VISUAL LANGUAGE FOR EXPLORING MASSIVE RDF DATA SETS

By

JUSTON MORGAN

A thesis submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WASHINGTON STATE UNIVERSITY
School of Engineering and Computer Science

MAY 2010

To the Faculty of Washington State University:

The members of the Committee appointed to examine the thesis of JUSTON MORGAN find it satisfactory and recommend that it be accepted.

Wayne O. Cochran, Ph.D., Chair

Orest Pilskalns, Ph.D.

Scott Wallace, Ph.D.

VISUAL LANGUAGE FOR EXPLORING MASSIVE RDF DATA SETS

Abstract

by Juston Morgan, M.S.
Washington State University
May 2010

Chair: Wayne O. Cochran

We demonstrate a novel method for visually exploring and browsing large collections of semistructured data modeled in RDF, a W3C standard for emerging web applications. The method hinges on a theoretical coupling between query language expressivity and structural summaries of data. For standard RDF query languages, this amounts to a bisimulation partitioning of the data. We adapt the classic Kanellakis-Smolka algorithm (KSA) for interactively computing the bisimulation relation, allowing user interaction through a graphical user interface (GUI). The GUI allows users to intuitively filter and structure results, implemented under the hood as a refinement of the underlying bisimulation partition by using KSA. Data is initially presented in the GUI as a single node, representing the totality of the data, and from which the user can iteratively search the data by repeatedly calling a filter or refinement step. The actions on a node cause new nodes to be created, which are connected to the previous node. A new node will contain a subset of the partition from the previous node. Any non-empty node can be used to further refine the search. This paper, will overview our approach and illustrate a current working prototype based on the methodology.

ACKNOWLEDGEMENT

I want to thank Dr. George Fletcher for all of his guidance, inspiration and friendship throughout this project. I also want to thank Dr. Cochran for stepping in as my advisor, and the rest of the committee members for everything they did to help me along the way.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

The *World Wide Web* (W3) has completely changed how information is shared. Web browsers allowed users to follow Hypertext links, which are links between two documents that could be located anywhere on the web, and search engines allowed for searching the linked documents. Over the years the main element of W3 has been linked documents and recently there has been an increased effort to create links to the raw data behind these documents. The raw data was usually stored in formats such as XML, CVS or marked up in HTML tables which remove most of the data's inherent structure and semantics. The structure of the data is very important for creating meaningful links to the data. Currently one of the best set of practices for publishing and linking structured data on the Web is Linked Data [8]. Linked Data depends on documents containing data in *Resource Description Framework* (RDF) format, which allows for creating typed statements that link arbitrary things.

The availability of RDF data has increased along with the popularity of the semantic web. RDF data contain a wealth of information that can be easily parsed by a computer. The problems with RDF data arise when attempting to find relationships while reading data with the human eye. This is a very difficult task because the RDF files are not sorted in any way and sections that would be similar could be separated by thousands of lines of data. Many different visual query languages have been purposed for browsing and displaying RDF data, because its much easier to have a computer display the data in some meaningful format. Even with all of the different RDF visual query languages available we feel that our system, *RDF Relationship Display* (R^2D) provides the first formally justified visualization of RDF, where the words formal was taken from [14]. RDF data is out there and is only going to become more prevalent in the future, so there is a need for good, logical, and easy to use RDF browsers.

Our method is based on a theoretical coupling between query language expressivity and structural summaries of data, amounting to a bisimilar partitioning of the data. To interactively compute the bisimilar partition we adapted the classic *Kanellakis-Smolka Algorithm* (KSA), and created a *Graphical User Interface* (GUI) that allows the user to view and perform tasks on the data. The partition, which is incrementally created by our version of KSA, is displayed as a set of blocks connected by edges on the canvas portion of the GUI. Each block contains at least one triple and all triples present in this block are bisimilar to each other. The overall graph that is created on the canvas can be constructed with a *SPARQL* query. Our definition of bisimilar and how it relates to the edges displayed between blocks is explained further in the next section.

The rest paper is structured as the followed. Chapter 2 gives detailed descriptions of RDF, *SPARQL* , KSA, our working definition of bisimilar and how it relates to RDF. In chapter 3 a review of many visual query languages are given and then each is compared to R^2D . A detailed explanation of R^2D can be found in chapter 4 and chapter 5 presents our conclusions and thoughts for the future of R^2D .

CHAPTER TWO

BACKGROUND

2.1 RDF and *SPARQL*

In this section we give the basic definitions that are at the core of R^2D . RDF is the *World Wide Web Consortium* (W3C) standard for representing information in the Semantic Web [17]. RDF stores information in subject-predicate-object triples, which allows for easy computer readability. Humans however have a hard time following RDF triples in very large files. The main problem for humans is recognizing the relationship between triples that are not near one another in large collection of RDF data. The terms *Uniform Resource Identifiers* (URIs), literal, and blank-node refer to the elements that make up the subjects, predicates and objects, and we call the set of all elements the atoms ($\mathring{A}$). An RDF file consists of triples that are made up of an enumerable set of $\mathring{A}$.

Definition 1 A RDF triple is an object t, where t = $\langle a_s, a_p, a_o \rangle \in \mathring{A} \times \mathring{A} \times \mathring{A}$. Where a_s = subject(t), a_p = predicate(t), and a_o = object(t).

Definition 2 A graph *G* is a finite set of triples [13]. Let:
$S(G) = \{subject(t) \mid t \in G\}$,
$P(G) = \{predicate(t) \mid t \in G\}$, and
$O(G) = \{object(t) \mid t \in G\}$,

The domain of *G* is the set of atoms occurring in *G* , denoted as $\mathring{A}(G) = S(G) \cup P(G) \cup O(G)$.

Example 1 Subset of $\mathring{A}$'s that will be used in further examples.
{John, Paul, Tim, Doug, William, Steve, empNo, ID, dept, department, directory, ext, Shipping, Sales, I T, Help Desk, Services, Integers[0-200]}

Table 2.1: RDF Triples

Triple ID	Subject	Predicate	Object
0	John	empNo	112
1	Paul	empNo	132
2	Tim	empNo	145
3	112	dept	Shipping
4	132	dept	Sales
5	145	dept	I T
6	Shipping	ext	027
7	Sales	ext	013
8	I T	ext	002
9	Steve	ID	156
10	William	ID	187
11	Doug	ID	152
12	156	department	S/R
13	187	department	Services
14	152	department	Help Desk
15	S/R	directory	05
16	Services	directory	06
17	Help Desk	directory	14

RDF triples created from the set of $\mathring{A}$ in Example 1.

The query language recommended by the W3C for RDF data is *SPARQL* [20]. *SPARQL* queries are very similar to SQL queries in structure. If, for example, there was a database that contained a table that held employee data, where empID was the key and it contained 2 other fields called name and salary. This table would first have to be converted into RDF, which would contain triples such as those in Table 2.2 before a *SPARQL* query could be run. Once the table was converted to RDF constructing a *SPARQL* query to return a certain empID's salary would be a very simple task. Example 2 gives the SQL query for returning the salary from and empID as well as what this would look like as a *SPARQL* query.

Example 2 SQL vs *SPARQL* for "Retrieve employee 1234's salary."

SQL

SELECT salary
FROM employees
WHERE empID = "1234"

SPARQL
SELECT ?sal
WHERE {empID:1234, LB:salary, ?sal .}

Table 2.2: Relational Database to RDF

Subject	Predicate	Object
emps:1234	LB:name	”John”
emps:1234	LB:salary	36000

RDF triples built from a table in a relational database.
Named employees, with columns empID, name and salary.

Example 3 Simple *SPARQL* query of ”Retrieve those who have empNo and dept”.
SELECT ?p ?t
WHERE { ?p empNo ?d . ?d dept ?t . }

Table 2.3: *SPARQL* Results

John	112	Shipping
Paul	132	Sales
Tim	145	I T

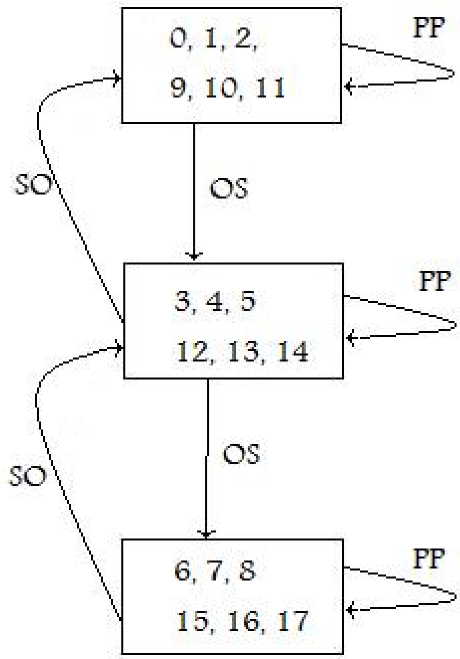
Result of the above *SPARQL* query on the data from Table 2.1.

2.2 Bisimilarity and KSA

We can view an RDF graph as a traditional directed graph, having triples as nodes, and there is an edge labeled XY from triple t to triple s if and only if position X of triple t and position Y of triple s contain the same atom (see Table 2.4 for all possible edge labels).

In our work the term bisimilar is used to describe equivalence between triples. Two triples s and t in an RDF graph are *bisimilar* if in the corresponding directed graph, if there is an edge labelled XY from t to some node t’, then there exists a node s’ such that there is an edge labelled XY from s to s’ and t’ and s’ are bisimilar, and vice versa. This equivalently comes from *SPARQL* query equivalence, where two triples are indistinguishable by

Figure 2.1: Graph Showing KSA Partitioning



View of the KSA partition on the triples from Table 2.1 labeled with edge types.

Table 2.4: Possible Edge Types

	Subject	Predicate	Object
Subject	SS	SP	SO
Predicate	PS	PP	PO
Object	OS	OP	OO

All nine possible edge types

SPARQL queries if and only if they are bisimulation equivalent [13]. The data in Table 2.1 is an example of a subset of data from two simple databases containing the same basic information in two schemas that were merged into one RDF file. In Figure 2.1 a sample graph has been built from the data in Table 2.1, by using R^2D . The figure shows the subset of triples that are contained in each block. The triples in the same block are said to be bisimilar. This means that all of the triples in each block are bisimulation equivalent because they are *SPARQL* query equivalent. Figure 2.1 shows that after running R^2D on this information the first block contained the subset of triples {0,1,2,9,10,11}, which are shown in Table 2.5. Looking at this data its easy to see how the triples are grouped with those that are all contain similar data. Tables 2.6 and 2.7 show the data from blocks 2 and 3 respectively.

Table 2.5: Block Data from Figure 2.1

Triple ID	Subject	Predicate	Object
0	John	empNO	112
1	Paul	empNO	132
2	Tim	empNO	145
9	Steve	ID	156
10	William	ID	187
11	Doug	ID	152

Bisimilar Triples in Block 1

Table 2.6: Block Data from Figure 2.1

Triple ID	Subject	Predicate	Object
3	112	dept	Shipping
4	132	dept	Sales
5	145	dept	IT
12	156	department	S/R
13	187	department	Services
14	152	department	Help Desk

Bisimilar Triples in Block 2

We implemented our version of KSA that allows the user to select one of the 9 edge types, to begin refining the data with. The 9 edge types are all of the possible combinations of S, P and O and are given in Table 2.4. The code for this function is shown in Listing A.1 and it returns the full partition (P) as well as the newest block created on this iteration. This function along with a few others allow users to browse RDF data for bisimilar relationships without having any knowledge of the structure of the data.

Table 2.7: Block Data from Figure 2.1

Triple ID	Subject	Predicate	Object
6	Shipping	ext	027
7	Sales	ext	013
8	IT	ext	002
15	S/R	directory	05
16	Services	directory	06
17	Help Desk	directory	14

Bisimilar Triples in Block 3

CHAPTER THREE

RELATED WORK

Many *visual query languages* (VQL) have been purposed for RDF, XML and *relational databases* (RDBs). The following is a brief overview of how these three three are different, a few VQLs for each, and how each of these compare to R^2D .

The main difference between RDF, XML and RDBs are in their data models. An advantage to using the RDF data model over XML or relational databases is in its simplicity. The RDF data model resembles a graph where the XML data model is a tree with several types of nodes, and RDB utilize flat tables as a data model. The ordering of RDF properties does not matter unlike the ordering of elements that are required in XML. RDF also makes use of URIs and other things that XML and RDB are agnostic to. One common misconception is that RDF is some sort of simple XML format. There is an XML serialization format for RDF data [5] as well as another format called *Notation 3* (N3) [6]. Table 3.1 shows how RDF triples are displayed in N3.

Real world information and their relationship can not always be neatly packed into hierarchies, as in XML or tables, as in relational databases. This information is more easily stored as a graph which can easily be converted into RDF triples that can be stored in a tabular manner using N3, shown in Table 3.2. This allows the information to be easily visualized as either a table or graph at the atomic level.

Table 3.1: N3

Notation 3
John Loves Jill. Jill Loves Tom. John Knows Tom. Tom WorksFor Jill.

RDF triples shown in N3 format.

Table 3.2: RDF triples in a Table

Num	Subject	Predicate	Object
1	John	Loves	Jill
2	Jill	Loves	Tom
3	John	Knows	Tom
4	Tom	WorksFor	Jill

Generic RDF triples in tabular format.

3.1 Challenges of Visual Querying of Data

There are many challenges when it comes to trying to visually query data. Some of these are: allowing users to input queries, the users knowledge level, structuring and displaying the output of the queries, and allowing queries on the results of previous queries, filters and/or constraints on the output. Papers such as [14], [1], and [22] give insight for visualizing data. [14] gives concepts for hyper-graphs which are graphs where the relation being specified does not have to be binary nor even of fixed arity. [1] gives concepts for graph-oriented user interfaces. [22] gives reasons for why *Great Big Graphs* (GBG) might not be the best way to present RDF data. These papers contained a wealth of information on how to deal with the challenges mentioned above. Indeed, the study of visual query languages is a very mature area of research [10]. In what follows, we just highlight approaches to visual querying most related to R2L, placing our approach in the broader landscape of visual tools for information systems.

3.2 Visual Query Languages

A few examples of visual query languages that use a data model other then RDF are *Query-by-Example* (QBE) [24], XML-GL [11], and *XQuery-by-Example* (XQBE) [9]. QBE is a high level data base management language for relational databases. XML-GL is a graphical language used for querying and restructuring XML documents. XQBE is a visual query language for expressing a large subset of XQuery in a visual form, and can be considered as an evolution of XML-GL. All three of these VQL's allow for visualizing queries constructed by users that only need a very general understanding of the query language.

