Topic 5: Battlespace Understanding and Management

Reasoning about Unknown Objects and Dependencies in C2 Networks Using Value of Information

Dr. Brian Drabble

Operations & Information Management Group,
University of Bradford School of Management and Law,
Emm Lane, Bradford,
West Yorkshire, BD9 4JL,
UK.

bdrabble1@gmail.com

Keywords: "Network Analysis and Modelling", "Understanding Plan Effects", "Constraint and Option Management".

Supported by the Naval Postgraduate School, San Diego, CA, USA. Contract N00244-14-1-0056 NAVSUP, September 1st 2014 to April 30th 2015

Abstract

This paper describes an approach to reasoning about unknown objects and dependences in C2 network analysis using Value of Information (VOI). C2 systems and their attributes are modeled as a series of networks describing the inter-dependencies between persons, groups, locations, resources, concepts, etc. This allows for the capture of the required infrastructural, organizational, procedural, etc., aspects of networks. Due to a lack of information C2 network models often contain unknown nodes and dependency links. Unknown nodes are ones that are suspected members of the network and could be instantiated to one of several values. For example, an insurgent network must have a leader node but it could be one of several people. Dependency links are ones that suggest possible ways to address a dependency. A C2 Center depends on electrical power which could be supplied via one or more sources. The algorithms described here firstly provide analysts with the ability to track constraints between unknown nodes and links allowing forced instantiations (options reduced to one), empty options (no instantiations possible) and reduced set sizes to be identified. Secondly, it provides analysts with the ability to assess the VOI for an unknown node/link in terms of what other decisions regarding unknowns are made computationally easier by their instantiation. The approach has been integrated with a planning capability to create plans to resolve unknown nodes/links and to bring about desired network effects. It has been applied to several domains including counter insurgency, air campaign planning and countering Weapons of Mass Destruction development.

1. Overview

This paper describes an approach to reasoning about unknown objects and dependencies in C2 network analysis using Value of Information (VOI). Due to a lack of information C2 network models often contain unknown nodes and dependency links. Unknown nodes are ones that are suspected to be part of the network and could be instantiated to one of several values. For example, an insurgent network must have a leader node but it could be one of several people. Dependency links are ones that suggest possible ways to address a dependency. A C2 Center depends on electrical power which could be supplied via one or more sources. VOI is used to identify and rank the potential information value gained through either resolving an unknown to a specific instance or by identifying the specific node(s) addressing a network dependency. The use of VOI supports an analyst by decreasing their overall cognitive workload, increasing their productivity and increasing their overall situational awareness. It reduces cognitive workload by dynamically identifying and ranking key unknowns as the structure and content of the target system change over time. It increases productivity by guiding analysts to focus on the most relevant and important unknowns. It increases situational awareness by identifying "connections" between objects and unknowns within a network, identifying the 2nd, 3rd, etc. order effects of an analyst's decisions. The techniques described would indicate to an analyst that VOI for resolving the location of an insurgent safe house will directly reduce the lack of information on the networks C2 structure by 20%. An indirect effect of reducing the unknown should reduce the candidates for the group's leader by 50% and subsequently the group's financier by 80%. Additionally, it allows for better use of intelligence collection and analysis capabilities. It achieves this by reducing the number of analyst generated requests for information to a smaller set. This smaller set provides the same maximal information value, removes redundant collections and shows how results from different collections can be combined to deduce additional information. The paper is structured as follows. Section 2 provides an

introduction to the concept of VOI as it pertains to reasoning about network unknowns. Section 3 provides an introduction to the dependency based modelling technique used to describe target systems such as C2 networks, Weapons of Mass Destruction (WMD) programs, insurgent networks, etc. Section 4 provides details of the algorithms developed to reason with network unknowns. Section 5 provides details of the evaluation of the algorithms and Section 6 provides a summary and pointers to further research. A list of references and pointers are also provided.

2. Introduction

Analysts are faced on a daily basis with decisions regarding which information would provide greatest insights into their problem. Additionally, they are faced with the problem of what information would make other decisions and options easier to resolve. For example, if both the leader and the financier in an insurgent network are currently unidentified (i.e. unknown) then which should be the priority for resolution?[1] If the financier is strongly suspected to be related by familial links to the group's master bomb maker then, would resolving the identity of the master bomb maker reduce the potential set of financier candidates and hence make the resolution of the leader and financier easier? Alternately, if plans have been developed to raid potential insurgent safe houses to disrupt their training capabilities and the groups financier is then identified, would the easier option of arresting him have the same desired level of disruption on insurgent training? In order to address these types of questions a series of network analysis algorithms were developed. These algorithms allow analysts to develop models of C2, social, infrastructural, political, etc. networks at different levels of abstraction and aggregation. These network models capture the dependency between objects, its nature and strength. For example, a physical object or node such as an Air Operations Center (AOC) is heavily dependent on an electrical substation (another physical node) to provide it with electrical power. The AOC is also highly dependent on actor nodes (people, groups, etc.) to work collectively to develop order, assessments, reports, etc. for other dependent nodes in the C2 network. Understanding how different networks of actor and physical nodes interact through their inter and intra-dependencies[2] is key to understanding their function and how changes propagate through them. One key aspect of change is the level of information known about a node and the implications this has for understanding how the networks function. This is especially important if the level of knowledge concerning a node increases as it has potential implications for other nodes in the network. To explore this issue, a series of algorithms were developed to explicitly reason about the level of information known and more importantly unknown about network nodes and dependency links [1]. These algorithms employed the concept of VOI to understand the added value to the network of increasing the level of information known on a node or dependency link. These algorithms collectively provide a capability to reason with unknowns in a network. Information levels on a node or link could be increased by adding to its existing information (aggregation), by analyzing what is already known (refinement) or by identifying new information based on the level of information on other nodes and/or dependency links (synthesis). Discovering the identity of the insurgent groups leader adds to the knowledge that the network has such a person and who the leader

[1] The instantiation of the financier or leader node to a specific individual or group.

[2] Intra dependencies are within networks (e.g. electrical substation on a transmission line). Inter dependencies are between networks (e.g. AOC on a Communication Center in a network of exchanges, repeaters, etc.)

is (aggregation). Identifying the specific substation(s) and transmission line(s) addressing the AOCs dependency on electrical power refines the knowledge about its dependencies and vulnerabilities[3] (refinement). Identification of one or more of the nodes of a mobile SAM site (radar trucks, transport vehicles, etc.) in a locale could indicate the presence of a SAM capability that needs to be added to the network (synthesis). The VOI based algorithms for reasoning across these three areas were successfully developed and integrated with the Cassandra network dependency and consequence analysis toolkit [2, 3, 4, 5, 6]. Cassandra provided network modeling and visualization capabilities, APIs to external data sources and an architecture within which the algorithms for reasoning with unknowns could be tested and evaluated. Details of the integration with Cassandra are provided later in the paper.

3. Reasoning with Dependencies

The purpose of this section is to provide an overview of the dependency based representation used to describe C2 and other types of target systems. To address questions raised earlier requires technologies that can identify the importance of a node within a target system (e.g. C2 network, insurgent network, weapons development program, etc.). Firstly, this allows collection and analysis decisions to be prioritized. Secondly, it identifies the **direct** and **indirect** consequences on node information levels. That is, what is known versus what is potentially knowable about a network node or link.

3.1 Modelling Networks via their Dependencies

The importance of a node can be identified through a range of different measures and these include:

- The cumulative weighted dependency that other nodes have on it (**dependent** analysis)
- The cumulative weighted dependency that a node has on other nodes (**dependee** analysis)

Dependency can be stated directly between any pair of nodes in the target system and reflects the strength of the dependency that one node has on another. The strength of a dependency is measured using a scale of 1 – 10 where 1 reflects a weak dependency and 10 a critical one. The direct dependencies are then used to define a transitive dependency between two nodes linked by one or more intermediary nodes. For example, if the insurgency group's leader is dependent on a master bomb maker and the master bomb maker is dependent on a safe house then the leader is transitively dependent on the safe house. In many applications of C2, transport, social, political and other types of networks, it is often the cumulative transitive dependencies that define the importance of a node and not the usually smaller set of direct pairwise dependencies. Cassandra provides various network analysis algorithms to generate a dependee and dependent score for each node in a network. A node can be a specific instance (person, place, resource, etc.) or a more abstract one such as a set of nodes defined by type, function, location or a subnetwork or system defining a capability such as electrical power, transport, fielded military units, etc. The dependent or dependee score for a node allows it to be ranked with other user selected nodes to identify its importance. The scores can also be used to compare nodes in terms of the ratio of their dependent or dependee scores. A screenshot from Cassandra is shown in Figure 1. This shows part of a model of a C2 system using data from JEFX [7], or Joint Expeditionary

[3] Vulnerability is an objects direct susceptibility to a phenomenon (C2 Center to overpressure from a weapon). Dependencies are indirect vulnerabilities which are exploited by impacting the nodes addressing the dependency.

