ARMY RESEARCH LABORATORY

A Visual Analytic for High-Dimensional Data Exploitation: The Heterogeneous Data-Reduction Proximity Tool

by Timothy Hanratty and John Richardson

ARL-TR-6502 **July 2013**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5067

ARL-TR-6502 **July 2013**

A Visual Analytic for High-Dimensional Data Exploitation: The Heterogeneous Data-Reduction Proximity Tool

Timothy Hanratty and John Richardson
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
July 2013	Final	November 2011–September 2012

4. TITLE AND SUBTITLE
A Visual Analytic for High-Dimensional Data Exploitation: The Heterogeneous Data-Reduction Proximity Tool

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Timothy Hanratty and John Richardson

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: RDRL-CII-C
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-6502

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Visual analytics is a growing area of research within the Department of Defense that targets the effective interleaving of analytical reasoning with interactive interfaces. Capitalizing on the human capacity for spatial reasoning, visual analytics enhance the decision maker's understanding of the underlying decision space by augmenting the assimilation of complex relationships. The Tactical Information Fusion Branch has developed a visual analytic application, entitled the Heterogeneous Data-Reduction Proximity Tool (HDPT). The goal of the HDPT is to complement traditional social network analysis by allowing the exploitation of a social network based upon the calculated similarity of the individuals (against a known reference set) as opposed to the traditional organizational structure. The result is an improved understanding of the human terrain that effectively incorporates qualitative and quantitative information into the decision making process.

15. SUBJECT TERMS
visual analytics, data reduction, multidimensional scaling, Gower

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Timothy Hanratty
Unclassified	Unclassified	Unclassified	UU	80	19b. TELEPHONE NUMBER (*Include area code*) 410-278-3084

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

The authors wish to thank Mr. Mark Thomas, branch chief of the Tactical Information Fusion Branch, for his guidance. The program is improved as a result.

1. Introduction

The mission of the Tactical Information Fusion Branch is the research and development of advanced information analytics to assist Soldiers in determining, using, and sharing relevant information and improving the synthesis of data for decisions. Challenging these efforts is not only by the unprecedented increase in the types and amount of information available, but the human judgment necessary to evaluate them in the presence of incomplete and inconsistent information within time-critical environments (*1, 2*). Innovated methods are required that allow the efficient and effective transformation of data from information to knowledge. One process that will tackle this challenge is visual analytics.

Visual analytics is a growing area of research within the Department of Defense (DOD) that targets the effective interleaving of analytical reasoning with interactive interfaces. Capitalizing on the human capacity for spatial reasoning, visual analytics enhance the decision maker's understanding of the underlying decision space by augmenting the assimilation of complex relationships (*3, 4*). Visual analytics is a multidisciplinary field that has yielded significant results in an array of paradigms including business, medicine, and defense (*1, 5*).

The Tactical Information Fusion Branch has developed a visual analytic application, entitled the Heterogeneous Data-Reduction Proximity Tool (HDPT). The goal of the HDPT is to complement traditional social network analysis by allowing the exploitation of a social network based upon the calculated similarity of the individuals (against a known reference set) as opposed to the traditional organizational structure. The result is an improved understanding of the human terrain that effectively incorporates qualitative and quantitative information into the decision making process.

This report documents the design and development of the HDPT visual analytic application. The outline of the remainder of the report is as follows. In section 2, background information on the statistical approach undertaken and the rational for choosing the particular proximity calculation is provided. In section 3, the system-level design and instantiation of the HDPT as a web application linked to the Distributed Common Ground Systems-Army (DSGS-A) is discussed. The report concludes with lessons learned and the way forward in section 4.

2. Background

For many military applications, extracting information from high-dimensional data sets is a persistent and complicated task. This is especially true when the data sets are of mixed-data type, wherein the attributes defining the objects to be compared take on values from differing

measurement scales. Moreover, the data of interest are typically amorphous; that is, not linked to an explicit theory to assist the researcher in making inferences or predicting structure. For this reason, multidimensional scaling (MDS) was chosen as the specific candidate for visualizing the structure of the data and Gower's similarity coefficient as the algorithm for calculating the proximity matrices. The following section provides a background on both MDS and Gower's coefficient.

2.1 Multidimensional Scaling

Originating out of the fields of mathematical psychology and social sciences, MDS is a data analysis approach used to visually ascertain the similarity or dissimilarity between the pairwise "distances" among a given set of objects (*6–8*). The values of the distances**,** sometimes called proximity measures or similarity measures, can be obtained either as perceived subjective measures or calculated objectively within the pairwise comparison of the given set of objects. Most often, the objects are vectors of the form X = $(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ with the components $\mathbf{x}_k$ known as attributes, variables, or factors collectively providing the basis for comparison of the objects. Given a similarity matrix for a set of objects, each object is projected as a point in *n* space; arranged so the distances between the objects have the strongest possible relation to the similarity matrix. The intrinsic power of MDS is that it allows the dimensionality reduction of a complex *n* space to a human interpretable two- or three-dimensional space. It is this projection that promotes the exploratory analysis of the data's hidden structure.

Traditionally, MDS is divided into two major categories: metric and nonmetric. Regarding metric MDS, the measures within the similarity matrix are true mathematical distances and satisfy the four axioms of the metric definition (*9*). Given that $\delta(X,Y)$ is the distance function and X, Y, and Z are objects within the similarity matrix:

(1) $\delta(X,X) = 0$ (identity)

(2) $\delta(X,Y) \geq 0$ for all X,Y (nonnegative)

(3) $\delta(X,Y) = \delta(Y,X)$ for all X,Y (symmetry)

(4) $\delta(X,Y) \leq \delta(X,Z) + \delta(Z,Y)$ for all X,Y,Z (triangle inequality)

In order to handle the many applications in which the similarity measures do not satisfy the constraints of being a metric, Shepard (*10*) and Kruskal (*11*) developed a method known as nonmetric MDS. Nonmetric MDS relaxes the space to preserve ordinal (ranked) data.

In addition to the metric versus nonmetric characteristic, the other key differentiating the MDS characteristic is in the definition of the distance measure that is selected. Different distance measures allow different calculations and projections of the original objects, potentially expanding the exploratory nature of MDS. For example, selecting the typical Euclidean distance would result in the common "as-the-crow-flies" straight line distance calculation between two

points. On the other hand, selecting the city block distance would result in the calculations being based on the sum of the perimeter distances between the two points, or "walking city blocks." Interestingly, the projection of a Euclidean measure between two points results in a circle of possible candidate positions, whereas the projection of a city block measure between two points results in a rectangle of possible positions. A third, less common, distance measure is called the supremum, in which the distance between two points is reduced to the maximum perimeter distance between two points. Figure 1 depicts an example of three distance measures: Euclidean, City Block, and Supremum.

The Minkowski metric shown in equation 1 is the most commonly used function from which most other metrics (distances) are derived: for r=1, equation 1 becomes the city block distance measure; for r=2, equation 1 becomes the Euclidean distance measure; and as r↑∞, equation 1 becomes the supremum distance measure (*7*, *8*):

Minkowski metric: $$\delta(X,Y) = \left(\sum_{i=1}^{d} \left| x_i - y_i \right|^r \right)^{1/r}, \quad (1)$$

where r≥1

City block distance: $$\delta(X,Y) = \sum_{i=1}^{d} \left| x_i - y_i \right|, \quad (2)$$

Euclidean distance: $$\delta(X,Y) = \left(\sum_{i=1}^{d} \left| x_i - y_i \right|^2 \right)^{1/2}, \quad (3)$$

Supremum distance: $$\delta(X,Y) = \max_d \left| x_{id} - y_{dk} \right|. \quad (4)$$

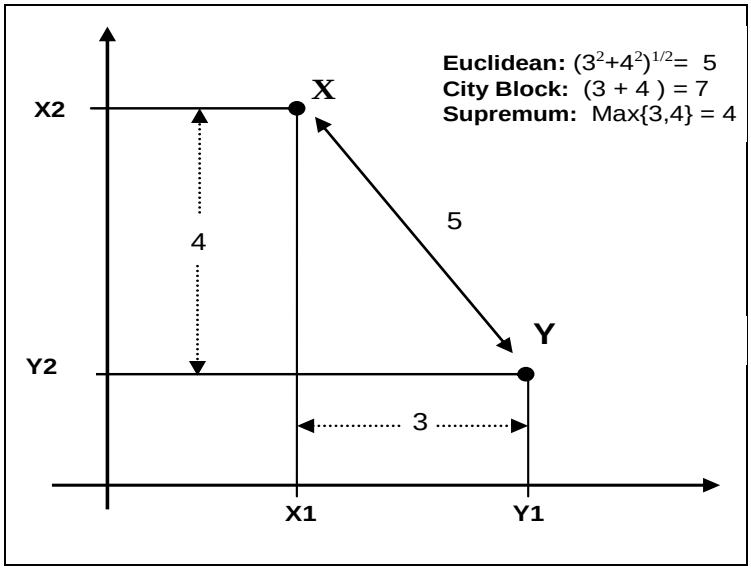

Figure 1. Minkowski metrics.

The hypotenuse of the right triangle corresponds to the Euclidean distance, the height of the triangle, the supremum distance, and the sum of the height plus the base, which is the city block distance. The essential concept is that a difference (distance) may be expressed in a variety of ways, depending on the specific metric invoked.

To measure how effective MDS reduces the input similarity matrix to its projection space, MDS calculates a Stress or "badness of fit" function. In general terms, the Stress function is the summation of the differences between the interobject distance measurements found in the input similarity matrix (D_{ij}) and their associated projected distances (d_{ij}). The goal is to minimize the Stress function, keeping the difference between $D_{ij} \approx d_{ij}$ as close to zero as possible. One of the more popular Stress functions is offered by Kruskal (*11*) and seen in equation 5:

$$\text{Stress}_1 = \left(\frac{\sum_{i}^{j} (D_{ij} - d_{ij})^2}{\sum_{i}^{j} Dij^2} \right)^{\frac{1}{2}}. \quad (5)$$

Various rules of thumb exist on how best to interpret the stress score; in general the lower the score the better. Another useful metric in determining the fit of an MDS projection is the Shepard plot (*12*). With a Shepard plot, the values of the given interobject distance measurements found in the input similarity matrix (D_{ij}) are plotted on the *x* axis against their associated projected distances (d_{ij}) on the *y* axis. A good fit will cluster about a 45° line if using metric MDS; otherwise, it will cluster about the central monotone line if using the nonmetric MDS analysis, and it has few outliers (*12*).

Differing from other forms of multivariate statistics, specifically principal component analysis (PCA), MDS does not constrain the data to be normally distributed. With that understanding, it becomes apparent that the hidden power behind meaningful MDS analysis is found in the construction of the similarity matrix and its projection into the reduced space.

Table 1 uses the mileage between 10 American cities as the objective similarity measure (*7*).

Table 1. Between-city mileage similarity matirix (*7*).

	ATL	CHI	DEN	HOU	LA	MIA	NY	SF	SEA	DC
ATL		587	1212	701	1936	604	748	2139	2183	543
CHI	587	0	920	940	1745	1188	713	1858	1732	597
DEN	1212	920	0	879	831	1726	1631	949	1028	1494
HOU	701	940	879	0	1374	968	1420	1645	1898	1220
LA	1936	1745	831	1374	0	2339	2451	347	959	2300
MIA	604	1188	1726	968	2339	0	1092	2594	2734	923
NY	748	713	1631	1420	2451	1092	0	2571	2408	205
SF	2139	1858	949	1645	347	2594	2571	0	678	2442
SEA	2182	1732	1021	1891	959	2734	2408	678	0	2329
DC	543	597	1494	1220	2300	923	205	2442	2329	0

The associated MDS 2-D visualization output would appear something like that found in figure 2. Note the geometric model allows one to discern the underlying structure and allow human interpretation.

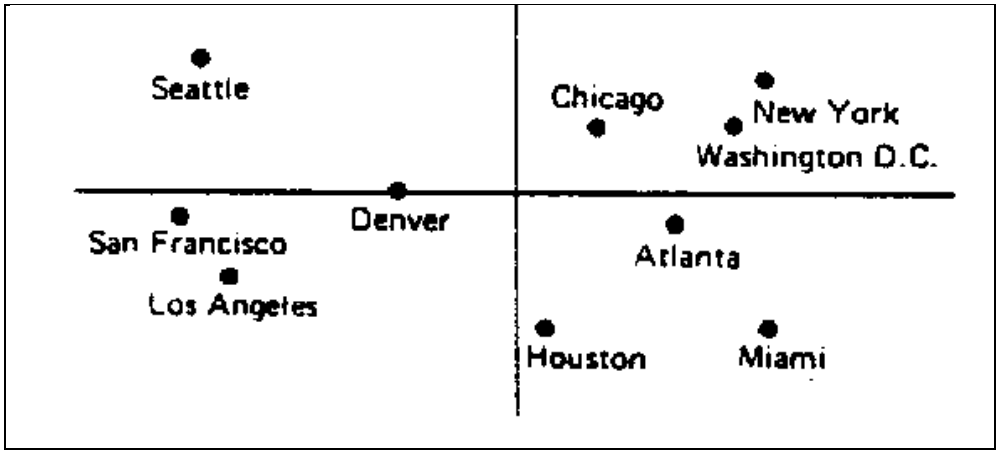

Figure 2. MDS cities visualization (*7*).

2.2 Gower's Similarity Coefficient

For calculated similarities, ideally, all of the defining attributes should be the same data type (*12*). Unfortunately, for many real-world problems, disparate scales of measure are common place and the calculation is problematic. According to Stevens (*13*), there are four scales or levels of measurement: nominal, ordinal, interval, and ratio. The lowest measurement level is the nominal scale. Nominal data are used to categorize data by name (e.g., morning, afternoon, and evening). The next higher measurement level is ordinal. With ordinal data, the numbers assigned represent an order or ranking (e.g., first, second, and third). The interval and ratio measurement levels represent the last two, higher level scales of measurement. For many MDS software

packages, the answer to the mixed scales of measure challenge is to brute force the data with *up-conversions* or *down-conversions*. Up-conversions treat lower level data as if they are higher level (e.g., nominal data represented by a number become actual interval values), whereas down-conversions work the opposite fashion (e.g., interval data can become ordinal). An obviously improved method for handling mix scales of measures is required.

Gower (*14*) was one of the first to confront the combination of quantitative and qualitative (mixed scales of measure). Given an array of objects with *k* attributes, the global similarity value ($\mathbf{S}_{ij}$) between two objects is defined as the summation of the individual attribute similarities ($\mathbf{s}_{ijk}$) multiplied by a possible weighting factor. Here, $\mathbf{s}_{ijk}$ corresponds to the measure of local similarities assigned to the object pair ($\mathbf{x}_i$,$\mathbf{x}_j$), restricted to attribute *k.* The summation of the individual similarities is divided by the summation across all weights. Gower's metric allows for the weighing of individual attributes and the possibility of missing data:

$$S_{ij} = \frac{\sum_{1}^{k} s_{ijk} w_k}{\sum_{1}^{k} w_k}, \tag{6}$$

where w_k is the weight assigned to an individual attribute category.

For situations in which the individual similarity cannot be computed for an attribute *k*, set w_k = 0, removing it from contributing to the numerator and demoninator calculation.

The calculation for the individual similarities is given below, where x_{ik} and x_{jk} are the k^{th} attribute for objects x_i and x_j, respectively. R_k is defined as the range for that particular quantitative attribute:

$$s_{ijk} = \begin{cases} \begin{cases} 1, & \text{if } \mathrm{X_{ik}} = \mathrm{X_{jk}} \\ 0, & \text{if } \mathrm{X_{ik}} \neq \mathrm{X_{jk}} \end{cases} & \text{, k is qualitative (nominal or binary)} \\ 1 - \dfrac{\left|\mathrm{X_{ik}} - x_{jk}\right|}{R_k} & \text{, k is quantitative (interval or ratio)} \end{cases}$$

In recent years, numerous extensions for similarity measurement calculations have been attempted in a wide array of subject areas from image processing to medical informatics. Approaches taken include, but are not limited to, rough sets (*15*), fuzzy logic (*16*), and ordinal extensions (*17*).

Of particular interest for this effort is the works by Wang (*18*). Wang (*18*) offers a computationally straightforward method for calculation fuzzy similarities between two fuzzy sets. The equations follow, where $\mu_A(x)$ and $\mu_B(x)$ are the associated fuzzy functions:

$$S_{ijk} = \begin{cases} \begin{cases} 1, & \text{if } X_{ik} = X_{jk} \\ 0, & \text{if } X_{ik} \neq X_{jk} \end{cases} & \text{, k is qualitative (nominal or binary)} \\ 1 - \dfrac{\left| x_{ik} - x_{jk} \right|}{R_k} & \text{, k is quantitative (interval or ratio)} \\ fuzzy\_sim(X_{ik}, X_{jk}) & \text{, k is fuzzy value} \end{cases} \tag{7}$$

where fuzzy_sim is defined as

$$\text{Fuzzy Similarity}(A,B) = \begin{cases} \dfrac{\sum_{1}^{n} \left[1 - \left| \mu_A(x_i) - \mu_B(x_i) \right| \right]}{n} & \text{, fuzzy functions and fuzzy element} \end{cases} \tag{8}$$

2.3 An Applied Example

The use of Gower's general coefficient is illustrated as a representative sample of terrorist activity recorded in table 2. The 11 events detailed are characterized by the five features (attributes) listed in the first row, the assigned weights follow in row 2, and the scales of measurement follow in row 3.

Table 2. A representative set of terrorist event data.

Attribute	Day	Location	Time	Prim/Attack	Sec/Attack
Weight	1	1	1	1	1
Scale	Nominal	Nominal	Interval	Nominal	Binary
Event 1	Saturday	Alpha Sector	18:00	SAF	NO
Event 2	Wednesday	Charlie Sector	12:00	IED	YES
Event 3	Saturday	Alpha Sector	19:00	SAF	NO
Event 4	Saturday	Bravo Sector	19:00	SAF	NO
Event 5	Wednesday	Charlie Sector	10:00	IED	YES
Event 6	Saturday	Bravo Sector	18:00	SAF	NO
Event 7	Wednesday	Charlie Sector	11:00	IED	YES
Event 8	Tuesday	Echo Sector	19:00	VBIED	NO
Event 9	Wednesday	Delta Sector	11:00	IED	YES
Event 10	Thursday	Foxtrot Sector	10:00	VBIED	YES
Event 11	Sunday	Delta Sector	20:00	VBIED	YES

Days of the week are not listed chronologically, so that an ordinal relationship is not in effect, and they are treated as nominal as are the location sectors. The times of attack are recorded on a 24-h clock, providing interval/ratio scale measures. The primary attack modes are as follows: small arms fire (SAF), improvised explosive device (IED), and vehicle borne improvised explosive device (VBIED). These modes are multilevel nominal, and secondary attack modes are treated as binary.

An assessment of the similarity between event 1 and event 2 using Gower's general coefficient requires the evaluation of $S_{12} = {}_{k=1}\sum^{5} w_{12k}\, s_{12k} / {}_{k=1}\sum^{5} w_{12k}$. The local similarities, $s_{ijk,}$ between the event pair (i, j) for attributes *k*, as defined by equation 5, is described below.

For events 1 and 2, the local similarities s_{12k}, $k = 1, \ldots, 5,$ take on the following values:

$s_{121} = 0$ Saturday $\neq$ Wednesday

$s_{122} = 0$ Alpha sector $\neq$ Charlie sector

$s_{123} = 0.75$ $1 - |18 - 12| /24 = 0.75$

$s_{124} = 0$ SAF $\neq$ IED

$s_{125} = 0$ no $\neq$ yes

The global similarity between events 1 and 2 is then calculated as

$$S_{12} = (0 + 0 + 0.75 + 0 + 0) / 5 = 0.15,$$

with corresponding dissimilarity $\sqrt{(1 - S_{12})} = 0.922$.