3.2.1 QBE

QBE was created with the intent to allow users to query, update, define, and control a relational database even if they know very little about relational data bases [9]. The operations available in QBE mimic those of manual table manipulation. QBE starts out with a two-dimensional skeleton table and the user is free to start to fill in the table with examples of the desired solution in appropriate table spaces. There are many differences between QBE and R^2D . These include how the data is displayed, how users interact with the data, and how queries are formulated.

3.2.2 XML-GL

XML–GL is a query language for XML-GDM data [11]. An XML–GL query results in the creation of a new XML document. The four parts of XML–GL query are *extract*, *match*, *clip*, and *construct*. The extract part contains the scope of the query, which includes the target documents and the elements inside these documents. The match part, which is optional, and contains any logical conditions that must be satisfied by the target elements. The clip part is where sub-elements are specified on the extracted elements from the match part. The construct part, also optional, specifies any new elements that should be included in the result document. Graphically, XML–GL has two graphs side by side separated by a vertical line where the left side contains a visualization of the extract and match parts, and the right side contains the clip and construct parts. One similarity between XML–GL and R^2D is in the display of the data. Both systems use a graph-like-view to display data and any node in the graph can have more then one edge.

3.2.3 XQBE

XQBE is a visual query language for XML thats based on QBE. It allows for simple and complex XQuery queries to be visualized. They recommend using simple transformations and discourage its use for extremely complex transformations [9]. XQBE is regarded as a direct decedent of XML–GL and in appendix A of [9] they give a detailed comparison of XQBE and XML–GL. The results of XQuery queries are displayed as two graphs separated by a vertical line and edges that cross this vertical line are call binding edges. Since XQBE is similar to XML-GL and QBE it compares the same to R^2D as they did. Visually XQBE uses graphs to display query results with labeled edges between nodes adding to the expressivity of the language.

3.3 Visual Query Languages for RDF

When looking at a raw RDF file its difficult for humans to see the structure or follow all of the links, which is why there have been many attempts to create tools for visualizing RDF data. A few examples of VQL for RDF are Tabulator [7], Fenfire [16], [15], Graphite [12], Explorator [3], NITELIGHT [23], GRQL [4], and RDF-GL [18]. Tabulator is a powerful generic RDF browser that allows users to follow URIs and displays the data in tabular form. Fenfire is a RDF browser that allows for visualizing all of the subject–predicate–object relationships of a focused item that is either a subject or object. [15] presents a graphical notation for representing queries on semistructured RDF data, that is meant to be both easy to use and sufficiently expressive to cover a wide range of queries. Graphite is a tool that allows for visually constructing queries over RDF data at the atomic level. Explorator is a tool for user directed exploration of RDF data from either dereferencing an URI or a *SPARQL* query against a *SPARQL* Endpoint. A *SPARQL* Endpoint is just a machine-friendly interface towards a knowledge base. NITELIGHT is a Web-based graphical tool for semantic query construction based on the *SPARQL* specification. GRQL utilizes the

RDF/S data model for constructing queries expressed in a declarative language such as RQL. RDF-GL is the first graphical query language based on *SPARQL* , designed for RDF. Links in massive RDF data files are difficult for humans to follow so its imperative to have tools that allow these links to be visualized.

3.3.1 Tabulator

Tabulator is an extensive tool that allows for visualizing and following URIs of a specified RDF document in a variety of ways [7]. Tabulator has two distinct modes, exploration and analysis, which the user can easily switch between. In exploration mode the user is able to explore the RDF graph in a tree view where nodes of the tree can be expanded to get more information and links that may contain more RDF data about a given node are implicitly followed. In analysis mode the user is able to define a pattern to be searched for. The result of this query can be displayed as a table, calendar, and map. The only similarities between Tabulator and R^2D are they both use RDF data and give the user a way to search and explore the data.

3.3.2 Fenfire

Fenfire is a RDF browser that gives a graph view, where blocks are subjects and objects and edges are the predicates between them [16]. The graph initially has a focus block in the middle and all triples that contain information in the focus as either a subject or object are displayed. Those triples that have the focus as an object are displayed with the subject in a new box to the left of the focus and connected to the focus by the predicate of the triple. Similarly if the triple contains the focus as a subject the object of this triple appears in a block to the right of the focus again connected by the predicate. An example would be triples like “John IsA Man.” and “Jill Loves John.” where John is the focus. Then “Man” would be to the right of “John” connected by the “IsA” predicate and “Jill” would be to the left of “John” connected by the “Loves” predicate. Although R^2D ends up looking similar to Fenfire that is where the similarities end. R^2D does not deal with RDF data at the atomic level instead each of our blocks contains a subset of the complete set of RDF data and edges show the bisimilar relation between blocks.

3.3.3 RDF Facets

Andreas Harths paper on Graphical Representation of RDF Queries gives a graphical notation for representing queries for semistructured data [15] by use of what they call RDF facets. A simple definition of an RDF facet is a filter condition over the RDF graph. The facets can be done on either the subject or object and multiple facets done on the same variable amounts to a join. They give a subset of RDF queries that can be visualized in their graphical notation. RDF facets and R^2D both allow users to explore RDF data and display the results of the exploration in a graph. The way the exploration is done is vastly different and the graph that is produced by using RDF facets are at the atomic level where each node of the graph is a subject or object connected by the predicate, where as in R^2D each node contain one or more complete triples.

3.3.4 Graphite

Graphite is a visual query tool for large RDF graphs [12]. Graphite allows users to construct query patterns that return exact matches as well as near matches. The user interface has two main parts, the query area where users construct the query subgraphs is on the left side and on the right side is the result area that shows the exact and near matches in a way that is easy for the user to flip between. Graphite and R^2D both use graphs to display data which is the only similarity between the two programs.

3.3.5 Explorator

Explorator is an open-source exploration search tool for RDF graphs [3]. It provides a QBE interface along with a custom model of operations. It allows the user to explore URIs as if they were a *SPARQL* Endpoint which can be queried with *SPARQL* . Explorator allows users to build these queries, even if they do not know what *SPARQL* is, by using an intermediate function call that is easy to use. The main similarity between Explorator and R^2D is allowing the user to set a filter on the S, P, or O locations. They both also allow for exploring data but in very different ways.

3.3.6 NITELIGHT

NITELIGHT is a graphical editing environment for the construction of semantic queries based on *SPARQL* [23]. The interface for NITELIGHT has 5 main components which are the canvas, toolbar, ontology browser, properties panel and the result viewer. The canvas is where the graphical rendering of *SPARQL* queries occurs and once they appear they are selectable and can be manipulated by different functions in the toolbar. The ontology browser provides users with a starting point for query specification, and to facilitate the process of query formulation. The properties panel includes different operations that may be available on a selected item on the canvas. The results viewer displays the *SPARQL* query of the current data on the canvas. The graph that is constructed from the query is at the atomic level where nodes are subject or objects and the edges connecting them are the predicates, which is different then the graph constructed in R^2D .

3.3.7 GRQL

GRQL is a tool that a relies on RDFS data model using queries expressed in RQL [4]. This means that a user can explore graphically though the individual RDFS class and property definitions. RDFS is the schema definition for RDF. GRQL gives users the ability to browse and place filters on RDFS descriptions with out having expert knowledge of RQL or RDF.

3.3.8 RDF-GL

RDF-GL is a VQL for RDF that is based on *SPARQL* . There are three RDF-GL elements: boxed, circle and arrow and extra information is assigned to the elements basd on their shape and color. Very complex *SPARQL* queries can be recreated by using the elements to build a visualization of it. Their main focus so far as been on the SELECT query and they state that future research will be on the FROM, FROM NAMED and GRAPH elements of *SPARQL* . The only similarities between RDF-GL and R^2D are both use RDF data that the attempt to search on by the use of *SPARQL* queries. RDF-GL does this directly while R^2D currently does not display the results as a *SPARQL* query the result can always be obtained by running a *SPARQL* query on the data.

3.4 Visual Query Language Conclusion

Though many of these applications deal with RDF data and or display the data as a graph none of them do what R^2D does. All of these applications are ”resource” centric, which is to say that edges are the predicates of the triples, whereas we are ”relationship” centric, where edges are relationships between triples. We allow the user to explore the data, search for key words in any of the three fields (S, P, O), and allow the user to save the data from any block to be used later. In the graph each block contains only the triples that participate in the given edge type and this participation cascades throughout the entire graph. This allows for potentially visualizing the structure in the data, as a whole, instead of the structure that may be present at the atomic level.

CHAPTER FOUR

R^2D

This chapter gives a detailed explanation of R^2D . We discuss how to get into the initial state, explain the actions available from the task bar, all of the possible methods done on the blocks, and finally, go through a simple example demonstrating the functionality of R^2D using the data from table 2.1.

4.1 Start Up

R^2D begins by prompting the user to open a file that must be one of three different file extensions. The first extension are `.txt` file containing RDF N3 triples shown in Figure [6], another extension type are `.n3` files which also contain the data in N3 format along with some extra data used for decoding the information, and the finally there are the `.TRIPLES` extension where each triple is separated by $|*|$ (format used by DBPedia). If the file meets one of these specifications then R^2D enters its initial state shown in Figure 4.1. Here a single block is displayed on the canvas labeled with its ID number (always 0 for the initial block) and its current filter of ****** which means no filter. Once R^2D has reached this point it is ready for the user.

There are two main parts of the GUI: the task bar at the top and the canvas. The task bar contains all of the actions that the user can perform independent of the blocks, and the canvas is the available area for the blocks to be moved and where all of the newly created blocks appear.

4.2 Task Bar

The task bar contains nine buttons allowing the user to do many actions very quickly. These buttons are, in order, from left to right, “Start Over”, “New File”, “Load Query”,“Save

Figure 4.1: Initial State

Query", "Save All", "Options", "Tutorial", "Help", and "Exit". The functionality for each button is explained below.

The "Start Over" button opens a dialog box asking whether to start over or not, which is set up as a safety mechanism in case the user changed their mind or did not mean to hit this button. Then if the user then selects yes another dialog window opens asking to save the current data. If the user selects no to starting over then no actions are done and the user is returned to current state as if nothing happened. If the user selects yes a second time a native OS file browsing window opens so that the user can save the information and then the canvas is reset to the initial state with the last loaded file. If the user selects no to the saving the information then the canvas is reset to the initial state with the last loaded file right away. This allows for a quick and easy way to return to the initial state when methods done on the blocks have resulted in a set of empty blocks.

The “New File” button allows the user to start over with a new file. When this button is selected a dialog box opens asking if the user is sure they wish to open a new file, again as a safety mechanism. If the user selects no then nothing is done and the user can proceed as if they never hit this button. If the user selects yes then another dialog box opens asking the user if they wish to save the current data to a file before the data is lost. After saving or not a native OS file selection browser opens that has a filter for the three types of files that can be loaded. Upon selecting a file, that satisfies the conditions stated in the “Start Up” section, the GUI is cleared to the initial state with the new file. This button allows the user to continue working on different files with out having to restart the program.

“Load Query” and “Save Query” buttons are currently non functional and will be discussed in more detail in the future works section.

The “Save All” button saves all of the data for each block into a text file. The text file contains the block information followed by all of the triples that are associated with this block. The file is saved with the triples being in N3 format which allows for this file to be opened as a new file. The block information consists of the block ID and then the edge type, parent block ID, and current SPO filter if available.

The “Options” button opens a small window that contains three options along with a button labeled “Go”. These options are ”Change BG Color”, “Change Label Color”, and “Change Block Color”. All three options open an OS specific color choosing window. The first option is used to set the background of the canvas and all subsequent option and data windows to the selected color. The second option is used for changing the color of the block labels. The final option is used for changing the fill color of the blocks displayed on the canvas along with any new blocks. These options allows the user to fully customize the color scheme in case the current scheme is difficult for them to read.

The “Tutorial” button opens a window that describes the basic functions that can be done on the blocks. These functions include how to drag the blocks around, how to open the method window, explains what each of the four methods are and how to use them. It also explains how to use the “Options” button to change the colors of the GUI.

The “Help Query” button opens a window and displays information the user can expect to obtain while performing the different actions on the blocks. It provides an explanation of what the arrows on the edges mean, in terms of which block is the parent and which is the child based on where the arrow points, along with how to read the arrow label. Other points that are explained using this button are how to add a filter and then change or remove this filter along with how filters will propagate to other blocks by the edge types. After reading this section a user should be able to use R^2D with confidence.

Lastly, the “Exit” button exits the program.

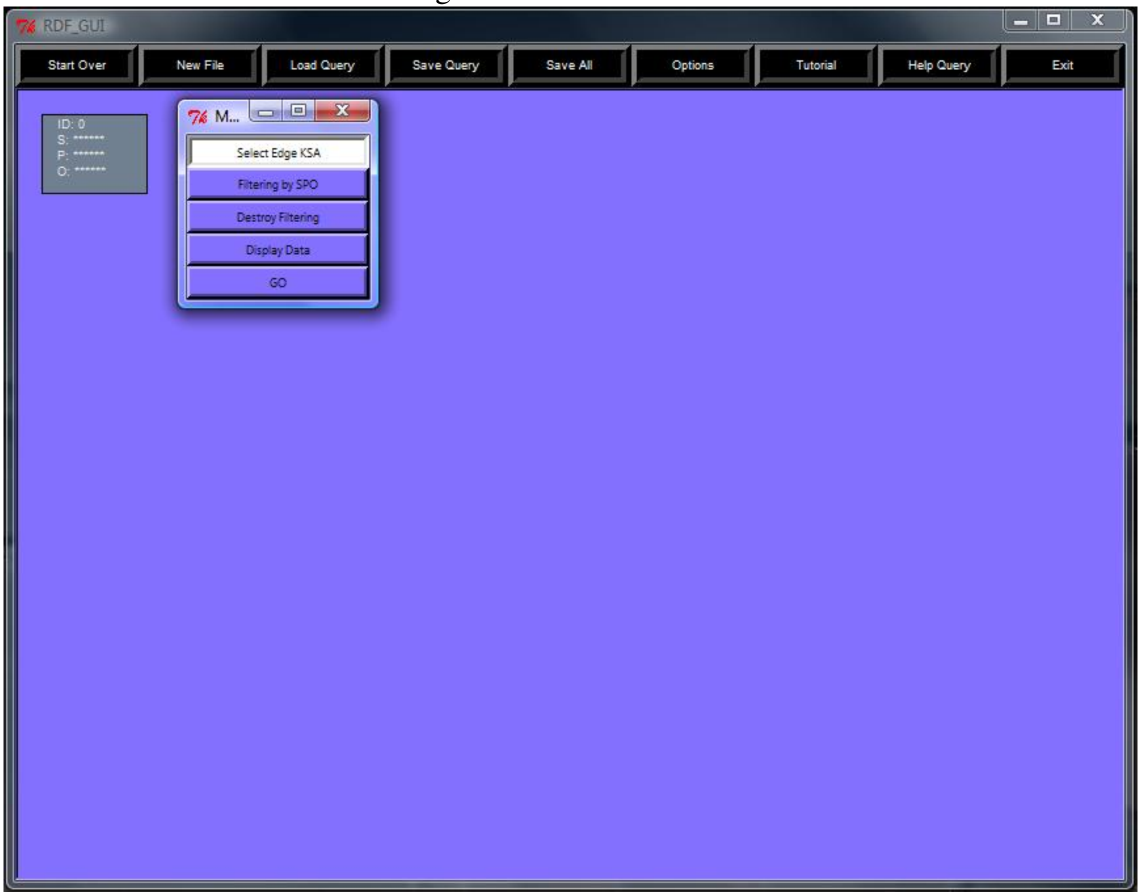
4.3 Blocks

To keep things simple there are only two actions, moving (left mouse button) and open methods (right mouse button). This is done to avoid having to do complicated clicks or key presses on a block to access the different methods available. Moving a block is accomplished by pressing down the left mouse button and then moving the mouse and then releasing the left mouse button which leaves the block at the last location before the left mouse button was released. Opening the method window is done by right clicking on the block shown if Figure 4.2.