Force Experiment which was the periodic US Air Force-led operational experiment created to evaluate new technologies and war fighting concepts in a simulated wartime battle environment. A dependent analysis of the C2 capabilities identified Mather AFB with a score of 65.0 as the node most depended upon by others. This is shown in the analysis pane at the right of Figure 1.. The node "C2 Centers – WOC" (Wing Operations Center) with a score of 46.0 has approximately a third less importance to the C2 network than Mather AFB. Mather AFB provided capabilities including C2, Communications, air defense, etc., which many nodes/node sets are directly or indirectly dependent. The analysis is updated dynamically as nodes and/or links are added/deleted/modified. For example, if Mather's functionality was degraded 75% which other nodes have increased dependency scores as the network compensates

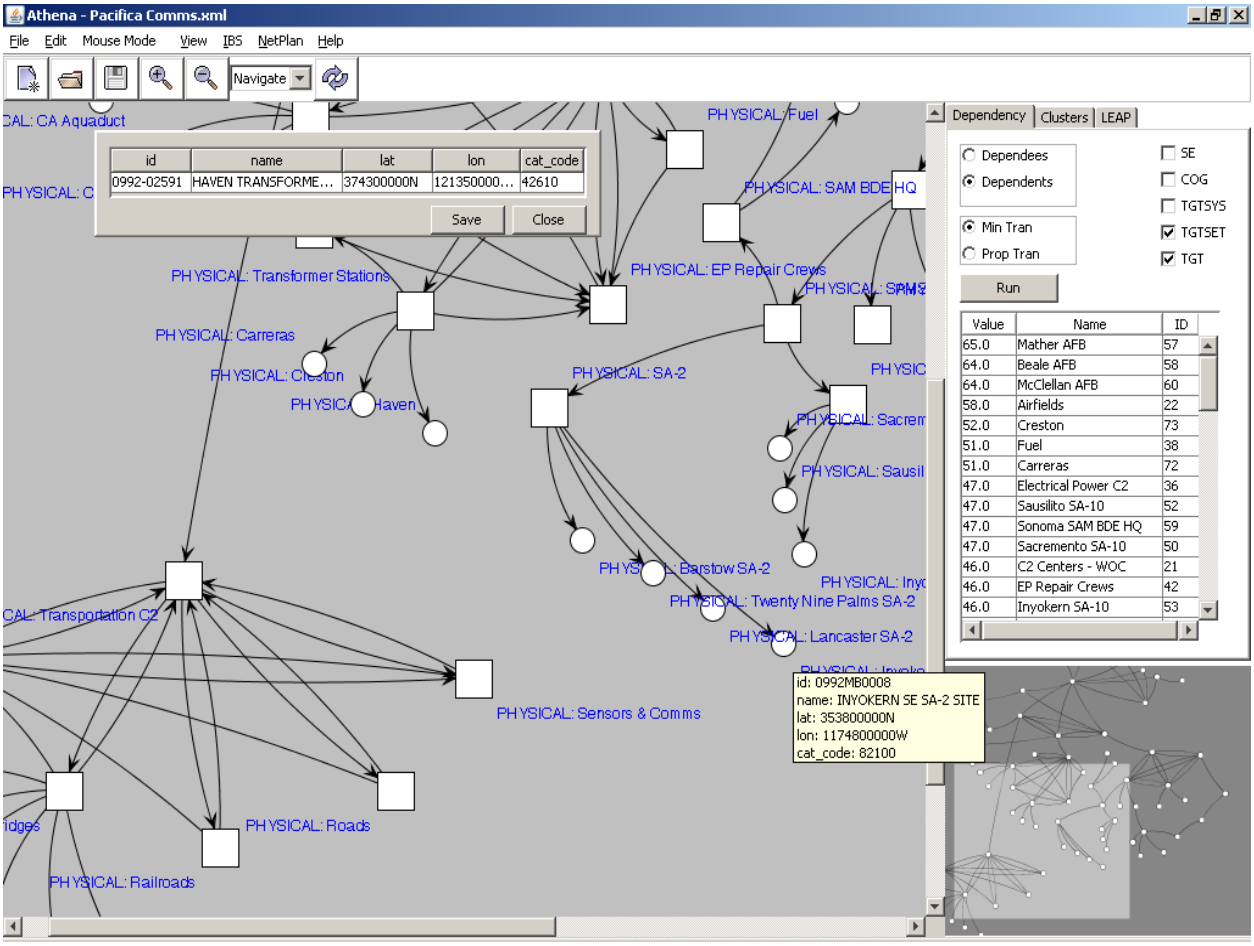

Figure 1: Example Network Model and Dependency Analysis

Reasoning with direct and transitive dependencies allows for the identification and ranking of the most important nodes and hence which aspects of the network the analyst should focus on. Direct dependencies can be placed between any pair of target system nodes hence a dependency could be placed on a node that has yet to be instantiated to a specific value. For example, the master bomb maker is dependent on a safe house from where they can construct devices. However, the actual location may be unknown or only suspected to be one of a list of suspected locations. This dependency exists irrespective of whether the location is known. Additionally, its value does not reflect any

confidence measure that any location identified is the one that actually provides the capabilities. The network models allow a probability measure on the link from the master bomb maker to a location. A similar measure can also be associated with the location reflecting confidence the location exists.

3.2 Analyzing Network Unknowns

The dependency network model makes no distinction between instantiated nodes and unknown nodes. Hence it is possible to calculate a cumulative dependency score for unknown nodes and rank them based on their overall importance to the network. A high dependent score would aid the analyst to discover that the location they suspect as being the base of the master bomb maker is also suspected to be the same location being used by other suspected insurgents as a safe house. Alternatively, the unknown node could be an electrical substation that supplies power to an AOC and is also suspected of supplying power to several other key facilities within an Integrated Air Defense. If the analyst were to resolve the safe house location or the substation, then it would provide valuable additional information on the network's functions. It would also allow specific nodes to be targeted to effect network behavior. For example, if the safe house is identified, then it could be raided and the capabilities of the master bomb maker disrupted. The identification of the safe house may make other decisions easier or force other decisions to a specific value. For example, if there are two different suspected safe houses, then knowing the identity of one may indirectly identify the other by elimination. Alternatively, the identification of the network's financier would make the identification of its leader easier[4] as they are linked by a familial or clan relationship. Hence decisions regarding one unknown can be propagated to other unknowns to identify forced decisions (i.e. a node must take a specific value or be linked to a specific node), option reductions (i.e. the options to instantiate a node or satisfy a dependency is reduced by 1 to n options) and empty option decisions (i.e. the options to instantiate a node or provide a link is reduced to 0). If a propagation results in an empty option then, the network has transitioned to an inconsistent state. An analyst can then rescind one or more decisions to explore other options. Aiding the analyst to identify when a network entered an inconsistent state was viewed as a significant advantage of the approach. Understanding the key decisions causing the inconsistency allowed analysts to retrace their reasoning steps. This essentially employs VOI in reverse to indicate what the analyst no longer knows if they decide to undo a previous decision by turning a node or link back into an unknown.

The dependency and unknown reasoning assumes that all nodes and links in a network model are correct and truly reflect the state of the domain. However, in most scenarios there is an element of uncertainty as to whether one or more nodes are part of the network (**membership**) or that a link correctly identifies the node addressing a node's dependency (**dependency**). An example of the former would be where an analyst is confident the insurgents have at least one safe house but there may be a second one. An example of the latter is where an analyst has limited confidence that the first suspected safe house is the one being used by the master bomb maker. Adding probabilities for membership and dependency into the network representation allowed analysts to provide measures of their confidence in the presence of a node or link. Probability values are calculated separately using information stored in OBMIS [4]. OBMIS is an ontology based management Information system allowing Cassandra to interact

[4] The instantiation is made easier due to the reduction in the options for the leader node.

with 3rd party tools and systems. The ontology employed by OBMIS takes into account a wider range of constraints, relationships and information than those available to Cassandra. For example, knowing there is a strong familial relationship between the insurgent groups leader and financier and knowing each was spotted at the same location yesterday allows OBMIS to develop a high dependency probability between the leader and financier node. Values can also be provided by the analyst. Analysts found the ability to reason with probabilities especially useful when they were retracing their reasoning steps due to the network becoming inconsistent. If a forced decision for the identity of the financier was to a node with a low membership probability or the leader and the financier link had a low probability, then this could be the reason why the option set for the master bomber became empty.

However, the primary purpose for propagating probability measures is to provide analysts with additional insight into dependency scores. Dependency analysis may identify that the insurgent group's financier is transitively dependent on an unknown safe house. However, if the dependency chain from the financier to the safe house contains one or more low probability values then this may indicate that such a dependency maybe not as strong as indicated or in the worst case may not exist at all. Hence, the analyst should concentrate on dependency chains that contain higher probability values even though the chain itself may have a lower dependency score. Consider the example, shown in Figure 1, it may be the case that focusing on the "C2 Centers – WOC" node with a dependency score of 46.0 has a greater probability of affecting the network's behavior then focusing on Mather AFB with the score of 65.0. This could be due to the Mather AFB score being reliant on several low membership and/or dependency probabilities, having a lower overall average probability score per node, having a larger number of outliers, etc. Each of these measures are an aspect of the VOI concept as they provide insight into other decisions and analysis results. For example, if the score for Mather AFB is questionable due to low membership and/or dependencies probabilities then what other node scores should be questioned as they contain the same or a large common subset of nodes and links?