Because Gower's coefficient is a similarity score, $1–S_{ij}$ is the corresponding dissimilarity. Appropriately normalized, both take on complementary values in the unit interval [0, 1], that is, similarity + dissimilarity ≡ unity. The mapping $\sqrt{(1 - S_{12})}$ serves simply to enlarge the small values that will always be encountered, because they can never exceed unity, and it has no impact on the final result. However, with greater importance, dissimilarity corresponds more fully to the axioms supporting the distance model introduced in section 2.1.

The following section details the design and development of the HDPT visual analytic application. Specifically, the HDPT's development for concept demonstration in the U.S. Army's Command, Control, Communication, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) On-The-Move (OTM) 2012 exercise (E12) and associated scenario.

3. HDPT System Overview

Although there exists research into the development of qualitative and quantitative similarity analysis, few have been effectively coupled with a visualization framework, and none have been interactively coupled with a tactical military decision support environment. Toward that end, HDPT is the software instantiation of a visual analytic technique that effectively combines MDS with the flexibility of mixed-scale Gower's similarity calculation. The domain of interest for this instantiation is the assessment of individuals from a tactical social network. This specific implementation was designed for the concept demonstration at the C4ISR OTM E12 that was integrated with the DCGS-A program.

At a high level of abstraction, the concept of operation for this exercise was as follows. The HDPT started with a reference data set representing individuals with known group affiliations: insurgents, innocents, and criminals. As intelligence was collected on new individuals within an area of operation, HDPT computed and plotted their relative positions with regard to the existing reference data. The resulting analytic portrayed the relative position of the new individual's orientation within the known human terrain (insurgent vs. innocent vs. criminal). The goal of the tool was to give a military analysts further understanding of the local operational human environment and assist in defining future information requests. The remainder of this paper will

focus on the design and development of the HDPT web application demonstrated at the E12 C4ISR OTM.

As shown in figure 3, there are three major components that comprise the HDPT system: (1) the HDPT web application, (2) the Global graph, and (3) the statistics engine.

The primary HDPT component and subject of this paper is the HDPT web application. HDPT is a web application that is deployable within the Ozone Widget Framework (OWF). It accesses the Global graph via a web service as its data source. To calculate the similarity it uses Gower's similarity and multidimensional scaling algorithms contained in an *R* statistical computing environment. The Global graph and OWF are both products by Potomac Fusion [x] and part of the DCGS-A program of record (POR).

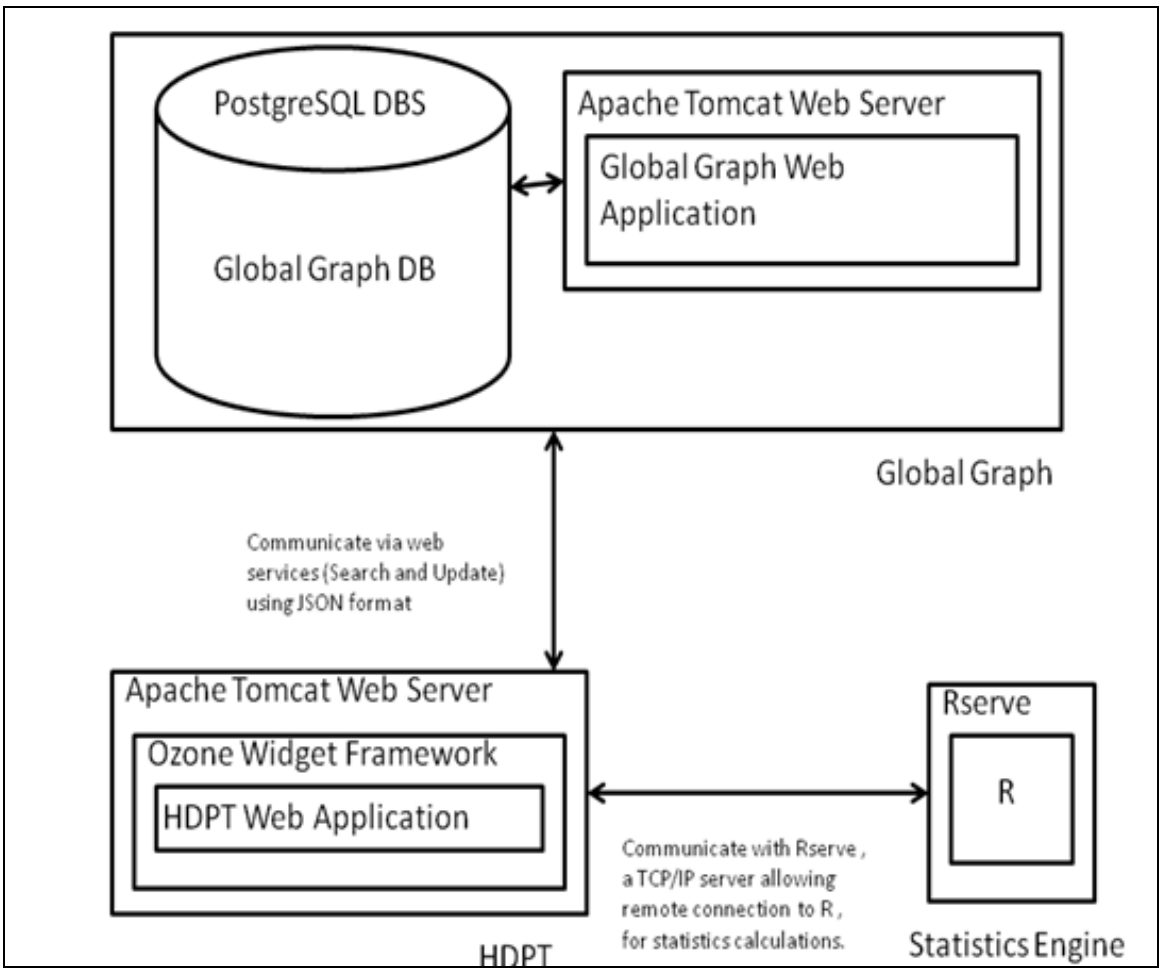

Figure 3. HDPT system diagram.

3.1 HDPT Web Application

The HDPT web application is the core of the HDPT system and serves as the user interface for performing similarity analysis. The primary components of the HDPT web application used in E12 are displayed in figure 4 and include the following: (1) HDPT Menu bar, (2) Search window panel, and (3) Plot window panel.

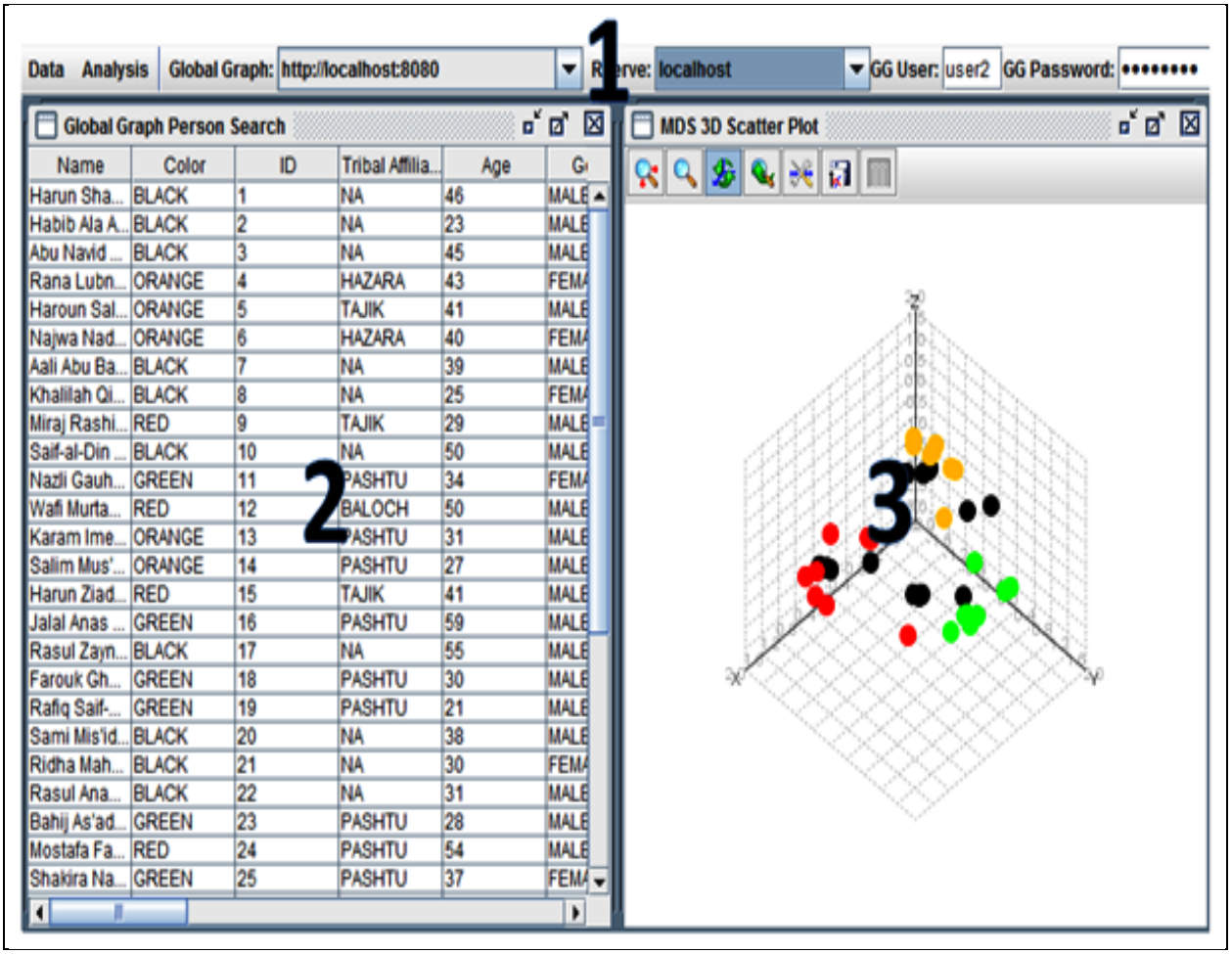

Figure 4. HDPT web application components: (1) Menu bar, (2) Search window, and (3) Plot window.

Figure 5 illustrates the overall Unified Modeling Langauge class diagram for the HDPT web application. The associated source code can be found in appendix C, HDPT Web Application Source code. The following sections will underscore the characteristics of the three primary components:

- HDPT Menu bar
- Search window panel
- Plot window panel

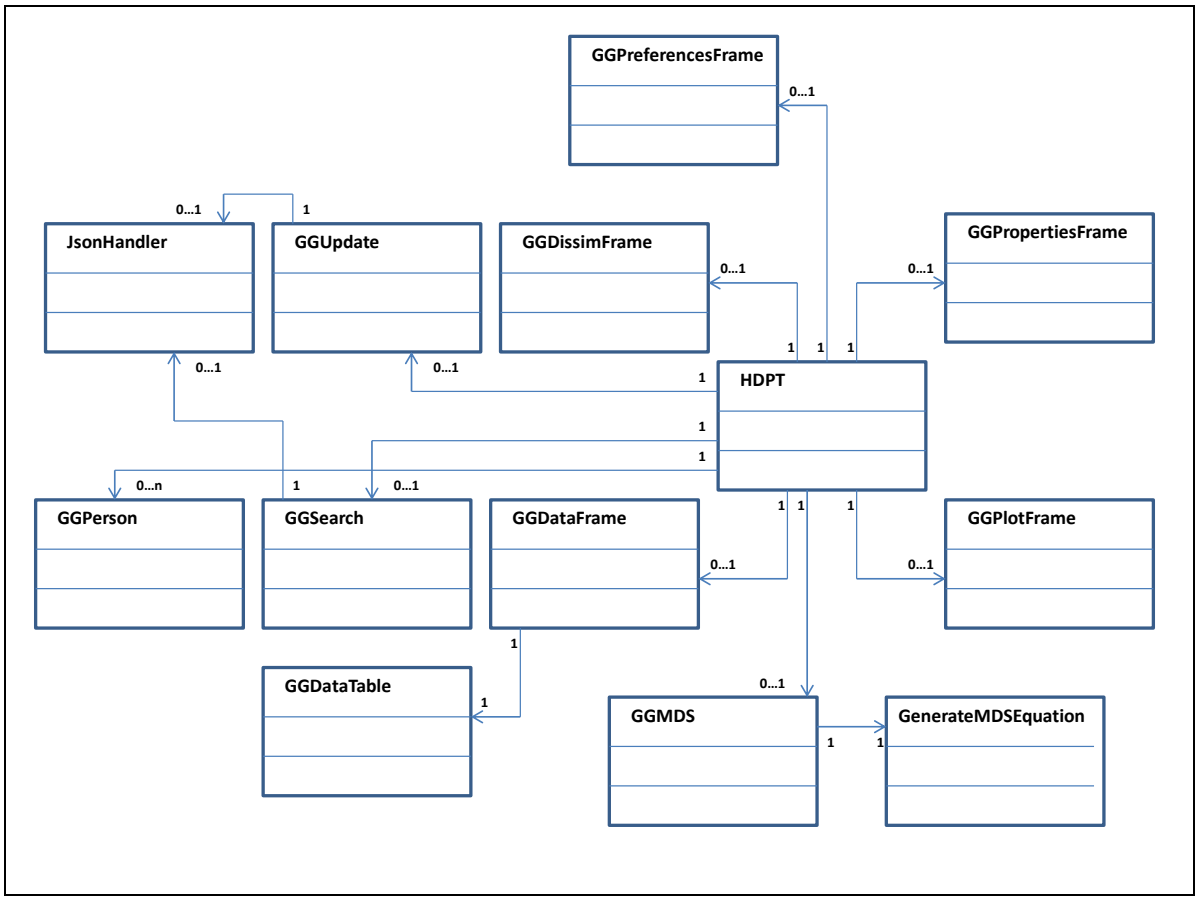

Figure 5. HDPT web application UML class diagram.

3.1.1 HDPT Menu Bar

The HDPT Menu bar contains all of the actions for loading data, executing the analysis, and configuring the HDPT. The Menu bar allows access to the Data Menu, Analysis Menu, and the configuration components.

3.1.1.1 Data Menu

The HDPT Data Menu contains all of the actions for loading data into the tool and propagating updates back to the data source. The data source used by HDPT during the E12 exercise was the DCGS-A Global graph [x]. In this exercise, a SQL version of the Global graph was used that consisted of a PostgreSQL database and associated web services for searching and updating the database (figure 3). A Representational State Transfer (REST) web service protocol was used for communication between HDPT and the data source using a JavaScript Object Notation (JSON) data structure. REST is a lightweight alternative protocol to mechanisms like Simple Object Access Protocol (SOAP) and Remote Procedure Calls (RPC) . SOAP, RPC, Typically with REST, simple HTTP is used to make the connections. Likewise, JSON is a lightweight data-interchange format designed for exchanging human-readable structured text. The selection of these protocols greatly facilitated connection and interaction with the DCGS-A framework.

1. Load All GG Data: Selecting **Load All GG Data** will load all data created for the E12 exercise that is resident in the Global graph into the HDPT Search window. This action was added to HDPT to provide a means to directly load all data associated with the OTM exercise in to HDPT. ALL OTM scenario data in the Global graph, created by ARL, was tagged with a searchable identifier that allowed the entire set to be retrieved by a single search. The JSON structure used with the web service search is shown in appendix A. The unified modeling language (UML) sequence diagram associated with selecting this command is shown in figure 6.

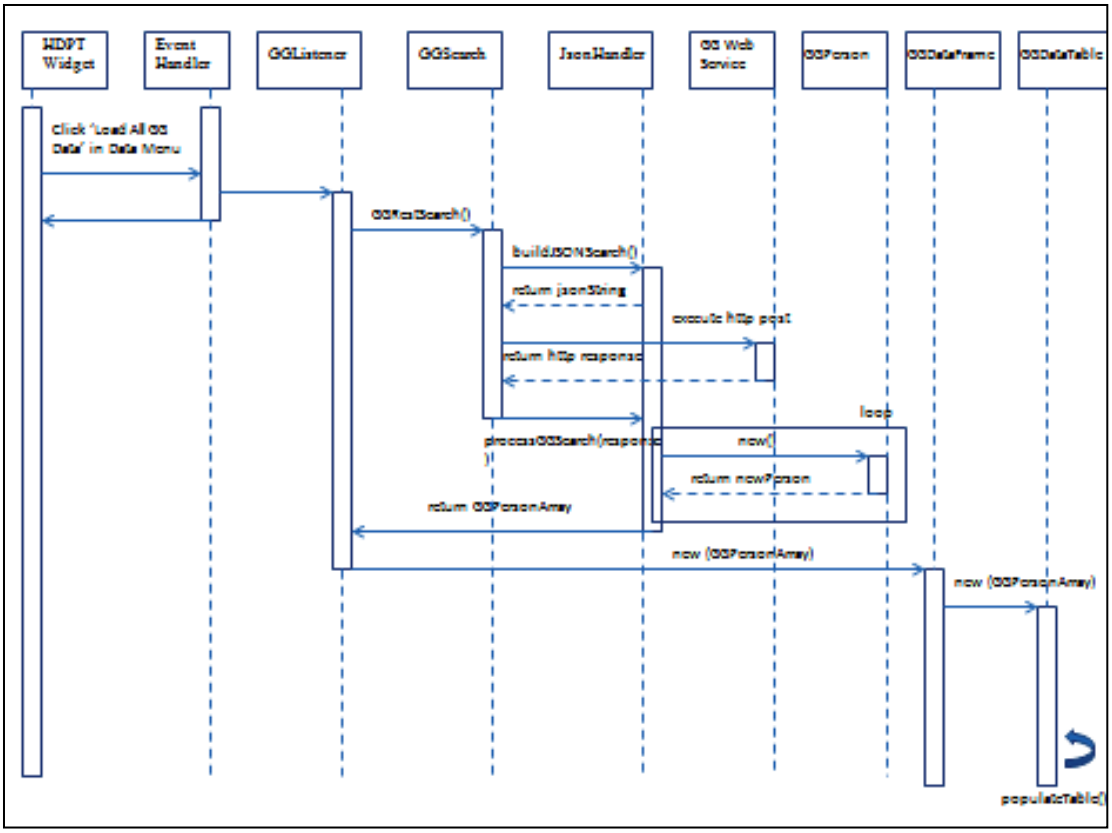

Figure 6. Load all Global graph data sequence diagram.

2. Load GG Data from Map: Selecting Load GG Data from Map will load data from a specific geographic area in the HDPT Search window. To use this action, the user must first use the HDPT map tool, shown in figure 7. The map tool returns a set of bounding coordinates that are used to search the Global graph. The map tool is a javascript map developed using the OpenLayers and OpenStreetMap Appliccation Programming Interface APIs and data. The JSON structure used with the web service search is shown in appendix A.