The method window contains the methods that can be performed on the block. There are four methods including: Select Edge KSA, Filtering by SPO, Destroy Filtering, and Data Display. Once one of the four methods is selected the ”Go” button at the bottom is then pressed to start the selected method.

The Select Edge KSA method opens another window that contains all nine possible

Figure 4.2: Method Menu



edge types shown in Figure 4.4(a). Once and edge is chosen, R^2D goes into our KSA function and if it was successful then a new block is created on the canvas, and then all of the blocks are updated. The new block also adds a labeled arrow connecting the new block to the originally clicked on block with the arrow head pointing to this block, shown in Figure 4.4(b). This method helps to display the bisimilar relations that are present in the RDF triples.

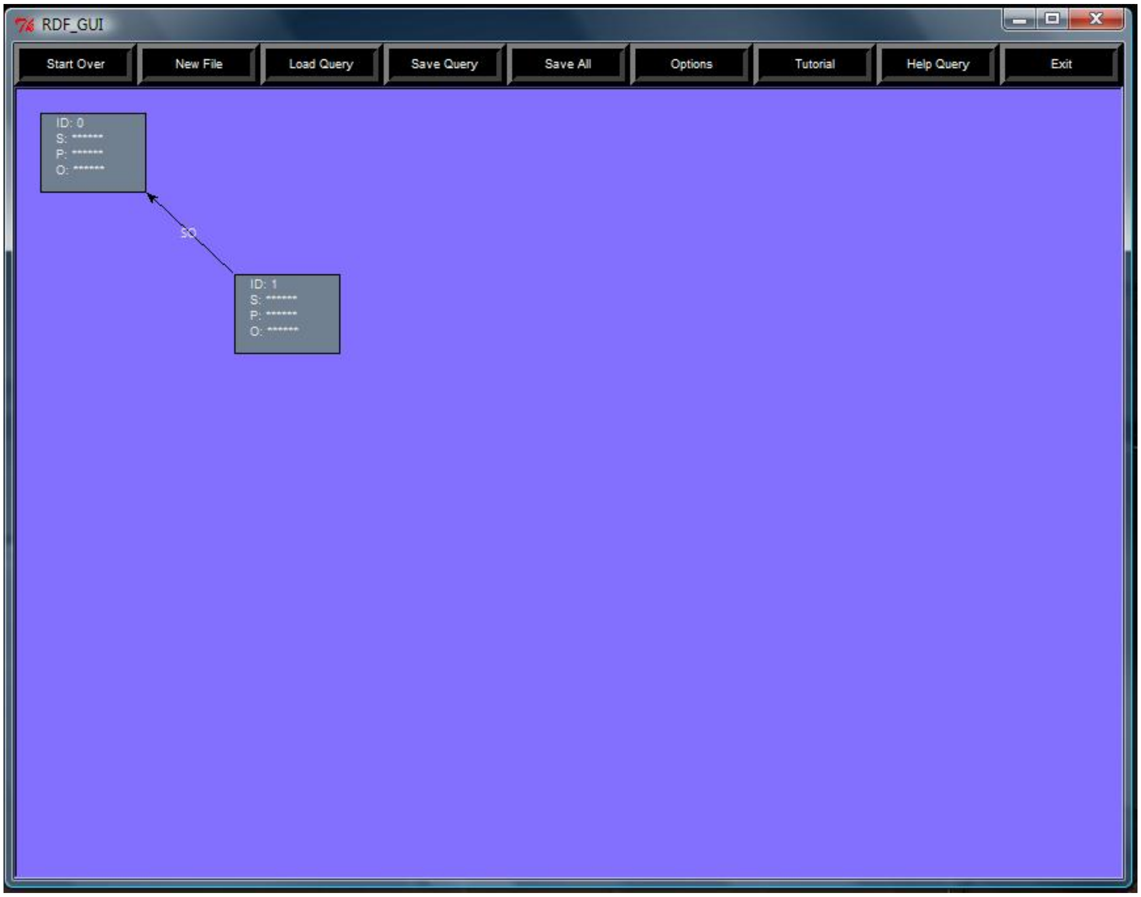
The Filtering by SPO method opens another window that has three fields, one for subject, predicate, and object. Filters can be entered on one, all three, or none of the fields. The filter window is shown in Figure 4.5(a). The filter will propagate to other blocks depending on the edge types between them. The propagation is shown in detail in Figure 4.3.

Figure 4.3: Select Edge KSA (a-b)

(a) Select Edge Menu

In Figure 4.5(a) the user has selected to filter on block 1 and has entered “Shipping” into the subject field. Figure 4.5(b) shows that ”Shipping” has propagated to the object filter of block 0 because of the SO edge between the blocks. Figure 4.5(c) shows that block 0 contains twelve triples before the filtering and Figure 4.5(d) shows that block 0 now only contains the triple where “Shipping” is the object.

The Destroy Filter method removes the filter from all of the blocks in the graph, which is useful when a filter has caused the graph to look empty. If a blank string is used as a filter it removes the filter that was present on this block at that given position. At this time



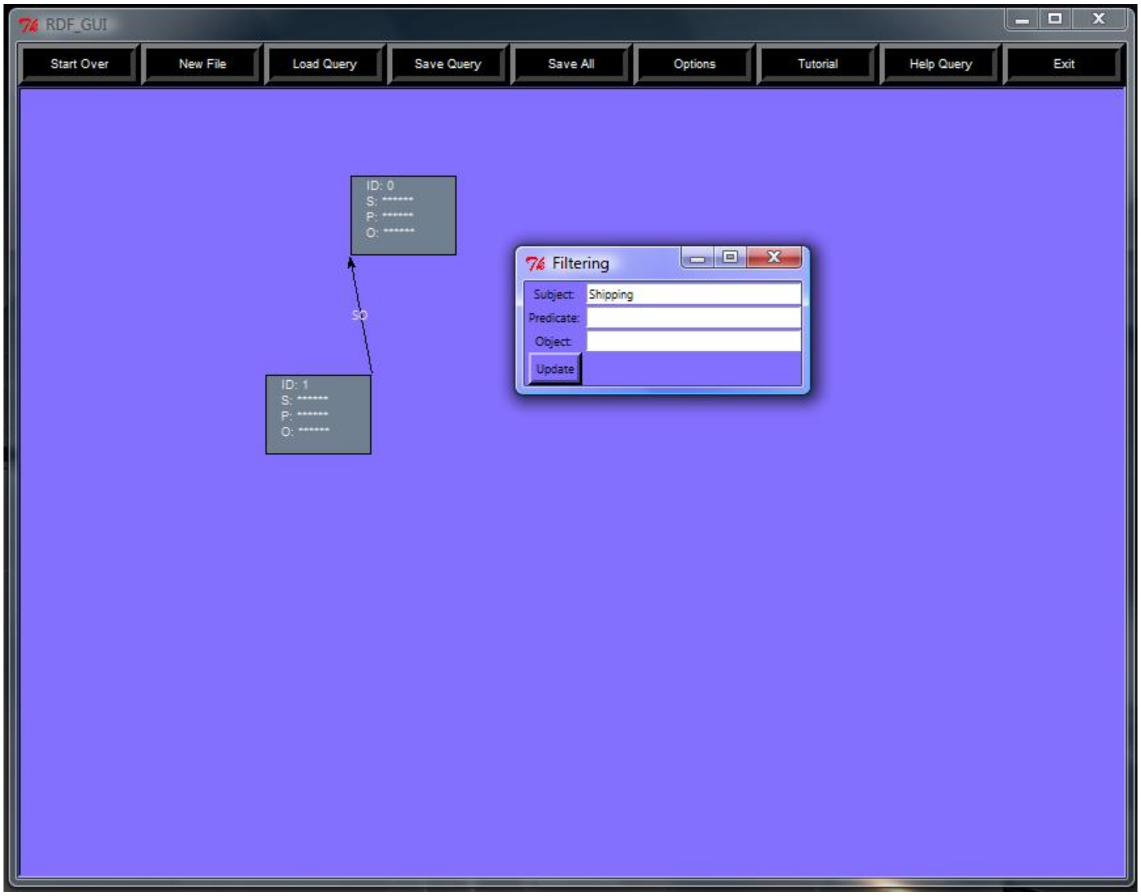
(b) After Edge Selection

Creating new blocks using Select Edge KSA method.

the empty string does not properly propagate so using the Destroy Filtering method is the preferred method for removing filters. The filter is done on top of the data blocks and does not directly alter the data so filtering and then destroying the filter can be done over and over without having to use the "Start Over" command.

The Data Display method displays the data of the current block in its own window, labeled with the block number. Figures 4.6(a), 4.6(b), and 4.6(c) shows how the data display windows look. The data in this new window can be saved to a text document, or closed using the exit button. This saved text document can be opened as a new file. The

Figure 4.4: Filter Propagation (a-d)

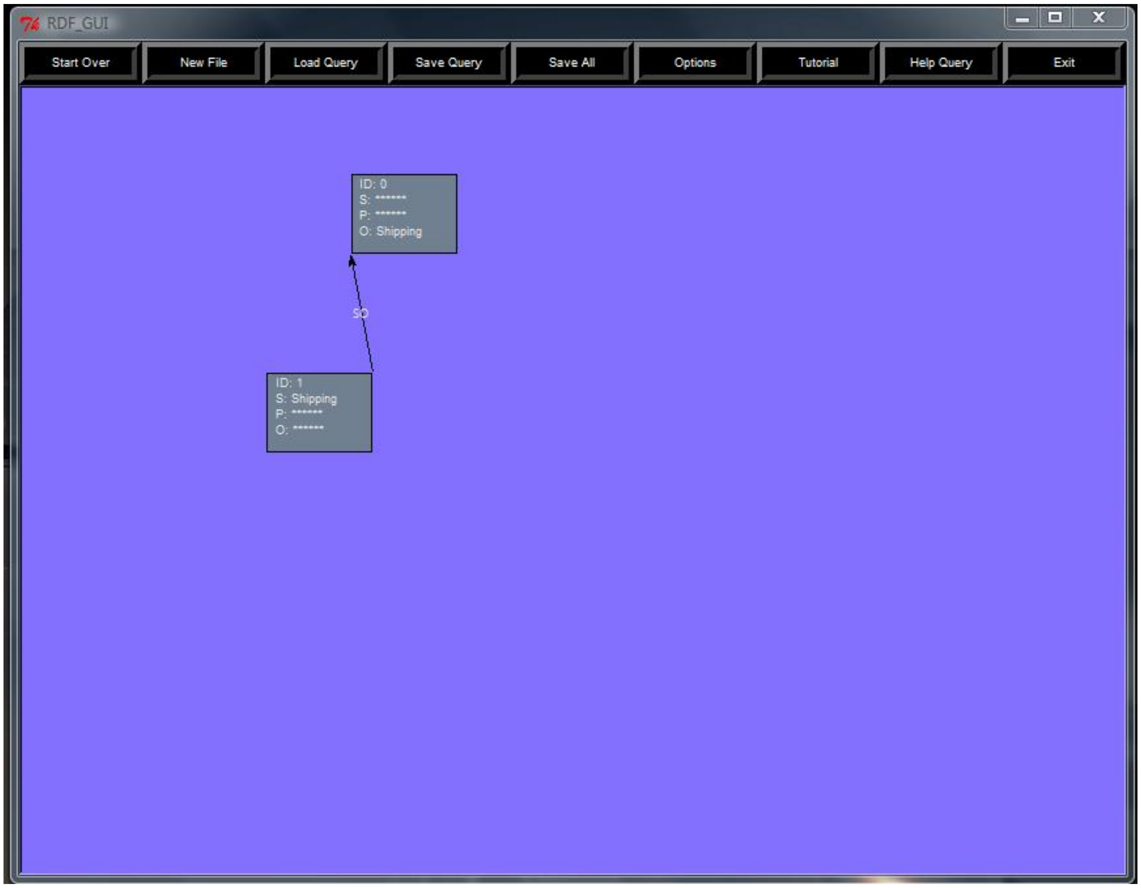


(a) OS Edge Filter on Block 1

window can also be left open and used to compared against the same block after a filter or adding a new edge to see how the data has changed.