4. Reasoning with Unknown Nodes and Dependencies

The following five sections provide details of the functionality developed to support reasoning with unknown nodes and unknown dependencies. They provide details of the GUI through which the analyst interacts with the algorithm and its integration with Cassandra.

4.1 Reasoning with Unknown Nodes

An analyst can insert an unknown node to represent a person, group, location, resource, concept, etc. The unknown node is associated with one or more instance sets which define the possible instances for the unknown node. For example, an unknown insurgent could be associated with both instance sets "Suspected Hezbollah Insurgents" and "Suspected Hamas Insurgents" if the analyst believes the insurgent is a member of one of these groups. By default, the instance set for an unknown node is the group of nodes that share its basic type: Physical, Actor or Concept. However, in a large network (i.e. > 2000 nodes) this may result in a default instance set of 100s. Hence the analyst is encouraged to create the smallest instance set possible. Pairs of nodes involving one or two unknowns can be linked by a constraint which limits the potential instances that the unknown node(s) can be instantiated to. As described earlier an analyst may want to constrain the unknown leader and financier to be two separate

people or that a suspected safe house and suspected bomb factory shares the same location. Additional constraint information from OBMIS can be used to dynamically update the constraints. For example, if an OBMIS update from a SITREP states the insurgent leader is in Iraq and the financier is in Europe, then it can be deduced to a certain level of probability they are not the same person. To support this the following inter-node constraint types are provided:

- **same-as**: The nodes must be instantiated to the same instance.
- **not-same-as**: The nodes cannot be instantiated to the same instance.

If a constraint is placed between an unknown node and an instantiated one, then only a not-same-as constraint is allowed. If an analyst were to insert a same-as constraint between an instantiated and unknown node, then this would immediately force the unknown node to be instantiated to the value of the instantiated node. Analysts often used this as a "short cut" to try different instantiations for an unknown node and identify the n-order effects of the decision.

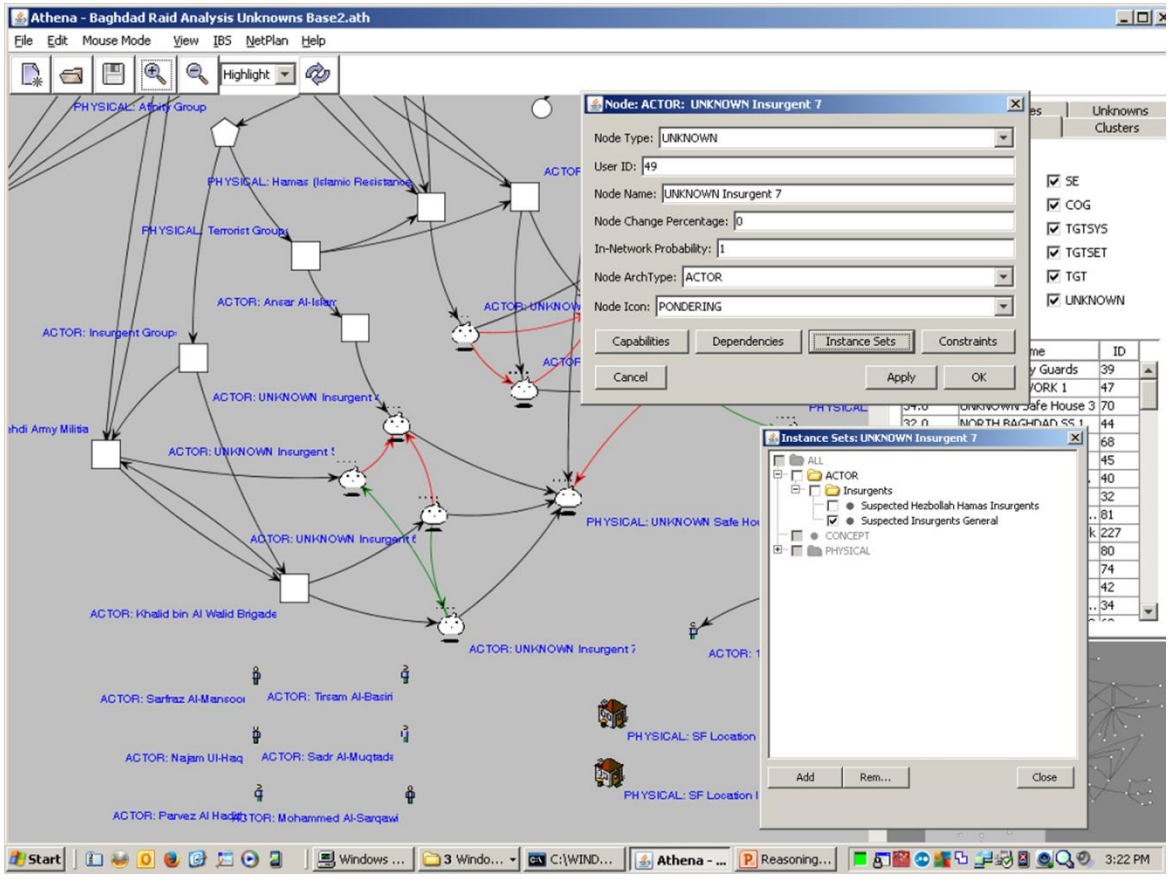

Figure 2: Same-as and Not-Same-As Constraints Example

Figure 2 shows a screen shot from a DARPA funded insurgency scenario containing several unknown nodes and their constraints. These are persons, groups, etc. that have leadership, financial, planning or training roles. Unknown nodes are shown as a cloud icon, same-as constraint as green links and not-same-as constraints as a red links. The two lower green links in Figure 2 shows that *unknown insurgent 7*

is the same-as *unknown insurgent 6* and the same-as *unknown insurgent 5*. The two upper red not-same-as links show that *unknown insurgent 5* and *unknown insurgent 6* are not the same as *unknown insurgent 4*. The lower right portal in Figure 2 shows the instance set (i.e. suspected insurgents general) for *unknown insurgent 7* and the upper portal describes general information for *unknown insurgent 7*. Constraints and instance sets for an unknown can be created, modified or deleted by the analyst via Cassandra or exported and imported via OBMIS.

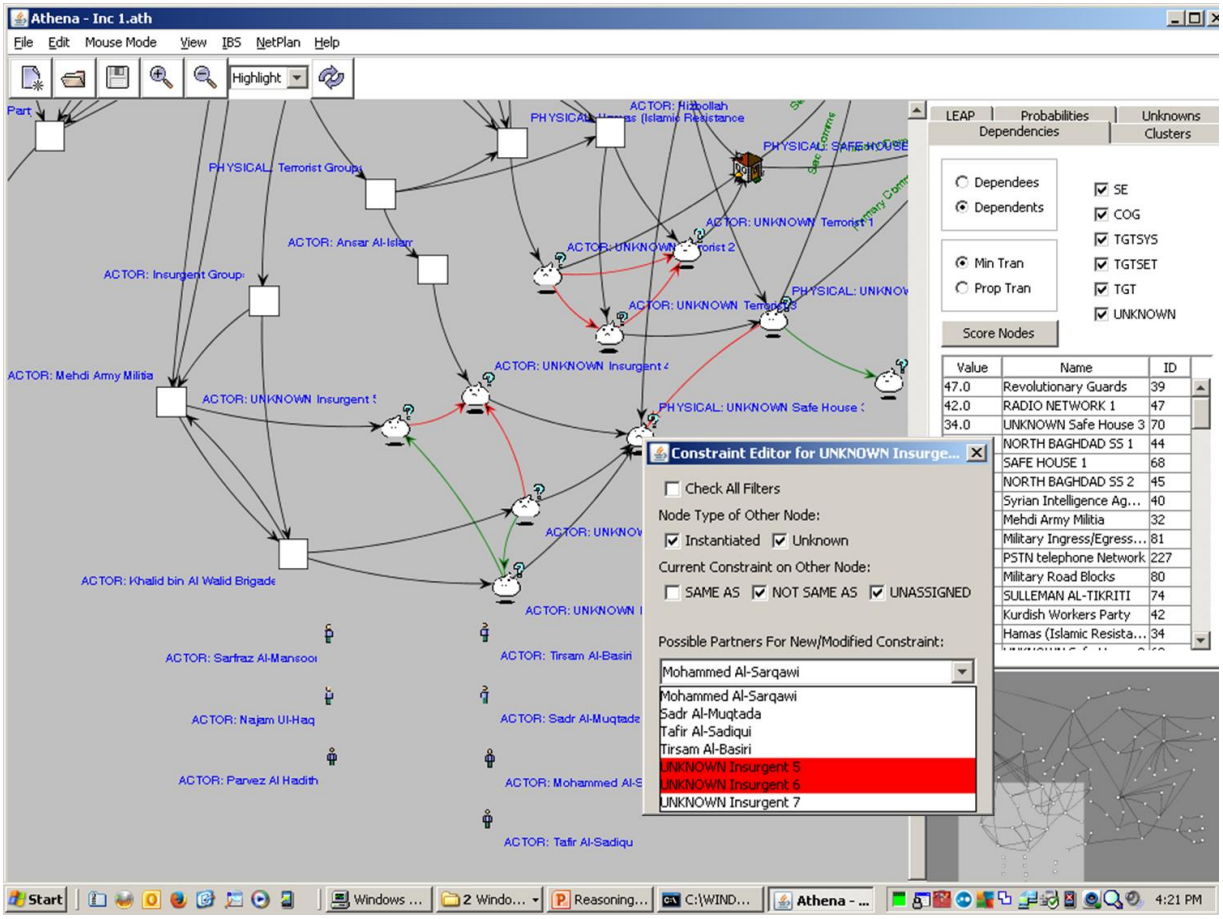

Figure 3: Constraint Editing Portal

Figure 3 shows the editing portal for the *unknown insurgent 4*. The *Instantiated* and *unknown* tick boxes at the top are used by the analyst to filter which nodes in the instance set(s) should be displayed. In this case selecting both means all entries in the instance set(s) should be displayed. The *same-as*, *not-same-as* and *unassigned* options across the center of the portal filter the instance options via the *instantiated* and *unknown* options.