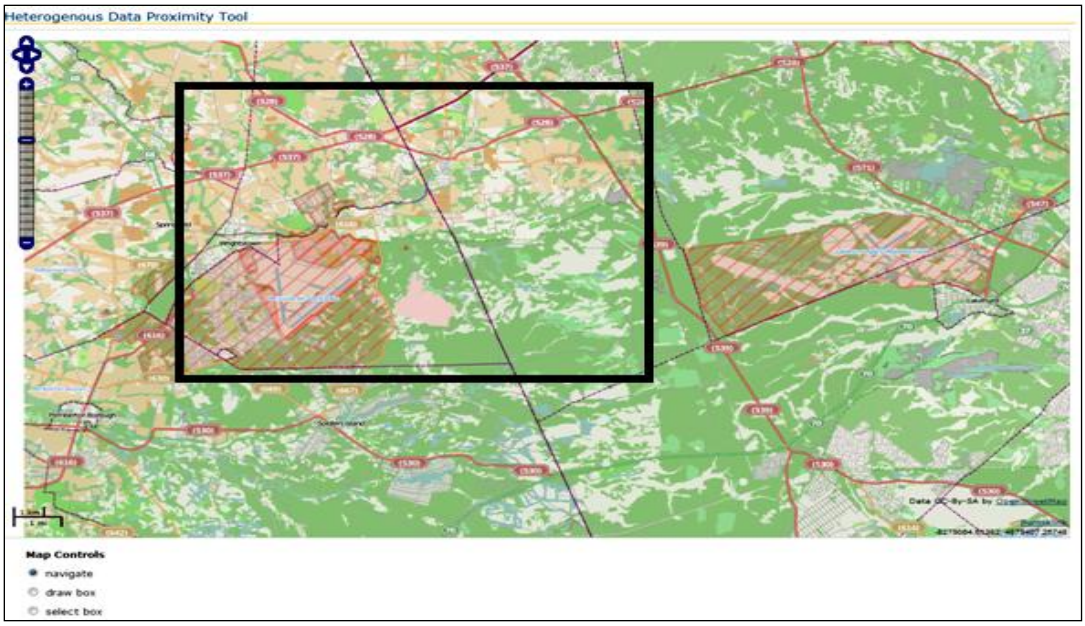

Figure 7. HDPT bounding box data selection.

3. Update GG Data: Selecting **Update GG Data** propagates all data changes that the user has made in the HDPT Search window back to the Global graph. The Search window contains a table, where each row is a person found during a Global graph search (from menu actions 1 and 2 above), and each column is one of the attributes known about the person. Each of these attributes, other than Name and ID, were editable during E12. If an attribute of any row is edited, then the Update GG Data action becomes available in the data menu. This action will call the Global graph update web service to write the new attribute value to the Global graph.

   The update web service requires a JSON structure of the person's complete set of attributes to perform the update. This JSON structure is actually obtained from the Global graph search results and saved as a variable in the java object created for each person. When the Search window detects an event that changes an attribute value for a person, the JSON structure variable for that person is edited to reflect the new value. Furthermore, a flag is set to mark that person as modified. The update menu action will update all persons that have a flag signaling they have been modified.

For the E12 exercise, the cells in the Search window table were preconfigured to only allow attributes to be modified within a narrow set of options (except for age, which could be any positive whole number). An example of a row being edited is shown in figure 8. In the E12 exercise, the attributes examined were static, and their name, data type, and range of values were a part of the HDPT code. Appendix B lists the attributes, possible values, and data type supported by HDPT for the E12 exercise.

Name	Color	ID	Tribal Affilia...	Age	Gender	Marital Stat...
Harun Sha...	BLACK	1	NA	46	MALE	NA
Habib Ala A...	BLACK	2	PASHTU	23	MALE	NA
Abu Navid ...	BLACK	3	BALOCH	45	MALE	NA
Rana Lubn...	ORANGE	4	HAZARA	43	FEMALE	SINGLE
Haroun Sal...	ORANGE	5	TAJIK	41	MALE	SINGLE
Najwa Nad...	ORANGE	6	NA	40	FEMALE	SINGLE
Aali Abu Ba...	BLACK	7		39	MALE	NA

Figure 8. Example of editing attributes within the Search window.

4. Edit Weights: Selecting **Edit Weights** will launch the Column Properties window (figure 9). This window displays the attributes of each of the columns in the Search window. The **Edit Weights** menu action only becomes clickable when the Search window is active. To begin, in the Search window, each column (except Name, Color, and ID) has a weight from 0 to 1. The weight is used during Gower's similarity calculations. Briefly, during Gower's calculations, each column is assigned to a vector. All of the vectors are then used as inputs to Gower's calculation. The weight of a vector influences how much it impacts the calculation. By default, all columns start with a weight of 1. The Column Properties window also contains a Data Type column, which displays the data type for the attribute (Nominal [NOM], Ordinal [ORD], Numeric [NUM], and Binary [BIN]). The data type is another variable used in Gower's calculations.

Column Properties

Column Name	Data Type	Weight
Tribal Affiliation	NOM	1
Age	NUM	1
Gender	BIN	1
Marital Status	BIN	1
Nationality	BIN	1
Place of Birth	BIN	1
AffiliationsID	ORD	1
AllegiancesID	NOM	1
Criminal Record	BIN	1
Education Level	BIN	1
Employment Type	NOM	1
Military Record	NOM	1
Religion	BIN	1
Skill	NOM	1
AddressID	NOM	1
EquipmentID	NOM	1
VehicleID	NOM	1

Figure 9. Column properties.

3.1.1.2 Analysis Menu

The Analysis menu contains the actions for executing Gower's and MDS calculations and generating the results as an interactive three-dimensional (3-D) scatter plot.

1. Plot: The **Plot** menu action becomes available once there is a Search window active. When this action is selected, the data currently displayed in the Search window is processed into a 3-D scatter plot. The process begins by initializing a connection to Rserve. Rserve is an open-source server that manages connections to *R*, the statistical engine used by HDPT. Consequently, *R* and Rserve must be running and configured on a networked computer reachable by HDPT. Next, HDPT formats the attributes and weights of the Search window and passes them to *R* to perform Gower's similarity calculation, specifically the gowdis() function in *R*. Finally, Gower's dissimilarity matrix is used as the arguments for the *R* function cmdscale(), which returns a set of 3-D coordinates that are plotted in the MDS 3-D Scatter Plot window. The UML sequence diagram for the Plot routine is shown in figure10.

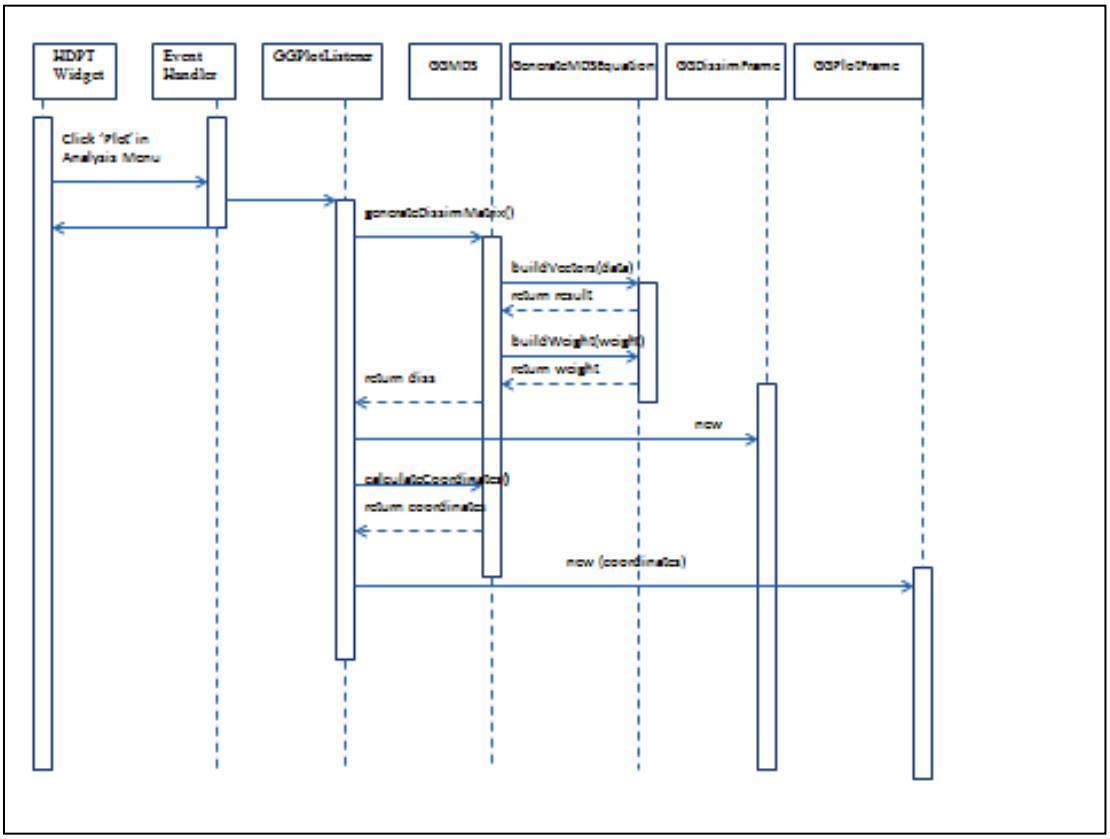

Figure 10. HDPT plot UML sequence diagram.

2. Preferences: The **Preferences,** as seen in figure 11, action brings up the Preferences window, which allows the configuration of the Plot window panel. There are two preferences that can be modified: attribute threshold and links displayed. Both of these attributes change the way the links between the nodes in the plot are handled. By default, when interacting with the plot, the user can right-click on a node, and links will be drawn to the three most similar nonblack nodes. These nodes are determined by HDPT evaluating the results in Gower's dissimilarity matrix, in which values closer to zero connote greater

similarity. The number of links drawn can be increased or decreased by using the “Change Max Links Slider.” Furthermore, if the user wanted to eliminate links to nodes with only a few attributes defined, then the “Use attribute threshold to filter links” option could be selected.

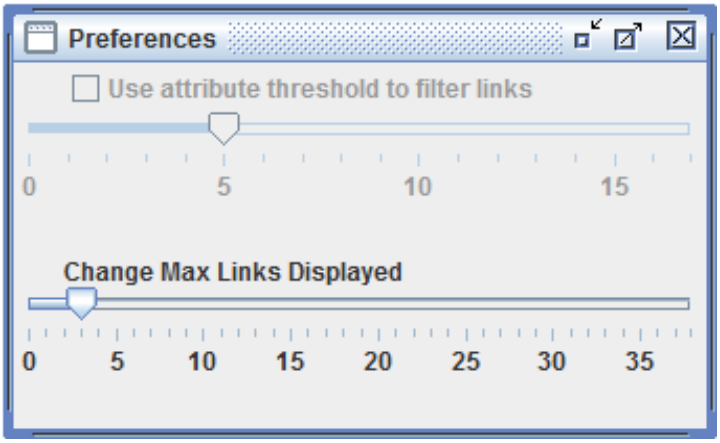

Figure 11. Preferences window.

3.1.1.3 Configuration Menu

The server Menu barallows the user to configure the server parameters for the Global graph and Rserve. There is a dropdown menu for each server selection that contains many commonly used servers for the OTM exercise. In addition, the user must enter their username and password for accessing the Global graph. HDPT uses these values for connecting to the Global graph REST web services and the statistics engine *R*.

3.1.2 Search Window Panel

The Search window panel is the part of the HDPT web application, in which the data set returned from a Global graph search is displayed, as seen in figure 4. The window consists of a table, in which each row is a person and each column is an attribute. The result returned from Global graph web search service is a JSON structure containing all of the matching people. HDPT processes the JSON structure into a separate Java object for each person. With that, the Search window has a limited number of user interactive features. In addition to attribute editing, which was discussed previously as part of the **Data Menu** actions, the Search window panel allows rows to be sorted according to the values in any column. Clicking the mouse on the column heading will cause the rows to sort alphabetically (words) or number order (digits) according to the data in that column. Finally, if the Plot window is currently active, clicking on a row in the Search window will highlight the node that corresponds to that row in the 3-D scatter plot. The source code for the Search window is found in appendix C.

3.1.3 Plot Window Panel

The Plot window panel displays a 3-D scatter plot of the MDS results. The development of the visualization used the JMathPlot open-source graphics library. A sample Plot window panel is shown in figure 12. In this case, the node under investigation is highlighted using a neutral yellow color and has the three "most" similar reference nodes linked with straight lines, sharing similarity between the two nodes from the criminal set (orange color) and one from the friendly set (green).

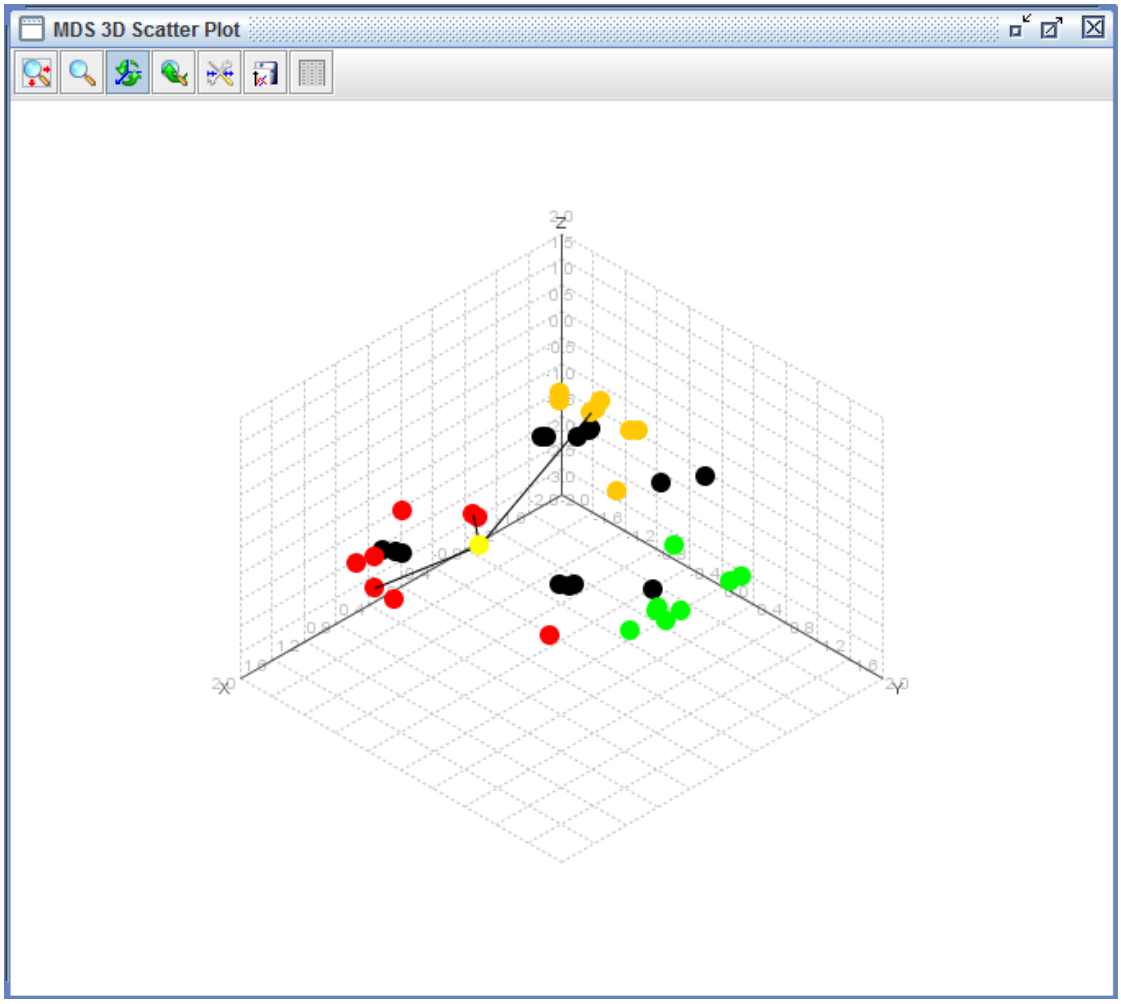

Figure 12. Plot window panel showing links.

To assist the Soldier's understanding of the underlying decision space, HDPT provided two important capabilities to the 3-D visual analytic that were used extensively throughout the E12 exercise. First, as shown in figures 13a and 13b, HDPT provided the ability to freely rotate the decision space along any axis. The projection of a 3-D decision space onto a 2-D screen can be problematic; objects that appear close to one another in 2-D can actually be far apart. The ability to rotate along any axis was critical to correctly interpreting the relation projections of the decision space.

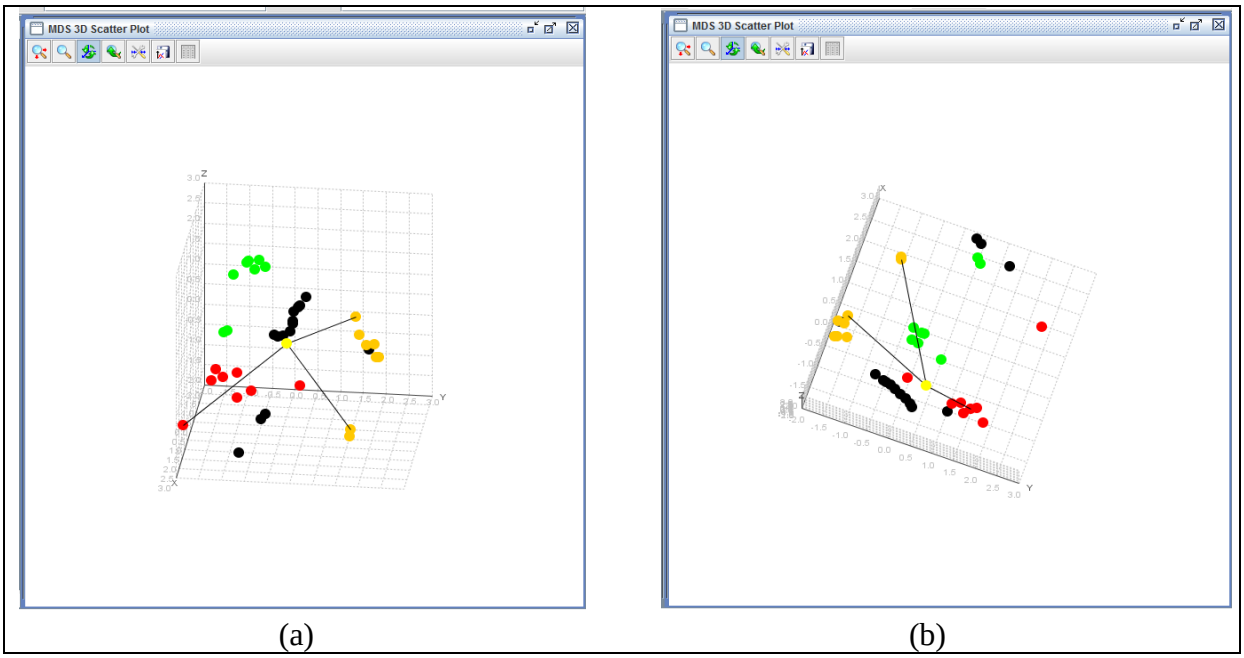

Figure 13. (a) View before rotation (b) HDPT rotated view.

A second capability that was used at length was the zoom. As shown in figure 14a and b, the zoom capability permitted users the ability to examine in finer detail the related nodes that were clustering close to the node in question.