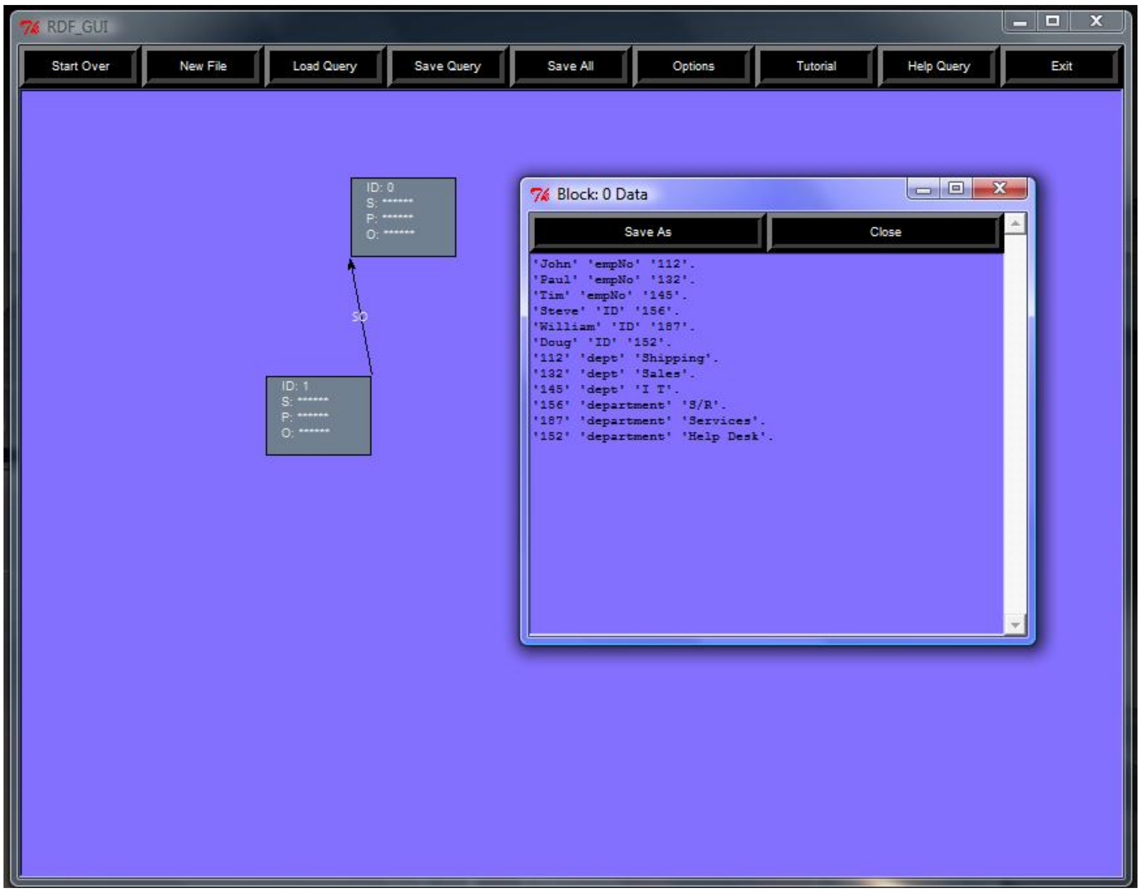
4.4 Example

Here is a brief example of how R^2D could be used on the data from table 2.1. The initial block contains all of the data thats present in the table shown in Figure 4.6(a). Creating an edge on this initial block (block 0) of the type SO will produce Figure 4.6(b) which also displays the data from both blocks after creating the edge. Creating another SO edge on

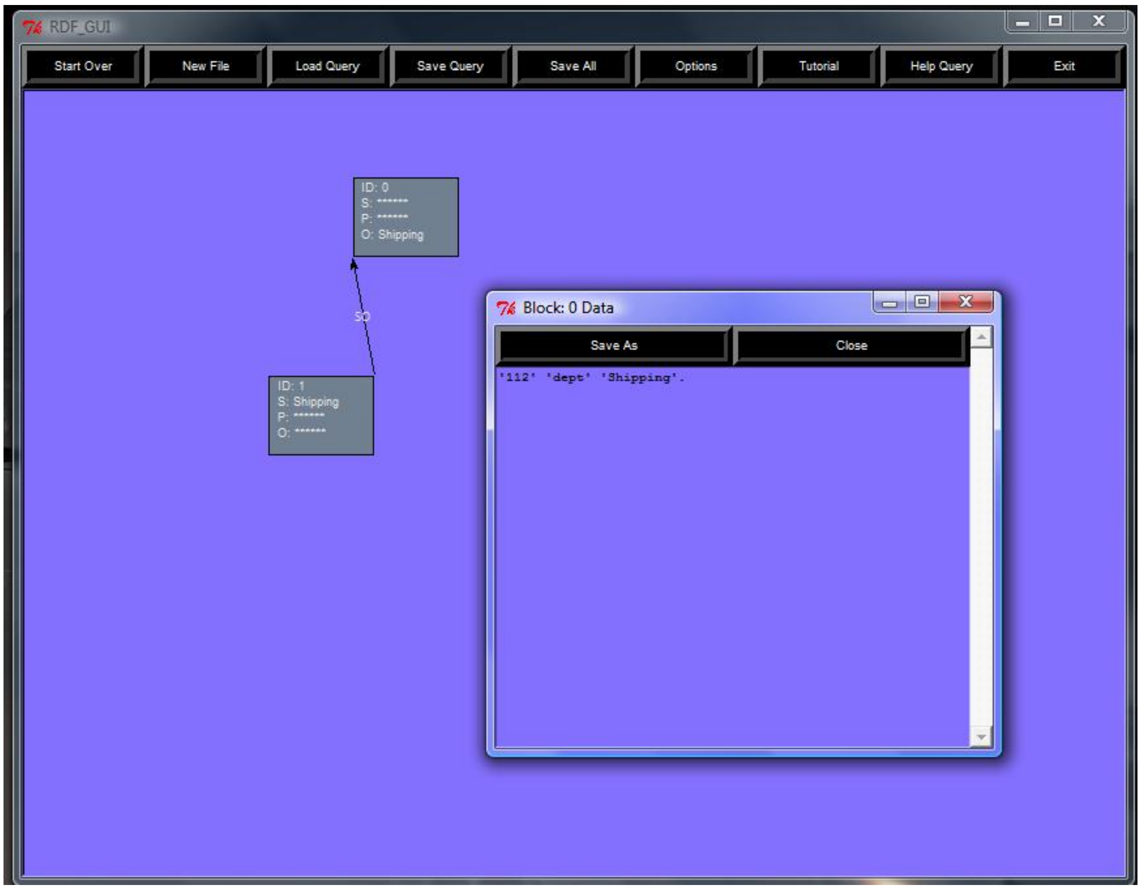


(b) Filter Jumping along OS Edge to Block 0

the new block (block 1) leads to Figure 4.6(c). This figure also contains the data from all three blocks. As you can see looking at the data that has been partitioned into the different blocks that data that has the same structure has been grouped together. This data set is very simple but it shows the potential power of R^{2}D .



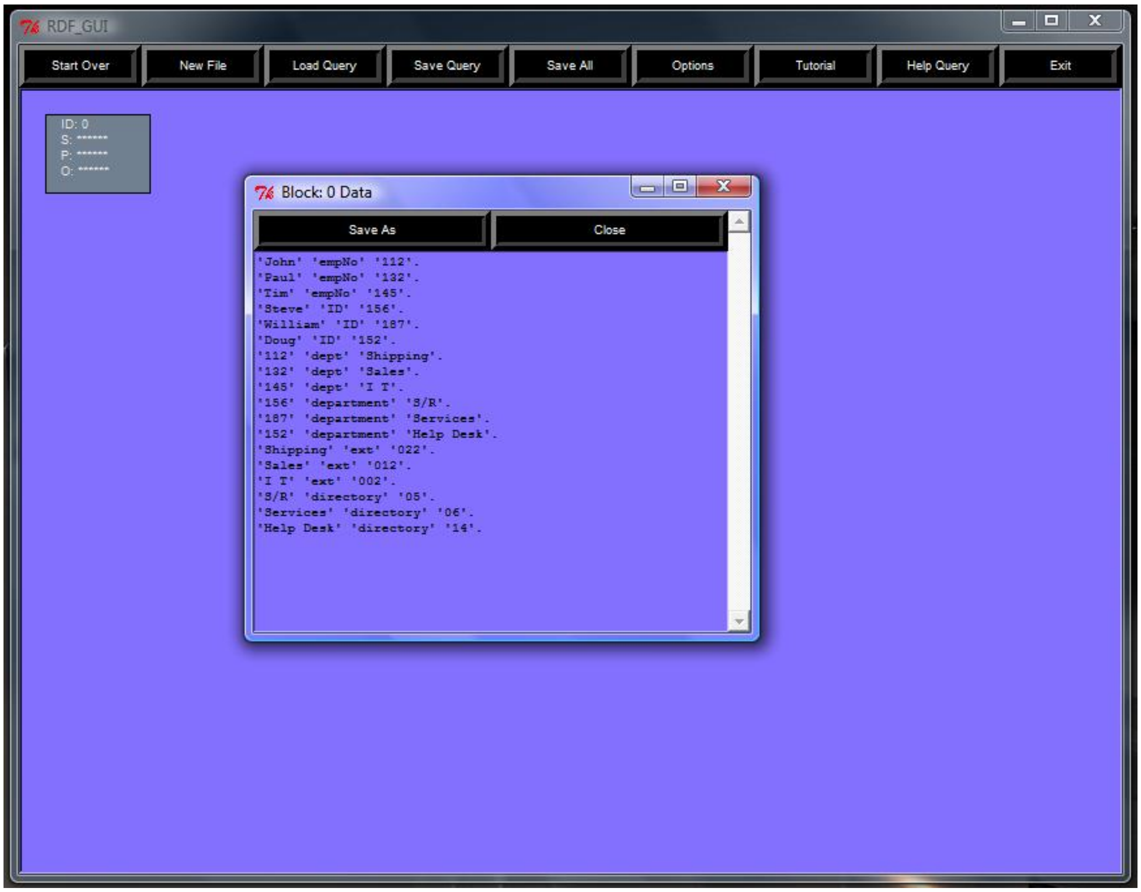
(c) Block 0 Data Before Filter



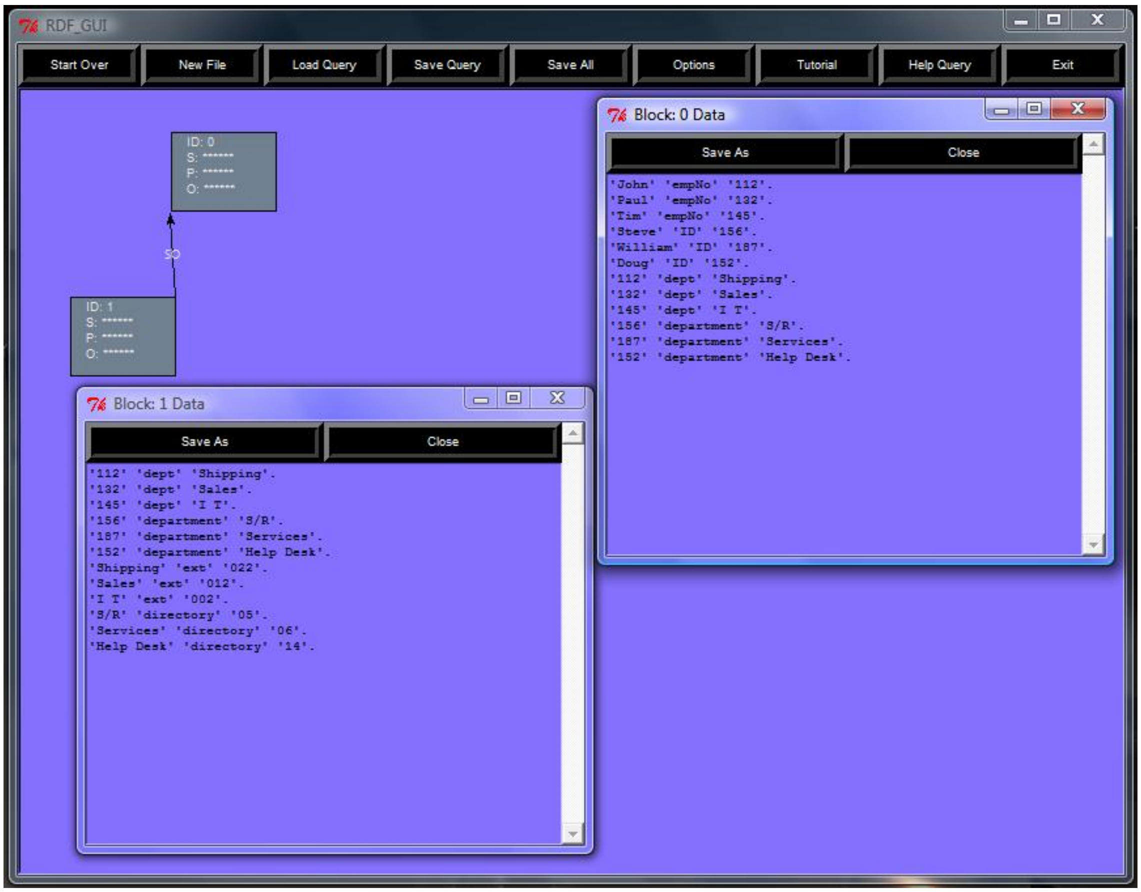
(d) Block 0 Data After Filter

How Filtering Propagates Along Edges.

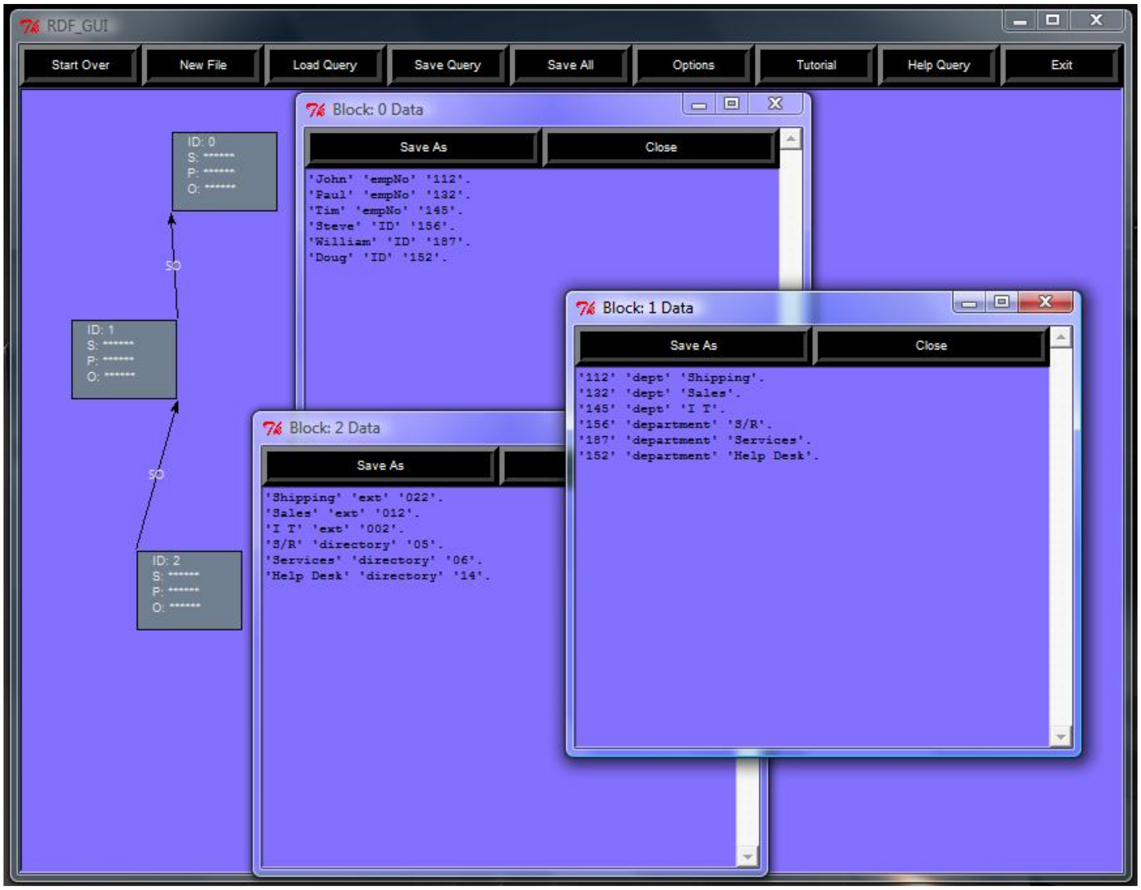
Figure 4.5: Data Display (a-d)



(a) Block 0



(b) Creation of Block 1



(c) Creation of Block 2

CHAPTER FIVE

CONCLUSION AND FUTURE WORK

There have been many different ideas and applications for visually browsing RDF data. Our application follows a path less traveled in that its based on a sound mathematical algorithm. The bisimilar relationships that are created can be recreated with *SPARQL* queries if the user was so inclined but our method allows the user to have no knowledge of what a *SPARQL* query even is. We set out to create a visual tool for browsing RDF data using a modified KSA. Our prototype meets these goals with a varying degree of success, but meets them none the less. With very little knowledge of the data a user can open a RDF file and begin to create filters or edges on the data. It currently takes what amounts to brute force to find any meaningful relationships in the data but starting with zero knowledge of the data or its structure brute force was acceptable for R^2D .