- The *not-same*-option displays any options that are related to *unknown insurgent 4* by a not-same-as constraint (shown in red). That is any unknown or instantiated node that cannot be the same as *unknown insurgent 4*. This identifies *unknown insurgents 5* and *6* respectively and there are currently no not-same-as constraints to instantiated nodes in the option set(s).

- The selection of the *unassigned* option displays any instance set options that currently have no *same-as* or *not-same-as* relationship to *unknown insurgent 4*. This lists the instantiated nodes and *unknown insurgent 7* as these are not involved in a constraint with *unknown insurgent 4*.

The setting of these portal options allows the analyst to view the current state of the constraints with respect to *unknown insurgent 4*. The analyst then uses the portal to add, delete or modify constraints. Figure 4 shows the selection of the instantiated node *Mohammed Al-Sarqawi* which results in the removal of the instances list and its replacement with a set of option buttons. Figure 4 shows the selection of the relationship same-as which instantiates the *unknown insurgent 4* node to the instance *Mohammed Al-Sarqawi*. Alternatively, the selection of *Mohammed Al-Sarqawi* and the not-same-as option would insert a not-same-as between *unknown insurgent 4* and *Mohammed Al-Sarqawi*. The selection of the *unassigned* option would remove any constraints between the two nodes.

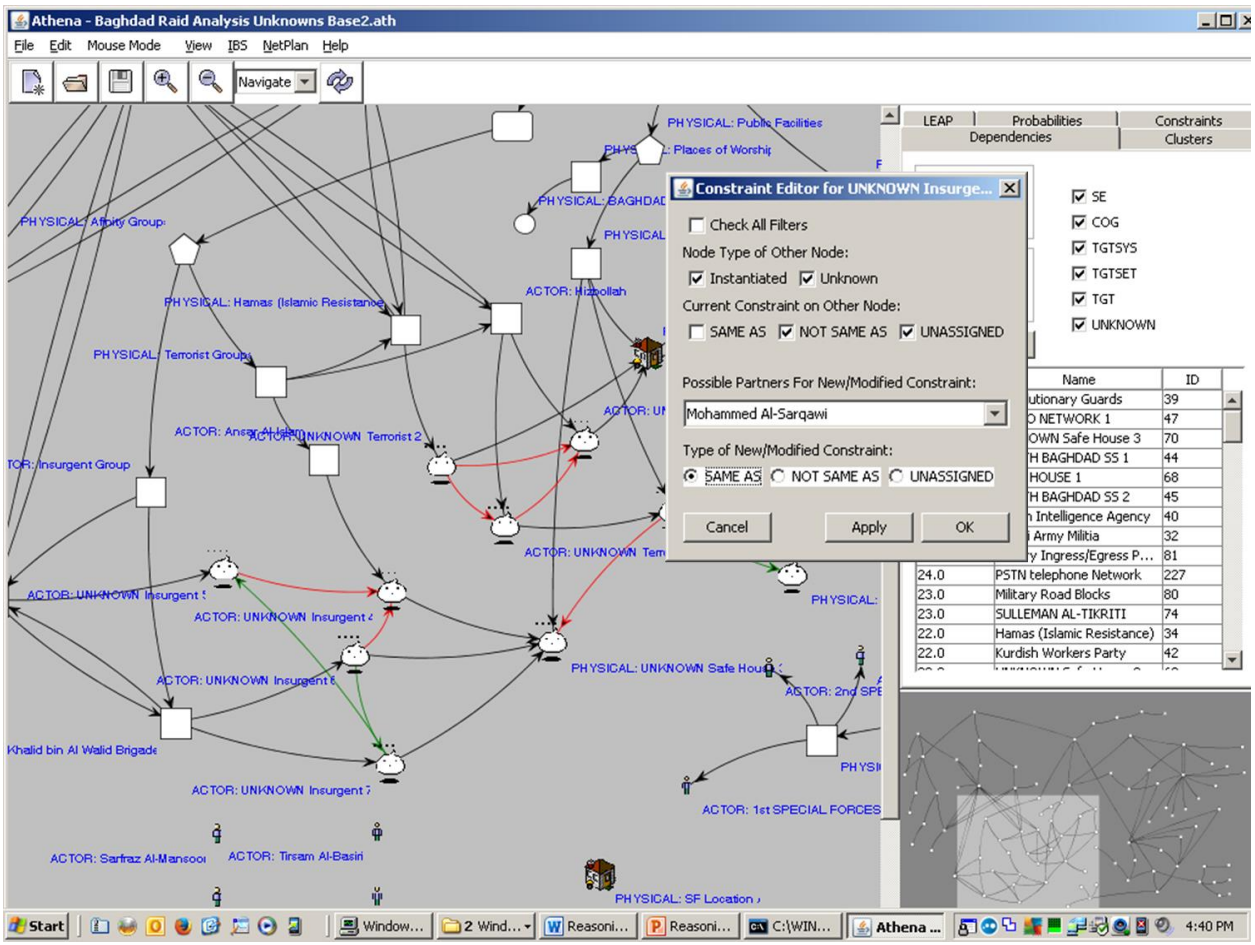

Figure 4: Assigning an Unknown node to a Specific Instance

The instantiation of an unknown node to a specific instance or the insertion/modification of an *unassigned* or *not-same-as* constraint results in a propagation that identifies the direct and n-order effects of the decision. This is achieved selecting the *Apply* button at the base of the portal. As described previously this can result in forced decisions, option reductions or in the worst case, one or more zero

options. In the case of an option reduction the impacted unknown nodes and number of options removed are listed for the analyst to view before selecting the *OK* button. In the case of a zero option, each is listed for the analyst to review. Figure 5 shows an example of a direct forced instantiation. The instantiation of *unknown insurgent 7* to *Mohammed Al-Sarqawi* forces the instantiation of *unknown insurgent 6* and *unknown insurgent 5* to *Mohammed Al-Sarqawi* as they are linked to *unknown insurgent 7* by same-as constraints. The result of the forced instantiation is to collapse the network to remove the three unknown nodes and assign their dependency links to *Mohammed Al-Sarqawi*. The same propagation and analysis is also carried out for a constraint change (e.g., same to not-same, same-as to unassigned). Additionally, all capabilities and dependencies that were asserted for the three unknown nodes are mapped to the *Mohammed Al-Sarqawi* node creating a union with its current values.

