SANDIA REPORT
SAND2009-6772
Unlimited Release
Printed October 2009

Graph Algorithms in the Titan Toolkit

Brian Wylie and William McLendon

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2009-6772
Unlimited Release
Printed October 2009

Graph Algorithms in the Titan Toolkit

Brian Wylie
Data Analysis and Visualization
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1323
bnwylie@sandia.gov

William McLendon
Scalable System Software
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS1323
wcmclen@sandia.gov

Abstract

Graph algorithms are a key component in a wide variety of intelligence analysis activities. The Graph-Based Informatics for Non-Proliferation and Counter-Terrorism project addresses the critical need of making these graph algorithms accessible to Sandia analysts in a manner that is both intuitive and effective. Specifically we describe the design and implementation of an open source toolkit for doing graph analysis, informatics, and visualization that provides Sandia with novel analysis capability for non-proliferation and counter-terrorism.

ACKNOWLEDGMENTS

The authors would like to thank the entire Titan team for valuable technical discussions on design, integration and implementation. We would also like to thank Bruce Hendrickson and Jon Berry for their expert guidance and advice on graph algorithms.

Contents

FIGURES

Nomenclature

ARG	Attributed Relational Graph
BGL	Boost Graph Library
PBGL	Parallel Boost Graph Library
MTGL	Multi-Threaded Graph Library
ST	Source and Target (e.g. find a graph path from Source to Target)
CSG	Connection SubGraph
DOE	Department of Energy
SNL	Sandia National Laboratories
SMU	Strategic Management Unit
WFO	Work for Others

1. INTRODUCTION

The Graph-Based Informatics for Non-Proliferation and Counter-Terrorism project started as a project to combine graph algorithms and visualization into a functional toolkit. As the project progressed it became the genesis for a much larger and broader effort that is now called the ***Titan Informatics Toolkit***. The authors believe the work conducted in this project will have a significant long term impact to both the Sandia analysis capabilities and to the global informatics community.

1.1. Titan Informatics Toolkit

The Titan Informatics Toolkit is an expansion of the Visualization ToolKit (VTK) to support the ingestion, processing, and display of informatics data. By leveraging the VTK engine, Titan provides a flexible, component based, pipeline architecture for the integration and deployment of algorithms in the fields of intelligence, semantic graph and information analysis.

The Titan project represents one of the first software development efforts to address the merging of scientific visualization and information visualization on a substantive level. The VTK parallel client-server layer will provide an excellent framework for doing scalable analysis on distributed memory platforms. In the same way that scientific visualization applications can be built with VTK, you can now build information visualization and analysis applications with Titan. As shown in Figure 1, applications can be built by combining the components as appropriate for your specific domain needs.