R^2D began as a simple visualization tool for displaying the results of our version of KSA. It has transformed into a much more sophisticated program that allow users to do much more then just visualize data. The future holds many more changes to R^2D including but not limited to: *SPARQL* query output, optimization of our KSA bisimulation function, giving the user the ability to see one step out, and changes to the overall appearance of the GUI. The output of a *SPARQL* query of a current graph could be very helpful if R^2D was used to test a small subset of a large data set with an unknown structure. The user would be able to experiment with different edge combinations on the small data set and then receive *SPARQL* queries that would then be run on the entire data set. The optimization of the KSA functions would allow for faster results and larger data sets to be input. Allowing the user to see all of the possible edges from all of the current blocks would help ease the pain of trying to find the structure by brute force. One purposed change to the GUI is displaying all of the possible edges and blocks one step out from the current blocks, which would allow the user to see whats possible instead of having to do this by brute force. Another idea in the works for the GUI is to rewrite the block movement functions and have new blocks appear in a more convenient fashion.

BIBLIOGRAPHY

[1] Marc Andries, Marc Gemis, Jan Paredaens, Inge Thyssens, and Jan Van den Bussche. Concepts for graph-oriented object manipulation. In *EDBT '92: Proceedings of the 3rd International Conference on Extending Database Technology*, pages 21–38, London, UK, 1992. Springer-Verlag.

[2] Renzo Angles. A Nested Graph Model for Visualizing RDF Data. In *3rd Alberto Mendelzon International Workshop on Foundations of Data Management*, Arequipa, Peru, 2009.

[3] Samur Araújo and Daniel Schwabe. Explorator: a tool for exploring RDF data through direct manipulation. In *Proceedings of the Linked Data on the Web Workshop*, Beijing, China, 2009.

[4] Nikolaos Athanasis, Vassilis Christophides, and Dimitris Kotzinos. Generating on the fly queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL). In *International Semantic Web Conference*, pages 486–501, 2004.

[5] Dave Beckett. RDF/XML Syntax Specification (Revised), 10 Feburary 2004. `http://www.w3.org/TR/rdf-syntax-grammer`.

[6] Tim Berners-Lee. Notation 3. `http://www.w3.org/DesignIssues/Notation3`.

[7] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and analyzing linked data on the semantic web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, 2006.

[8] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems*, 2009.

[9] Daniele Braga, Alessandro Campi, and Stefano Ceri. XQBE (XQuery by Example): A visual interface to the standard XML query language. *ACM Trans. Database Syst.*, 30(2):398–443, 2005.

[10] Tiziana Catarci, Maria Costabile, Stefano Levialdi, and Carlo Batini. Visual Query Systems for Databases: A Survey. In *Journal of Visual Languages and Computing*, pages 215–260. Academic Press Limited, 1997.

[11] Stefano Ceri, Sara Comai, Ernesto Damiani, Piero Fraternali, Stefano Paraboschi, and Letizia Tanca. XML-GL: A Graphical Language for Querying and Restructuring XML Documents. *Computer Networks*, 31(11-16):1171–1187, 1999.

[12] Duen Horng Chau, Christos Faloutsos, Hanghang Tong, Jason I. Hong, Brian Gallagher, and Tina Eliassi-Rad. GRAPHITE: A visual query system for large graphs. In *ICDM Workshops*, pages 963–966. IEEE Computer Society, 2008.

[13] George H. L. Fletcher. An algebra for basic graph patterns. In *Logic in Databases*, Rome, Italy, 2008.

[14] David Harel. On visual formalisms. *Commun. ACM*, 31(5):514–530, 1988.

[15] Andreas Harth, Sebastian Ryszard Kruk, and Stefan Decker. Graphical representation of rdf queries. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 859–860, New York, NY, USA, 2006. ACM.

[16] Tuukka Hastrup, Richard Cyganiak, and Uldis Bojars. Browsing linked data with fenfire. In *Proceedings of the Linked Data on the Web Workshop*, Beijing, China, 2008.

[17] Ivan Herman, Ralph Swick, and Dan Brickley. Resource Description Framework (RDF). `http://www.w3.org/RDF`.

[18] Frederik Hogenboom, Viorel Milea, Flavius Frasincar, and Uzay Kaymak. RDF-GL: A SPARQL-Based Graphical Query Language for RDF. In *In Y. Badr, A. Abraham, A.-E. Hassanien And R. Chbeir (Eds.), Emergent Web Intelligence: Advanced Information Retrieval*, Springer Verlag, Berlin, 2010.

[19] Paris C. Kanellakis and Scott A. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Inf. Comput.*, 86(1):43–68, 1990.

[20] Eric Prud'hommeaux and Lee Feigenbaum. SPARQL Qinproceedingsuery Language for RDF Errata. `http://www.w3.org/2001/sw/DataAccess/query-errata`.

[21] Davide Sangiorgi. On the origins of bisimulation and coinduction. *ACM Trans. Program. Lang. Syst.*, 31(4):1–41, 2009.

[22] M.C Schraefel and D. Karger. The pathetic fallacy of rdf. In *International Workshop on the Semantic Web and User Interaction (SWUI)*, Athens, Georgia, 2006.

[23] Paul R. Smart, Alistair Russell, Dave Braines, Yannis Kalfoglou, Jie Bao, and Nigel R. Shadbolt. A visual approach to semantic query design using a web-based graphical query designer. In *EKAW*, pages 275–291, 2008.

[24] Moshé M. Zloof. Query-by-Example: A data base language. *IBM Systems Journal*, 16(4):324–343, 1977.

APPENDIX

APPENDIX ONE
SOURCE CODE

Listing A.1: User defined KSA

```
## setTriples is the tripleTable (self.tripleTable) for this file ##
    ## setEdges is the set of edges (self.edgeNode) for this file ##
    ## theGraph list of all triple IDs
    ## edteType list of the edge type ['SS', 'SP', ..., 'OO] that the ##
    ## user is asking for ##
    ## returns the created Partition list and a listing of the newest ##
    def userDefKSBiSim(self, setTriples, setEdges, theGraph, edgeType):
        block2 = ''
        P = range(len(setTriples))
        P.sort()
        self.blockDict[min(P)] = P[1:]
        self.blockHashList.append(min(P))
        P = [P]
        spliterSet = P[:]
        sSet = set([])
        count = 0
        while spliterSet != []:
            S = spliterSet[count % len(spliterSet)]
            spliterSet.remove(S)
            for l_type in edgeType:
                C = self.findEdge(l_type, self.edgeNode, S)
                if C == []:
        pass
                #need to add in the new items to the tables
                elif C[0] not in self.blockHashList:
                    self.blockHashList.append(C[0])
                else:
                    self.blockDict[C[0]] = C[1:]
                if C != []:
                    for block in P:
                        bSet = set(block)
                        cSet = set(C)
                        interBC = cSet.intersection(bSet)
                        if ( interBC != set([]) ) and (interBC != bSet):
                            block2 = bSet - interBC
                            P, spliterSet = self.cleanPartition(P, list(spliterSet), list(block), list(interBC), list(block2))
                        else:
                            pass
                else:
                        pass
                        count += 1
        return P, list(block2)
```

Listing A.2: Source Code

```

"""RDF Relational Display, version 3.0
input: filename"""

import sys, os, time, tkMessageBox
#import sets
from Tkinter import *
from tkColorChooser import askcolor
from tkMessageBox import *
from tkFileDialog import askopenfilename
from tkFileDialog import asksaveasfilename


class rdfGui:



    def __init__(self, parent, fi):

#################### Global Sizes ######################################

        ## Frame ##
        FRW = 800
        FRH = 600

        ## Canvas ##
        CANW = (.5 * FRW)
        CANH = FRH

        ## Boxes ##
        self.BoxW = 80
        self.BoxH = 60

        ##block label default##
        self.bld = "******"
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

#################### Global Color Options for GUI #####################

        self.bgColor = "light slate blue"
        self.bgColorText = "light slate blue"
        self.menuButtonBGColor = "black"
        self.menuButtonFont = ('helvetica', 9)
        self.menuButtonFGColor = "white"
        self.recColor = "slate grey"
        self.recColorOutline = "red"
        self.recColorMoving = "dark slate blue"
        self.textColor = "black"
        self.labelTextColor = "snow2"
```

```

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

################### Global List and Dicts ###########################


        self.recInfoList = [] # [(rectID, [blockList])]
        self.recList = [] # [(rectID, (x,y))]
        self.recLineList =[]
        self.recLabel = []
        #index is the same as the recList, 1 filter for each rec
        self.filterList = []
        self.lineLabel = []
        self.edgeLabelDic = {}
        self.bboxDic = {}
        self.dataDic = {}
        self.tempLineList = []
        self.recListName = [0]
        self.recCount = 1
        self.oldx = 0
        self.oldy = 0
        self.activeBlock = 0
        self.tempStr = ''
        self.tagList = []

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#


##################### GUI Initialize ################################

        self.myParent = parent
        parent.title("RDF_GUI")
        self.myContainer = Frame(parent, width=FRW, height=FRH)

        #only activate one of the following two function calls
        ##### MenuBar for New, Save, Exit, Help
        #       self.makeMenuBar(self.myParent, self.myContainer)

        ##### Buttons for New, Save, Exit, Help ########
        self.makeButtonMenu(self.myParent, self.myContainer)


        self.canvas = Canvas(self.myContainer, bg = self.bgColor, \
                        relief=SUNKEN)
        self.canvas.config(bd=2,width = 2*CANW, height= CANH)
        self.canvas.config(highlightthickness=0)
        self.canvas.pack(side=LEFT, expand=YES, fill= BOTH)
        self.myContainer.pack()

################## RdfTableBuilder Class instantiation ############
        if fi != '':
```

```
            self.rtb = RdfTableBuilder(self, fi)
            self.KSpar = self.rtb.KSBiSim(self.rtb.tripleTable, \
                    self.rtb.edgeNode, [range(len(self.rtb.edgeNode))])
        else:
            self.fileD(1)
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

    ## No longer used but saved in case I wanted to use a scroll bars##
    def howToScroll(self, window, width, length):
        #scroll bar on/off code
        window.config(scrollregion= (0,0,width, length))
        sbary = Scrollbar(window)
        sbarx = Scrollbar(window, orient=HORIZONTAL)
        sbary.config(command=window.yview)
        sbarx.config(command=window.xview)
        window.config(yscrollcommand=sbary.set)
        window.config(xscrollcommand=sbarx.set)
        sbary.pack(side=RIGHT, fill=Y)
        sbarx.pack(side=BOTTOM, fill=X)


    ## Makes the menu bar ##
    def makeMenuBar(self, parent, thisCan):

        menu = Menu(thisCan)
        parent.config(menu=menu)
        filemenu = Menu(menu)
        helpmenu = Menu(menu)
        tutmenu = Menu(menu)
        menu.add_cascade(label="File", menu=filemenu)
        menu.add_cascade(label="Help", menu=helpmenu)
        menu.add_cascade(label="Tutorial", menu=tutmenu)
        filemenu.add_command(label="Start Over", command=self.startOver)

        filemenu.add_command(label="New File", command=self.newQuery)
        filemenu.add_command(label="Load Query", command=self.callback)
        filemenu.add_separator()

        filemenu.add_command(label="Save Query", command=self.callback)
        filemenu.add_command(label="Save All", command=self.callback)
        filemenu.add_separator()

        filemenu.add_command(label="Exit", command=parent.destroy)
        helpmenu.add_command(label="Help Query", command=self.helpQ)
        tutmenu.add_command(label="Tutorial", command=self.tutorial)



    ## Makes the Buttons instead of the menu bar ##
    def makeButtonMenu(self, parent, thisCan):