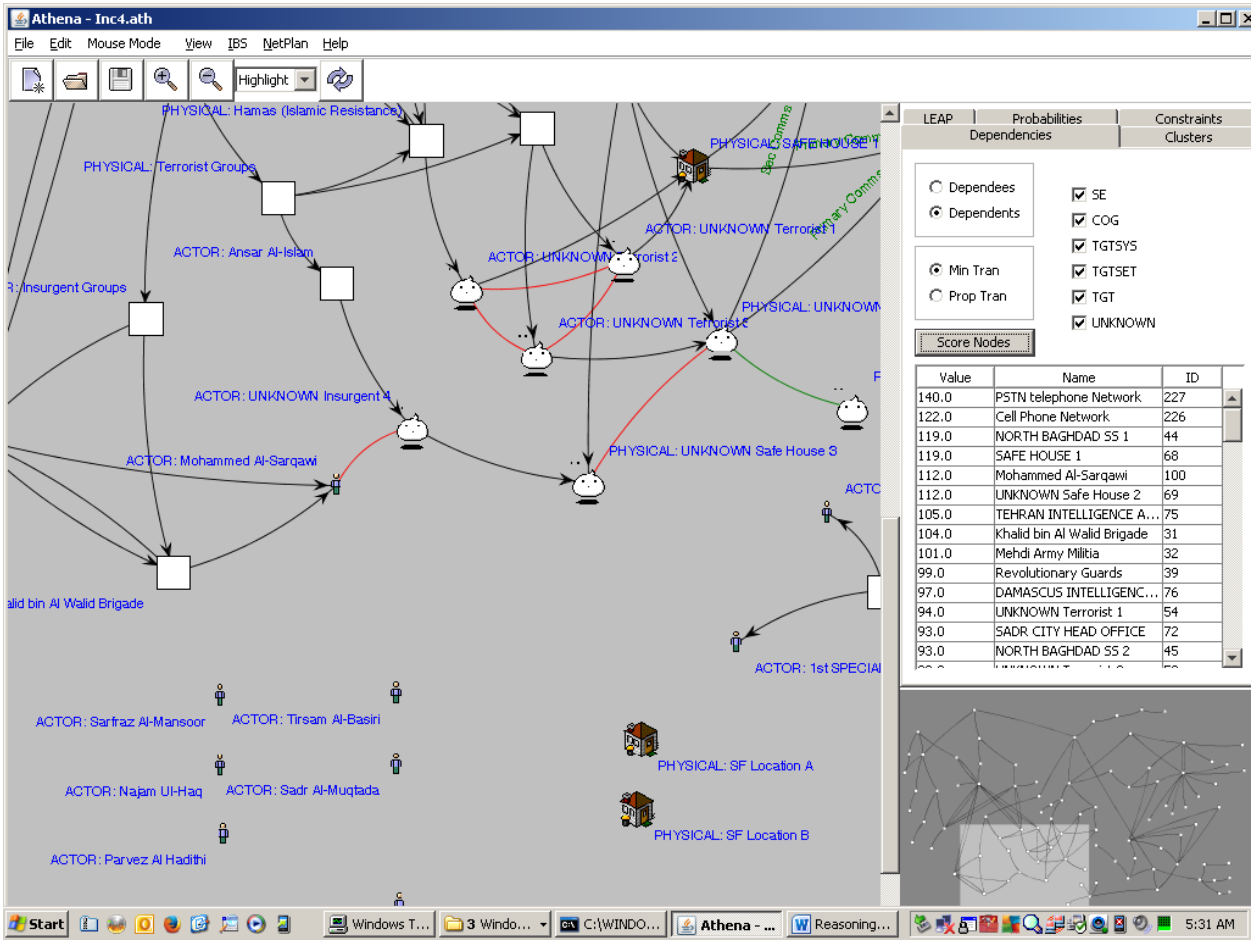

Figure 5: Forced Instantiation via Same-As and Not-Same-As Constraints

The *not-same-as* constraints that were between *unknown insurgent 5* and *unknown insurgent 4* and between *unknown insurgent 6* and *unknown insurgent 4* are inherited by *Mohammed Al-Sarqawi* to ensure *unknown insurgent 4* cannot be *Mohammed Al-Sarqawi*. As described previously, Cassandra maintains a model of the capabilities, dependencies and vulnerabilities of a node. Any inconsistencies are highlighted for viewing by the analyst. Consider the three unknown nodes linked by red not-same-as constraints at the top of Figure 5. If they share an instance set of 3 values and one instantiated, then

the option set for the remaining 2 unknown nodes is reduced to 2. If a further unknown is instantiated, this leaves the remaining unknown with a single value and the network is modified accordingly. Often analysts would see dramatic structural changes due to one or two selected or forced instantiations.

4.2 Analysis of Direct and Indirect Impacts of Constraints Changes

To support the analyst in prioritizing which unknown node to focus on, a capability was provided to rank nodes based on the following values for each of the unknown's potential options.

- **Instances**: Identify which instance sets are impacted and the number of instances removed.
- **Instantiations**: Identify which unknown nodes are forced directly or indirectly to a specified instance and the value of the instance.
- **Score**: Rank nodes with equal Instance or Instantiation values via their dependency scores.

The analysis shown in the middle right pane of Figure 6 lists the different instance options for each analyst selected unknown node.

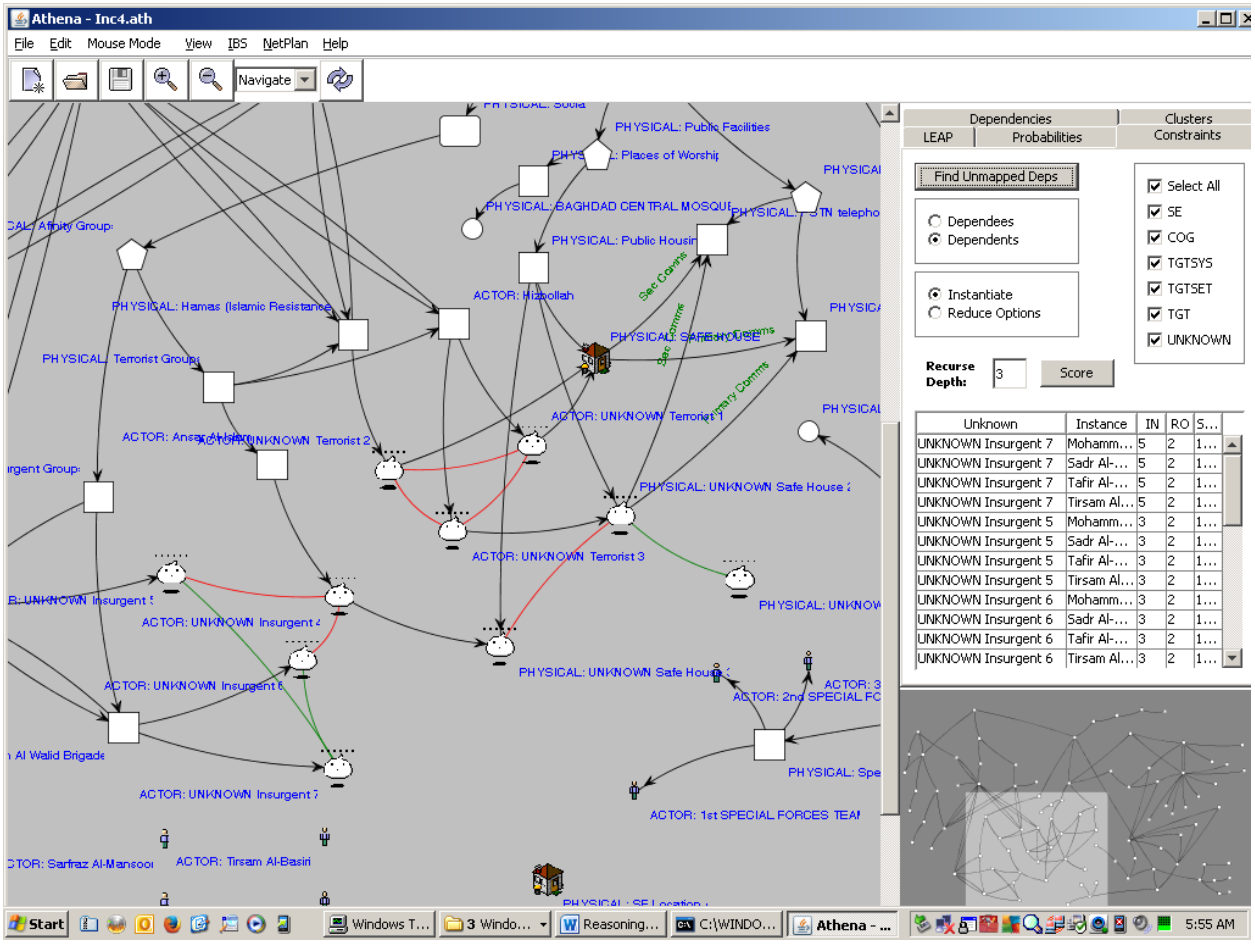

Figure 6: Analysis of Instantiations and Option Reductions

For example, *unknown insurgent 7* could be instantiated to *Tafir Al-Siddiqui, Tirsam Al-Basiri*, *Sadr Al-Muqtada* or *Mohammed Al Sarqawi*. This set is populated based on its direct and indirect *same-as* and

not-same-as constraints. The column headed **IN** identifies the number of forced (direct and indirect) instantiations and the column **RO** identifies the total number of instance options for the specified instantiation. For example, the first entry shows instantiating *unknown insurgent 7* to *Mohammed Al-Sarqawi* results in 5 forced instantiations and 2 option reductions. **Score** provides the nodes cumulative dependee or dependent score and is selectable by the analyst.

Clicking on a selected node in the analysis pane highlights in the network display the nodes involved in the **IN** and **RO** values. These values can be compared for different instances within the same unknown and/or with corresponding values for other unknown nodes. For example, if it can be shown that *unknown insurgent 5* is Tafir Al-Siddiqui with **IN** and **RO** scores of 5 and 2 then this would have greater impact on the network (in terms of forced instantiations) than if he were *unknown insurgent 6* with **IN** and **RO** scores of 3 and 2 respectively. Scrolling down the analysis pane would show an **RO** score of 6 for one option for *unknown insurgent4* hence this would have the greatest impact in terms of option reduction. However, with an **IN** score of 1 it would have limited impact on forced instantiations. Using VOI allows the analyst to explore various tactics in deciding which unknown node to focus upon and its relative importance. The analyst may want to quickly reduce the option sets of unknowns by trying to instantiate the unknowns with the largest **RO** scores allowing analysts to quickly identify unknowns with zero option solutions. Alternatively, the analyst may want to quickly develop an understanding of the most important persons, places, etc. in the scenario by choosing the unknown with the largest **IN** score.