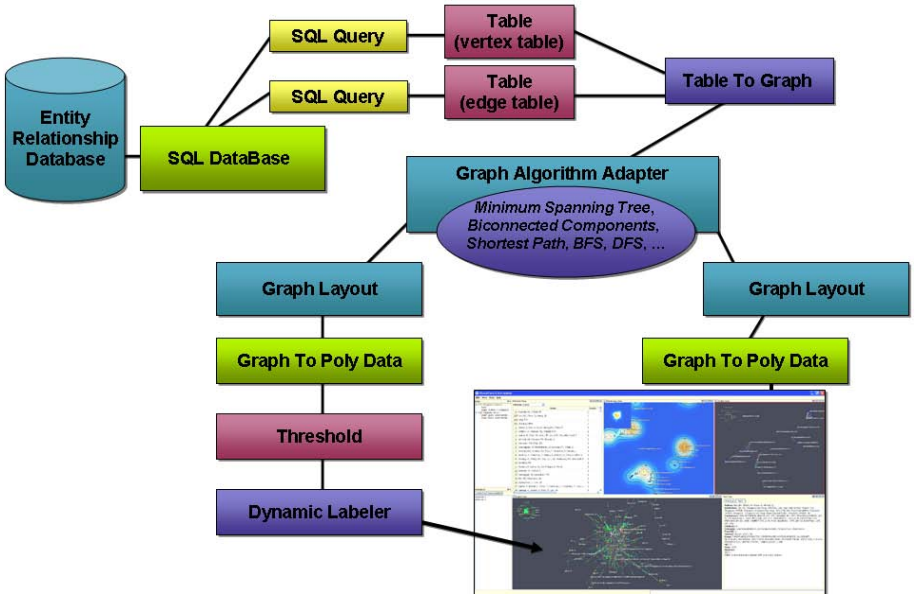

Figure 1: The Titan Informatics Toolkit component architecture enables the construction of targeted domain specific applications.

2. GRAPH LIBRARIES IN TITAN

There are a number of popular graph libraries that are available as either open source or internally within Sandia National Laboratories. The authors of these libraries have spent considerable time and effort developing efficient algorithms to solve a range of common graph algorithms. Titan leverages existing work through the use of graph-adapters which allow other libraries to treat the VTK graph data objects as though they are native. We should note that the fact that the VTK graph data structure within Titan can be adapted to multiple independent graph libraries illustrates the generality of the data structure itself.

Titan has focused on the integration of three external libraries for the core of its graph algorithms. The following sections will provide a brief overview.

2.1 Boost Graph Library (BGL)

Graph algorithms are an essential part of many informatics applications and a unified toolkit needs to provide that functionality. The Boost C++ library [1] is a large collection of templated C++ classes dedicated to generic programming, available online at http://www.boost.org . A popular algorithm library within this collection is the Boost Graph Library (BGL) [2]. This library implements many common graph "kernel" operations such as breadth-first search (BFS) and depth-first search (DFS), which can be used as the basis for many useful graph algorithms.
A few of the algorithms available within Titan include:

- *Bi-connected Components*
- *Breadth First Search*
- *Connected Components*
- *Minimum Spanning Tree*
- *Network Centrality*
- *Strongly Connected Components*

The Titan team decided to use these BGL algorithms instead of reimplementing them from scratch. To use a vtkGraph with a BGL algorithm, callers simply include vtkBoostGraphAdapter.h. This "data-less" adapter implements the required BGL concepts for vtkDirectedGraph and vtkUndirected-Graph, thus allowing BGL algorithms to process Titan graphs directly. The interface between Titan and BGL follows the VTK pipeline model. Referencing the "hello world" code example in Figure 2 we see that the header files for the adapter and for the BGL algorithm are included. We create a BGL algorithm, put the algorithm in the pipeline, and then color the nodes of the graph by the results of the algorithm. From the developers perspective the BGL functionality is a pipeline component. In practice all the BGL algorithms can be run in similar fashion. We often string up combinations of various BGL algorithms in pipelines. In fact, one of the layout algorithms in the Titan toolkit (G-Space [3]) runs three BGL algorithms to compute geodesic distances.

```
#include "vtkBoostBreadthFirstSearch.h"
#include "vtkGraphLayoutView.h"
#include "vtkRandomGraphSource.h"
#include "vtkRenderWindowInteractor.h"

int main(int argc, char* argv[])
{
  // Create a random graph
  vtkRandomGraphSource* source = vtkRandomGraphSource::New();

  // Create BGL algorithm and put it in the pipeline
  vtkBoostBreadthFirstSearch* bfs = vtkBoostBreadthFirstSearch::New();
  bfs->SetInputConnection(source->GetOutputPort());

  // Create a view and add the BFS output
  vtkGraphLayoutView* view = vtkGraphLayoutView::New();
  view->AddRepresentationFromInputConnection(bfs->GetOutputPort());

  // Color vertices and label based on BFS search
  view->SetVertexColorArrayName("BFS");
  view->SetVertexLabelArrayName("BFS");

  // Start view interaction
  view->GetInteractor()->Start();
}
```

Figure 2: Example usage of the Titan Informatics Toolkit; leveraging powerful graph algorithm libraries is easy and intuitive.

