Scalable Platforms for Graph Analytics and Collaborative Data Science

Amol Deshpande
Associate Professor
Department of Computer Science and UMIACS
University of Maryland at College Park

These slides at: http://ter.ps/a37

Joint work with many students and collaborators
Big Data

- Explosion of data, in pretty much every domain
  - Sensing devices and sensor networks (IoT) 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
  - 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 (10GB-500GB)...

- **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...
Focus of My Research Group at UMD

- 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
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
A graph captures a set of entities/objects, and interconnections between pairs of them.

Graphs also often called networks.

Entities/objects represented by vertices or nodes.

Interconnections between pairs of vertices called edges.

Also called links, arcs, relationships.

An undirected, unweighted graph

A directed, edge-weighted graph
Background: Graphs

- **Graph** captures a set of entities/objects, and interconnections between pairs of them

  - *Graphs* also often called *networks*

  - Entities/objects represented by *vertices or nodes*

  - Interconnections between pairs of vertices called *edges*
    - Also called *links, arcs, relationships*

- Graph theory, graph algorithms very well studied in Computer Science

  - Not as much work on managing large volumes of graph-structured data, or doing analytics over them
Increasing interest in querying and reasoning about the underlying graph (network) structure in a variety of disciplines.
Motivation

- Underlying data hasn’t necessarily changed that much
  - Aside from the data volumes and easier availability

- However, several new realizations:
  - Reasoning about graph structure provides useful and actionable insights *(network science/complex network analysis)*
  - Lose too much information/intuitions if graph structure ignored
  - Not easy to write many natural queries or tasks using traditional tools
    - Especially relational databases like Oracle
  - Hard to efficiently process inherently graph-structured queries or complex network analysis tasks using existing tools
    - A major concern with increasingly large graphs seen in practice
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?”
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

Knowledge Graph

World Wide Web

Federal funds networks

Financial transaction networks

Stock Trading Networks

Disease transmission networks

Citation networks

Communication networks

Networks in different domains:

- Social networks
- Knowledge Graph
- World Wide Web
- Federal funds networks
- Financial transaction networks
- Stock Trading Networks
- Disease transmission networks
- Citation networks
- Communication networks
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: 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
Examples of Graph Analysis Tasks

- **Community Detection:** partitioning the vertices into (potentially overlapping) groups based on the interconnections between them
  - Provide insights into how networks function; identify functional modules; improve performance of Web services...

- **Analyzing “ego-networks”**
  - Properties of neighborhoods around a large number of nodes

- **Building models of evolution**
  - Measuring properties of networks
  - Constructing evolution models that can explain those

\[V_1 \rightarrow V_2 \rightarrow V_3 \rightarrow V_4\]

**Identify Social circles**

- **Community Detection**
  - High school friends
  - Office colleagues
  - Work place friends
  - Family members

**Counting network motifs**

- Feed-fwd Loop
- Feed-back Loop
- Bi-parallel Motif
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
- Knowledge Graph
- World Wide Web
- Stock Trading Networks
- Federal funds networks
- Network evolution
- Community detection
- Centrality analysis
- Graph cleaning/inference
- Specially constructed graphs
- Message passing
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 analysis framework** [VLDB Demo’14, VLDBJ’15]
  
  - **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, arXiv’15]
    
    - A distributed indexing structure for retrieving historical snapshots
    
    - Temporal/evolutionary analytics framework, built on top of Apache Spark
  
  - **Subgraph pattern matching and counting** [ICDE’12, ICDE’14]
  
  - **GraphGen: graph analytics over relational data** [VLDB Demo’15]
Outline

- Graph Data Management
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
Scaling Graph Analysis Tasks

- Graph analytics/network science tasks too varied

- Hard to build general platforms like Hadoop/Dryad/Spark
  - What is a good programming abstraction to provide?
    - Needs to cover a large fraction of use cases, and be easy to use
    - MapReduce works very well for other analysis tasks, but 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
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

**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?*
NScale Programming Framework

• 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

```java
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.
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

Graph Extraction and Loading

MapReduce (Apache Yarn)

Subgraph extraction

Extracted Subgraphs

1
2
3
4
5
6
7
8
9
10
11
12

SG-1
SG-2
SG-3
SG-4
NScale: LCC Computation Walkthrough

**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

MapReduce (Apache Yarn)

Subgraph extraction

Cost Based Optimizer

Data Rep & Placement

Sample bin packing using Shingles

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

MapReduce (Apache Yarn)

Subgraph extraction

Cost Based Optimizer

Data Rep & Placement

[Diagram showing a complex graph with nodes and edges, illustrating the process flow.]
NScale: LCC Computation Walkthrough

Distributed execution of user computation

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
Node Master
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>
Outline

- Graph Data Management
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
Widespread use of “data science” in many many domains

A typical data analysis workflow

EDIT: Append Column
NEW: Add file

EDIT: Correct “addresses”

EDIT: Project columns

EDIT: Partition rows

CSV from data.gov

1000s of versions
Collaborative Data Science

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

- Increasingly the “pain point” is managing the process, especially during collaborative analysis
  - Many private copies of the 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 are pretty 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
Work being done in collaboration with Sam Madden (MIT) and Aditya Parameswaran (UIUC)
DataHub: A Collaborative Data Science Platform

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

*DON’T!*

We suggest removing the following types of files:

- Database dumps
- Log files

**Use extensions***

[GitHub Help](https://help.github.com/articles/working-with-large-files/)

[Stack Overflow Question](https://stackoverflow.com/questions/29393447/why-cant-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 in Git?*
  - *Repository and file limits in Git - Large text files in Git*
  - *Large configuration 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
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 changes did Alice make after January 01, 2015?

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
  - A Framework for Distributed Graph Analytics
- DataHub: A platform for collaborative data science
  - Recreation/Storage Tradeoff in Version Management [VLDB’15]
**Storage cost** is the space required to store a set of versions

\[
100 \text{ MB} + 101 \text{ MB} + 102 \text{ MB} = 303 \text{ MB}
\]

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

\[
(100 + 101 + 102) = 303 \text{ MB}
\]

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

Example: Unix diff, xdelta, XOR, etc.

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

**Scenario 1**
- Storage cost: \(100 + 30 + 10 = 140\) MB
- Total Access Cost: \(370\) MB

**Scenario 2**
- Storage cost: \(100 + 30 + 11 = 141\) MB
- Total Access Cost: \(341\) MB

**Scenario 3**
- 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></th>
<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>
<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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
</tbody>
</table>
Baselines

“Null” Version

Minimize Storage Cost
Recreation Cost: No constraint

Minimum Cost Arborescence (MCA)
Edmonds’ algorithm
Time complexity = $O(E + V \log V)$

Minimize Recreation Cost
Storage Cost: No constraint

Shortest Path Tree (SPT)
Dijkstra’s algorithm
Time complexity = $O(E \log V)$
Comparing Different Solutions

**MCA Storage Cost**

**SPT Recreation Cost**

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

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

Extensions

- Include user defined functions – e.g., custom “diff” functions for two versions
- Additional graph traversal operators

Engagement with users to refine the constructs

Implementation Challenges

*Data is stored in a compressed fashion, to exploit overlaps between versions*

Need new query execution and optimization strategies

*Version graph can become very large in a “dynamic update” environment*

Need scalable methods to handle the version graph
Thanks !!

More at: http://www.cs.umd.edu/~amol

Questions ?