4.3 Reasoning with Unmapped Dependencies

An unmapped dependency is one for which a node with the required capability has not been identified and mapped. An unmapped dependency can be defined in two different ways:

- The node with an unmapped dependency is linked to one or more nodes in the network but the mapping of node capability to node dependency has not yet been defined (Type 1)
- The node with the dependency is not linked to any other node within the network (Type 2)

Figure 7 shows the dependency link specification from the *unknown insurgent 7* and *unknown safe house 3*. The top portal shows the insurgent's dependencies on the left and the capabilities of the safe house on the right. The lower part of this portal shows the Relationship mapping "*Protection: Arrest*" inserted by the analyst between the two nodes. The strength of the dependency is shown in the lower portal has a value of 7.0[5]. A Type 1 unmapped dependency therefore occurs when no relationship or value has been provided for the dependency link Relationship specification. Examples of Type 2 unmapped dependency are shown at the bottom of Figure 8 which shows a collection of nodes (actors and physical locations) that are not currently linked into the network. Unmapped dependency reasoning allows analysts to rank nodes based on the number and type of unmapped dependencies and their dependent or dependee scores. This identifies unknown nodes that are important but have low information levels with regard to knowing all its dependencies. This also allows the analyst to compare the informational value of focusing on an instantiated node versus an unknown node.

[5] These can be inserted either by the analyst of generated by the link analysis algorithm LinkCalc [4]

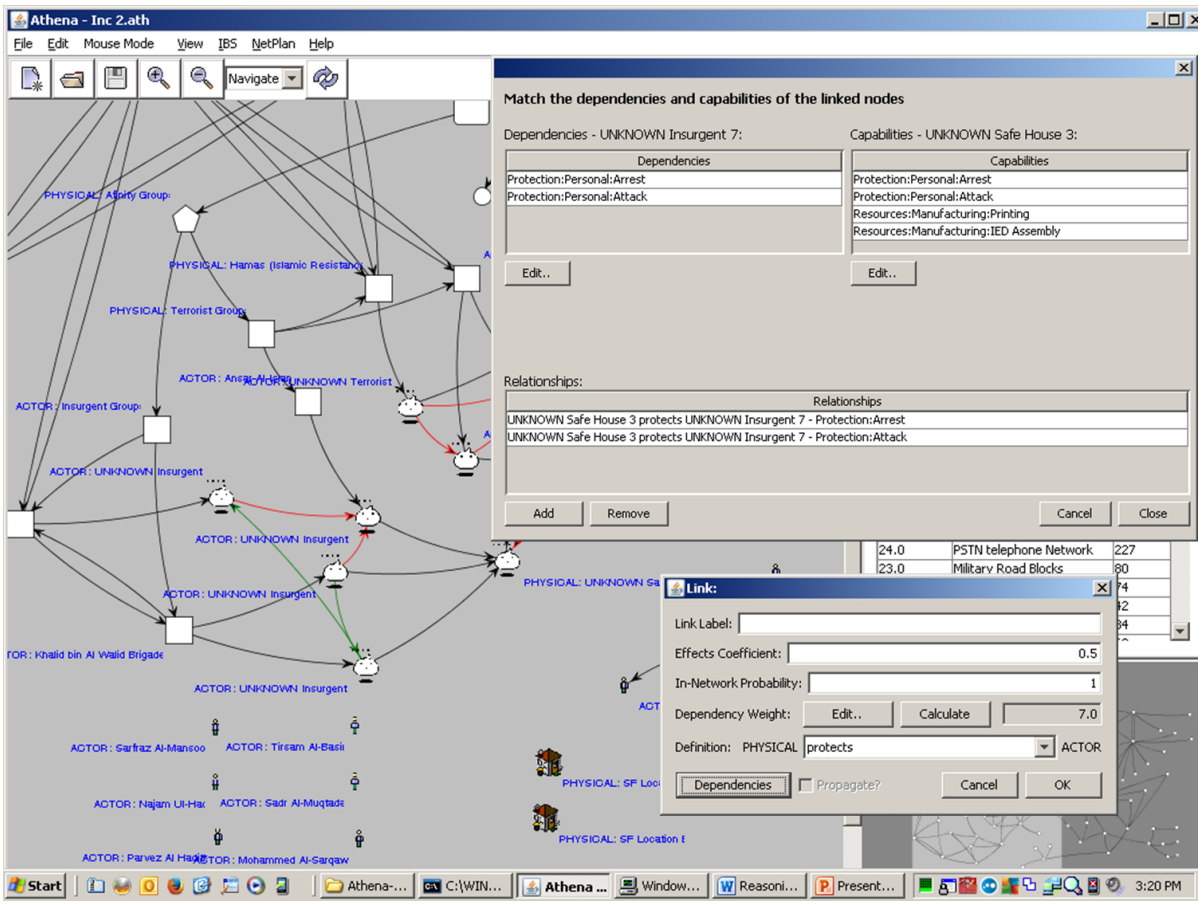

Figure 7: Example Dependency Mapping Specification

A node can be affected **directly** by means of it vulnerabilities or **indirectly** by means of its dependencies. Hence, the greater the level of information on a node's dependencies the more potential options there are to disrupt its capabilities. Identifying a safe house may reduce the options for an unknown insurgent (i.e. who is seen at the safe house location reduces the potential insurgent options). Again, the VOI to the analysts is knowing the direct effect (number of unmapped dependencies addressed) and the indirect effect of making the instantiation of the unknown insurgent easier. Thus giving them the ability to rank nodes. Analysts would frequently couple this type of analysis with RO and IN analysis to explore a hypothesis. For example, if the analyst can prove that *unknown insurgent 6*, the group's leader, is Mohammed Al-Sarqawi (by addressing his dependency on a safe location) then his IN score of 5 will result in several key members of the group being identified. The advantages of combining VOI from different methods is described in Section 6. Figure 8 shows an unmapped dependency analysis. This shows nodes *unknown insurgent 6* and *unknown insurgent 7* with dependent scores of 111.00 respectively and 2 unmapped dependencies are the most important with respect to the C2 leadership of the network. The Khalid bin Al-Walid Brigade with a dependent score of 104.0 is an important leadership node but only has one unmapped dependency. Hence. focusing on the unmapped dependencies of the two unknowns would provide more options for disrupting them then focusing on the Brigade. Combining this with IN and RO analysis may the indirect benefits of focusing on the unknown nodes.