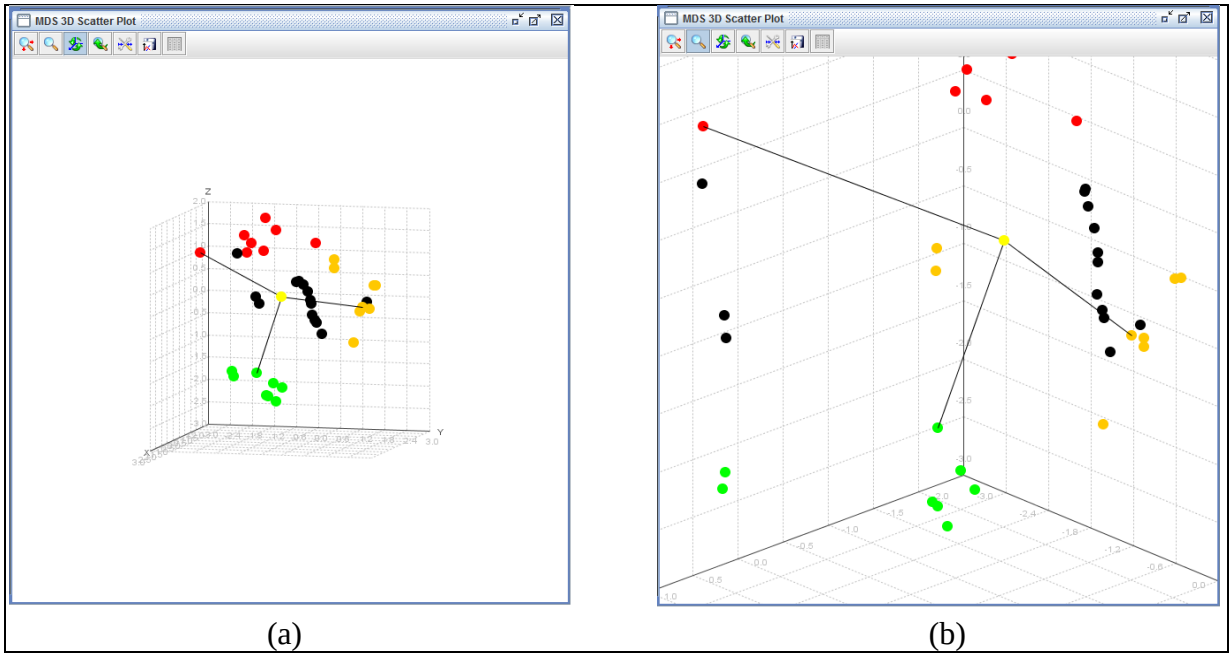

Figure 14. (a) View before zoom (b) HDPT zoomed view.

In this example, the node in question (yellow node) shows a high similarity to three neighboring nodes; one from each of the categorical types.

The Plot window panel also possessed several interactive features to assist in exploratory analysis. Left-clicking on a node in the display highlighted it and the corresponding row in the Search window, so that the node is referenced back to its original data. Second, as discussed previously, right-clicking on a node will draw links from itself to three or more of its most similar reference node neighbors.

4. Conclusions

Exploratory data analysis, cluster analysis, pattern recognition, data fusion, and data mining benefit from the acuity of visual knowledge extraction from high-dimensional data representations. MDS is a powerful technique holding the potential to contribute in all of these areas. HDPT was developed to demonstrate the utility of similarity analysis via a visual analytic in understanding the human terrain. The deployment of HDPT in a tactical environment (Ozone Widget and Global graph) made it possible to successfully demonstrate this technology and test the power of similarity analysis at the E12 exercise.

An important extension of this work is the development of formal procedures to determine how the value of information (VoI) could influence attribute weighting. Not all information should be weighted equally. Depending on the context of the current operational tempo, the weight assigned each piece of information and is dependent on the combined assessment of the *reliability of the source* along with the assessment of its *credibility or content* (*19*). Research is underway to investigate how VoI would be used not only to weight individual attributes, but how to combine complementary and contradictory results. Additionally, methods need to be developed to effectively model linguistic information, given over 80% of intelligence information is contained in the form of unstructured text. It is paramount to develop improved methodologies to process and exploit large and disparate unstructured data sets. The area of computational linguistics holds some promise for tackling this daunting challenge.

5. References

1. Thomas, J. J.; Cook, K. Illuminating the Path: The R&D Agenda for Visual Analytics. *IEEE Computer Society*, http://nvac.pnl.gov/agenda.stm, 2005; p 4.
2. Greitzer, F. L.; Noonan, C. F.; Franklin, L. R. Cognitive Foundations for Visual Analytics. Technical report, Pacific Northwest National Laboratory (PNNL), Richland, WA, 2011.
3. Börner, K.; Chen, C.; Boyack, K. Visualizing Knowledge Domains, in Blaise Cronin (Ed.), *Annual Review of Information Science & Technology*, Vol. 37,Medford, NJ: Information Today, Inc./American Society for Information Science and Technology, chapter 5, pp. 179–255, 2003.
4. Heer, J.; Card, S.; Landay, J. Prefuse: A Toolkit For Interactive Information Visualization. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, Portland, OR, 2–7 April 2005.
5. Hanratty, T.; Hammell II, R.; Yen, J.; McNeese, M.; Oh, S.; Kim, S.; Minotra, D.; Strater, L.; Cuevas, Colombo, D. Knowledge Visualization to Enhance Human-Agent Situation Awareness within a Computational Recognition-Primed Decision System. *IEEE Workshop on Situation Management at MILCOM*, 2009.
6. Torgerson, W. S. Multidimensional Scaling: I. Theory and Method. *Psychometrika* **1952**, *17*, 401–419.
7. Young, F. *Understanding Multidimensional Scaling Kotz-Johnson (Ed.) Encyclopedia of Statistical Sciences*, Vol. 5, Copyright (c) by John Wiley & Sons, Inc accessed online http://forrest.psych.unc.edu/teaching/p208a/mds/mds.html. (1985).
8. Cox, T.; Cox, M. Multidimensional Scaling, CRC Press 2nd Edition, 2001.
9. Mead, A. *Review of the Development of Multidimensional Scaling Methods, the Statistician* **1992**, *41* (1), 27–39.
10. Shepard, R. N. The Analysis of Proximities: Multidimensional Scaling With an Unknown Distance Function. *Psychometrika* **1962**, *27*, 125–139; 219–246.
11. Kruskal, J. B. Nonmetric Multidimensional Scaling: A Numerical Method. *Psychometrika* **1964**, *29*, 115–129.
12. Heady, R.; Lucas, J. PERMAP Operation Manual, University of Louisiana at Lafayette and Agnes Scott College, 23 March 2007.

13. Stevens, S. S. On the Theory of Scales of Measurement. Science **1946**, *103* (2684), 677–680.

14. Gower, J. A General Coefficient of Similarity and Some of Its Properties Biometrics **1971**, *27*, (4), 857–871.

15. Coppock, S.; Mazlack, L. Multi-Modal Data Fusion: A Description to be presented at *KES'2004, 8th International Conference on Knowledge-Based Intelligent Information & Engineering Systems* in Wellington, New Zealand, 2004.

16. Ji, Y.; Massanari, R.; Ager, J.; Yen, J.; Miller, R.; Ying, H. A Fuzzy Logic-Based Computational Recognition-Primed Decision Model. *Information Science* **2007**.

17. Podani, J. Extending Gower's General Coefficient of Similarity to Ordinal Characters, *Taxon* **1999**, 48, 331–340.

18. Wang, W. New Similarity Measures on Fuzzy Sets and on Elements. *Fuzzy Sets Syst*. **1997**, *85* (3), 305–309.

19. Hanratty et al. Capturing the Value of Information in Complex Military Environments. *IEEE Fuzzy Systems* 2012.

INTENTIONALLY LEFT BLANK.

Appendix A. Sample JSON Structure of Global Graph Search Result

This appendix appears in its original form, without editorial change.

Below is a sample JSON structure for a single person returned by the Global Graph REST service after executing a search. The values that were extracted from the JSON and used as attributes in HDPT are highlighted.


```
{
"id":"ebb9635f-d502-4a9c-9580-69a193b83426",
"classification":"Person",
"fgiSourceProtected":false,
"addresses":[{"id":"","city":"VNV"}],
"age":34,
"criminalRecords":[{"id":"","verdict":"Guilty"}],
"description":"Blue Minivan",
"education":[{"id":"","educationalLevel":"High"}],
"employment":[{"id":"","employerType":"BC"}],
"ethnicity":"HAZARA",
"gender":{"raw":"","physicalValue":"MALE"},
"handicapDisabilities":[],
"identification":[],
"languages":[{"id":"","isNativeLanguage":false}],
"maritalStatus":"MARRIED",
"medicalRecords":[],
"militaryService":[{"id":"","dutyOrPosition":"Served"}],
"personTravels":[],
"placeOfBirth":"BOA",
"significance":[],
"skills":[{"id":"","skill":"EL"}],
"citizenships":[],
"nationality":{"raw":"","physicalValue":"MUSLMA"},
"religions":[{"id":"","religionName":"Rad"}],
"displayName":"Ziyad Guda Sultan",
"names":[{"id":"",
"fullName":"Ziyad Guda Sultan","isDisplayName":false}],
"targetInfo":[],
"trackInfo":[],
"remarks":[{"id":"","subject":"C4ISR OTM","details":"Bomb"}],
"locations":[{"id":"","coordinates":[{"latitude":40.0,"longitude":-
74.45,"altitude":"NaN"}],"geometryType":"POINT"}],
"allegiances":[],
"battleDamageAssessment":{"id":"","isFirstVisit":false,"date":1336536000000},
"currentAssessment":{"id":"","isFirstVisit":false,"date":1336536000000},
"affiliations":[{"id":"","affiliationGroupName":"NFL","natureOfAffiliation":"2","
affiliation":"insurgent"}],
"externalKeys":[],
"classificationLevel":{"raw":"","physicalValue":"U"},
"electronicLocations":[],
"mapLocations":[{"lat":40.0,"lon":-74.45,"altitude":"NaN"}]
}
```

Appendix B. Person Attributes

This appendix appears in its original form, without editorial change.

Name
Values: No restrictions
Data type: Not used in similarity calculations

Color
Values: {BLACK (UNKNOWN), GREEN(FRIENDLY), ORANGE(CRIMINAL), RED(INSURGENT)}
Data type: Not used in similarity calculations

ID
Values: Sequential integer assigned from 1 to n, where n is the number of entities returned from a Global Graph search
Data type: Not used in similarity calculations

Tribal Affiliation
Values: {PASHTU, BALOCH, HAZARA, TAJIK, NA}
Data type: Nominal

Age
Values: Integer from 1 to 100
Data type: Number (continuous)

Gender
Values: {MALE, FEMALE, NA
Data type: Binary

Marital Status
Values: {SINGLE, MARRIED, NA}
Data type: Binary

Nationality
Values: {MUSLMA, AFGHAN, NA}
Data type: Binary

Place of Birth
Values: {BIA, BOA, NA}
Data type: Binary

AffiliationsID
Values: {HOSTILE, DISLIKE, NEUTRAL, JOKER, LIKES, FRIENDLY, NA}
Data type: Ordinal

AllegiancesID
Values: {BGI, TFH, WWD, NFL, CWJGA, NPT, NA}
Data type: Nominal

Criminal Record
Values: {GUILTY, NONE, NA}
Data type: Binary
Education Level
Values: {HIGH, LOW, NA}
Data type: Binary

Employment Type
Values: {NE, WC, BC, NA}
Data type: Nominal

Military Record
Values: {NEVER SERVED, SERVING, SERVED, NA}
Data type: Nominal

Religion
Values: {MLD, RAD, NA}
Data type: Binary

Skill
Values: {PH, WR, EL, ME, CO, DR, FI, NA}
Data type: Nominal

Address ID
Values: {VNV, VV, HAV, CCV, GT, UV, HOV, NA}
Data type: Nominal

EquipmentID
Values: {KNIFE, GANG COLORS, BOMB, VIDEO CAMERA, CELL PHONE, UNIFORM, BRIEFCASE, NA]
Data type: Nominal

VehicleID
Values: {BLUE MOTORCYCLE, SILVER COMPACT CAR, BLUE MINIVAN, GREY SEDAN, BROWN PICKUP TRUCK, BLACK SUV, BURGUNDY LUXURY SEDAN, NA}
Data type: Nominal

INTENTIONALLY LEFT BLANK.

