Knowledge Discovery Toolkit Status Report

John R. Gilbert
University of California, Santa Barbara

KDT Spring Mind Meld
March 5, 2012

Support: Intel, Microsoft, DOE Office of Science, NSF
Knowledge Discovery Toolbox
http://kdt.sourceforge.net/

A general graph library with operations based on linear algebraic primitives
Knowledge Discovery Toolbox
http://kdt.sourceforge.net/

- Aimed at domain experts who know their problem well but don’t know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors

A general graph library with operations based on linear algebraic primitives
Knowledge Discovery Toolbox
http://kdt.sourceforge.net/

- Aimed at domain experts who know their problem well but don’t know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors
- A collaboration among UCSB, UCB, and Lawrence Berkeley Lab
- Open source software, released under New BSD license
- v0.1 released March 2011; v0.2 expected March 2012

A general graph library with operations based on linear algebraic primitives
**Knowledge Discovery Workflow**

1. Cull relevant data
2. Build input graph
3. Analyze input graph
4. Visualize result graph

- Gene data
- Email
- Twitter
- Facebook
- Video
- Sensor
- Web
- ...
Knowledge Discovery Workflow

1. Cull relevant data
2. Build input graph
3. Analyze input graph
4. Visualize result graph

- Gene data
- Email
- Twitter
- Facebook
- Video
- Sensor
- Web
- ...

Disk-based technologies
KDT
Viz tools
Domain Expert vs. Graph Expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics \textit{(e.g., degree())}
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

- Hypergraphs and sparse matrices
- Graph primitives \textit{(e.g., bfsTree())}
- SpMV / SpGEMM on semirings
Domain Expert vs. Graph Expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (e.g., degree)
  - vectors
- Clustering / components
- Centrality / authority: betweenness, PageRank

```python
# bigG contains the input graph
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)

clus = G.cluster('Markov')

clusNedge = G.nedge(clus)

smallG = G.contract(clus)

# visualize
```

- Hypergraphs and sparse matrices
- Graph primitives (e.g., bfsTree())
- SpMV / SpGEMM on semirings
Domain Expert vs. Graph Expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (e.g., degree)
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank
- Hypergraphs and sparse matrices
- Graph primitives (e.g., bfsTree())
- SpMV / SpGEMM on semirings
Domain Expert vs. Graph Expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (e.g., degree)
  - vectors
- Clustering / components
- Centrality / authority: between centrality, PageRank

- Hypergraphs and sparse matrices
- Graph primitives (e.g., bfsTree)
- SpMV / SpGEMM on semirings

```python
# bigG contains the input graph comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)
clus = G.cluster('Markov')
clusNedge = G.nedge(clus)
smallG = G.contract(clus)

# visualize
L = G.toSpParMat()
d = L.sum(kdt.SpParMat.Column)
L = -L
L.setDiag(d)
M = kdt.SpParMat.eye(G.nvert()) - mu*L
pos = kdt.ParVec.rand(G.nvert())
for i in range(nsteps):
    pos = M.SpMV(pos)
```
A few KDT applications

Markov Clustering (MCL) finds clusters by postulating that a random walk that visits a dense cluster will probably visit many of its vertices before leaving.

We use a Markov chain for the random walk. This process is reinforced by adding an inflation step that uses the Hadamard product and rescaling.

PageRank says a vertex is important if other important vertices link to it.

Each vertex (webpage) votes by splitting its PageRank score evenly among its out edges (links). This broadcast (an SpMV) is followed by a normalization step (ColWise). Repeat until convergence.

PageRank is the stationary distribution of a Markov Chain that simulates a "random surfer".

Betweenness Centrality says that a vertex is important if it appears on many shortest paths between other vertices. An exact computation requires a BFS for every vertex. A good approximation can be achieved by sampling starting vertices.

Gaussian belief propagation (GaBP) is an iterative algorithm for solving the linear system of equations $Ax = b$, where $A$ is symmetric positive definite. GaBP assumes each variable follows a normal distribution. It iteratively calculates the precision $P$ and mean value $\mu$ of each variable; the converged mean-value vector approximates the actual solution.
Graph API (v0.2)

Real applications

Community Detection

Network Vulnerability Analysis

Applets

centrality('exactBC')
centrality('approxBC')
pageRank

cluster('Markov'), cluster('kmeans'), ...