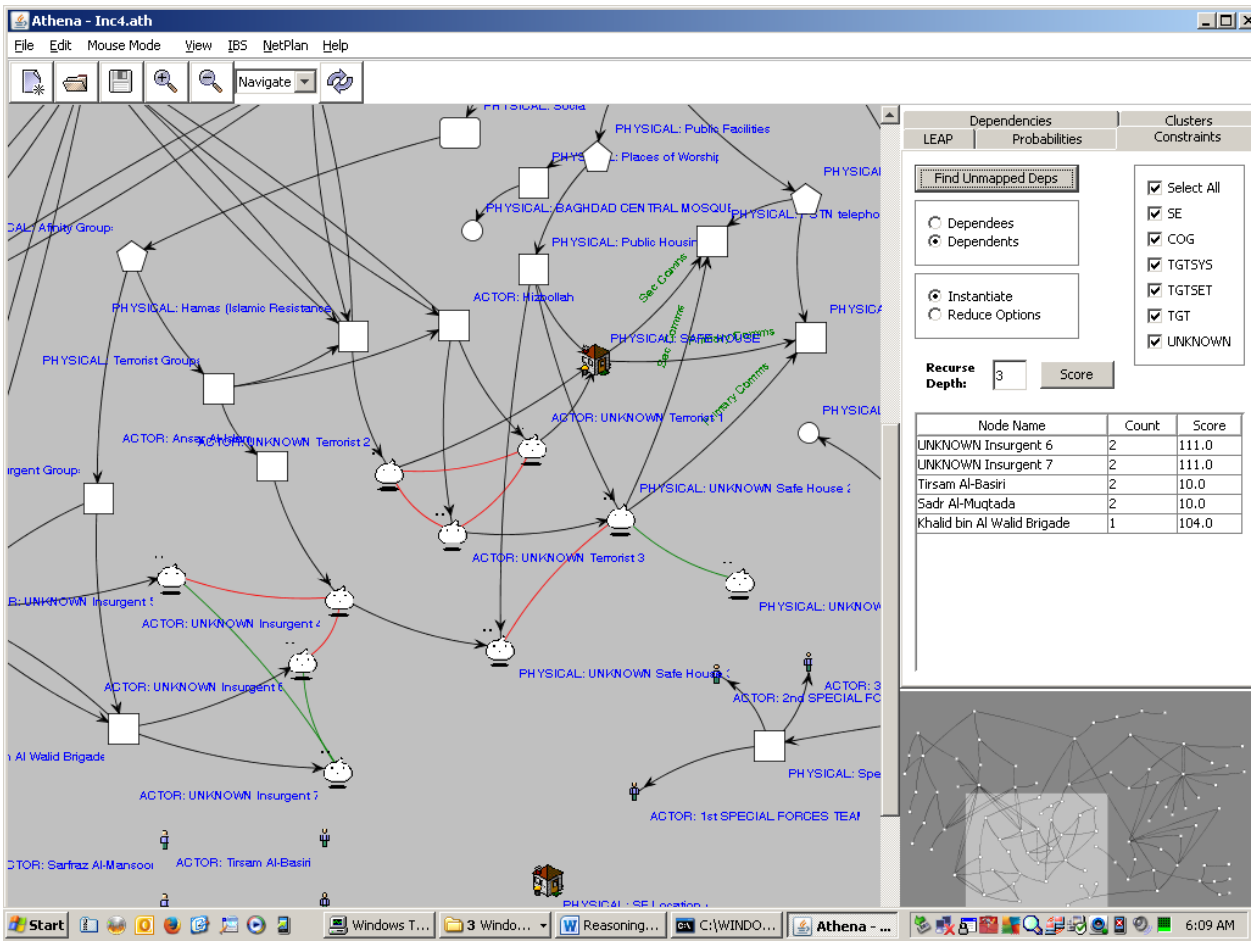

Figure 8: Identification and Ranking of Unmapped Dependencies

4.4 Reasoning with Node and Link Probabilities

Analyst specified membership and dependency probabilities are used to identify the cumulative probability associated with a dependency between any pair of analyst selected nodes. The right hand analysis pane of Figure 9 shows a dependee analysis for the terrorist group Hamas, showing a strong 7.0 dependency on *unknown safe house 3*. Given the membership and dependency values in the dependency chain from Hamas and the unknown safe house, what is the probability that the dependency does exist and is as strong as indicated? An analyst can select a pair of nodes and the algorithm will identify the ***Max*** probability path between them. Figure 10 shows the probability calculation for the dependency chain of Hamas on *unknown safe house 3*. This shows a value of 0.71 or 71% that Hamas is dependent on the safe house. The nodes listed are the ones involved in the dependency chain and the known ones are highlighted in grey in the network display. The "Prob" values normally show the dependency probability between the stated node and the node above. However, for confidentiality reasons these have been substituted with dummy values. The slider bar shown top right of Figure 10 allows the analyst to modify the probability calculation by increasing the weighting for membership values over dependency values or vice versa. In this example with the slider in the middle position membership and dependency probabilities carry equal weight.

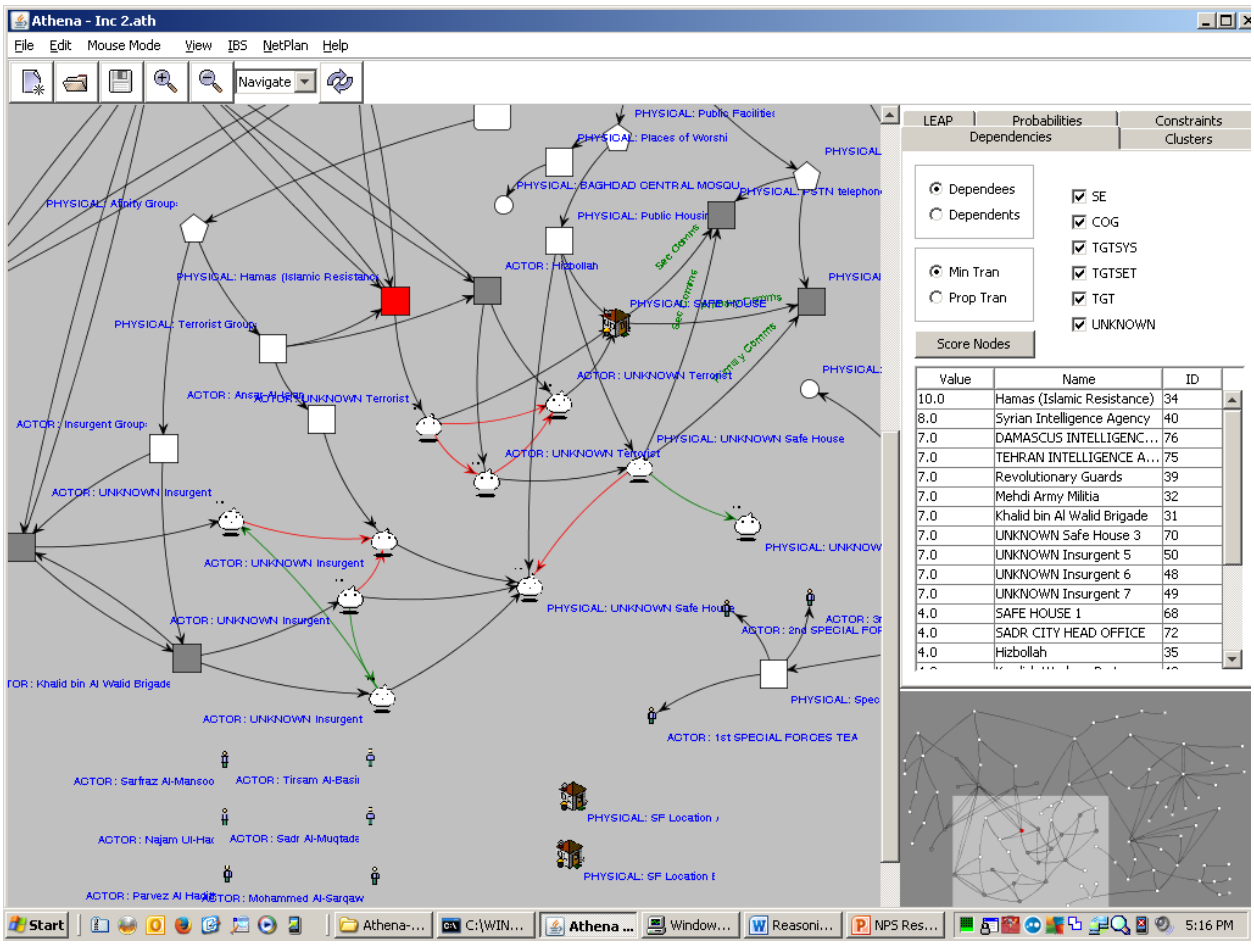

Figure 9: Dependee Score for the Hamas Node

If the analyst is seeking to identify options to affect the Hamas group, they would prefer to focus on ones with a high probability of existing indicated by a high max probability score. For example, if the dependency chain contains a dependency probability of 0.4 that *unknown insurgent 6* addresses the groups needs for leadership then despite a high probability that he uses *unknown safe house 3*, the overall confidence in the strength of the dependency chain could be questioned. Focusing on *unknown insurgent 6* to strengthen the probability he is providing leadership would potentially increase the **Max** probability of the dependency chain. Identifying which membership and/or dependencies in a dependency chain have low probabilities may also identify other pairwise dependencies that could be questioned as they contain the same or a large sub-set of the same low probabilities. This avoids the need for the analyst to explore all possible pairwise dependencies in the network.

4.5 Algorithm Overview

The basic dependency reasoning capabilities employ modified Floyd-Warshall [8] and Dijkstra's [9] algorithms to identify and rank the most important nodes in the network. Floyd-Warshall is a graph analysis algorithm for finding shortest paths in a weighted graph with positive or negative edge weights (but with no negative cycles) and also for finding transitive closure of a relation R. A single execution of

the algorithm will find the lengths (summed weights) of the shortest paths between all pairs of vertices, though it does not return details of the paths themselves.