2.2 Parallel Boost Graph Library (PBGL)

As a distributed memory toolkit, VTK currently provides a myriad of functionality around parallel scientific processing and visualization. The Parallel Boost Graph Library (PBGL) is a generic C++ library for high-performance parallel and distributed graph algorithms. The vtkGraph data structure, along with some distributed helper classes, enables the PBGL functionality to work in the same way as the BGL classes. PBGL is now part of the Boost Library proper as of release 1.40.0 (August 2009).

2.3 Multi-Threaded Graph Library (MTGL)

The Multi-Threaded Graph Library (MTGL) developed at Sandia National Laboratories (Jonathan Berry – Org 1416) targets shared memory platforms such as the massively multi-threaded Cray MTA/XMT [8], and can be used with the Sandia Qthreads library [9] on chip multiprocessors such as the Sun Niagara [10] and multi-core workstations. The MTGL API shares many of the same concepts as the Boost Graph Library (BGL). As in the BGL, the visitor pattern enables users to apply methods at key points of algorithm execution. MTGL users write adapters over their graph data structures as BGL users do, but there is no assumption that Boost and STL are available. MTGL algorithms for connected components are well designed and have scaled almost perfectly on the Cray MTA-2.

3. TARGETTED GRAPH ALGORITHMS

The current focus of graph algorithms within the Titan toolkit falls within the domain of informatics to search through social networks or perhaps TCP/IP network traffic for intrusion detection, etc. This graphs constructed to model this type of data are often referred to as "semantic graphs" but are formally called Attributed Relational Graphs (ARGs), the distinguishing characteristic of an ARG is the data contained on its nodes and edges.

The following sections provide details on three algorithms implemented within the Titan toolkit.

3.1 Temporal Search

There are multiple ways to consider temporal data within a graph. One way might be to consider the evolution of a graph with respect to time. Another might be to consider the time associated with the links in a graph and how this might relate to the spread of information (or contamination) through the network. Our work to date has focused on the latter of these two.

The temporal search query was implemented in Titan natively. The ARG used by this algorithm treats nodes as entities and lets edges represent a discrete interaction between two nodes occurring at a particular time. Using the Enron email database as an example in which people are represented by nodes and emails are edges containing a timestamp showing when it was sent.

Temporal search asks the question: "Given a root node and a starting time, what other nodes are reached and at what times?" The goal of this search is to discover two things (a) where might an infection spread if it originates from a given point in the graph and (b) when is the earliest time a node might become infected. Our temporal search labels nodes with their 'earliest' infection time and marks edges as valid if they leave an infected node. A special label is given to the earliest infection edge emanating from infected nodes.

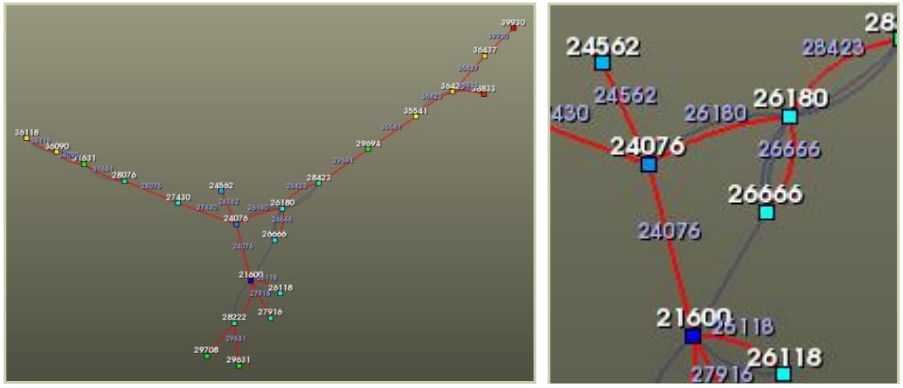

Figure 3: This figure shows a temporal search result on a graph of cell phone calls. The node and edge labels represent time-stamps for each call. The search was rooted at the call with time "21600"; red edges indicate the path of fastest propagation from the root node and blue edges are forward-in-time edges that are not the earliest.