```

```
        buttonBD = 8
        bWidth = 12
        bCan = Canvas(thisCan, height=20, width=800, bg="black")

        bSO = Button(bCan, text="Start Over", command=self.startOver, \
                     relief=RAISED)
        bNew = Button(bCan, text="New File", command=self.newQuery, \
                      relief=RAISED)
        bLoad = Button(bCan, text="Load Query", command=self.callback, \
                       relief=RAISED)
        bSave = Button(bCan, text="Save Query", command=self.callback, \
                       relief=RAISED)
        bSaveAll = Button(bCan, text="Save All", \
                          command=lambda: self.saveAs(''), relief=RAISED)
        bExit = Button(bCan, text="Exit",\
                       command=parent.destroy, relief=RAISED)
        bHelp = Button(bCan, text="Help Query", command=self.helpQ,\
                       relief=RAISED)
        bTut = Button(bCan, text="Tutorial", command=self.tutorial,\
                      relief=RAISED)
        bOption = Button(bCan, text="Options",\
                         command=self.optionMenuBar, relief=RAISED)

        bSO.pack(side=LEFT)
        bSO.config(width = bWidth, bd=buttonBD,\
                   bg=self.menuButtonBGColor, \
                   fg=self.menuButtonFGColor, \
                   font=self.menuButtonFont)

        bNew.pack(side=LEFT)
        bNew.config(width = bWidth, bd=buttonBD,\
                    bg=self.menuButtonBGColor, \
                    fg=self.menuButtonFGColor, \
                    font=self.menuButtonFont)

        bLoad.pack(side=LEFT)
        bLoad.config(width = bWidth, bd=buttonBD,\
                     bg=self.menuButtonBGColor, \
                     fg=self.menuButtonFGColor, \
                     font=self.menuButtonFont)

        bSave.pack(side=LEFT)
        bSave.config(width = bWidth, bd=buttonBD,\
                     bg=self.menuButtonBGColor,\
                     fg=self.menuButtonFGColor,\
                     font=self.menuButtonFont)

        bSaveAll.pack(side=LEFT)
        bSaveAll.config(width = bWidth, bd=buttonBD,\
                        bg=self.menuButtonBGColor,\
```

```
                              fg=self.menuButtonFGColor,\
                              font=self.menuButtonFont)

          bOption.pack(side=LEFT)
          bOption.config(width = bWidth, bd=buttonBD,\
                            bg=self.menuButtonBGColor,\
                            fg=self.menuButtonFGColor,\
                            font=self.menuButtonFont)

          bExit.pack(side=RIGHT)
          bExit.config(width = bWidth, bd=buttonBD,\
                          bg=self.menuButtonBGColor,\
                          fg=self.menuButtonFGColor,\
                          font=self.menuButtonFont)

          bHelp.pack(side=RIGHT)
          bHelp.config(width = bWidth, bd=buttonBD,\
                          bg=self.menuButtonBGColor,\
                          fg=self.menuButtonFGColor,\
                          font=self.menuButtonFont)

          bTut.pack(side=RIGHT)
          bTut.config(width = bWidth, bd=buttonBD,\
                         bg=self.menuButtonBGColor,\
                         fg=self.menuButtonFGColor,\
                         font=self.menuButtonFont)

          bCan.pack(side=TOP, fill=BOTH)

     # returns color tuple and string representation of the selected color
     def AskForColor(self, title='Pick Color'):
          ctuple, cstr = askcolor(title=title)
          return ctuple, cstr


     # return "yes" for Yes, "no" for No
     def AskQuestion(self, title='Title', message='your question here.'):
          a = askquestion( title, message )
          if a:
               if a != 'no':
                    return 'yes'
          else:
               return a


     ## Opens a dialog box asking if the user wants to open a new file ##
     def newQuery(self):

          if self.AskQuestion('Open New File', "Open new file?") == 'yes':
               if self.AskQuestion('Save', "Save before clearning data?")\
```

```
              == 'yes':
                self.saveAs('')
            else:
                pass
            self.fileD(0)

        ## user decided they did not want to open a new file ##
        else:
            pass


    ## If the user wanted to open a new file a file dialog box opened ##
    ## If fname is a valid file name selected by the user then the file ##
    ## is opened and the program starts over ##
    ## If flag ==1 then this is the start of the program and there is ##
    ## no need to clean the data. ##
    ## If flag == 0 then clean data before starting over. ##
    def fileD(self, flag):

        fname = askopenfilename(title = "Choose New File",\
                                filetypes=[('txt files', '*.txt')])
        print fname
        if fname is None:
            showerror("Error!", "File unable to open")
            pass
        else:
            if fname == '':
                showerror("Error!", "File unable to open")
            elif flag == 0:
                self.clearData(fname)
            else:
                self.rtb = RdfTableBuilder(self, fname)
                self.KSpar = self.rtb.KSBiSim(self.rtb.tripleTable,\
                                 self.rtb.edgeNode,\
                                 [range(len(self.rtb.edgeNode))])
                self.driver('', "start", '')

    ## User is asked if they wish to start over with the existing file ##
    ## If the user wants to start over they are prompted asking ##
    ## whether they want to save or not ##
    ## The program is then started over at the begining ##
    def startOver(self):

        if self.AskQuestion('Starting Over',\
                            "Do you wish to start over?") == 'yes':
            if self.AskQuestion('Save',\
                                "Save before clearning data?") == 'yes':
                self.saveAs('')
            else:
```

```
                pass
            self.clearData('')
            self.driver('', "start", '')

        else:
            pass


    ## fname is the name of the file to be opened if the user wanted ##
    ## to open a new file ##
    ## This function sets all of the data structures back to initial ##
    ## starting configurations ##
    def clearData(self, fname):

        del self.recInfoList[:]
        del self.recList[:]
        del self.recLineList[:]
        del self.recLabel[:]
        del self.filterList[:]
        del self.lineLabel[:]
        self.edgeLabelDic.clear()
        self.bboxDic.clear()
        self.dataDic.clear()
        del self.tempLineList[:]
        del self.recListName[:]
        self.recListName = [0]
        self.recCount = 1
        self.oldx = 0
        self.oldy = 0
        self.activeBlock = 0
        self.tempStr = ''
        del self.tagList[:]

        self.canvas.delete(ALL)
        if fname != '':
            self.rtb.clearRDFData()
            self.rtb = RdfTableBuilder(self, fname)
            self.KSpar = self.rtb.KSBiSim(self.rtb.tripleTable,\
                                          self.rtb.edgeNode,\
                                          [range(len(self.rtb.edgeNode))])

            self.driver('', "start", '')

    def readTextFile(self,filename):
        line = filename.readline()
        text = ""
        while line:
            text = text + line
            line = filename.readline()
        return text

```

```
    def helpQ(self):
        try:
            helpFile = open('helpFile.rtf', 'r')
            hq = self.readTextFile(helpFile)
        except IOError:
            hq = "Could not open Help file, please make sure its" +\
                 " in the proper directory"

        self.textToCan(hq, "Help Query", 0)
        helpFile.close()

    ## Prints the tutorial information to a text window ##
    ## May need to change this to open a tutorial file later and ##
    ## display its content ##
    def tutorial(self):

        try:
            tutorialF = open('tutorial.txt', 'r')
            tut = self.readTextFile(tutorialF)
        except IOError:
            tut = "Could not open Tutorial file, please make" +\
            "sure its in the proper directory"

        self.textToCan(tut, "Tutorial", 0)
        tutorialF.close()

    ## returns the value from the radio option menu ##
    def getOselect(self):
        return self.optionVar.get()


    ## Creates the radio option menu ##
    ## Allows for changing colors, fonts and size of rectangle ##
    def optionMenuBar(self):

        self.optionVar = StringVar()
        self.optionVar.set('cc')
        self.optionWin = Toplevel()
        self.optionWin.title("Options")
        self.optionWin.config(bg=self.bgColor)
        Radiobutton(self.optionWin, text='Change BG Color',\
                    value = 'cc', bg=self.bgColor, bd=4,\
                    variable=self.optionVar, width = 25,\
                    relief=SUNKEN, indicatoron=0).pack(side=TOP)
        Radiobutton(self.optionWin, text='Change Label Color',\
                    value = 'cf', bg=self.bgColor, bd=4,\
                    variable=self.optionVar, width = 25,\
                    relief=SUNKEN, indicatoron=0).pack(side=TOP)
        Radiobutton(self.optionWin, text='Change Block Color',\
                    value = 'cr', bg=self.bgColor, bd=4,\
                    variable=self.optionVar, width = 25,\
```

```
                    relief=SUNKEN, indicatoron=0).pack(side=TOP)
        Button(self.optionWin, text="Go",bg=self.bgColor,\
                relief=RAISED, bd=4, width = 25,\
                command=self.optionGet).pack(side=BOTTOM)


    ## Uses the value from the radio option menu and calls ##
    ## functions for chaning GUI appearence##
    def optionGet(self):

        opt = self.getOselect()

        if str(opt) == 'cc':
            rgb, c = self.AskForColor()
            if c != None:
                self.bgColor = c
                self.bgColorText = c
                self.canvas.config(bg= c)

        elif opt == 'cf':
            rgb, c = self.AskForColor()
            if c != None:
                self.labelTextColor = c
                self.tempLineList = self.recLineList[:]
                self.redrawLine()
                for item in self.recLabel:
                    self.canvas.itemconfigure(item, fill= c)

        elif opt == 'cr':
            rgb, c = self.AskForColor()
            if c != None:
                self.recColor = c
                for rec in self.recList:
                    self.canvas.itemconfigure(rec[0], fill=c)

        else:
            pass
        self.optionWin.destroy()


    ## Saves text to the file specified by the user ##
    ## possible errors in the file are caught to avoid crashing ##
    def saveAs(self, text):

        fname = asksaveasfilename(title="Save File As...",\
                                  filetypes=[('txt files', '*.txt')] )

        if type(fname) is unicode:
            try:
                fo = open(fname, 'w')
                if text is '':
```

```
                    count = 0
                    #saving all data from all blocks
                    for i in self.recList:
                        fo.write("Rec: " + str(count) +"\n")
                        fo.write("Filter\n" + "Subject: " +\
                                 self.filterList[count][0] \
                                 + "\nPredicate: " +\
                                 self.filterList[count][1] \
                                 + "\nObject: " +\
                                 self.filterList[count][2] + "\n")
                        p, edge = self.findParent(i[0])


                        if p is not None:
                            fo.write("Has edge: "+ str(edge)+\
                                 " to block: " + str(p)+ "\n")
                        fo.write("Contains the following triples:\n")
                        temp = self.displayFilter(self.dataDic[i[0]],\
                                                  self.filterList[count])
                        fo.write(temp + "\n\n")
                        count += 1
                    fo.flush()
                    fo.close()
                else:
                    fo.write(text)
                    fo.flush()
                    fo.close()
                return 1
            except IOError:
                showerror("Error!", "Unable to Open File: " +fname )
                return 0
        else:
            showwarning("Warning!", "File Selection Cancelled")
            return 0


    ## Displays the data from the block into a new window ##
    ## Background color can be controlled by chaning bgColorText ##
    # in __init__ ##
    def textToCan(self, text, title, menuYN):

        self.txtwin = Toplevel()
        self.txtwin.title(title)
        scrollbar = Scrollbar(self.txtwin)
        scrollbar.pack(side=RIGHT, fill=Y)
        if menuYN is 1:
            W = self.rtb.longestTriple
        else:
            W = 60

        if W > 500:
```

```
            W = 500
        elif W < 60:
            W = 60
        halfW = int(W/2)
        bw = halfW-3

        if menuYN is 1:
            bc = Canvas(self.txtwin, height=20, width=W,\
                        bg="black", relief=RAISED)
            bSaveAs = Button(bc, text="Save As",\
                              command=lambda: self.saveAs(text),\
                              relief=RAISED)
            bSaveAs.config(bd=7, width=bw, bg=self.menuButtonBGColor,\
                            fg=self.menuButtonFGColor,\
                            font=self.menuButtonFont)
            bSaveAs.pack(side=LEFT)
            ex = Button(bc, text="Close", command=self.txtwin.destroy,\
                        relief=RAISED)
            ex.config(bd=7, width=bw, bg=self.menuButtonBGColor,\
                      fg=self.menuButtonFGColor,\
                      font=self.menuButtonFont)
            ex.pack(side=RIGHT)
            bc.pack(side=TOP, fill=BOTH)

        self.txt = Text(self.txtwin, bg=self.bgColorText,\
                        width = W, wrap=WORD,\
                        yscrollcommand=scrollbar.set)
        self.txt.pack()
        scrollbar.config(command=self.txt.yview)

        self.txt.config(state=NORMAL)
        self.txt.delete(1.0, END)
        self.txt.insert(1.0, text)
        self.txt.config(state=DISABLED)


    ## Generic command for button press testing ##
    def callback(self):
        print "filemenu action"


    ## Makes a new rectangle at cX, cY in the canvas ##
    ## Color of the Background, Text, and ActiveOutline ##
    ## color can be changed in __init__ ##
    ## self.recColor == Background color ##
    ## self.recColorOutline == outline color of the block when ##
    ## the mouse is over it ##
    ## self.textColor == color of text in the rectangle ##
    def makeRect(self, cX, cY):
        self.recListName[self.recCount-1] = \
                        self.canvas.create_rectangle(cX, cY,\
```

```
                                                            cX+self.BoxW,\
                                                            cY+self.BoxH,\
                                                            fill=self.recColor)
        self.bboxDic[self.recListName[self.recCount-1]] =\
                                          (cX,cY,cX+self.BoxW,cY+self.BoxH)
        self.canvas.tag_bind(self.recListName[self.recCount-1],\
                             '<Enter>', self.entered)
        self.canvas.tag_bind(self.recListName[self.recCount-1],\
                             '<ButtonPress-1>', self.recLClick)
        self.canvas.tag_bind(self.recListName[self.recCount-1],\
                             '<ButtonPress-3>', self.recRC)
        self.canvas.tag_bind(self.recListName[self.recCount-1],\
                             '<B1-Motion>', self.recDrag)
        rec = "\nID: " + str(self.recCount-1)
        ts = "\nS: " + self.bld
        tp = "\nP: " + self.bld
        to = "\nO: " + self.bld
        self.recLabel.append(self.canvas.create_text(cX+30,cY+20,\
                                                  font=self.menuButtonFont,\
                                                  text = rec + ts + tp + to,\
                                                  fill= self.labelTextColor))
        self.canvas.tag_bind(self.recLabel[-1], '<Enter>',\
                             self.entered)
        self.canvas.tag_bind(self.recLabel[-1], '<ButtonPress-1>',\
                             self.recLClick)
        self.canvas.tag_bind(self.recLabel[-1], '<ButtonPress-3>',\
                             self.recRC)
        self.canvas.tag_bind(self.recLabel[-1], '<B1-Motion>',\
                             self.recDrag)
    #add rec to list
        self.recList.append((self.recListName[self.recCount-1], (cX, cY)))
        self.filterList.append(['', '', ''])

    ## Action done each time the mouse goes over a rectangle/block ##
    ## Used mainly for debugging for now ##
    def entered(self, event):
        pass



    ## Displayes the contents of the closes block ##
    def printtext(self, event):

        self.findClosest(event.x, event.y)
        pstr = self.displayFilter(\
            self.dataDic[self.recList[self.activeBlock][0]],\
            self.filterList[self.activeBlock])
        self.textToCan(pstr, "Block: " +\
                       str(self.activeBlock) + " Data", 1)

