$</td>
<td>PTime, Minimum Cost Arborescence (MCA)</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>$C &lt; \infty$</td>
<td>min ${\max {R_i</td>
<td>1 \leq i \leq n}}$</td>
<td>PTime, Shortest Path Tree (SPT)</td>
</tr>
<tr>
<td>P3</td>
<td>$C \leq \beta$</td>
<td>min ${\sum_{i=1}^{n} R_i}$</td>
<td>NP-hard, LAST* Alg</td>
<td>NP-hard, LMG Algorithm</td>
</tr>
<tr>
<td>P4</td>
<td>$C \leq \beta$</td>
<td>min ${\max {R_i</td>
<td>1 \leq i \leq n}}$</td>
<td>NP-hard, MP Algorithm</td>
</tr>
<tr>
<td>P5</td>
<td>min $C$</td>
<td>$\sum_{i=1}^{n} R_i \leq \theta$</td>
<td>NP-hard, LAST* Alg</td>
<td>NP-hard, LMG Algorithm</td>
</tr>
<tr>
<td>P6</td>
<td>min $C$</td>
<td>$\max {R_i</td>
<td>1 \leq i \leq n} \leq \theta$</td>
<td>NP-hard, MP Algorithm</td>
</tr>
</tbody>
</table>
Baselines

“Null” Version

**Minimize Storage Cost**

Recreation Cost: No constraint

**Minimize Recreation Cost**

Storage Cost: No constraint

**Minimum Cost Arborescence (MCA)**

Edmonds’ algorithm

Time complexity = $O(E + V \log V)$

**Shortest Path Tree (SPT)**

Dijkstra’s algorithm

Time complexity = $O(E \log V)$
Comparing Different Solutions

**MCA Storage Cost**

- **Dataset: DC**
- **Type = CSV files**
- **#Versions = 100010**
- **#Deltas = 18086876**
- **Average version size = 347.65 MB**
- **MCA Recreation Cost = 11.5 PB**
- **SPT Storage Cost = 34 TB**

**SPT Recreation Cost**

*Storage budget of 1.1X the MCA reduces total recreation cost by 1000X*