The addition of the constraint that an edge can only be followed if its timestamp occurs after the node from which it originates differentiates this search from standard shortest-path type searches in that time-stamps are absolute values and are not incremental as in weighted shortest-paths. Given this constraint, we are also not guaranteed to take all available paths that might exist in the static graph. Figure 4 illustrates how a temporal search might progress through a simple graph. Note that node D is never visited because its interaction with B precedes B's earliest possible infection time.

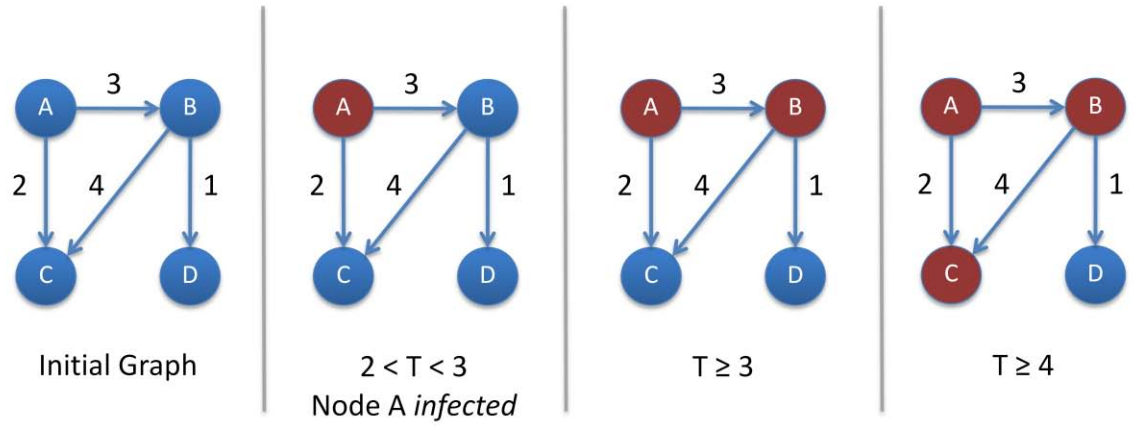

Figure 4: A simple graph illustrating 4 actors and the times associated with their connections. At time 2.5 node A is *infected.* The final state at T>=4 A,B, and C are infected but D is not because it interacted with B before B was infected.

3.2 Multi-ST/Connection Subgraph

The Multi-ST and Connection Subgraphs (CSG) algorithms described here perform a similar task. These algorithms can be considered as path-finding algorithms. Generally speaking both algorithms take a set of nodes within a graph and are tasked with finding path(s) within the input graph that connects the endpoints. Along the way, these algorithms attempt to optimize the paths they find in a very different way, resulting in results that can be quite different given the same inputs.

The Multi-ST search is an extension of a common graph ST search algorithm. The ST search is a two-phase search containing a *discovery* phase and a *recording* phase. In *discovery*, ST expands search frontiers using BFS from two nodes in a graph that we call the Source and the Target. If a path does exist between S and T, the frontiers of our BFS will eventually meet. Once we find our meeting point the discovery phase ends and we enter the recording phase. The recording phase will backtrack from the meeting point back to S and T using information saved during discovery and record the path it takes.

Since the ST search at its core only supports two endpoints, we extended the algorithm to support >2 endpoints by permuting a set of endpoints and merge results from individual searches. The current ST and Multi-ST search algorithms are implemented in MTGL.