```

```

    ## Function controls the draging of the blocks ##
    ## The postion of the block is updated as its moved along with ##
    ## the text and the arrows are redrawn on release ##
    def recDrag(self, event):

        event.widget.itemconfigure (self.recList[self.activeBlock][0],\
                                    fill =self.recColorMoving)
        self.canvas.move(self.recList[self.activeBlock][0],\
                         event.x-self.oldx, event.y-self.oldy)
        self.canvas.move(self.recLabel[self.activeBlock],\
                         event.x-self.oldx, event.y-self.oldy)
        event.widget.bind('<ButtonRelease-1>', self.redrawing)
        self.oldx, self.oldy = event.x, event.y
        self.destroyConLines()


    ## Funtion for redrawing connecting lines to the proper corners ##
    def smartLine(self, bbox1, bbox2):

        #check if bbox1 is to the left or right of bbox2
        if(bbox1[0] <= bbox2[0]):
            #bbox1 left of bbox2
            lr = 1
        else:
            lr = 0

        if(bbox1[1] <= bbox2[1]):
            #bbox1 is above bbox2
            ud = 1
        else:
            ud = 0

        if(lr == 1 and ud == 1):
            #bbox1 is above and to the left of bbox2
            return (bbox1[2], bbox1[3]), (bbox2[0], bbox2[1])
        elif(lr == 1 and ud == 0):
            ##bbox1 is to the left and below bbox2
            return (bbox1[2], bbox1[1]), (bbox2[0],bbox2[3])
        elif(lr == 0 and ud == 1):
            #bbox1 is to the right and above bbox2
            return (bbox1[0],bbox1[3]), (bbox2[2],bbox2[1])
        else:
            #bbox1 is to the right and below bbox2
            return (bbox1[0], bbox1[1]), (bbox2[2],bbox2[3])


    ## Once the block stops moving from recDrag the position ##
    ## and lines need to be updated ##
    def redrawing(self, event):

```

```
        x,y = event.x, event.y
        event.widget.itemconfigure (self.recList[self.activeBlock][0],\
                                    fill =self.recColor)
        tbbox = event.widget.bbox(self.recList[self.activeBlock][0])
        self.bboxDic[self.recList[self.activeBlock][0]]= tbbox
        self.recList[self.activeBlock] = \
                        (self.recList[self.activeBlock][0],\
                         (tbbox[0],tbbox[1]))
        event.widget.unbind('<ButtonRelease-1>')

        self.redrawLine()

    def redrawLine(self):
        ## redraws all lines and stores the info for each ##
        for line in self.tempLineList:
            orgB = line[1]
            desB = line[2]
            start, end = self.smartLine(self.bboxDic[orgB],\
                                        self.bboxDic[desB])
            self.conLine(start,end,orgB, desB, line[3])


    ## Sets self.activeBlock to nearest block to the action done ##
    ## Possible actions are: mouse over, right click, left ##
    ##click, left double click, and mouse button 3 clicked ##
    def findClosest(self, x, y):

        for i, pos in self.recList:
            if(pos[0] <= x and (x <= (pos[0]+self.BoxW)) and \
               pos[1] <= y and (y <= (pos[1]+self.BoxH))):
                self.activeBlock = self.recList.index((i, pos))

            else:
                pass


    ## Function for left clicking on block ##
    ## sets self.activeBlock to get ready to drag ##
    def recLClick(self, event):
        self.oldx,self.oldy = event.x,event.y
        self.findClosest(event.x, event.y)


    ## makes the radiobutton for the Filtering ##
    def mkCombo(self):
        sel = ['S', 'P', 'O']
        self.varobj = StringVar()
        self.varsub = StringVar()
        self.varpre = StringVar()
        self.winCombo = Toplevel()
        self.winCombo.title("Filtering")
```

```
        self.winCombo.config(bg=self.bgColor)
        Label(self.winCombo, bg=self.bgColor,\
              text='Subject: ').grid(column=0, row=0)
        Label(self.winCombo, bg=self.bgColor,\
              text='Predicate: ').grid(column=0, row=1)
        Label(self.winCombo, bg=self.bgColor,\
              text='Object: ').grid(column=0, row=2)
        entryS = Entry(self.winCombo, width = 32,\
                       textvariable=self.varsub)
        entryP = Entry(self.winCombo, width = 32,\
                       textvariable=self.varpre)
        entryO = Entry(self.winCombo, width = 32,\
                       textvariable=self.varobj)
        entryS.grid(column=1, row=0)
        entryP.grid(column=1, row=1)
        entryO.grid(column=1, row=2)
        Button(self.winCombo, text="Update",bg=self.bgColor,\
               relief=RAISED, bd=4,\
               command=self.filterBlock).grid(column=0, row=3)


    ## concats the block filter strings to the first 9 characters ##
    def setBlockText(self, ts, tp, to, block):
        self.filterList[block] = [ts, tp, to]
        if len(ts) > 9:
            ts = ts[:8]
        if len(tp) > 9:
            tp = tp[:8]
        if len(to) > 9:
            to = to[:8]
        #self.filterList[block] = [ts, tp, to]
        if ts == '':
            s = "\nS: " + self.bld
        else:
            s = "\nS: "+ts

        if tp == '':
            p = "\nP: " + self.bld
        else:
            p = "\nP: " + tp

        if to == '':
            o = "\nO: " + self.bld
        else:
            o = "\nO: " + to
        rec = "\nID: " + str(block)
        self.canvas.itemconfigure(self.recLabel[block],\
                                  text = rec+s+p+o)
```

```
    ## parent is the parent of the curent block ##
    ## edgeT is the type of edge child->parent ##
    def updateBlockFilterData(self, parent, edgeT, actBlock):


        if edgeT == 'SS':
            if self.filterList[actBlock][0] != '':
                self.filterList[parent][0] = \
                                self.filterList[actBlock][0]

            else:
                pass
        elif edgeT == 'SP':
            if self.filterList[actBlock][0] != '':
                self.filterList[parent][1] = \
                                self.filterList[actBlock][0]

            else:
                pass
        elif edgeT == 'SO':

            if self.filterList[actBlock][0] != '':
                self.filterList[parent][2] = \
                                self.filterList[actBlock][0]

            else:
                pass
        elif edgeT == 'PS':
            if self.filterList[actBlock][1] != '':
                self.filterList[parent][0] = \
                                self.filterList[actBlock][1]

            else:
                pass
        elif edgeT == 'PP':
            if self.filterList[actBlock][1] != '':
                self.filterList[parent][1] = \
                                self.filterList[actBlock][1]

            else:
                pass
        elif edgeT == 'PO':
            if self.filterList[actBlock][1] != '':
                self.filterList[parent][2] = \
                                self.filterList[actBlock][1]

            else:
                pass
        elif edgeT == 'OS':
            if self.filterList[actBlock][2] != '':
                self.filterList[parent][0] = \
```

```
                                    self.filterList[actBlock][2]

            else:
                pass
        elif edgeT == 'OP':
            if self.filterList[actBlock][2] != '':
                self.filterList[parent][1] = \
                                    self.filterList[actBlock][2]

            else:
                pass
        elif edgeT == 'OO':
            if self.filterList[actBlock][2] != '':
                self.filterList[parent][2] = \
                                    self.filterList[actBlock][2]

            else:
                pass


        self.setBlockText(self.filterList[parent][0],\
                          self.filterList[parent][1],\
                          self.filterList[parent][2], parent)


    ## Fuction gets the data from the filter window.  Sets the block ##
    ## to the current filter ##
    ## Calls the depthff function to update the rest of the graph ##
    ## with this filter. ##
    def filterBlock(self):


        thisActiveB = self.activeBlock

        s = self.varsub.get()
        p = self.varpre.get()
        o = self.varobj.get()

        self.setBlockText(s, p, o, thisActiveB)
        self.depthff(0, self.recList[thisActiveB], thisActiveB)
        self.winCombo.destroy()


    ## checks if this triple has the filtered word at the correct loc ##
    ## word is the filter string
    ## loc is s, p, o location
    ## rdfT is the triple informatino
    def checkIn(self, word, loc, rdfT):

        if word == rdfT[loc]:
            #keep this triple
```

```
            return True
        else:
            #remove this triple
            return False

    ##takes the block data and filter for this block and returns ##
    ## the string of triples ##
    ## blkdata is the data from the self.dataDic for the selected block
    ## thisFilter is the current filter list for the selected block
    def displayFilter(self, blkdata, thisFilter):

        printStr =''
        for block in blkdata:

            for item in block:
                triple = self.rtb.tripleTable[item][1:]

                if thisFilter == ['','','']:
                    printStr = printStr + str(triple) + '\n'

                else:
                    #words would be the filter on s, p, then o
                    for word in thisFilter:
                        if word == '':
                            #there is a filter on this block do nothing
                            pass
                        else:
                            if word == triple[thisFilter.index(word)]:
                                #keep item
#                                print "saving triple:", triple

                                printStr = printStr + str(triple) + '\n'
                            else:
                                #do nothing on non matched words
                                pass
        return printStr


    #makes the radiobutton for selecting edge type
    def mkRadio(self):
        options = ['SS', 'SP', 'SO', 'PP', 'PS', 'PO', 'OO', 'OP', 'OS']
        self.var = StringVar()
        self.var.set('SS')
        self.winRad = Toplevel()
        self.winRad.title("Select Edge Type")
        self.winRad.config(bg=self.bgColor)
        Radiobutton(self.winRad, text = 'SS', value = 'SS',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'SP', value = 'SP',\
                    bg=self.bgColor, bd=7,\
```

```
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'SO', value = 'SO',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'PS', value = 'PS',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'PP', value = 'PP',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'PO', value = 'PO',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'OS', value = 'OS',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'OP', value = 'OP',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)
        Radiobutton(self.winRad, text = 'OO', value = 'OO',\
                    bg=self.bgColor, bd=7,\
                    variable=self.var).pack(side=LEFT)

    ## returned: stuff -> all partition blocks from KSA. ##
    ## returned: testing -> each triple from present KSA ##
    ## blocks that satisfies this edge ##
    def getStuff(self, edge, blockID):
        tList = []
        testing = []
        tList.append(edge)
        curPartition = self.dataDic[self.recList[blockID][0]]
        stuff = self.searchBlockList(curPartition, tList)
        return stuff



    ## if a selection has been made after clicking on the OK##
    ## button on the menu ##
    ## The data associated with the active block ##
    ## This data is sent to searchBlockList which returns a ##
    ## list of blocks with the selected edge type (tList) ##
    def getState(self):
        tmp = self.getRadSelect()
        if tmp is not '':
            stuff = self.getStuff(tmp, self.activeBlock)
            if stuff != []:
                self.driver([], "new", tmp)
                self.depthff(1, self.recList[-1], self.activeBlock)

            self.winRad.destroy()

```

```
    ## returns radio button selection on the select edge type ##
    ## radiobutton menu ##
    def getRadSelect(self):
        return self.var.get()


    ## Searches through the partition associated with this ##
    ## partition for a certain edge type ##
    ## patition is
    ## edgeT is the edge type (1 of nine possible ss, sp, so, ect)
    ## returns list of partion blocks ##
    def searchBlockList(self, partition, edgeT):
        tlist = []
        for block in partition:
            for et in edgeT:

                temp = self.rtb.findEdge(et, self.rtb.edgeNode, block )
                #Do not add the same partition block more then once
                if temp != [] and temp not in tlist:
                    tlist.append(temp)
        return tlist


    ## Creats the generic button window for choosing which ##
    ## of the 4 methods to run ##
    def recRC(self, event):
        self.findClosest(event.x, event.y)
        self.rcVar = StringVar()
        self.rcVar.set('s')
        self.rcWin = Toplevel()
        self.rcWin.title("Methods")
        self.rcWin.config(bg=self.bgColor)
        Radiobutton(self.rcWin, text='Select Edge KSA', value = 's',\
                    bg=self.bgColor, bd=4, variable=self.rcVar,\
                    width = 25, relief=SUNKEN,\
                    indicatoron=0).pack(side=TOP)
        Radiobutton(self.rcWin, text='Filtering by SPO', value = 'f',\
                    bg=self.bgColor, bd=4, variable=self.rcVar,\
                    width = 25, relief=SUNKEN,\
                    indicatoron=0).pack(side=TOP)
        Radiobutton(self.rcWin, text='Destroy Filtering', value = 'k',\
                    bg=self.bgColor, bd=4, variable=self.rcVar,\
                    width = 25, relief=SUNKEN,\
                    indicatoron=0).pack(side=TOP)
        Radiobutton(self.rcWin, text='Display Data', value = 'd',\
                    bg=self.bgColor, bd=4, variable=self.rcVar,\
                    width = 25, relief=SUNKEN,\
                    indicatoron=0).pack(side=TOP)
        Button(self.rcWin, text="GO",bg=self.bgColor, relief=RAISED,\
               width = 25, bd=4,\
```

```
                command=lambda: self.rcGet(event)).pack(side=BOTTOM)


    ## Used to get the selection from the method window ##
    def rcGet(self, event):
        method = self.rcVar.get()
        # Make new Edge #
        if method == 's':
            self.ksaEdge(event)
        # Display Data #
        elif method == 'd':
            self.printtext(event)
        # Open filtering Window #
        elif method == 'f':
            self.tStr = ''
            self.mkCombo()
        # Destory filtering #
        elif method == 'k':
            self.destroyFilter()
        else:
            pass
        self.rcWin.destroy()


    def destroyFilter(self):

        for block in self.recList:
            self.setBlockText('','','',self.recList.index(block))

    ## event handler for left clicking on box. ##
    ##Filtering selection ##
    def recLDClick(self, event):
        self.tStr =''
        self.mkCombo()


    ## event handler for right clicking on box. ##
    ## KSA on selected edge type ##
    def ksaEdge(self, event):
        self.recListName.append(self.recCount)
        self.recCount += 1
        #self.findClosest(event.x, event.y)
        self.mkRadio()
        selItem = ''
        selItem = Button(self.winRad, bg=self.bgColor,\
                         width=4,text = 'GO', relief=RAISED,bd=7,\
                         command = self.getState).pack(side=BOTTOM)


    ## Destoys all lines when rectangle begins to be dragged.##
    ## Will be redrawn on release ##
```

```
    def destroyConLines(self):

        for line in self.recLineList:
            self.canvas.delete(line[0])
            self.canvas.delete(line[4])
            #add in destroy label id as well
            #add in label id to recLineList

        if self.recLineList != []:
            self.tempLineList = self.recLineList[:]
            self.recLineList = []


    #### Creates a line from orig block to dest block ####
    def conLine(self, origTup, destTup, origBlock, destBlock, edge):

        self.recLineList.append([self.canvas.create_line(origTup[0],\
                                                         origTup[1],\
                                                         destTup[0],\
                                                         destTup[1],\
                                                         smooth='true',\
                                                         width=1,\
                                                         arrow=FIRST),\
                                 origBlock, destBlock, edge,
                                 self.canvas.create_text(\
                                     ((origTup[0] + destTup[0])/2),\
                                     ((origTup[1] + destTup[1])/2),\
                                     text = str(edge),\
                                     fill= self.labelTextColor)])


    def driver(self, data, ty, edge):
        if ty is "start":
            self.makeRect(20, 20)
                                          #[list(range(len(data)))]
            self.dataDic[self.recList[0][0]] = self.KSpar

        else:
            self.makeRect(40, 160)
            self.dataDic[self.recList[-1][0]] = data
            B1 = self.bboxDic[self.recList[self.activeBlock][0]]
            B2 = self.bboxDic[self.recList[-1][0]]
            start, end = self.smartLine(B1, B2)
            self.conLine(start, end, self.recList[self.activeBlock][0],\
                         self.recList[-1][0], edge)

    def idToindex(self, bid):

        for j in self.recList:
            if bid in j:
                return self.recList.index(j)
```

```
    ## returns the parnet and edge type between them ##
    def findParent(self, child):

        for line in self.recLineList:
            if child in line:
                ind = line.index(child)
                #index 2 is that of child that has parent at index 1
                if ind == 2:
                    return line[1], line[3]
                else:
                    pass
            else:
                pass
        return None, ''


    ## returns the child and edge type between them ##
    def findChild(self, parent, cl):
        for line in self.recLineList:
            if parent in line:
                ind = line.index(parent)
                #index 1 is that of the Parent so child is at index 2
                if ind == 1 and [line[2], line[3]] not in cl:
                    return line[2], line[3]
                else:
                    pass
            else:
                pass
        return None, ''


    ## flag indicates whether a new block was created flag == 1 ##
    ## or if its a filter fix flag == 0 ##
    ## node is the type: self.recList item ##
    def depthff(self, flag, node, activeB):

        N = node[0]

        #phase 1
        #while N has parent P
        while N != None:
            P = N
            N, eg = self.findParent(N)
            if eg != '':
                teg = eg[1]+eg[0]
                curParent = self.idToindex(N)
                if flag == 1:
                    s = self.getStuff(teg, curParent)
                else:
                    self.updateBlockFilterData(curParent, eg, activeB)
```

```
                    activeB = curParent
                    s = ''

                oldData = self.dataDic[self.recList[curParent][0]]

                if s != '':
                    for i in s:
                        if i not in oldData:
                            s.remove(i)

                    self.dataDic[self.recList[curParent][0]] = s


        #phase 2
        fixupStack = []
        fixupStack.append(P)
        childList =[]
        while fixupStack != []:
            item = fixupStack.pop()

            #generate chidren
            while 1:
                child, eg = self.findChild(item, childList)
                if child != None and [child, eg] not in childList:
                    childList.append([child, eg])
                else:
                    break
            parBlk = self.idToindex(item)

            #update children
            for j in childList:
                chiBlk = self.idToindex(j[0])
                edge = j[1]
                if flag == 1:
                    pi = self.dataDic[self.recList[parBlk][0]]
                    nData = self.searchBlockList(pi, [j[1]])
                    self.dataDic[self.recList[chiBlk][0]] = nData

                #filter update

                self.updateBlockFilterData(chiBlk,\
                                           edge[1]+edge[0], parBlk)
            #push (append) childeren
            for i in childList:
                fixupStack.append(i[0])
            childList = []

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
##################### RDF Table Builder ###########################
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
```

```
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

## Class for doing all of the KSA process ##
class RdfTableBuilder:

    longestTriple = 30
    tripleID = 1
    blockID = 1


    #list holding the [target, type] relationships and the index
    #of the list is the source node
    edgeNode = []

    #list for holding block edge information.
    blockEdge = []
    tripleTable = []
    tempTripleTable = []
    blockHashList = []

    #dictionary for holding the atom and the locations of that
    #atom and what its relationship is
    atomList = {}
    blockDict = {}
    tripleBlock = {}

    def __init__(self, gui, fileName):

        try:
            fileOpen = open(fileName, 'r')
            #self.__readFile(fileOpen)
            self.__NTripleParse(fileOpen)
        except IOError:
            print "Could not open file:", fileName
            gui.fileD(1)
          # sys.exit()

    def clearRDFData(self):

        self.longestTriple = 30
        self.tripleID = 1
        self.blockID = 1
        del self.edgeNode[:]
        del self.blockEdge[:]
        del self.tripleTable[:]
        del self.blockHashList[:]

        self.blockDict.clear()
        self.atomList.clear()
        self.tripleBlock.clear()