Appendix C. HDPT Source Code

This appendix appears in its original form, without editorial change.

```
package hdpt;

import java.awt.BorderLayout;
import java.awt.Cursor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.beans.PropertyVetoException;
import java.util.Vector;
import javax.swing.JApplet;
import javax.swing.JComboBox;
import javax.swing.JDesktopPane;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPasswordField;
import javax.swing.JProgressBar;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.JToolBar;
import javax.swing.event.InternalFrameEvent;
import javax.swing.event.InternalFrameListener;

public class HDPT extends JApplet {

	private static final long serialVersionUID = 1L;
	public static final int ALL = 0;
	public static final int MAP = 1;
	public static final int UPDATE = 2;
	private JDesktopPane desk = null;
	private JScrollPane scroll = null;
	private GGMDS ggMDS = null;
	private double[] diss;
	private double[][] coordinates;
	private JMenu dataMenu;
	private JMenu analysisMenu;
	private GGDataFrame ggDataFrame = null;
	private GGDissimFrame ggDissimFrame = null;
	private GGPreferencesFrame ggPreferencesFrame = null;
	private GGPlotFrame ggPlotFrame = null;
	private GGPropertiesFrame ggPropertiesFrame = null;
	private GGCreateFrame ggInsertFrame = null;
	private Object[] result = null;
	private Object[] geoSearch;
	private JMenuItem ggItem;
	private JMenuItem ggGeoItem;
	private JMenuItem ggInsertItem;
	private JMenuItem ggPlotItem;
	private JMenuItem ggPreferencesItem;
	private JMenuItem ggUpdateItem;
	private JMenuItem ggWeightsItem;
	private JTextField userField;
	private JPasswordField pwdField;
	private JComboBox<String> ggHost;
	private String[] ggHosts = { "http://localhost:8080","http://10.1.200.82:8443",
	"https://zulu.idvrn.arl.army.mil:8443" };
	private JComboBox<String> rserveHost;
	private String[] rserveHosts = {
"localhost","10.1.200.87","10.1.200.82","zulu.idvrn.arl.army.mil",zeus.idvrn.arl.army.mil"};
	private GGSearch search;
	private GGUpdate update;
	private boolean geo = false;

	public void init() {
		// Execute a job on the event-dispatching thread:
		// creating this applet's GUI.
		try {
			javax.swing.SwingUtilities.invokeAndWait(new Runnable() {
```

```
                        public void run() {
                                createGUI();
                        }
                });
        } catch (Exception e) {
                System.err.println("createGUI didn't successfully complete");
        }
        search = new GGSearch(this);
        update = new GGUpdate(this);
        geoSearch = new Object[4];

}

public void createGUI() {

        this.desk = new JDesktopPane();
        this.scroll = new JScrollPane(desk);
        this.getContentPane().setLayout(new BorderLayout());
        this.getContentPane().add(scroll, BorderLayout.CENTER);
        buildMenu();
}

private void buildMenu() {

        JToolBar toolBar = new JToolBar("Still draggable");
        this.getContentPane().add(toolBar, BorderLayout.NORTH);

        this.setJMenuBar(new JMenuBar());

        this.dataMenu = new JMenu("Data");
        this.getJMenuBar().add(this.dataMenu);

        this.analysisMenu = new JMenu("Analysis");
        this.getJMenuBar().add(this.analysisMenu);

        ggItem = new JMenuItem("Load All GG Data");
        ggItem.addActionListener(new ggListener(this));
        this.dataMenu.add(ggItem);

        ggGeoItem = new JMenuItem("Load GG Data from Map");
        ggGeoItem.addActionListener(new ggListener(this));
        this.dataMenu.add(ggGeoItem);
        this.ggGeoItem.setEnabled(false);

        ggInsertItem = new JMenuItem("Analyze Person");
        ggInsertItem.addActionListener(new ggInsertListener(this));
        this.dataMenu.add(ggInsertItem);
        this.ggInsertItem.setEnabled(true);

        ggUpdateItem = new JMenuItem("Update GG Data");
        ggUpdateItem.addActionListener(new ggUpdateListener(this));
        this.dataMenu.add(ggUpdateItem);
        this.ggUpdateItem.setEnabled(false);

        ggWeightsItem = new JMenuItem("Edit Weights");
        ggWeightsItem.addActionListener(new ggWeightsListener(this));
        this.dataMenu.add(ggWeightsItem);
        this.ggWeightsItem.setEnabled(false);

        ggPlotItem = new JMenuItem("Plot");
        ggPlotItem.addActionListener(new ggPlotListener(this));
        this.analysisMenu.add(ggPlotItem);

        ggPreferencesItem = new JMenuItem("Preferences");
        ggPreferencesItem.addActionListener(new ggPreferencesListener(this));
        this.analysisMenu.add(ggPreferencesItem);

        ggHost = new JComboBox<String>(ggHosts);
        ggHost.setEditable(false);
        rserveHost = new JComboBox<String>(rserveHosts);
        rserveHost.setEditable(false);
        userField = new JTextField("user2");
        pwdField = new JPasswordField("ggsecret");
        toolBar.add(new JLabel("  Global Graph: "));
        toolBar.add(this.ggHost);


        toolBar.add(new JLabel("  Rserve: "));
        toolBar.add(this.rserveHost);
```

```
        toolBar.add(new JLabel("GG User: "));
        toolBar.add(this.userField);


        toolBar.add(new JLabel("GG Password: "));
        toolBar.add(this.pwdField);
}

public String getGGHost() {
        return (String) ggHost.getSelectedItem();
}

public String getRserveHost() {
        return (String) rserveHost.getSelectedItem();
}

public String getUser() {
        // System.out.println((String)userField.getText());
        return (String) userField.getText();
}

public String getPwd() {
        // System.out.println(String.valueOf(pwdField.getPassword()));
        return String.valueOf(pwdField.getPassword());
}

public void setSearchBounds(Object top, Object left, Object bottom,Object right) {

        geoSearch[0] = top;
        geoSearch[1] = left;
        geoSearch[2] = bottom;
        geoSearch[3] = right;
        this.ggGeoItem.setEnabled(true);
        geo = true;
}

public JTable getGGDataTable() {             // Used to pass the data table to plot
                                             // frame so it can correlate to plot
        return ggDataFrame.getGGDataTable();
}

public boolean dataExists() {         // Used by preferences frame to determine if
                                      // any data has been loaded, so that the max
                                             // links slider can be configure
        if (ggDataFrame == null)
                return false;
        else
                return true;
}

public void selectPlot(int plot, boolean selected) {// by the data table to correlate
                                                    // plot with a row selected
        if (ggPlotFrame != null)
                ggPlotFrame.getPlots(plot, selected);

}

public Vector<Vector<Object>> getGGDissimData() { // Passes the dissimilarity so it
                                             // can be used by plot frame to determine
                                             // closest neighbors

        if (ggDissimFrame != null)
                return ggDissimFrame.getCellData();
        else
                return null;
}

public class ggListener implements ActionListener, Runnable {
        HDPT callback;
        ActionEvent e;

        public ggListener(HDPT callback) {
                this.callback = callback;
        }

        public void actionPerformed(ActionEvent event) {

                this.e = event;
                (new Thread(this)).start();

}
```

```
		@Override
		public void run() {
			try {

				if (e.getSource() == ggGeoItem)
					result = search.GGRestSearch(MAP, geoSearch);
				else
					result = search.GGRestSearch(ALL, null);

				if (ggDataFrame != null) {
					ggDataFrame.dispose();
					ggDataFrame = null;
				}

				if (ggDissimFrame != null) {
					ggDissimFrame.dispose();
					ggDissimFrame = null;
				}

				if (ggPlotFrame != null) {
					ggPlotFrame.dispose();
					ggPlotFrame = null;
				}

				if (ggPropertiesFrame != null) {
					ggPropertiesFrame.dispose();
					ggPropertiesFrame = null;
				}

				ggDataFrame = new GGDataFrame(result, callback);
				ggDataFrame.addInternalFrameListener(new frameListener());
				desk.add(ggDataFrame);
				// dissimilarityMenu.setEnabled(true);
				ggPlotItem.setEnabled(true);
				ggUpdateItem.setEnabled(true);
				ggWeightsItem.setEnabled(true);
			} catch (Exception srchEx) {
				System.out.println("Unable to perform Global Graph Search\n"
						+ srchEx);
			}

		}
	}

	public class ggUpdateListener implements ActionListener, Runnable {

		HDPT callback;
		ActionEvent e;
		JProgressBar updateProgress;

		public ggUpdateListener(HDPT callback) {
			this.callback = callback;
		}

		public void actionPerformed(ActionEvent event) {

			this.e = event;
			(new Thread(this)).start();

		}

		@Override
		public void run() {
			callback.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));
			ggDataFrame.setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));

			ggItem.setEnabled(false);
			ggUpdateItem.setEnabled(false);
			if (geo)
				ggGeoItem.setEnabled(false);

			if (ggDataFrame.isDataEdited())
				try {
					update.GGRestUpdate(ggDataFrame.getGGData());
					ggDataFrame.setDataEdited(false);
				} catch (Exception e1) {
					System.out.println("Unable to Update the Global Graph.");
					// e1.printStackTrace();
				}
```

```
                ggItem.setEnabled(true);
                ggUpdateItem.setEnabled(true);
                if (geo)
                        ggGeoItem.setEnabled(true);

                callback.setCursor(Cursor
                                .getPredefinedCursor(Cursor.DEFAULT_CURSOR));
                ggDataFrame.setCursor(Cursor
                                .getPredefinedCursor(Cursor.DEFAULT_CURSOR));

        }
}

public class ggWeightsListener implements ActionListener {

        HDPT callback;

        public ggWeightsListener(HDPT callback) {
                this.callback = callback;
        }

        public void actionPerformed(ActionEvent e) {
                if (ggPropertiesFrame != null) {
                        ggPropertiesFrame.dispose();
                        ggPropertiesFrame = null;
                }

                ggPropertiesFrame = new GGPropertiesFrame(callback);
                desk.add(ggPropertiesFrame);
                ggPropertiesFrame.toFront();

        }
}

public class ggInsertListener implements ActionListener {

        HDPT callback;

        public ggInsertListener(HDPT callback) {
                this.callback = callback;
        }

        public void actionPerformed(ActionEvent e) {
                if (ggInsertFrame != null) {
                        ggInsertFrame.dispose();
                        ggInsertFrame = null;
                }

                ggInsertFrame = new GGCreateFrame(callback);
                desk.add(ggInsertFrame);
                ggInsertFrame.toFront();

        }
}

public class ggPreferencesListener implements ActionListener {

        HDPT callback;

        public ggPreferencesListener(HDPT _callback) {
                this.callback = _callback;
        }

        public void actionPerformed(ActionEvent e) {
                if (ggPreferencesFrame != null) {
                        ggPreferencesFrame.dispose();
                        ggPreferencesFrame = null;
                }

                ggPreferencesFrame = new GGPreferencesFrame(callback);
                desk.add(ggPreferencesFrame);
                ggPreferencesFrame.toFront();

        }
}

public class ggPlotListener implements ActionListener, Runnable {

        HDPT callback;
        ActionEvent e;
```

```
        public ggPlotListener(HDPT callback) {
                this.callback = callback;
        }

        public void actionPerformed(ActionEvent event) {

                this.e = event;
                (new Thread(this)).start();
        }

        @Override
        public void run() {
                if (ggMDS == null) {
                        try {
                                ggMDS = new GGMDS(callback);
                        } catch (Exception e1) {
                                System.out.println("Unable to connect to R. ");
                                ggMDS = null;
                                // e1.printStackTrace();
                        }
                }

                if (ggMDS != null) {
                        try {
                                diss = ggMDS.generateDissimMatrix(ggDataFrame.getGGData());
                        } catch (Exception dissEx) {
                        System.out.println("Unable to calculate Dissimilarity Matrix.");
                                diss = null;
                        }

                        if (diss != null) {

                                if (ggDissimFrame != null) {
                                        ggDissimFrame.dispose();
                                        ggDissimFrame = null;
                                }

                                if (ggPlotFrame != null) {
                                        ggPlotFrame.dispose();
                                        ggPlotFrame = null;
                                }

                                ggDissimFrame = new GGDissimFrame(diss, callback);
                                desk.add(ggDissimFrame);
                                try {
                                        ggDissimFrame.setIcon(true);
                                } catch (PropertyVetoException e1) {//1.printStackTrace();
                                }
                                // mdsMenu.setEnabled(true);

                                try {
                                        coordinates = ggMDS.calculateCoordinates();
                                        ggMDS = null;
                                } catch (Exception mdsEx) {
                                        System.out.println("Unable to calculate MDS.");
                                        coordinates = null;
                                }

                                if (coordinates != null) {
                                ggPlotFrame = new GGPlotFrame(coordinates, ggDataFrame
                                                        .getGGData().getPeople(), callback,
                                                        callback.getGGDissimData());
                                        ggDataFrame.getGGDataTable().clearSelection();
                                        desk.add(ggPlotFrame);
                                }
                        }
                }
        }
}

public class frameListener implements InternalFrameListener {

        public void internalFrameActivated(InternalFrameEvent arg0) {
        }

        public void internalFrameClosed(InternalFrameEvent arg0) {
                ggPlotItem.setEnabled(false);
                ggUpdateItem.setEnabled(false);
                ggWeightsItem.setEnabled(false);
        }
        public void internalFrameClosing(InternalFrameEvent arg0) {
```

```
                }
                public void internalFrameDeactivated(InternalFrameEvent arg0) {
                }
                public void internalFrameDeiconified(InternalFrameEvent arg0) {
                }
                public void internalFrameIconified(InternalFrameEvent arg0) {
                }
                public void internalFrameOpened(InternalFrameEvent arg0) {
                }

        }
}

package hdpt;

import javax.swing.*;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableModel;
import java.awt.BorderLayout;

public class GGDataFrame extends JInternalFrame {

        private static final long serialVersionUID = 1L;
        private JTable table;
        private DefaultTableModel model;
        private JScrollPane scrollPane;
        private GGDataTable dataTable;
        private boolean edited = false;

        public GGDataFrame(Object[] data, HDPT callback) {
                // this.callback = callback;
                this.dataTable = new GGDataTable(data);
                this.setTitle(dataTable.getTitle());
                this.model = new
DefaultTableModel(ColumnProperties.getColumnNames(),dataTable.getRowCount());
                this.table = new JTable(model);
                this.table = dataTable.populateTable(table, model);
                this.table.setRowSelectionAllowed(true);
                this.table.setAutoCreateRowSorter(true);
                this.table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                this.table.getModel().addTableModelListener(new dataTableListener(callback));
                this.table.getSelectionModel().addListSelectionListener(new
rowSelectionListener(table, callback));

        this.table.getSelectionModel().setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
                this.scrollPane = new JScrollPane(table);
                this.setContentPane(scrollPane);

                this.setSize(callback.getContentPane().getSize().width / 2,((BorderLayout)
callback.getContentPane().getLayout())
                                .getLayoutComponent(BorderLayout.CENTER).getSize().height);
                this.setLocation(0, 0);
                this.setVisible(true);
                this.setResizable(true);
                this.setMaximizable(true);
                this.setIconifiable(true);
                this.setClosable(true);
                this.setDefaultCloseOperation(JInternalFrame.DISPOSE_ON_CLOSE);

        }

        public boolean isDataEdited() {
                return edited;
        }
        public void setDataEdited(boolean b) {
                edited = b;
        }
        public GGDataTable getGGData() {
                return dataTable;
        }
        public JTable getGGDataTable() {
                return table;
        }
        public class dataTableListener implements TableModelListener {
                HDPT callback;

                public dataTableListener(HDPT _callback) {
                        callback = _callback;
```

```
                }

                public void tableChanged(TableModelEvent e) {

                        int col = e.getColumn();
                        int row = e.getFirstRow();
                        // System.out.println(row);
                        model.removeTableModelListener(this);

                        dataTable.handleChange(col, row, model);
                        edited = true;
                        model.addTableModelListener(this);
                }

        }

        public class rowSelectionListener implements ListSelectionListener {
                JTable table;
                HDPT callback;
                public rowSelectionListener(JTable _table, HDPT _callback) {
                        table = _table;
                        callback = _callback;
                }

                public void valueChanged(ListSelectionEvent e) {
                        ListSelectionModel lsm = (ListSelectionModel) e.getSource();

                        if (!lsm.getValueIsAdjusting()) {
                                int first = e.getFirstIndex();
                                int last = e.getLastIndex();
                                if (lsm.isSelectedIndex(first)) {
                                        // System.out.println(table.getValueAt(first, 2));
                                        callback.selectPlot(
                                                (Integer) table.getValueAt(last, 2) - 1, false);
                                        callback.selectPlot(
                                                (Integer) table.getValueAt(first, 2) - 1, true);
                                } else {
                                        // System.out.println(table.getValueAt(last, 2));
                                        callback.selectPlot(
                                                (Integer) table.getValueAt(first, 2) - 1, false);
                                        callback.selectPlot(
                                                (Integer) table.getValueAt(last, 2) - 1, true);
                                }
                        }
                }

        }

}
package hdpt;

import javax.swing.DefaultCellEditor;
import javax.swing.JComboBox;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;

public class GGDataTable {

        private TableColumn nameColumn;
        private TableColumn idColumn;
        private TableColumn colorColumn;
        private String[] colorTypes = { "RED", "BLUE", "GREEN", "BLACK", "ORANGE" };
        private TableColumn tribeColumn;
        private String[] tribeTypes = { "PASHTU", "BALOCH", "HAZARA", "TAJIK", "NA" };
        private TableColumn genderColumn;
        private String[] genderTypes = { "MALE", "FEMALE", "NA" };
        private TableColumn maritalColumn;
        private String[] maritalTypes = { "SINGLE", "MARRIED", "NA" };
        private TableColumn nationalityColumn;
        private String[] nationalityTypes = { "MUSLMA", "AFGHAN", "NA" };
        private TableColumn pobColumn;
        private String[] pobTypes = { "BIA", "BOA", "NA" };
        private TableColumn equipmentColumn;
        private String[] equipmentTypes = { "Pistol", "Knife", "Gang Colors",
                        "Bomb", "Video Camera", "Cell Phone", "Uniform", "Briefcase", "NA" };
        private TableColumn vehicleColumn;
        private String[] vehicleTypes = { "White Panel Truvk", "Blue Motorcycle",
                        "Silver Compact Car", "Blue Minivan", "Grey sedan",
                        "Brown Pickup Truck", "Black SUV", "Burgundy Luxury Sedan", "NA" };
```

```
private TableColumn affiliationColumn;
private String[] affiliationTypes = { "Hostile", "Dislike", "Neutral",
                "Joker", "Likes", "Friendly", "NA" };
private TableColumn allegianceColumn;
private String[] allegianceTypes = { "BGI", "TFH", "WWD", "NFL", "CWJGA",  "NPT", "NA" };
private TableColumn criminalColumn;
private String[] criminalTypes = { "Guilty", "None", "NA" };
private TableColumn educationColumn;
private String[] educationTypes = { "High", "Low", "NA" };
private TableColumn employmentColumn;
private String[] employmentTypes = { "NE", "WC", "BC", "NA" };
private TableColumn militaryColumn;
private String[] militaryTypes = { "Never Served", "Serving", "Served",
                "NA" };
private TableColumn religionColumn;
private String[] religionTypes = { "Mld", "Rad", "NA" };
private TableColumn skillColumn;
private String[] skillTypes = { "PH", "WR", "EL", "ME", "CO", "DR", "FI",
                "NA" };
private TableColumn addressColumn;
private String[] addressTypes = { "TSV", "VNV", "VV", "HAV", "CCV", "GT",
                "UV", "HOV", "NA" };
private boolean[] editArray;
private String title = "Global Graph Person Search";
GGPerson[] people = null;

public GGDataTable(Object[] data) {
        people = (GGPerson[]) data;
        editArray = new boolean[people.length];
        resetEditArray();

}

protected void resetEditArray() {
        for (int i = 0; i < editArray.length; i++) {
                editArray[i] = false;
        }
}

public void setRowEdited(int i) {
        editArray[i] = true;
}

public boolean isRowEdited(int i) {
        return editArray[i];
}

public String getTitle() {
        return title;
}

public int getRowCount() {
        return people.length;
}

public GGPerson[] getPeople() {
        return people;
}

public JTable populateTable(JTable t, DefaultTableModel m) {

        JTable tmp = t;

        ColumnProperties.initweights();

        for (int i = 0; i < people.length; i++) {
                tmp.setValueAt(people[i].getName(), i, 0);
                tmp.setValueAt(people[i].getColorString(), i, 1);
                tmp.setValueAt(i + 1, i, 2);

                people[i].setID(i + 1);

                tmp.setValueAt(people[i].getTribe(), i, 3);
                tmp.setValueAt(people[i].getAge(), i, 4);
                tmp.setValueAt(people[i].getGenderString(), i, 5);
                tmp.setValueAt(people[i].getMaritalStatusString(), i, 6);
                tmp.setValueAt(people[i].getNationalityString(), i, 7);
                tmp.setValueAt(people[i].getPlaceOfBirthString(), i, 8);
                tmp.setValueAt(people[i].getVerboseAttitude(), i, 9);
                tmp.setValueAt(people[i].getAllegiance(), i, 10);
                tmp.setValueAt(people[i].getCriminalRecString(), i, 11);
```

```
            tmp.setValueAt(people[i].getEducationString(), i, 12);
            tmp.setValueAt(people[i].getEmployment(), i, 13);
            tmp.setValueAt(people[i].getMilRecord(), i, 14);
            tmp.setValueAt(people[i].getReligionString(), i, 15);
            tmp.setValueAt(people[i].getSkill(), i, 16);
            tmp.setValueAt(people[i].getAddress(), i, 17);
            tmp.setValueAt(people[i].getEquipment(), i, 18);
            tmp.setValueAt(people[i].getVehicle(), i, 19);
    }

    nameColumn = tmp.getColumnModel().getColumn(0);
    JTextField nameFld = new JTextField(20);
    nameFld.setEditable(false);
    nameColumn.setCellEditor(new DefaultCellEditor(nameFld));

    colorColumn = tmp.getColumnModel().getColumn(1);
    JComboBox colorBox = new JComboBox(colorTypes);
    colorColumn.setCellEditor(new DefaultCellEditor(colorBox));

    idColumn = tmp.getColumnModel().getColumn(2);
    JTextField idFld = new JTextField(5);
    idFld.setEditable(false);
    idColumn.setCellEditor(new DefaultCellEditor(idFld));

    tribeColumn = tmp.getColumnModel().getColumn(3);
    JComboBox tribeBox = new JComboBox(tribeTypes);
    tribeColumn.setCellEditor(new DefaultCellEditor(tribeBox));

    // no editor for age

    genderColumn = tmp.getColumnModel().getColumn(5);
    JComboBox genderBox = new JComboBox(genderTypes);
    genderColumn.setCellEditor(new DefaultCellEditor(genderBox));

    maritalColumn = tmp.getColumnModel().getColumn(6);