Unfortunately, one problem with the ST search is that it only captures the *shortest* path between nodes. This is desirable in many cases such as finding driving directions on a map, etc. but graphs that model many informatics problems have a power-law degree distribution. These power-law graphs are generally sparse and have a low diameter resulting from a relatively small set of nodes that are highly connected. Shortest-paths algorithms will tend to gravitate towards these highly connected nodes, but those paths may not represent an interesting relationship between two nodes (i.e., In a social network the shortest path connecting myself to Kevin Bacon may pass through Bill Clinton but it's unlikely that path represents a substantive chain of relationships.)

With the rise of graph-based informatics on social network, a number of algorithms and heuristics have been proposed to try and find meaningful connections between individuals that avoid the "Bill Clinton" type nodes. We implemented one such heuristic to find these Connection Subgraphs [11] in MTGL and use it within Titan.

The CSG heuristic attempts to find meaningful paths between nodes in a graph by adding a penalty to highly connected nodes as well as to long paths. We accomplish this by modeling the graph as an electrical network in which the source node is treated as a voltage source and the target node is treated as a sink (ground). We then compute a conductance for every edge that is inversely proportional to the degrees of the nodes they connect. Finally, every node is connected to a ground through a link that has a fixed conductance. We then compute the amount of electric current that is delivered from S to T through this network and find the interesting connections by backtracking from T to S along connections that have the highest current values.

In this heuristic, the high resistance along edges that connect high-degree nodes will tend to force current away from these highly connected nodes. This satisfies the avoidance criteria for highly connected nodes. Additionally, the fixed conductance from each vertex to a global sink provides a 'tax' for each subsequent hop along a path we take providing an increasing penalty for long paths. The result is a path connecting S and T, which tends to avoid highly connected nodes and long paths if possible in the hope that the result represents a more meaningful connection between the endpoints.

3.3 Isomorphic Graph Queries

Database queries are generally quite effective for simple queries but as the query grows in complexity traditional SQL syntax and function can break down. For example:
"Show me all instances where payload X was transferred to computer Y, and then computer Y signaled known bad computer Z, and Z pulled payload X".

The SQL command to encapsulate this logic would be both significantly complex and extremely slow to execute. The "white board" version of this query might look something like this:

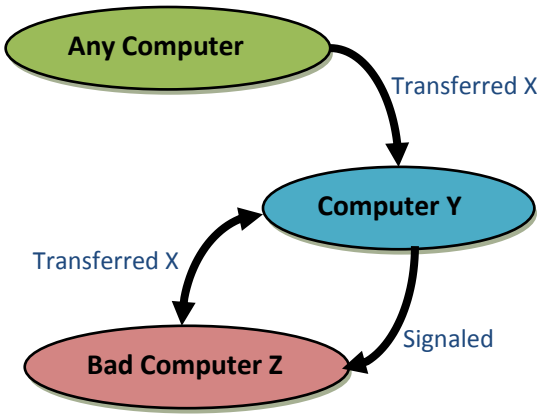

Figure 5: An example isomorphic graph query.

The diagram in Figure 5 is a graph and is the natural way to represent these types of queries. The research domain that addresses these types of semantic graph queries is called "Isomorphic Graph Search". Subgraph Isomorphism is an NP-Complete problem in general, but heuristics are available which can gain approximate results quickly when vertices and edges in a graph are colored. This data can be used to quickly prune down the possibilities.

Heuristic Description

Input

G_1 : Input graph.
G_2 : Graph, presumably much smaller than G_1, which we wish to locate within G_1, if it exists.

Output

Are there instances of G_2 within G_1? If so, return them.

Implementation

The basic procedure is the following:

1. Generate a walk, **W**, which covers all edges and vertices in G_1 containing the pattern of colors visited on each step. This will follow a vertex-edge-vertex-edge-vertex ... pattern. Think of this as unrolling G_1 into a linked list. Generating an Eulerian path works but not all graphs will have one. Relaxing the each-edge-is-traversed-exactly-once requirement lets us quickly get a walk that visits every vertex and edge in G_1.