Graph500

Building blocks

DiGraph
bfsTree, isBfsTree
plus utility (e.g., DiGraph,nvert, toParVec,degree,load,UFget,+,*
sum,subgraph,reverseEdges)

HyGraph
bfsTree, isBfsTree
plus utility (e.g., HyGraph,nvert, toParVec,degree,load,UFget)

(Sp)ParVec
(e.g., +,* ,|,&,>,==,[],
abs,max,sum,range,
norm,hist,randPerm,
scale,topK)

SpParMat
(e.g., +,* ,SpMM, SpMV, SpRef,
SpAsgn)

CombBLAS

SpMV, SpMM, etc.
Combinatorial BLAS: A matrix-based graph library

- Also sparse & dense vectors, distributed and local
- Matrix operations over user-defined (and some built-in) semirings
- Highly templated C++
- Reference implementation in MPI

Architecture of matrix classes
Sparse array-based primitives

Sparse matrix-matrix multiplication (SpGEMM)

Element-wise operations

Sparse matrix-dense vector multiplication

Sparse matrix indexing

Matrices on various semirings: $(x, +)$, $(\text{and, or})$, $(+, \text{min})$, ...
Indexing sparse arrays in parallel (coarsen graphs, extract subgraphs, etc.)

**SpRef:** \( B = A(I, J) \)

**SpAsgn:** \( B(I, J) = A \)

**SpExpAdd:** \( B(I, J) += A \)

\( A, B \): sparse matrices  
\( I, J \): vectors of indices

\( \text{SpRef} \) using mixed-mode sparse matrix-matrix multiplication (\text{SpGEMM}). Ex: \( B = A([2,4], [1,2,3]) \)
Strong scaling of \textbf{SpRef}

random symmetric permutation $\Leftrightarrow$ relabeling graph vertices

- RMAT Scale 22; edge factor=8; $a=.6$, $b=c=d=.4/3$
- Franklin/NERSC, each node is a quad-core AMD Budapest
KDT v0.2: Attributed Semantic Graphs and Filters

Example:

- Vertex types: Person, Phone, Camera
- Edge types: PhoneCall, TextMessage, CoLocation
- Edge attributes: StartTime, EndTime
- Calculate centrality just for PhoneCalls and TextMessages between times sTime and eTime

```python
def vfilter(self, vTypes):
    return self.type in vTypes

def efilter(self, eTypes, sTime, eTime):
    return ((self.type in eTypes) and
            (self.sTime > sTime) and
            (self.eTime < eTime))

wantedVTypes = (People)
wantedETypes = (PhoneCall, TextMessage)
start = dt.now() - dt.timedelta(hours=1)
end = dt.now()
bc = G.centrality('approxBC', filter=
    (vfilter, wantedVTypes),
    (efilter, wantedETypes,
    start, end))
```
Implementing filters: Options

- Prefilter to extract the relevant subgraph
  - Simple, but too much time / memory for many use cases

- Write filters in Python, call back from CombBLAS
  - Simple & flexible, but hurts performance

- Write filters as semiring ops in C++, wrap in Python
  - Good performance, but hard to write new filters

- Work in progress: Write filters in Python subset, compile with SEJITS (selective embedded just-in-time specialization)
KDT Team (2011-12)

- David Alber, Microsoft
- Victor Amelkin, UCSB
- Aydin Buluc, LBNL
- Varad Deshmukh, UCSB
- Kevin Deweese, UCSB
- John Gilbert, UCSB
- Shoaib Kamil, UC Berkeley
- Chris Lock, UCSB
- Adam Lugowski, UCSB
- Steve Reinhardt, Cray
- Lijie Ren, UCSB
- Veronika Strnadova, UCSB
- Yun Teng, UCSB
- Drew Waranis, UCSB
- Support: Intel, Microsoft, DOE Office of Science, NSF