    JComboBox maritalBox = new JComboBox(maritalTypes);
    maritalColumn.setCellEditor(new DefaultCellEditor(maritalBox));

    nationalityColumn = tmp.getColumnModel().getColumn(7);
    JComboBox nationalityBox = new JComboBox(nationalityTypes);
    nationalityColumn.setCellEditor(new DefaultCellEditor(nationalityBox));

    pobColumn = tmp.getColumnModel().getColumn(8);
    JComboBox pobBox = new JComboBox(pobTypes);
    pobColumn.setCellEditor(new DefaultCellEditor(pobBox));

    affiliationColumn = tmp.getColumnModel().getColumn(9);
    JComboBox affiliationBox = new JComboBox(affiliationTypes);
    affiliationColumn.setCellEditor(new DefaultCellEditor(affiliationBox));

    allegianceColumn = tmp.getColumnModel().getColumn(10);
    JComboBox allegianceBox = new JComboBox(allegianceTypes);
    allegianceColumn.setCellEditor(new DefaultCellEditor(allegianceBox));

    criminalColumn = tmp.getColumnModel().getColumn(11);
    JComboBox criminalBox = new JComboBox(criminalTypes);
    criminalColumn.setCellEditor(new DefaultCellEditor(criminalBox));

    educationColumn = tmp.getColumnModel().getColumn(12);
    JComboBox educationBox = new JComboBox(educationTypes);
    educationColumn.setCellEditor(new DefaultCellEditor(educationBox));

    employmentColumn = tmp.getColumnModel().getColumn(13);
    JComboBox employmentBox = new JComboBox(employmentTypes);
    employmentColumn.setCellEditor(new DefaultCellEditor(employmentBox));

    militaryColumn = tmp.getColumnModel().getColumn(14);
    JComboBox militaryBox = new JComboBox(militaryTypes);
    militaryColumn.setCellEditor(new DefaultCellEditor(militaryBox));

    religionColumn = tmp.getColumnModel().getColumn(15);
    JComboBox religionBox = new JComboBox(religionTypes);
    religionColumn.setCellEditor(new DefaultCellEditor(religionBox));

    skillColumn = tmp.getColumnModel().getColumn(16);
    JComboBox skillBox = new JComboBox(skillTypes);
    skillColumn.setCellEditor(new DefaultCellEditor(skillBox));

    addressColumn = tmp.getColumnModel().getColumn(17);
    JComboBox addressBox = new JComboBox(addressTypes);
    addressColumn.setCellEditor(new DefaultCellEditor(addressBox));
```

```
        equipmentColumn = tmp.getColumnModel().getColumn(18);
        JComboBox equipmentBox = new JComboBox(equipmentTypes);
        equipmentColumn.setCellEditor(new DefaultCellEditor(equipmentBox));

        vehicleColumn = tmp.getColumnModel().getColumn(19);
        JComboBox vehicleBox = new JComboBox(vehicleTypes);
        vehicleColumn.setCellEditor(new DefaultCellEditor(vehicleBox));

        return tmp;

}

public void handleChange(int col, int row, DefaultTableModel t) {
        int person = row;

        switch (col) {
        case 0:
                people[person].setName((String) t.getValueAt(row, col));
                t.setValueAt(people[person].getName(), row, col);
                people[person].updateNode(col);
                setRowEdited(person);
                break;
        case 1:
                people[person].setColor((String) t.getValueAt(row, col));
                people[person].setOrganizationByColor(people[person]
                                .getColorString());
                t.setValueAt(people[person].getColorString(), row, col);
                people[person].updateNode(col);
                setRowEdited(person);
                break;
        case 2:
                t.setValueAt(people[person].getID(), row, col);
                // people[person].updateNode(col);
                break;
        case 3:
                people[person].setTribe((String) t.getValueAt(row, col));
                t.setValueAt(people[person].getTribe(), row, col);
                people[person].updateNode(col);
                setRowEdited(person);
                break;
        case 4:
                try {
                        if (Integer.parseInt((String) t.getValueAt(row, col)) >= 0) {
                                people[person].setAge((String) t.getValueAt(row, col));
                                t.setValueAt(people[person].getAge(), row, col);
                                people[person].updateNode(col);
                                setRowEdited(person);
                        } else
                                t.setValueAt(people[person].getAge(), row, col);
                } catch (Exception intEx) {
                        t.setValueAt(people[person].getAge(), row, col);
                }
                break;
        case 5:
                people[person].setGender((String) t.getValueAt(row, col));
                t.setValueAt(people[person].getGenderString(), row, col);
                people[person].updateNode(col);
                setRowEdited(person);
                break;
        case 6:
                people[person].setMaritalStatus((String) t.getValueAt(row, col));
                t.setValueAt(people[person].getMaritalStatusString(), row, col);
                people[person].updateNode(col);
                setRowEdited(person);
                break;
        case 7:
                people[person].setNationality((String) t.getValueAt(row, col));
                t.setValueAt(people[person].getNationalityString(), row, col);
                people[person].updateNode(col);
                setRowEdited(person);
                break;
        case 8:
                people[person].setPlaceOfBirth((String) t.getValueAt(row, col));
                t.setValueAt(people[person].getPlaceOfBirthString(), row, col);
                people[person].updateNode(col);
                setRowEdited(person);
                break;
        case 9:
                people[person].setVerboseAttitude((String) t.getValueAt(row, col));
                t.setValueAt(people[person].getVerboseAttitude(), row, col);
```

```
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 10:
                        people[person].setAllegiance((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getAllegiance(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 11:
                        people[person].setCriminalRec((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getCriminalRecString(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 12:
                        people[person].setEducation((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getEducationString(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 13:
                        people[person].setEmployment((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getEmployment(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 14:
                        people[person].setMilRecord((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getMilRecord(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 15:
                        people[person].setReligion((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getReligionString(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 16:
                        people[person].setSkill((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getSkill(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 17:
                        people[person].setAddress((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getAddress(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 18:
                        people[person].setEquipment((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getEquipment(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                case 19:
                        people[person].setVehicle((String) t.getValueAt(row, col));
                        t.setValueAt(people[person].getVehicle(), row, col);
                        people[person].updateNode(col);
                        setRowEdited(person);
                        break;
                default:
                        break;
                }
        }

}
```

```
package hdpt;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.Header;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.auth.DigestScheme;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicHeader;
import org.apache.http.message.BasicHttpRequest;
import org.apache.http.protocol.BasicHttpContext;
import org.apache.http.protocol.HTTP;
import org.apache.http.util.EntityUtils;
import org.apache.http.impl.conn.BasicClientConnectionManager;
import org.apache.http.conn.scheme.PlainSocketFactory;

public class GGSearch {
	HDPT callback;
	public GGSearch(HDPT app) {
		callback = app;
	}
public Object[] GGRestSearch(int srchType, Object[] latlon)
	throws Exception {
		JsonHandler handleJSON = new JsonHandler(callback);
		// Set up the client and send the request
		// Code to ignore SSL
		SSLSocketFactory sf = new SSLSocketFactory(new TrustStrategy() {
			public boolean isTrusted(final X509Certificate[] chain,
					String authType) throws CertificateException {
				return true;
			}

		});

		Scheme httpsScheme = new Scheme("https", 443, sf);
		Scheme httpScheme = new Scheme("http", 80, new PlainSocketFactory());
		SchemeRegistry schemeRegistry = new SchemeRegistry();
		schemeRegistry.register(httpsScheme);
		schemeRegistry.register(httpScheme);

		ClientConnectionManager cm = new BasicClientConnectionManager(
				schemeRegistry);

		// String url = "http://10.1.200.82:8443/gg/search";
		// String url = "https://zulu.idvrn.arl.army.mil:8443/gg/search";
		// String url = "http://localhost:8080/gg/search";

		String url = callback.getGGHost() + "/gg/search";
		// System.out.println(url);
		HttpPost post = new HttpPost(url);

		post.setHeader("Accept", "application/json");
		post.setHeader("Content-Type", "application/json");
		post.setHeader("User-Agent", "Jakarta Commons-HttpClient/3.1");

		StringEntity se = new StringEntity(handleJSON.buildJSONSearch(srchType,
				latlon));
		se.setContentEncoding(new BasicHeader(HTTP.CONTENT_TYPE,
				"application/json"));
		post.setEntity(se);

		HttpClient client = new DefaultHttpClient(cm);

		HttpResponse response = client.execute(post);

		// System.out.println(response.getStatusLine());

		if (response.getStatusLine().getStatusCode() == HttpStatus.SC_UNAUTHORIZED) {
			if (response.containsHeader("WWW-Authenticate")) {
				final Header challengeHeader = response
						.getHeaders("WWW-Authenticate")[0];
				DigestScheme ds = new DigestScheme();
```

```
                                ds.processChallenge(challengeHeader);
                                final Header authHeader = ds.authenticate(
                                        // new UsernamePasswordCredentials(callback.getUser(),
                                                // "ggsecret"),
                                                new UsernamePasswordCredentials(callback.getUser(),
                                                        callback.getPwd()), new BasicHttpRequest(
                                                                HttpPost.METHOD_NAME, url),
                                                new BasicHttpContext());

                                post.addHeader(authHeader);
                                EntityUtils.consume(response.getEntity());

                                HttpClient newClient = new DefaultHttpClient(cm);
                                response = newClient.execute(post);

                                // System.out.println(response);
                                System.out.println("GG Search Connection Status: "
                                                + response.getStatusLine());

                        }
                }

                Object[] result = handleJSON.processGGSearch(response);
                EntityUtils.consume(response.getEntity());
                return result;

        }

}
package hdpt;

import java.util.Iterator;

import org.apache.http.HttpResponse;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.ObjectMapper;
import org.codehaus.jackson.node.ArrayNode;
import org.codehaus.jackson.node.ObjectNode;

public class JsonHandler {
        private GGPerson[] ggPersonArray = null;
        private ObjectMapper objectMapper = null;

        // private HDPT callback;

        public JsonHandler(HDPT hdpt) {
                objectMapper = new ObjectMapper();
                // callback = hdpt;
        }

        public String buildJSONSearch(int srchType, Object[] latlon) {

                // The top-level JSON object
                final ObjectNode jsonObj = objectMapper.createObjectNode();

                // The root object
                final ObjectNode rootObject = objectMapper.createObjectNode();

                // The conditions array
                ArrayNode conditionsArray = objectMapper.createArrayNode();

                // The first element in the conditions array
                ObjectNode conditionsObject = objectMapper.createObjectNode();

                switch (srchType) {
                case HDPT.ALL:
                        conditionsObject.put("property", "Person.remarks.subject");
                        conditionsObject.put("operator", "LIKE");
                        conditionsObject.put("value", "C4ISR OTM");
                        break;
                case HDPT.MAP:
                        conditionsObject.put("property", "Person.locations");
                        conditionsObject.put("operator", "BOUNDED_BY");
                        ArrayNode valueArray = objectMapper.createArrayNode();
                        valueArray.add("(" + latlon[0] + "," + latlon[1] + ")");
                        valueArray.add("(" + latlon[2] + "," + latlon[3] + ")");
                        conditionsObject.put("value", valueArray);
                        break;
                case HDPT.UPDATE:
                        conditionsObject.put("property", "Person.names.fullName");
                        conditionsObject.put("operator", "LIKE");
```

```
                conditionsObject.put("value", "*"); // replace * with name person to find
                break;
        }
        /*
         * if(latlon == null){
         * conditionsObject.put("property","Person.remarks.subject");
         * conditionsObject.put("operator","LIKE");
         * conditionsObject.put("value","C4ISR OTM");
         *
         * } else{ conditionsObject.put("property","Person.locations");
         * conditionsObject.put("operator","BOUNDED_BY"); ArrayNode valueArray =
         * objectMapper.createArrayNode(); valueArray.add("(" + latlon[0] + ","
         * + latlon[1] +")"); valueArray.add("(" + latlon[2] + "," + latlon[3]
         * +")"); conditionsObject.put("value", valueArray); }
         */
        conditionsArray.add(conditionsObject);
        rootObject.put("conditions", conditionsArray);
        jsonObj.put("maxresults", 200);
        jsonObj.put("root", rootObject);

        String jsonString = jsonObj.toString();
        // System.out.println(jsonString);

        return jsonString;

}

public Object[] processGGSearch(HttpResponse response) {
        JsonNode rootNode = null;

        try {
                rootNode = objectMapper.readValue(
                                response.getEntity().getContent(), JsonNode.class);
        } catch (Exception jsonEx) {
                System.out.println("There is a problem with the JSON search response:\n\n"
                                                + jsonEx.getStackTrace());
                return ggPersonArray;

        }

        if (rootNode != null) {
                Iterator<JsonNode> nodes = rootNode.getElements();
                // System.out.println(rootNode);
                nodes = (nodes.next()).getElements();
                JsonNode personArray = nodes.next();

                JsonNode personNode;
                GGPerson newPerson;
                ggPersonArray = new GGPerson[personArray.size()];
                for (int i = 0; i < personArray.size(); i++) {
                        newPerson = new GGPerson();
                        personNode = personArray.get(i);
                        newPerson.setNode(personNode);

                        // System.out.println(personNode + "\n\n");

                        try {
                                newPerson.setName(personNode.get("names").getElements()
                                                .next().get("fullName").getTextValue());
                        } catch (Exception nameEx) {
                                newPerson.setName("NA");
                        }

                        try {
                                newPerson.setOrganization(personNode.get("affiliations")
                                                .getElements().next().get("affiliation")
                                                .getTextValue());
                        } catch (Exception orgEx) {
                                newPerson.setOrganization("NA");
                        }

                        try {
                                newPerson.setTribe(personNode.get("ethnicity")
                                                .getTextValue());
                        } catch (Exception tribeEx) {
                                newPerson.setTribe("NA");
                        }

                        try {
                                newPerson.setAge(personNode.get("age").asText());
                        } catch (Exception ageEx) {
```

```
                newPerson.setAge("NA");
        }

        try {
                newPerson.setGender(personNode.get("gender")
                                .get("physicalValue").getTextValue());
        } catch (Exception genderEx) {
                newPerson.setGender("NA");
        }

        try {
                newPerson.setMaritalStatus(personNode.get("maritalStatus")
                                .getTextValue());
        } catch (Exception maritalEx) {
                newPerson.setMaritalStatus("NA");
        }

        try {
                newPerson.setNationality(personNode.get("nationality")
                                .get("physicalValue").getTextValue());
        } catch (Exception natEx) {
                newPerson.setNationality("NA");
        }

        try {
                newPerson.setPlaceOfBirth(personNode.get("placeOfBirth")
                                .getTextValue());
        } catch (Exception pobEx) {
                newPerson.setPlaceOfBirth("NA");
        }

        try {
                newPerson.setMilRecord(personNode.get("militaryService")
                                .getElements().next().get("dutyOrPosition")
                                .getTextValue());
        } catch (Exception milEx) {
                newPerson.setMilRecord("NA");
        }

        try {
                newPerson.setReligion(personNode.get("religions")
                                .getElements().next().get("religionName")
                                .getTextValue());
        } catch (Exception relEx) {
                newPerson.setReligion("NA");
        }

        try {
                newPerson.setSkill(personNode.get("skills").getElements()
                                .next().get("skill").getTextValue());
        } catch (Exception skillEx) {
                newPerson.setSkill("NA");
        }

        try {
                newPerson.setAddress(personNode.get("addresses")

        .getElements().next().get("city").getTextValue());
        } catch (Exception addrEx) {
                newPerson.setAddress("NA");
        }

        try {
                newPerson.setEmployment(personNode.get("employment")
                                .getElements().next().get("employerType")
                                .getTextValue());
        } catch (Exception empEx) {
                newPerson.setEmployment("NA");
        }

        try {
                newPerson.setAllegiance(personNode.get("affiliations")

.getElements().next().get("affiliationGroupName")
                                .getTextValue());
        } catch (Exception allEx) {
                newPerson.setAllegiance("NA");
        }

        try {
                newPerson.setAttitude(personNode.get("affiliations")
```

```
                        .getElements().next().get("natureOfAffiliation")
                                                .getTextValue());

                        } catch (Exception attEx) {
                                newPerson.setAttitude("NA");
                        }

                        try {
                                newPerson

                        .setCriminalRec(personNode.get("criminalRecords")

                        .getElements().next().get("verdict")
                                                                .getTextValue());
                        } catch (Exception crmEx) {
                                newPerson.setCriminalRec("NA");
                        }
                        try {
                                newPerson.setEducation(personNode.get("education")

                        .getElements().next().get("educationalLevel")
                                                .getTextValue());
                        } catch (Exception edEx) {
                                newPerson.setEducation("NA");
                        }
                        try {
                                newPerson.setVehicle(personNode.get("description")
                                                .getTextValue());
                        } catch (Exception vehEx) {
                                newPerson.setVehicle("NA");
                        }
                        try {
                                newPerson
                                                .setEquipment(personNode.get("remarks")
                        .getElements().next().get("details")
                                                                .getTextValue());
                        } catch (Exception eqEx) {
                                newPerson.setEquipment("NA");
                        }
                        try {
                                newPerson.setUUID(personNode.get("id").getTextValue());
                        } catch (Exception uuidEx) {
                                newPerson.setUUID("NA");
                        }

                        //
                        ggPersonArray[i] = newPerson;
                }
        }
        return ggPersonArray;
    }

    public String buildJSONUpdate(JsonNode node) {

        // The top-level JSON object
        ObjectNode updateObj = objectMapper.createObjectNode();

        // The root object
        ObjectNode entitiesObject = objectMapper.createObjectNode();

        // The conditions array
        ArrayNode personArray = objectMapper.createArrayNode();

        personArray.add(((ObjectNode) node));

        entitiesObject.put("Person", personArray);
        updateObj.put("entities", entitiesObject);

        String jsonString = updateObj.toString();

        // System.out.println(jsonString);
        return jsonString;

    }

    public Object[] getSearchResult() {
        return ggPersonArray;

    }
}
```

```
package hdpt;
import java.net.URI;
import java.security.cert.CertificateException;
import java.security.cert.X509Certificate;
import org.apache.http.conn.ssl.TrustStrategy;
import org.apache.http.Header;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.conn.ClientConnectionManager;
import org.apache.http.conn.scheme.Scheme;
import org.apache.http.conn.scheme.SchemeRegistry;
import org.apache.http.conn.ssl.SSLSocketFactory;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.auth.DigestScheme;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicHeader;
import org.apache.http.message.BasicHttpRequest;
import org.apache.http.protocol.BasicHttpContext;
import org.apache.http.protocol.HTTP;
import org.apache.http.util.EntityUtils;
import org.apache.http.impl.conn.BasicClientConnectionManager;
import org.apache.http.conn.scheme.PlainSocketFactory;

public class GGUpdate {
        HDPT callback;
        public GGUpdate(HDPT app) {
                callback = app;
        }
public void GGRestUpdate(GGDataTable data) throws Exception {
        JsonHandler handleJSON = new JsonHandler(callback);
                GGPerson[] people = data.getPeople();

                // Set up the client and send the request
                // Code to ignore SSL

                SSLSocketFactory sf = new SSLSocketFactory(new TrustStrategy() {
                        public boolean isTrusted(final X509Certificate[] chain,
                                        String authType) throws CertificateException {
                                return true;
                        }

                });

                Scheme httpsScheme = new Scheme("https", 443, sf);
                Scheme httpScheme = new Scheme("http", 80, new PlainSocketFactory());
                SchemeRegistry schemeRegistry = new SchemeRegistry();
                schemeRegistry.register(httpsScheme);
                schemeRegistry.register(httpScheme);

                ClientConnectionManager cm = new BasicClientConnectionManager(
                                schemeRegistry);

                HttpPost post = new HttpPost();
                post.setHeader("Accept", "application/json");
                post.setHeader("Content-Type", "application/json");
                post.setHeader("User-Agent", "Jakarta Commons-HttpClient/3.1");

                StringEntity se;
                HttpClient client;
                HttpResponse response;
                Header challengeHeader;
                DigestScheme ds;
                Header authHeader;
                HttpClient newClient;

                for (int i = 0; i < people.length; i++)
                        if (data.isRowEdited(i)) {
                String url = callback.getGGHost() + "/gg/entity/Person/"+ people[i].getUUID();
                                        // System.out.println(url);
                                        post.setURI(new URI(url));