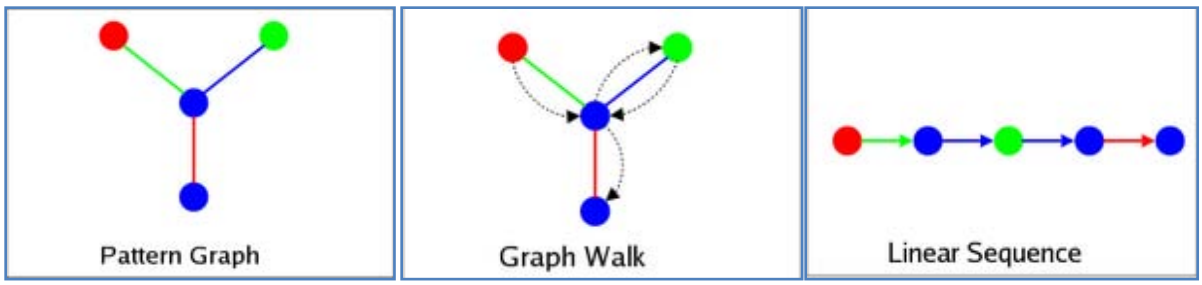

Figure 6: Diagrammatical representation of the isomorphic graph search. A user specifies a pattern of interest, an Eulerian tour is conducted (graph walk) and that walk is stored as a linear sequence.

2. Perform a filtered traversal of G_2 starting at vertices with the same color as W[0], following edges which respect the source-edge-target color pattern in W, until we get to the end of W.
3. At this point, any vertices in G_2 that are still active were reached by some valid traversal respecting the pattern encoded in W. So, the next step is to reverse our pattern, W, and re-traverse G_2 starting at the active vertices at the end of the previous phase, but this time we record the vertices and edges we follow.
4. Return the vertices and edges found.

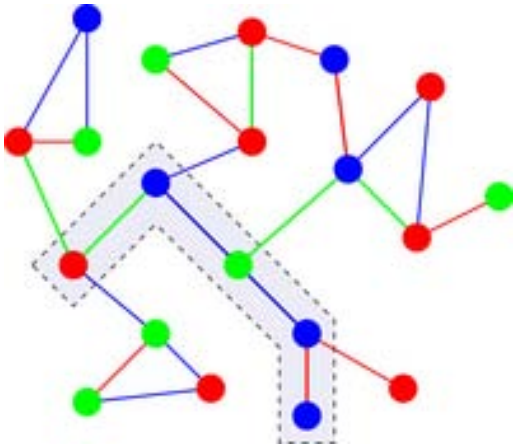

Figure 7: All instances of the user specified pattern are identified and highlighted.

Limitations

Classic (sub)graph isomorphism approaches that yield exact answers are well known to be NP complete, giving rather severe limitations to the size of graphs that isomorphism can be computed. We avoid this through the addition of two major constraints to this heuristic.

First, this heuristic based approach can only produce an inexact solution. While the existence of an inexact solution from this method does not guarantee that an exact match exists, we can say with confidence that if there are exact matches, they will be wholly contained within the

solution returned by this method. Further refinement could be done using exact solvers against the set of results if they are small enough to be tractable.

The second limitation to note is on the input data. This search only works for ARGs since the node/edge labels (i.e., colors) are what allow the aggressive pruning to occur in a way that we could not do if we looked only at the structure of the input graph. Though a limitation, we don't believe it a serious one for the types of problems we are likely to encounter as most useful data sets will contain node and edge attributes of some sort that is important to the problem.

4. RESULTS AND CONCLUSIONS

The Graph-Based Informatics for Non-Proliferation and Counter-Terrorism project was specifically targeted at combining graph algorithms and visualization. The project has significantly grown in scope since its inception and has now become the scalable software architecture (Titan) that links visualization, graph algorithms, linear algebra, tensor methods, statistics and deep text analysis. Within the specific scope of this project we achieved following goals:

- Data-less adapters to Boost, Parallel Boost, and MTGL graph algorithms.
- Multi-ST robust graph path-finding algorithm.
- Subgraph Isomorphic Predicate Engine.