```

```
    def getID(self, ID, flag):
        iter1 = self.tripleBlock.iteritems()
        found = 0

        for k, val in iter1:
            if(flag == 0): #searching for BID from a given TID
                if val == ID:
                    found = 1
            else:
                if k == ID:
                    found = 1
        if found == 0:
            pass

    def setBlockID(self, tripID, newID):
        iter2 = self.tripleBlock.iteritems()
        found = 0

        for k, val in iter2:
            if(k == tripID):
                found = 1
                self.tripleBlock[k] = newID
                return 1
        if found == 0:
            return 0


    def __addTripleList(self, s, p, o):

        thisTriple = []
        thisTriple.insert(0, 1)
        thisTriple.insert(1, s)
        thisTriple.insert(2, p)
        thisTriple.insert(3, o)
        self.tripleBlock[self.tripleID] = self.blockID
        self.tripleTable.append(thisTriple)
        self.tripleID +=1
        #gets the edgeNode ready to accept information for this triple
        self.edgeNode.append([1])


    #method for adding atoms to the atom list
    def __addAtom(self, tID, atom, spo):
        newList = []
        if spo == 0:
            char = 'S'
        elif spo == 1:
            char = 'P'
        else:
            char = 'O'
```

```
        newList.insert(0, tID)
        newList.insert(1, char)

        if atom in self.atomList:
            self.atomList[atom].append(newList)

        else:
              self.atomList[atom] = [newList]



    def __addEdge(self, source, target, edgeType):
        #checks if this index is still in the initialized state or not
        if(self.edgeNode[source] == [1]):
            tempList = [[target, edgeType]]
        else:
            tempList = self.edgeNode[source]

        #check if this edge already in the list.
            if([target, edgeType] in tempList):
                #do not add it again
                pass
            else:
                tempList.append([target, edgeType])
        self.edgeNode[source] = tempList


    #sourceIndex is the index of the cure
    def __matchUp(self, sourceIndex, triple):

        spoIndex = 0
        for item in triple:

            if self.atomList.has_key(item):

                #for each item in the triple, the edge table is updated
                itemVal = self.atomList[item]

                for q in itemVal:

                    edgeTypeForward = 'A'
                    edgeTypeBack = 'A'

                    if(spoIndex == 0):
                        edgeTypeForward = 'S' + q[1]
                        edgeTypeBack = q[1] + 'S'

                    elif(spoIndex == 1):
                        edgeTypeForward = 'P' + q[1]
                        edgeTypeBack = q[1] + 'P'
```

```

                    elif(spoIndex == 2):
                        edgeTypeForward = 'O' + q[1]
                        edgeTypeBack = q[1] + 'O'

                    else:
                        #needs to break here for illegal
                            pass

                    #edge type was found above
                    if(edgeTypeForward != 'A'):
                        #adds edge to the list (source, target)
                        self.__addEdge(sourceIndex, q[0],\
                                       edgeTypeForward)
                        #adds the other direction target, source)
                        self.__addEdge(q[0], sourceIndex,\
                                       edgeTypeBack)
            else:
                pass

            spoIndex += 1



    def __clean(self, info):
        info = info.lstrip(' ')
        return info.rstrip(' ')

    def __longest(self, new):
        if new > self.longestTriple:
            self.longestTriple = new+25


    def __NTripleParse(self, fileIn):
        tCount = 0
        line = fileIn.readline()
        while line:
            #print "line:", line
            numbNodes = len(self.tripleTable)

            if line.isspace():
                #print "line full of spaces"
                pass
            else:
                line = line.lstrip(' ')
                line = line.rstrip(' \n')
                if line.startswith('#'):
                    #print "line starts with #"
                    pass
                elif line.endswith('.'):
                    #print "line passed test"
```

```
                    templength = len(line)
                    self.__longest(templength)

                    line = line[:templength-1]


                    newString = line.split(None,2)
                    #print "newString\n", newString
                    for i in newString:
                        newString[newString.index(i)] = self.__clean(i)

                    if [1, newString[0], newString[1],\
                        newString[2]] in self.tripleTable:
                        pass
                    else:
                        self.__addTripleList(newString[0],\
                                             newString[1],\
                                             newString[2])
                        k = 0
                        #add in each atom from the new triple list
                        while k < 3:
                            self.__addAtom(numbNodes, newString[k], k)
                            k += 1
                        #after each item is added to the triple list
                        #add this triple in to making any
                        #new edges that need to be created
                        self.__matchUp( tCount, [newString[0],\
                                                 newString[1],\
                                                 newString[2]])
                        tCount += 1


                else:
                    #print "line didn not end in . or start with #"
                    pass


            line = fileIn.readline()
        fileIn.close()

    #method for reading the triple data in from the file
    def __readFile(self, fileIn):
        line = fileIn.readline()
        tCount = 0
        while line:
            #create the s p o table here with tripleID and blockID

            newString = line.split('"')
            numbNodes = len(self.tripleTable)
        #check here if item is already in list
```

```
            if [1, newString[1],\
                newString[3],\
                newString[5]] in self.tripleTable:
                pass
            else:
                               #s           p             o
                self.__addTripleList(newString[1],newString[3],\
                                     newString[5])
                k = 0
                #add in each atom from the new triple list
                while k < 3:
                    self.__addAtom(numbNodes, newString[(k*2)+1], k)
                    k += 1
                #after each item is added to the triple list add this
                #triple in to making any new edges that
                #need to be created
                self.__matchUp( tCount, [newString[1],\
                                         newString[3],\
                                         newString[5]])
                tCount += 1
            line = fileIn.readline()

        fileIn.close()




    def findEdge(self, edgeT, allEdges, ss):

        count = 0
        c = []
        for i in allEdges:
            for j in i:

                #the last condition is for not adding the edges to itself
                if edgeT == j[1] and count != j[0] and count not in c:
                    if j[0] in ss:
                        c.append(count)

            count += 1
        return c


    tt =[]

    def fastFilter(self, sub, pre, obj, data):

        tempTable = []
        nameList = []
        tripleList = []
```

```
        nameList.append(sub)
        nameList.append(pre)
        nameList.append(obj)
        t = ['S', 'P', 'O']
        count = 0

        for name in nameList:
            if name == "":
                pass
            else:
                if self.atomList.has_key(name):
                    for i in self.atomList[name]:
                        if i[1] == t[count]:
                            tripleList.append(i[0])
                else:
                    pass
            count += 1

        for t in tripleList:
            for block in data:
                if t in block:
                    tempTable.append(self.tripleTable[t])
                    break
                else:
                    pass
        return tempTable[:]



    #same inputs as the orginal KS but also a function for doing
    #what the user wants (currently not used)
    ## setTriples is the tripleTable (self.tripleTable) for this file ##
    ## setEdges is the set of edges (self.edgeNode) for this file ##
    ## theGraph list of all triple IDs
    ## ie [0, 1, 2, ..., len(self.tripleTable)]##
    ## edteType list of the edge type ['SS', 'SP', ..., 'OO'] that the ##
    ## user is asking for ##
    ## returns the created Partition list and a listing of the newest ##
    ## added block (list[Block2]) ##
    def userDefKSBiSim(self, setTriples, setEdges, theGraph, edgeType):

        block2 = ''
        P = range(len(setTriples))
        P.sort()

        self.blockDict[min(P)] = P[1:]
        self.blockHashList.append(min(P))
        P = [P]

        spliterSet = P[:]
        sSet = set([])
```

```
        count = 0
        while spliterSet != []:
            S = spliterSet[count % len(spliterSet)]
            spliterSet.remove(S)
            for l_type in edgeType:
                #function call here to get C
                C = self.findEdge(l_type, self.edgeNode, S)
                if C == []:
                    pass
                #need to add in the new items to the tables
                elif C[0] not in self.blockHashList:

                    self.blockHashList.append(C[0])
                else:
                    self.blockDict[C[0]] = C[1:]

                if C != []:
                    for block in P:

                        bSet = set(block)
                        cSet = set(C)
                        interBC = cSet.intersection(bSet)

                        if ( interBC != set([]) ) and (interBC != bSet):

                            block2 = bSet - interBC
                            P, spliterSet = \
                                self.cleanPartition(P, list(spliterSet),\
                                                    list(block),\
                                                    list(interBC),\
                                                    list(block2))

                        else:
                            pass
                else:
                    pass
                count += 1

        return P, list(block2)



    #Kanellakis-Smolka algorithm
    ## setTriples is the tripleTable (self.tripleTable) for this file ##
    ## setEdges is the set of edges (self.edgeNode) for this file ##
    ## theGraph list of all triple IDs
    ## ie [0, 1, 2, ..., len(self.tripleTable)]##
    ## returns the partition P
    def KSBiSim(self, setTriples, setEdges, theGraph):

        edgeTypeList = ['SS', 'SO', 'SP',\
```

```
                        'OS', 'OO', 'OP',\
                        'PS', 'PO', 'PP']

        P = range(len(setTriples))
        P.sort()
        maxLen = len(P)

        self.blockDict[min(P)] = P[1:]
        self.blockHashList.append(min(P))

        P = [P]
        spliterSet = P[:]
        sSet = set([])
        count = 0

        while spliterSet != [] and len(P) != maxLen:
            S = spliterSet[count % len(spliterSet)]
            spliterSet.remove(S)

            for l_type in edgeTypeList:
                #function call here to get C
                C = self.findEdge(l_type, self.edgeNode, S)
                if C == []:
                    pass
                #need to add in the new items to the tables
                elif C[0] not in self.blockHashList:
                    self.blockDict[C[0]] = C[1:]
                    self.blockHashList.append(C[0])

                else:
                    self.blockDict[C[0]] = C[1:]

                if C != []:
                    for block in P:
                        if maxLen == len(P):
                            break
                        bSet = set(block)
                        cSet = set(C)
                        interBC = cSet.intersection(bSet)

                        if  interBC != set([])  and interBC != bSet:

                            block2 = bSet - interBC
                            P, spliterSet = \
                               self.cleanPartition(P, list(spliterSet),\
                                                   list(block),\
                                                   list(interBC),\
                                                   list(block2))
                        else:
                            pass
                else:
```

```
                    pass
                count += 1

        return P

    def cleanPartition(self, part, ss, blk, intBC, b2):

        part.remove(blk)
        part.append(intBC)
        part.append(b2)

        if blk in ss:
            ss.remove(blk)
        ss.append(intBC)
        ss.append(b2)
        return part, ss

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
############################ Main #################################
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
if __name__=="__main__":

#    if len(sys.argv) != 2:
#        print __doc__
#    else:
#        rdf = RdfTableBuilder(sys.argv[1])
#        i = 0

    root = Tk()
    root.tk.call('tk', 'scaling', 1)
    root.tk.call('package', 'require', 'tile')
    root.tk.call('namespace', 'import', '-force', 'ttk::*')
    root.tk.call('ttk::setTheme', 'alt')

    if len(sys.argv)!=2:
        rt = rdfGui(root, '')
    else:
        rt = rdfGui(root, sys.argv[1])
        rt.driver('', "start", '')

    root.mainloop()
```