                  se = new
                stringEntity(handleJSON.buildJSONUpdate(data.getPeople()[i].getNode()));
                se.setContentEncoding(new BasicHeader(HTTP.CONTENT_TYPE,    "application/json"));
                post.setEntity(se);
                client = new DefaultHttpClient(cm);
                response = client.execute(post);

                                // System.out.println(response.getStatusLine());
```

```
                                if (response.getStatusLine().getStatusCode() ==
                                HttpStatus.SC_UNAUTHORIZED) {
                                        if (response.containsHeader("WWW-Authenticate")) {
                                challengeHeader = response
                                                                .getHeaders("WWW-Authenticate")[0];
                                                ds = new DigestScheme();
                                                ds.processChallenge(challengeHeader);
                                                authHeader = ds
                                                                .authenticate(new
                                                        UsernamePasswordCredentials(
                                                                                        callback.getUser(),
                                                                                        callback.getPwd()),
                                                                                        new BasicHttpRequest(

                                                                                HttpPost.METHOD_NAME, url),
                                                                                new BasicHttpContext());

                                                post.addHeader(authHeader);
                                                EntityUtils.consume(response.getEntity());

                                                newClient = new DefaultHttpClient(cm);
                                                response = newClient.execute(post);

                                                // System.out.println(response);
                                                // System.out.println("GG Update Connection Status:
" +
                                                // response.getStatusLine());

                                        }
                                }

                                System.out.println("GG Update Connection Status: "
                                                + response.getStatusLine());
                                EntityUtils.consume(response.getEntity());
                        }
                data.resetEditArray();

        }

}
package hdpt;

import java.awt.BorderLayout;
import java.beans.PropertyVetoException;
import java.util.Vector;
import javax.swing.JInternalFrame;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

public class GGDissimFrame extends JInternalFrame {
        private static final long serialVersionUID = 1L;
        private JTable table;
        private JScrollPane scrollPane;
        private DefaultTableModel model;
        private Object[][] dmatrix;
        private HDPT callback;

        public GGDissimFrame(double[] diss, HDPT callback) {
                super("Dissimilarity Matrix");

                this.callback = callback;
                int rows = this.callback.getGGDataTable().getRowCount();

                dmatrix = new Object[rows][rows];

                int mem = 0;

                for (int q = 0; q < rows; q++) {
                        for (int u = q; u < rows; u++) {
                                if (u == q)
                                        dmatrix[u][q] = 0.0;
                                else
                                        // if (mem < diss.length)
                                        dmatrix[u][q] = diss[mem++];
                        }
                }

                for (int q = 0; q < rows; q++) {
                        for (int u = 0; u < rows; u++) {
                                if (u > q)
```

```
                                dmatrix[q][u] = dmatrix[u][q];
                        }
                }

                Object[] label = new Object[rows];

                for (int z = 1; z <= rows; z++) {
                        label[z - 1] = "Object" + z;
                }
                this.model = new DefaultTableModel(dmatrix, label);
                this.table = new JTable(model);
                this.table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                this.scrollPane = new JScrollPane(table);
                this.setContentPane(scrollPane);
                this.setSize(
                                callback.getContentPane().getSize().width / 2,
                                ((BorderLayout) callback.getContentPane().getLayout())

        .getLayoutComponent(BorderLayout.CENTER).getSize().height / 2);
                this.setLocation(
                                0,
                                ((BorderLayout) callback.getContentPane().getLayout())

        .getLayoutComponent(BorderLayout.CENTER).getSize().height / 2);
                this.setVisible(true);
                this.setResizable(true);
                this.setMaximizable(true);
                this.setIconifiable(true);
                this.setClosable(true);
                this.setDefaultCloseOperation(JInternalFrame.DISPOSE_ON_CLOSE);
                try {
                        this.setIcon(true);
                } catch (PropertyVetoException e) {
                        e.printStackTrace();
                }
        }
        @SuppressWarnings("unchecked")
        public Vector<Vector<Object>> getCellData() {
                return model.getDataVector();
        }

}
package hdpt;

import org.rosuda.REngine.*;
import org.rosuda.REngine.Rserve.RConnection;

public class GGMDS {
        private REngine engine;
        private RConnection c;
        private GenerateMDSEquation gen;
        private GGDataTable data;

        public GGMDS(HDPT callback) throws Exception {

                c = new RConnection(callback.getRserveHost());
                // c = new RConnection("zulu.idvrn.arl.army.mil");
                // c = new RConnection("10.1.200.87");
                REXP x = c.eval("R.version.string");
                System.out.println(x.asString());

                gen = new GenerateMDSEquation();

        }

        public double[] generateDissimMatrix(GGDataTable in) throws Exception {

                data = in;
                REXP x = null;
                double[] diss = null;

                // Load the FD library that contains the gowdis function
                c.parseAndEval("library(\"FD\");");

                // Clear out return variable so that old results are not propagated
                // during an error
                c.parseAndEval("m <- NULL");

                // Build the attribute vectors to add be added to a dataframe
                c.parseAndEval(gen.buildVectors(data));
```

```
		// Generate an R expression to create a data frame containing all
		// the vectors that have been created
		String gower = "df <- data.frame(";
		String unique = null;

		REXP uniqueTest;
		int[] weights = new int[ColumnProperties.getAttributeCount()];
		for (int i = 0; i < ColumnProperties.getAttributeCount(); i++) {
			// Check to make sure vector does not contain all same values which
			// would cause an R error
			unique = "b <- length(unique(a" + i + "))";
			uniqueTest = c.parseAndEval(unique);
			// System.out.println(uniqueTest.asInteger());
			if (uniqueTest.asInteger() > 1) {
				gower += "a" + i + ",";
				weights[i] = ColumnProperties.getWeight(i);
			} else
				weights[i] = -1;
		}
		gower += ")";
		gower = gower.replace(",)", ")");
		// System.out.println(gower);
		c.parseAndEval(gower);

		// Create the weight vector
		c.parseAndEval(gen.buildWeight(weights));

		// Generate an R expression to evaluate the gowdis function using
		// the data frame and weights that have been created
		c.parseAndEval("m <- gowdis(df,w, ord='podani')");
		x = c.parseAndEval("m");

		diss = x.asDoubles();
		// System.out.println(x);

		return diss;
	}

	public double[][] calculateCoordinates() throws Exception {

		double[][] xyz = null;
		REXP mds = null;

		c.parseAndEval("x <- NULL");
		c.parseAndEval("library(\"MASS\");");
		c.parseAndEval("x <- cmdscale(m, k=3, add=TRUE)$points");
		mds = c.parseAndEval("x");
		double[] coords = mds.asDoubles();

		int zLim = coords.length;
		int xLim = zLim / 3;
		int yLim = (2 * zLim) / 3;

		double[] xCoord = new double[zLim / 3];
		double[] yCoord = new double[zLim / 3];
		double[] zCoord = new double[zLim / 3];

		int index;
		for (index = 0; index < xLim; index++)
			xCoord[index] = coords[index];

		for (index = xLim; index < yLim; index++)
			yCoord[index - xLim] = coords[index];

		for (index = yLim; index < zLim; index++)
			zCoord[index - yLim] = coords[index];

		xyz = new double[][] { xCoord, yCoord, zCoord };

		c.close();
		return xyz;

	}

	public void getType() {
		try {
			engine.parseAndEval("attributes(m)");

		} catch (REngineException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
```

```
                } catch (REXPMismatchException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
        }

}
package hdpt;
import java.util.ArrayList;
import java.util.List;
public class GenerateMDSEquation {
        private GGDataTable data;
        private GGPerson[] people = null;
        public String buildVectors(GGDataTable in) {
                data = in;
                people = data.getPeople();
                String input = "";
                String result = "";
                Object a = null;
                try {
                int personCount = people.length;
                int attributeCount = ColumnProperties.getAttributeCount();
                //Iterate through the data points and create R expressions to
                //assign vectors for each data column
                boolean isOrdinal = false;
                boolean isNominal = false;
                String[] types = ColumnProperties.getDataTypes();

                        for (int i = 0; i < attributeCount; i++) {
                                input = "";
                                if (types[i].compareTo("ORD") == 0){
                                        isOrdinal = true;
                                        input = input + "a" + (i) + " <- ordered(c(";
                                }
                                else{
                                        isOrdinal = false;
                                        input = input + "a" + (i) + " <- c(";
                                }
                                //add '' around nominal text values
                                if (types[i].compareTo("NOM") == 0)
                                        isNominal = true;
                                else
                                        isNominal = false;


                                for( int p = 0; p < personCount; p++)
                                {

                                        switch(i){
                                        case 0:a = people[p].getTribe();
                                        break;
                                        case 1:a = people[p].getAge();
                                        break;
                                        case 2:a = people[p].getGenderBinary();
                                        break;
                                        case 3:a = people[p].getMaritalStatusBinary();
                                        break;
                                        case 4:a = people[p].getNationalityBinary();
                                        break;
                                        case 5:a = people[p].getPlaceOfBirthBinary();
                                        break;
                                        case 6:a = people[p].getAttitude();
                                        break;
                                        case 7:a = people[p].getAllegiance();
                                        break;
                                        case 8:a = people[p].getCriminalRecBinary();
                                        break;
                                        case 9:a = people[p].getEducationBinary();
                                        break;
                                        case 10:a = people[p].getEmployment();
                                        break;
                                        case 11:a = people[p].getMilRecord();
                                        break;
                                        case 12:a = people[p].getReligionBinary();
                                        break;
                                        case 13:a = people[p].getSkill();
                                        break;
                                        case 14:a = people[p].getAddress();
                                        break;
                                        case 15:a = people[p].getEquipment();
                                        break;
```

```
                                        case 16:a = people[p].getVehicle();
                                        break;
                                        default:break;
                                        }

                                        if (a.toString().equals("-1"))
                                                a = "NA";
                                        String tmp = a.toString().trim();
                                        if(isNominal && !tmp.startsWith("'") && !tmp.equals("NA"))
                                                //if(isNominal && !tmp.startsWith("'"))
                                                input += "'" + a.toString().trim() + "'";
                                        else
                                                input +=a.toString().trim();
                                        input += ",";


                                }


                                if(isOrdinal)
                                        input += "))\n";
                                else
                                        input += ")\n";

                                input = input.replace(",)", ")");
                                //System.out.println(input.length());
                                //TODO:  FIX inputs that are to big or find out how to change R
input buffer
                                if(input.length() < 2000)
                                        result = result += input;

                        }


                }catch(Exception e){
                        System.out.println("Error in MDS.generateDissimMatrix communicating with
R.");
                }

                //System.out.println(result);
                return result;

        }


        public String buildWeight(int[] in){

                String weight = "w <- c(";
                for(int i = 0; i < in.length; i++){
                        //System.out.println(data.getWeight(i));
                        if (in[i] != -1){
                                weight+=in[i];
                                if(i+1 != in.length)
                                        weight += (",");
                        }
                }
                if(weight.endsWith(","))
                        weight = weight.substring(0,weight.lastIndexOf(","));
                weight += (")");

                //System.out.println(weight);
                return weight;

        }

        //eventually check command line length for feeding into R to keep it less than buffer:
usually 1024
        public static List<String> splitEqually(String text, int size) {
                // Give the list the right capacity to start with. You could use an array
                // instead if you wanted.
                List<String> ret = new ArrayList<String>((text.length() + size - 1) / size);

                for (int start = 0; start < text.length(); start += size) {
                        ret.add(text.substring(start, Math.min(text.length(), start + size)));
                }
                return ret;
        }



}
```

```
package hdpt;

import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.*;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;



public class GGPreferencesFrame extends JInternalFrame {

	private static final long serialVersionUID = 1L;
	private HDPT callback;
	private JCheckBox thresholdBox;
	private JSlider thresholdSlider;
	private JPanel prefPanel;
	private JLabel linkLabel;
	private JLabel disabledLabel;
	private JSlider linkSlider;
	public static int attributeThreshold = 5;
	public static int maxLinks = 3;
	public static boolean useThreshold = false;
	private String boxString = "Use attribute threshold to filter links";

	public GGPreferencesFrame(HDPT _callback){
		this.callback = _callback;
		this.setTitle("Preferences");

		//this.setContentPane(scrollPane);

		this.setSize(callback.getContentPane().getSize().width/4,
callback.getContentPane().getSize().height/4);
		this.setLocation(100,100);
		this.setVisible(true);
		this.setResizable(true);
		this.setMaximizable(true);
		this.setIconifiable(true);
		this.setClosable(true);
		this.setDefaultCloseOperation(JInternalFrame.DISPOSE_ON_CLOSE);

		setupPanel();
		setContentPane(prefPanel);

	}


	private void setupPanel(){
		prefPanel = new JPanel();
		prefPanel.setLayout(new BoxLayout(prefPanel,BoxLayout.Y_AXIS));

		thresholdBox = new JCheckBox(boxString, false);
		thresholdBox.addActionListener(new ggThresholdListener(callback));
		prefPanel.add(thresholdBox);
		thresholdBox.setEnabled(false);

		thresholdSlider = new JSlider(JSlider.HORIZONTAL, 0,
ColumnProperties.getAttributeCount(), 5 );
		thresholdSlider.addChangeListener(new ggThresholdCountListener(callback));
		thresholdSlider.setMajorTickSpacing(5);
		thresholdSlider.setMinorTickSpacing(1);
		thresholdSlider.setPaintTicks(true);
		thresholdSlider.setPaintLabels(true);
		prefPanel.add(thresholdSlider);
		thresholdSlider.setEnabled(false);

		prefPanel.add(Box.createRigidArea(new Dimension(0, 25)));

		linkLabel = new JLabel("Change Max Links Displayed");
		disabledLabel = new JLabel("DISABLED");
		prefPanel.add(linkLabel);

		if(callback.dataExists()){
			linkSlider = new JSlider(JSlider.HORIZONTAL, 0,
callback.getGGDataTable().getRowCount() - 1, 3 );
			linkSlider.addChangeListener(new ggLinkCountListener(callback));
			linkSlider.setMajorTickSpacing(5);
			linkSlider.setMinorTickSpacing(1);
			linkSlider.setPaintTicks(true);
```

```
                linkSlider.setPaintLabels(true);
                prefPanel.add(linkSlider);
        }
        else
                prefPanel.add(disabledLabel);


    }


    public class ggThresholdListener implements ActionListener{

        HDPT callback;

        public ggThresholdListener(HDPT _callback) {
                this.callback = _callback;
        }

        public void actionPerformed (ActionEvent e) {
                if(thresholdBox.isSelected()){
                        useThreshold = true;
                        thresholdSlider.setEnabled(true);
                }
                else{
                        useThreshold = false;
                        thresholdSlider.setEnabled(false);
                        //System.out.println(useThreshold);
                }

        }
    }

    public class ggThresholdCountListener implements ChangeListener{

        HDPT callback;

        public ggThresholdCountListener(HDPT _callback) {
                this.callback = _callback;
        }

        public void stateChanged(ChangeEvent e) {
                JSlider source = (JSlider)e.getSource();

                if(!source.getValueIsAdjusting())
                        attributeThreshold = (int)source.getValue();


        }

    }
    public class ggLinkCountListener implements ChangeListener{

        HDPT callback;

        public ggLinkCountListener(HDPT _callback) {
                this.callback = _callback;
        }

        public void stateChanged(ChangeEvent e) {
                JSlider source = (JSlider)e.getSource();

                if(!source.getValueIsAdjusting())
                        maxLinks = (int)source.getValue();


        }

    }

}
```

```
package hdpt;

import javax.swing.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;
import javax.swing.table.DefaultTableModel;

public class GGPropertiesFrame extends JInternalFrame {

	private static final long serialVersionUID = 1L;
	private JTable table;
	private DefaultTableModel model;
	private JScrollPane scrollPane;

	public GGPropertiesFrame(HDPT callback) {
		// this.callback = callback;
		this.setTitle("Column Properties");
		this.model = new DefaultTableModel(new String[] { "Column Name",
				"Data Type", "Weight" }, ColumnProperties.getAttributeCount());
		this.table = new JTable(model);
		this.table = ColumnProperties.populateTable(table, model);
		this.table.setAutoCreateRowSorter(true);
		this.table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
		this.table.getModel().addTableModelListener(new dataTableListener());
		this.scrollPane = new JScrollPane(table);
		this.setContentPane(scrollPane);

		this.setSize(callback.getContentPane().getSize().width / 4, callback
				.getContentPane().getSize().height / 4);
		// this.setLocation(callback.getContentPane().getSize().width/2,100);
		this.setLocation(100, 100);
		this.setVisible(true);
		this.setResizable(true);
		this.setMaximizable(true);
		this.setIconifiable(true);
		this.setClosable(true);
		this.setDefaultCloseOperation(JInternalFrame.DISPOSE_ON_CLOSE);

	}

	public class dataTableListener implements TableModelListener {

		public dataTableListener() {
		}

		public void tableChanged(TableModelEvent e) {

			int col = e.getColumn();
			int row = e.getFirstRow();
			// System.out.println(row);
			model.removeTableModelListener(this);

			int newWeight;
			if (col == 1)
				;// do nothing now, maybe allow data type change in the future

			if (col == 2) {

				try {
					newWeight = Integer.parseInt((String) model.getValueAt(row,
							col));
					System.out.println(row);
					ColumnProperties.setWeight(row, newWeight);

				} catch (Exception ex) {
					System.out.println(ex.getMessage());
					model.setValueAt(ColumnProperties.getWeight(row), row,
col);
				}
			}

			model.addTableModelListener(this);
		}

	}

}
```

```
package hdpt;

import java.awt.Color;

import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.node.ArrayNode;
import org.codehaus.jackson.node.ObjectNode;

public class GGPerson {
	private String name = null;
	private String organization = null;
	private String address = null;
	private String employment = null;
	private String allegiance = null;
	private String tribe = null;
	private String nationality = null;
	private String attitude = null;
	private String verboseAttitude = null;
	private String milRec = null;
	private String skill = null;
	private String age = null;
	private String gender = null;
	private String maritalStatus = null;
	private String placeOfBirth = null;
	private String religion = null;
	private String criminalRec = null;
	private String education = null;
	private String equipment = null;
	private String vehicle = null;
	private int nationalityBinary = -1;
	private int genderBinary = -1;
	private int maritalStatusBinary = -1;
	private int placeOfBirthBinary = -1;
	private int religionBinary = -1;
	private int criminalRecBinary = -1;
	private int educationBinary = -1;
	private Color plotColor = Color.BLACK;
	private String plotColorString = "BLACK";
	private int id = -1;
	private String uuid = null;
	private JsonNode node = null;
	private double lat;
	private double lon;

	// node set/get
	protected void setNode(JsonNode n) {
		node = n;
	}

	public JsonNode getNode() {
		return node;
	}

	// uuid set/get
	protected void setUUID(String id) {
		uuid = id;
	}

	public String getUUID() {
		return uuid;
	}

	// Name set/get
	public void setName(String nm) {
		name = nm;
	}

	public String getName() {
		return name;
	}

	// Org set/get -> determines node color in scatter plot
	public void setOrganization(String org) {
		organization = org;
		setColor();
	}

	public void setOrganizationByColor(String org) {
		if (plotColorString.equalsIgnoreCase("RED"))
			organization = "Insurgent";
		else if (plotColorString.equalsIgnoreCase("ORANGE"))
```

```
                organization = "Criminal";
        else if (plotColorString.equalsIgnoreCase("GREEN"))
                organization = "Friendly";
        else if (plotColorString.equalsIgnoreCase("BLUE"))
                organization = "Mobile";
        else if (plotColorString.equalsIgnoreCase("BLACK"))
                organization = "Unknown";
        else
                organization = "Unknown";
}

public String getOrganization() {
        return organization;
}

public String getColorString() {
        return plotColorString;

}

public Color getColor() {
        return plotColor;
}

public void setColor() {
        if (organization.equalsIgnoreCase("Criminal")) {
                plotColor = Color.ORANGE;
                plotColorString = "ORANGE";
        } else if (organization.equalsIgnoreCase("Insurgent")) {
                plotColor = Color.RED;
                plotColorString = "RED";
        } else if (organization.equalsIgnoreCase("Friendly")) {
                plotColor = Color.GREEN;
                plotColorString = "GREEN";
        } else if (organization.equalsIgnoreCase("Mobile")) {
                plotColor = Color.BLUE;
                plotColorString = "BLUE";
        } else if (organization.equalsIgnoreCase("Unknown")) {
                plotColor = Color.BLACK;