In addition, the members of this project hope that the broader toolkit becomes the basis for a popular, long lived, open source toolkit for doing graph analysis, informatics, and visualization. We also believe the toolkit will attract external partnerships and WFO funding, while providing Sandia with novel analysis capability for non-proliferation and counter-terrorism.

4.1 SMU Benefit and Project Impact

The primary SMU benefiting from the research and development conducted in the "Graph-Based Informatics for Non-Proliferation and Counter-Terrorism" LDRD is DS&A, but the work may also be beneficial to the HSD and ST&E SMUs. All of the graph algorithms and visualization techniques developed as part of this LDRD have been deployed into an open source informatics toolkit called Titan (www.sandia.gov/Titan). The toolkit is quite flexible and can be combined in various ways for different problem domains, including strategically important problems to Sandia like large scale network packet monitoring (5600) and deep unstructured text analysis (5900). Our expectation is that the, already popular, open source toolkit will have a lifetime in excess of ten years, and that Sandia will be developing and using the toolkit perhaps across multiple SMUs during that lifetime.

The developers believe the open source toolkit will have significantly greater impact then an isolated in-house project. The Titan toolkit already has users/developers from around the world: Indiana University, Harvard, Cal-Tech, Leeds University and University of Utah all have known users or developers. We also believe that the toolkit's open standards and APIs will attract intelligence organizations dissatisfied with proprietary and inflexible commercial offerings. If these agencies see promise in a flexible, open source informatics toolkit, it may attract strategic long term partnerships and bring additional WFO funding into Sandia.

5. REFERENCES

[1] Wylie B., Baumes J., "A Unified Toolkit for Information and Scientific Visualization", Visual Data Analytics, 2009.

[2] Wylie B., Baumes J., Shead T., "Titan Informatics Toolkit", IEEE Visualization Tutorial, Columbus, OH, 2008.

[3] Wylie B., Baumes J., Shead T., "GSpace: A Linear Time Graph Layout", Visualization and Data Analysis, 2008.

[4] Cedilnik A., Baumes J., Ibanez L., Megason S., and Wylie B., "Integration of Information and Volume Visualization for Analysis of Cell Lineage and Gene Expression during Embryogenesis", Visualization and Data Analysis 2008.

[5] Boost C++ libraries. http://www.boost.org.

[6] J. G. Siek, L.-Q. Lee, and A. Lumsdaine. The Boost Graph Library: User Guide and Reference Manual, 2002.

[7] Press release: Sandia national labs visualizes some of the world's largest simulations using nvidia technology. http://www.nvidia.com/object/IO27539.html.

[8] J. Feo, D. Harper, S. Kahan, and P. Konecny. Eldorado. In *CF '05: Proceedings of the 2nd conference on Computing frontiers*, pages 28–34, New York, NY, USA, 2005. ACM.

[9] K. B. Wheeler, R. Murphy, and D. Thain. Qthreads: An api for programming with millions of lightweight threads. In *IEEE Workshop on MultiThreaded Architectures and Applications (MTAAP)*, 2008.

[10] Sun Niagara 2 processor. http://www.sun.com/processors/niagara.

[11] Faloutsos, C., McCurley, K. S., and Tomkins, A. 2004. Fast discovery of connection subgraphs. In *Proceedings of the Tenth ACM SIGKDD international Conference on Knowledge Discovery and Data Mining* (Seattle, WA, USA, August 22 - 25, 2004). KDD '04. ACM, New York, NY, 118-127. DOI= http://doi.acm.org/10.1145/1014052.1014068

6. DISTRIBUTION

1	MS1217	Ricardo A. Contreras	5925
1	MS1323	Brian Wylie	1424
1	MS1323	William McLendon	1423
2	MS9018	Central Technical Files	8944
2	MS0899	Technical Library	4536
1	MS0123	D. Chavez, LDRD Office	1011