                plotColorString = "BLACK";
        } else {
                plotColor = Color.BLACK;
                plotColorString = "BLACK";
        }
}

public void setColor(String c) {
        if (c.equalsIgnoreCase("GREEN")) {
                plotColor = Color.GREEN;
                plotColorString = "GREEN";
        } else if (c.equalsIgnoreCase("BLACK")) {
                plotColor = Color.BLACK;
                plotColorString = "BLACK";
        } else if (c.equalsIgnoreCase("BLUE")) {
                plotColor = Color.BLUE;
                plotColorString = "BLUE";
        } else if (c.equalsIgnoreCase("ORANGE")) {
                plotColor = Color.ORANGE;
                plotColorString = "ORANGE";
        } else if (c.equalsIgnoreCase("RED")) {
                plotColor = Color.RED;
                plotColorString = "RED";
        } else {
                plotColor = Color.BLACK;
                plotColorString = "BLACK";
        }
}

// Tribe set/get
public void setTribe(String trb) {
        tribe = trb;
}

public String getTribe() {
        return tribe;
}

// Age set/get
public void setAge(String num) {
        age = num;
}
```

```
public String getAge() {
	return age;
}

// Gender set/get
public void setGender(String gen) {
	gender = gen;

	if (gender.equalsIgnoreCase("MALE"))
		genderBinary = 0;
	else if (gender.equalsIgnoreCase("FEMALE"))
		genderBinary = 1;
	else {
		genderBinary = -1;
		gender = "NA";
	}
}

public String getGenderString() {
	return gender;
}

public int getGenderBinary() {
	return genderBinary;
}

// Marital Status set/get
public void setMaritalStatus(String ms) {
	maritalStatus = ms;

	if (maritalStatus.equalsIgnoreCase("SINGLE"))
		maritalStatusBinary = 0;
	else if (maritalStatus.equalsIgnoreCase("MARRIED"))
		maritalStatusBinary = 1;
	else {
		maritalStatusBinary = -1;
		maritalStatus = "NA";
	}
}

public String getMaritalStatusString() {
	return maritalStatus;
}

public int getMaritalStatusBinary() {
	return maritalStatusBinary;
}

// Nationality set/get
public void setNationality(String nat) {
	nationality = nat;

	if (nationality.equalsIgnoreCase("MUSLMA"))
		nationalityBinary = 0;
	else if (nationality.equalsIgnoreCase("AFGHAN"))
		nationalityBinary = 1;
	else {
		nationalityBinary = -1;
		nationality = "NA";
	}
}

public String getNationalityString() {
	return nationality;
}

public int getNationalityBinary() {
	return nationalityBinary;
}

// Place of Birth set/get
public void setPlaceOfBirth(String pob) {
	placeOfBirth = pob;

	if (placeOfBirth.equalsIgnoreCase("BIA"))
		placeOfBirthBinary = 0;
	else if (placeOfBirth.equalsIgnoreCase("BOA"))
		placeOfBirthBinary = 1;
	else {
		placeOfBirthBinary = -1;
		placeOfBirth = "NA";
```

```
        }
}

public int getPlaceOfBirthBinary() {
        return placeOfBirthBinary;
}

public String getPlaceOfBirthString() {
        return placeOfBirth;
}

// Military Record set/get
public void setMilRecord(String mil) {
        milRec = mil;
}

public String getMilRecord() {
        return milRec;
}

// Religion set/get
public void setReligion(String rel) {
        religion = rel;

        if (religion.equalsIgnoreCase("Mld"))
                religionBinary = 0;
        else if (religion.equalsIgnoreCase("Rad"))
                religionBinary = 1;
        else {
                religionBinary = -1;
                religion = "NA";
        }
}

public int getReligionBinary() {
        return religionBinary;
}

public String getReligionString() {
        return religion;
}

// Skill set/get
public void setSkill(String skl) {
        skill = skl;
}

public String getSkill() {
        return skill;
}

// Address set/get
public void setAddress(String addr) {
        address = addr;

        if (addr.equalsIgnoreCase("TSV")) {
                lat = 39.98;
                lon = -74.43;
        } else if (addr.equalsIgnoreCase("VNV")) {
                lat = 39.98;
                lon = -74.42;
        } else if (addr.equalsIgnoreCase("VV")) {
                lat = 39.97;
                lon = -74.43;
        } else if (addr.equalsIgnoreCase("HAV")) {
                lat = 40.01;
                lon = -74.55;
        } else if (addr.equalsIgnoreCase("CCV")) {
                lat = 40.02;
                lon = -74.56;
        } else if (addr.equalsIgnoreCase("GT")) {
                lat = 40.03;
                lon = -74.52;
        } else if (addr.equalsIgnoreCase("UV")) {
                lat = 40.05;
                lon = -74.46;
        } else if (addr.equalsIgnoreCase("HOV")) {
                lat = 40.01;
                lon = -74.45;
        } else {
                lat = -1;
```

```
                lon = -1;
                address = "NA";
        }
}

public String getAddress() {
        return address;
}

// Employment set/get
public void setEmployment(String emp) {
        employment = emp;
}

public String getEmployment() {
        return employment;
}

// Allegiance set/get
public void setAllegiance(String alg) {
        allegiance = alg;
}

public String getAllegiance() {
        return allegiance;
}

// Attitude set/get
public void setAttitude(String att) {
        attitude = att;

        if (att.equalsIgnoreCase("1"))
                verboseAttitude = "Hostile";
        else if (att.equalsIgnoreCase("2"))
                verboseAttitude = "Dislikes";
        else if (att.equalsIgnoreCase("3"))
                verboseAttitude = "Neutral";
        else if (att.equalsIgnoreCase("4"))
                verboseAttitude = "Joker";
        else if (att.equalsIgnoreCase("5"))
                verboseAttitude = "Likes";
        else if (att.equalsIgnoreCase("6"))
                verboseAttitude = "Friendly";
        else
                verboseAttitude = "NA";
}

public String getAttitude() {
        return attitude;
}

public void setVerboseAttitude(String att) {
        verboseAttitude = att;

        if (att.equalsIgnoreCase("Hostile"))
                attitude = "1";
        else if (att.equalsIgnoreCase("Dislikes"))
                attitude = "2";
        else if (att.equalsIgnoreCase("Neutral"))
                attitude = "3";
        else if (att.equalsIgnoreCase("Joker"))
                attitude = "4";
        else if (att.equalsIgnoreCase("Likes"))
                attitude = "5";
        else if (att.equalsIgnoreCase("Friendly"))
                attitude = "6";
        else
                attitude = "NA";
}

public String getVerboseAttitude() {
        return verboseAttitude;
}

public void setEquipment(String eq) {
        equipment = eq;
}

public String getEquipment() {
        return equipment;
}
```

```
public void setVehicle(String veh) {
	vehicle = veh;
}

public String getVehicle() {
	return vehicle;
}

public void setCriminalRec(String crm) {
	criminalRec = crm;

	if (criminalRec.equalsIgnoreCase("Guilty"))
		criminalRecBinary = 0;
	else if (criminalRec.equalsIgnoreCase("None"))
		criminalRecBinary = 1;
	else {
		criminalRecBinary = -1;
		criminalRec = "NA";
	}

}

public String getCriminalRecString() {
	return criminalRec;
}

public int getCriminalRecBinary() {
	return criminalRecBinary;
}

public void setEducation(String ed) {
	education = ed;

	if (education.equalsIgnoreCase("High"))
		educationBinary = 0;
	else if (education.equalsIgnoreCase("Low"))
		educationBinary = 1;
	else {
		educationBinary = -1;
		education = "NA";
	}

}

public String getEducationString() {
	return education;
}

public int getEducationBinary() {
	return educationBinary;
}

public void setID(int x) {
	id = x;
}

public int getID() {
	return id;
}

// synchronized so updates are not over-written
public synchronized void updateNode(int attr) {

	// System.out.println(node);

	switch (attr) {
	case 0:
		try {
			((ObjectNode) node.get("names").getElements().next()).put(
					"fullName", name);
		} catch (Exception nameEx) {
		}// add code if name ever editable
		break;

	case 1:
		try {
			((ObjectNode) node.get("affiliations").getElements().next())
					.put("affiliation", organization);
		} catch (Exception affEx) {
			ArrayNode orgsArray = ((ObjectNode) node)
					.putArray("affiliations");
```

```
                ObjectNode orgNode = orgsArray.addObject();
                orgNode.put("affiliation", organization);
        }
        break;

case 2:
        break;

case 3:
        if (tribe.equalsIgnoreCase("NA"))
                try {
                        ((ObjectNode) node).remove("ethnicity");
                } catch (Exception tribeEx) {
                }
        else
                ((ObjectNode) node).put("ethnicity", tribe);
        break;

case 4:
        ((ObjectNode) node).put("age", Integer.parseInt(age));
        break;

case 5:
        if (gender.equalsIgnoreCase("NA"))
                try {
                        ((ObjectNode) node).remove("gender");
                } catch (Exception genderEx) {
                }
        else
                try {
                        ((ObjectNode) node.get("gender")).put("physicalValue",
                                        gender);
                } catch (Exception genderEx) {
                        ((ObjectNode) node).putObject("gender").put(
                                        "physicalValue", gender);
                }
        break;

case 6:
        if (maritalStatus.equalsIgnoreCase("NA"))
                try {
                        ((ObjectNode) node).remove("maritalStatus");
                } catch (Exception marEx) {
                }
        else
                ((ObjectNode) node).put("maritalStatus", maritalStatus);
        break;

case 7:
        if (nationality.equalsIgnoreCase("NA"))
                try {
                        ((ObjectNode) node).remove("nationality");
                } catch (Exception natEx) {
                }
        else
                try {
                        ((ObjectNode) node.get("nationality")).put("physicalValue",
                                        nationality);
                } catch (Exception natEx) {
                        ((ObjectNode) node).putObject("nationality").put(
                                        "physicalValue", nationality);
                }
        break;

case 8:
        ((ObjectNode) node).put("placeOfBirth", placeOfBirth);
        break;

case 9:
        try {
                ((ObjectNode) node.get("affiliations").getElements().next())
                                .put("natureOfAffiliation", attitude);
        } catch (Exception attEx) {
                ArrayNode attsArray = ((ObjectNode) node)
                                .putArray("affiliations");
                ObjectNode attNode = attsArray.addObject();
                attNode.put("natureOfAffiliation", attitude);
        }
        break;

case 10:
```

```
        try {
                ((ObjectNode) node.get("affiliations").getElements().next())
                                .put("affiliationGroupName", allegiance);
        } catch (Exception allEx) {
                ArrayNode allsArray = ((ObjectNode) node)
                                .putArray("affiliations");
                ObjectNode allNode = allsArray.addObject();
                allNode.put("affiliationGroupName", allegiance);
        }
        break;

case 11:
        try {
                ((ObjectNode) node.get("criminalRecords").getElements().next())
                                .put("verdict", criminalRec);
        } catch (Exception crimEx) {
                ArrayNode crimsArray = ((ObjectNode) node)
                                .putArray("criminalRecords");
                ObjectNode crimNode = crimsArray.addObject();
                crimNode.put("verdict", criminalRec);
        }
        break;

case 12:
        try {
                ((ObjectNode) node.get("education").getElements().next()).put(
                                "educationalLevel", education);
        } catch (Exception edEx) {
                ArrayNode edsArray = ((ObjectNode) node).putArray("education");
                ObjectNode edNode = edsArray.addObject();
                edNode.put("educationalLevel", education);

        }
        break;

case 13:
        try {
                ((ObjectNode) node.get("employment").getElements().next()).put(
                                "employerType", employment);
        } catch (Exception empEx) {
                ArrayNode empsArray = ((ObjectNode) node)
                                .putArray("employment");
                ObjectNode empNode = empsArray.addObject();
                empNode.put("employerType", employment);
        }
        break;

case 14:
        try {
                ((ObjectNode) node.get("militaryService").getElements().next())
                                .put("dutyOrPosition", milRec);
        } catch (Exception milEx) {
                ArrayNode milsArray = ((ObjectNode) node)
                                .putArray("militaryService");
                ObjectNode milNode = milsArray.addObject();
                milNode.put("dutyOrPosition", milRec);
        }
        break;

case 15:
        try {
                ((ObjectNode) node.get("religions").getElements().next()).put(
                                "religionName", religion);
        } catch (Exception religionEx) {
                ArrayNode religionsArray = ((ObjectNode) node)
                                .putArray("religions");
                ObjectNode religionNode = religionsArray.addObject();
                religionNode.put("religionName", religion);
        }
        break;

case 16:
        try {
                ((ObjectNode) node.get("skills").getElements().next()).put(
                                "skill", skill);
        } catch (Exception skillEx) {
                ArrayNode skillsArray = ((ObjectNode) node).putArray("skills");
                ObjectNode skillNode = skillsArray.addObject();
                skillNode.put("skill", skill);
        }
        break;
```

```
                case 17:
                        try {
                                ((ObjectNode) node.get("addresses").getElements().next()).put(
                                                "city", address);
                        } catch (Exception cityEx) {
                                ArrayNode addressesArray = ((ObjectNode) node)
                                                .putArray("addresses");
                                ObjectNode addressNode = addressesArray.addObject();
                                addressNode.put("city", address);
                        }

                        try {
                                ((ObjectNode) node.get("locations").getElements().next()
                                                .get("coordinates").getElements().next()).put(
                                                "latitude", lat);
                                ((ObjectNode) node.get("locations").getElements().next()
                                                .get("coordinates").getElements().next()).put(
                                                "longitude", lon);
                                ((ObjectNode) node.get("locations").getElements().next()).put(
                                                "geometryType", "POINT");
                        } catch (Exception latlonEx) {
                                ArrayNode locationsArray = ((ObjectNode) node)
                                                .putArray("locations");
                                ObjectNode locationNode = locationsArray.addObject();
                                locationNode.put("geometryType", "POINT");
                                ArrayNode coordinatesArray = locationNode
                                                .putArray("coordinates");
                                ObjectNode latlonNode = coordinatesArray.addObject();
                                latlonNode.put("latitude", lat);
                                latlonNode.put("longitude", lon);
                        }
                        break;

                case 18:
                        try {
                                ((ObjectNode) node.get("remarks").getElements().next()).put(
                                                "details", equipment);
                        } catch (Exception eqEx) {
                                ArrayNode remarksArray = ((ObjectNode) node)
                                                .putArray("remarks");
                                ObjectNode remarksNode = remarksArray.addObject();
                                remarksNode.put("details", equipment);
                                remarksNode.put("subject", "C4ISR OTM");
                        }
                        break;

                case 19:
                        ((ObjectNode) node).put("description", this.getVehicle());
                        break;

                default:
                        break;
                }
                // System.out.println(node);

        }
}
```

```
package hdpt;

import java.text.NumberFormat;
import javax.swing.DefaultCellEditor;
import javax.swing.JComboBox;
import javax.swing.JFormattedTextField;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
public class ColumnProperties {

	private static String[] columnNames = { "Name", "Color", "ID",
			"Tribal Affiliation", "Age", "Association Rating",
			"Individual Sentiment Rating", "Nationality",
			"SNA Centrality Measure", "Positional Influence Potential",
			"Education Level", "Employment Type", "Religious Rad Ratio",
			"AddressID" };

	private static String[] dataTypes = { "NOM", "NUM", "ORD", "NUM", "BIN",
			"ORD", "ORD", "ORD", "NOM", "ORD", "NOM" };

	private static int[] weights = new int[dataTypes.length];

	public ColumnProperties() {

	}
	public static String[] getColumnNames() {
		return columnNames;
	}
	public static String[] getDataTypes() {
		return dataTypes;
	}

	public static int getAttributeCount() {
		return dataTypes.length;
	}

	public static void setWeight(int index, int value) {
		weights[index] = value;
	}

	public static int getWeight(int index) {
		return weights[index];
	}
	public static void initWeights() {

		for (int i = 0; i < dataTypes.length; i++) {
			weights[i] = 1;
		}

	}
	public static JTable populateTable(JTable t, DefaultTableModel m) {
		JTable tmp = t;
		for (int i = 0; i < dataTypes.length; i++) {
			tmp.setValueAt(columnNames[i + 3], i, 0);
			tmp.setValueAt(dataTypes[i], i, 1);
			tmp.setValueAt(weights[i], i, 2);
		}
		TableColumn nameColumn = tmp.getColumnModel().getColumn(0);
		JTextField fld = new JTextField(20);
		fld.setEditable(false);
		nameColumn.setCellEditor(new DefaultCellEditor(fld));
		nameColumn.setPreferredWidth(150);

		TableColumn typeColumn = tmp.getColumnModel().getColumn(1);
		JComboBox typeBox = new JComboBox(new String[] { "NUM", "NOM", "ORD",
				"BIN" });
		typeColumn.setCellEditor(new DefaultCellEditor(typeBox));
		typeColumn.setPreferredWidth(80);
		TableColumn weightsColumn = tmp.getColumnModel().getColumn(2);
		JFormattedTextField numberField = new
JFormattedTextField(NumberFormat.getIntegerInstance());
		weightsColumn.setCellEditor(new DefaultCellEditor(numberField));
		weightsColumn.setPreferredWidth(80);
		return tmp;
	}
	public static JTable populateInsertTable(JTable t, DefaultTableModel m) {
		return null;
	}
}
```

```
package hdpt;

import javax.swing.*;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.lang.reflect.Field;
import java.util.Arrays;
import java.util.Collections;
import org.math.plot.Plot3DPanel;
import org.math.plot.plots.ScatterPlot;
import org.math.plot.canvas.*;
import java.util.Vector;

public class GGPlotFrame extends JInternalFrame {

	private static final long serialVersionUID = 1L;

	private double[][] coordinates;
	private Object[] data;
	private Vector<Vector<Object>> dissim;
	private HDPT callback;
	private Plot3DPanel plots;
	private GGSimilarityLinkFrame simFrame;

	public GGPlotFrame(double[][] coordinates, Object[] data, HDPT _callback,
			Vector<Vector<Object>> _dissim) {
		super("MDS 3D Scatter Plot");

		this.coordinates = coordinates;
		this.data = data;
		this.callback = _callback;
		this.dissim = _dissim;

		this.setSize(
				callback.getContentPane().getSize().width / 2,
				((BorderLayout) callback.getContentPane().getLayout())

	.getLayoutComponent(BorderLayout.CENTER).getSize().height);
		this.setLocation(callback.getContentPane().getSize().width / 2, 0);
		this.setVisible(true);
		this.setResizable(true);
		this.setMaximizable(true);
		this.setIconifiable(true);
		this.setClosable(true);
		this.setDefaultCloseOperation(JInternalFrame.DISPOSE_ON_CLOSE);
		this.plots = this.createScatterPlot();
		this.add(plots);

		// JSplitPane splits = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
		// splits.setDividerLocation(callback.getContentPane().getSize().height*4/5);

		// splits.add(plots);
		// splits.add(new GGSimilarityLinkFrame(data, callback));
		// this.add(splits);
	}

	public void getPlots(int i, boolean b) {
		ScatterPlot plot = (ScatterPlot) (plots.getPlot(i));

		plot.setSelectedByTable(b);
		((Plot3DCanvas) (plots.plotCanvas)).repaint();
	}

	public int[] sortDissim(int row) {
		Vector<Double> tmp2 = new Vector<Double>();
		int[] links = new int[dissim.get(row).size()];

		for (int j = 0; j < dissim.get(row).size(); j++)
			tmp2.add(Double.valueOf(dissim.get(row).get(j).toString()));

		Collections.sort(tmp2);

		// for(int i = 0; i < tmp2.size(); i++)
		// System.out.println(tmp2.get(i));

		int scan = 1;
		int ref = 0;
		for (int index = 0; index < links.length; index++) {
```

```
                        links[index] = dissim.get(row).indexOf(tmp2.get(index));
                        // System.out.println(links[index]);
                        scan = links[index] + 1;
                        ref = index;
                        while (scan < links.length) {
                                if (dissim.get(row).indexOf(tmp2.get(ref), scan) != -1) {
                                        links[++index] = dissim.get(row).indexOf(tmp2.get(ref),
                                                        scan);
                                        scan = links[index] + 1;
                                } else
                                        scan = links.length;
                        }
                }

                // for(int index = 0; index < links.length; index++){
                // System.out.println(links[index]);
                // }

                /*
                 * links[0] = dissim.get(row).indexOf(tmp2.get(0));
                 *
                 * if (tmp2.get(0) != tmp2.get(1)) links[1] =
                 * dissim.get(row).indexOf(tmp2.get(1)); else links[1] =
                 * dissim.get(row).indexOf(tmp2.get(1), links[0]);
                 *
                 * if (tmp2.get(1) != tmp2.get(2)) links[2] =
                 * dissim.get(row).indexOf(tmp2.get(2)); else links[2] =
                 * dissim.get(row).indexOf(tmp2.get(2), links[1]);
                 *
                 * if (tmp2.get(2) != tmp2.get(3)) links[3] =
                 * dissim.get(row).indexOf(tmp2.get(3)); else links[3] =
                 * dissim.get(row).indexOf(tmp2.get(3), links[2]);
                 */
                // for(int i = 0; i < links.length; i++)
                // System.out.println(links[i]);
                return links;
        }

        public Plot3DPanel createScatterPlot() {
                Plot3DPanel p = new Plot3DPanel();

                double X, Y, Z;
                for (int j = 0; j < coordinates[0].length; j++) {
                        X = coordinates[0][j];
                        Y = coordinates[1][j];
                        Z = coordinates[2][j];

                        double[][] xyz = new double[1][3];
                        xyz[0][0] = X;
                        xyz[0][1] = Y;
                        xyz[0][2] = Z;

                        Color c;
                        try {
                                Field field = Color.class.getField(((GGPerson) data[j])
                                                .getColorString());
                                c = (Color) field.get(null);
                        } catch (Exception e) {
                                c = null; // Not defined
                        }

                        p.addScatterPlot(((GGPerson) data[j]).getName(), c,
                                        Arrays.copyOf(xyz, 1), (GGPerson) data[j],
                                        callback.getGGDataTable(), this);

                }

                p.setEditable(false); // editable brings up datasframe
                p.setNotable(true); // notable changes color, add comments etc

                // }

                // p.setLegendOrientation(PlotPanel.SOUTH);
                return p;
                // new FrameView(p).setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }

}
```

INTENTIONALLY LEFT BLANK.

List of Symbols, Abbreviations, and Acronyms

2-D	two-dimensional
3-D	three-dimensional
C4ISR	Command, Control, Communication, Computers, Intelligence, Surveillance and Reconnaissance
DOD	Department of Defense
DSGS-A	Distributed Common Ground Systems-Army
E12	exercise 12
HDPT	Heterogeneous Data-Reduction Proximity Tool
JSON	JavaScript Object Notation
MDS	multidimensional scaling
OTM	On-The-Move
OWF	Ozone Widget Framework
PCA	principal component analysis
POR	program of record
REST	Representational State Transfer
UML	unified modeling language
VoI	value of information

NO. OF COPIES	ORGANIZATION
1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB IMAL HRA
1 (PDF)	DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL
1 (PDF)	GOVT PRINTG OFC A MALHOTRA
19 (14 PDFs) 5 HCs)	DIR USARL RDRL CII C B BODT E BOWMAN F BRUNDICK B BROOME (1 HC) J DUMER T HANRATTY (4 HCs, 1 PDF) C HANSEN E HEILMAN S KASE M MITTRICK A NEIDERER K OGAARD J RICHARDSON H ROY M THOMAS