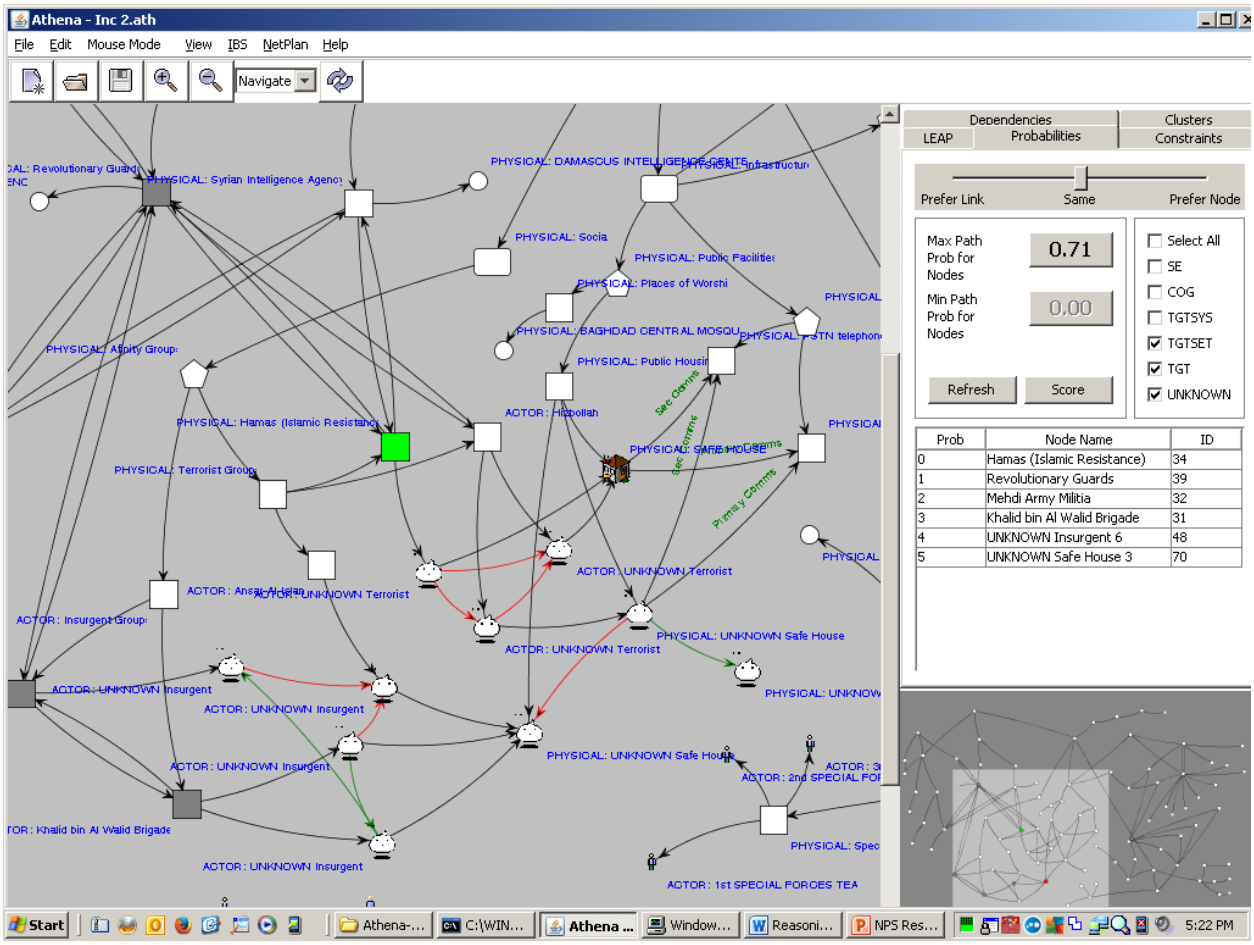

Figure 10: Probability Analysis for a Selected Pair of Network Nodes

The Floyd–Warshall algorithm compares all possible paths through the graph between each pair of vertices. It is able to do this with $\Theta(|V|^3)$ comparisons in a graph. This is extremely efficient considering that there may be up to $\Omega(|V|^2)$ edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal. Modifications were made to the Floyd-Warshall to allow classes of nodes and links to be included or excluded by means of user selections. The results of the Floyd-Warshall algorithm are used to generate the weighted scores for both dependent and dependee analysis

Dijkstra's algorithm is another algorithm for finding the shortest paths between nodes in a graph. It was conceived by computer scientist Edger W. Dijkstra in 1956. The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes, but a more common variant as employed in Cassandra fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest path tree. This algorithm is used to support the propagation of the direct effects on a node's capabilities and the indirect or n-order effects on other

node capability. Modifications were made to the algorithm to take into account probability values on nodes and dependency links.

The constraint reasoning algorithms propagates over a constraint graph comprising nodes and same-as and not-same-as constraints. The constraint graph overlays the dependency graph and the constraint links are handled separately from dependency links even though dependency and constraint links can connect the same pair of node. Propagation over same-as and not-same-as constraints occurs when a new constraint is added; it is assumed that at least one of the nodes involved in the constraint is unknown. The propagation may involve merging unknowns with instances (in the case of a same-as constraint). It may also involve reasoning based on counting members of instance sets and deducing forced assignments of unknowns to instances by means of not-same-as constraints. The reasoning is case-based and recursive: each change in the constraint graph starts a new round of reasoning by cases, leading to possible further changes in the graph in terms of instantiations and option reductions. A central aspect in developing the reasoning capability was to identify an appropriate set of cases to identify the required changes in the network. This aspect is especially important when collapsing the network due to combining multiple unknown nodes linked via same-as constraints which also have residual not-same-as constraints to other nodes in the network.

Prior to or during the development of a network, a hierarchy of instance sets needs to be defined and this is controlled by the analyst. Unknown nodes can be associated with multiple instance sets, based on attributes including capability, associations, etc. Hence the belief that a location may be a safe house or it may be a bomb factory" associates it with both the instance sets "Safe House" and "Bomb Factory". A strong assumption, but one that is used, is that associating an unknown with an instance set implies that it is identical to an instance in that set. There are various consistency checks that could be carried out. If an unknown is associated with both the "Safe House" and "Bomb Factory" sets, it should be the case that the instance is from the intersection of these instance sets. However, since the algorithm may want to turn hard assignments into probabilistic assignments at some point, this type of reasoning is generally ignored. Hence the algorithm allows an unknown to be associated with both "Safe House" and "Bomb Factory" even if there are no instances shared between them. Based on similar probabilistic grounds, the algorithm allows two unknowns to be considered the same without explicitly combining them into a single unknown. Since instances are assumed distinct, there is an implicit not-same-as constraint on all instance-instance pairs. Other types of pairs (unknown → instance, unknown → unknown) can be constrained by same-as, not-same-as, or neither.

Because of the strong assumption mentioned above, if an unknown is associated with an instance set containing *k* instances, and it is in *not-same-as* constraints with exactly *k-1* of these instances, it must be identical with the remaining instance in the set, which can be indicated by adding a *same-as* constraint between the unknown and instance. This essentially means the unknown and instance "merge" into a single node and the unknown is removed from the network while ensuring all capabilities, dependencies and links for the unknown node are integrated with those of the instance node. Essentially if *U* is the unknown and *I* is the instance, both *U* and *I* may have constraints on them prior to the merge, and in this case the constraints on *U* will be transferred to *I*. These constraints will necessarily be *same-as* and *not-same-as* constraints involving other unknowns. If such a constraint is of type *same-as* between *U* and

U1, this will become a *same-as* constraint between *U1* and *I*, which then forces *U1* to be merged with *I*, and a further transfer of constraints will occur. On the other hand, if the constraint between *U* and *U1* is of type *not-same-as*, this will become a *not-same-as* between *U1* and *I*. A check is performed to see if this new constraint forces *U1* to be assigned to an instance, according to the counting argument outlined in the preceding paragraph.

Within the algorithm, these cases are handled by a pair of functions that can call each other (and themselves) recursively. For simplicity, name these functions U_SAME_AS_I and U_NOT_SAME_AS_I. If U_SAME_AS_I is called initially, it might call either U_SAME_AS_I or U_NOT_SAME_AS_I, or both, depending on constraints on the unknown *U*. Additionally, U_NOT_SAME_AS_I might cause U_SAME_AS_I to be called if the counting argument on instance sets associated with *U* forces an instance to merge with *U*. This raises the issue of "how does the constraint propagation start?" Currently it is necessary to have an analyst assert an initial *same-as* or *not-same-as* constraint or modify an existing one within the network. In the future, if the further probabilities described in Section 6 are fully incorporated, it may be that *U* and *I* become constrained based on exceeding a threshold probability that a node is a specific instance. For example, if the three instances in the Safe House instance set are SF1, SF2 and SF3 and probability that SF2 is the instance reaches 90% then with a threshold of 80% this would force U to the instance SF2.

The probability reasoning algorithm propagates values over the nodes and links in the dependency graph. The algorithm identifies the maximum probability weighted path between two nodes selected by an analyst. The maximum probability path computation can consider only nodes, only links or both depending on an analyst supplied weighting. The computation is based on a function call to a Dijkstra's algorithm modified for the purpose. The basic Dijkstra algorithm was modified to handle the node and link weights by using the weight sums of logarithms of the node and link probabilities. The algorithm was implemented this way for computational convenience but is also justifiable mathematically.

The unmapped dependency reasoning algorithm employs a search algorithm that *walks* the dependency graph to identify all nodes that have an unmapped dependency taking into account unconnected subgraphs and singleton nodes.

5. Algorithm Evaluation and Testing

The algorithms were evaluated on two separate scenarios. The first focused on a counter insurgency scenario in Baghdad and the second focused on countering the development of WMD in Iran. The insurgency scenario contained 250 nodes and 600 dependency links where each node contained an average of 2.1 capabilities per node and 3.3 dependencies per node. Probability values were calculated for each nodes and dependency link. The WMD scenario contained 2650 nodes and 6818 dependency links where each node contained an average of 2.7 capabilities per node and 3.1 dependencies per node. Again, probability values were provided for each of the nodes and dependency links. For test purposes a number of unknown actor and physical nodes were either inserted into the network or existing instantiated nodes were turned into unknown nodes. Sets of *same-as* and *not-same-as* constraints were also added together with appropriate instance sets. The insurgency scenario was modified to create 3 different evaluation scenarios and the WMD modified to create 2 scenarios. The

evaluations were conducted with subject matters experts from the Aerospace Corporation who were selected by the project funders. Details of the scenarios and their content are provided in Table 1. The evaluation of the same-as and not-same-as propagation algorithms was conducted as follows:

1. Via information provided by VOI unknown node Instantiations were selected to enforce between 2 and 4 direct or indirect forced instantiations. All dependency links, capability and dependency attributes were handled correctly and merged with the appropriate nodes.
2. Unknown node instantiations were selected to cause between 1 and 6 option reductions across between 1 and 3 unknown instance sets. All reductions were correctly identified and those that forced an instantiation by means of a single option were identified and instantiated. All dependency links, capability and dependency attributes were handled correctly and merged with nodes with forced instantiations.
3. The analysis of unknown nodes to identify the number of instantiations and option reductions correctly identified all values for scenarios 1 through 5. Each scenario analysis was completed in 1 and 3 seconds depending on the size of the scenario.
4. The propagation of the effects of an instantiation or option reduction and the updates to the network nodes and links were completed in less than a second for each of scenarios 1 through 5.

Table 1: Scenario Evaluation Data Summary

Scenario	Scenario Number	Instantiated to Unknown	New Unknowns	Instance Sets	Constraints
Insurgency	1	9	5	5, average of 10 options	Same-as: 10, Not-same-as: 15
	2	15	5	10, average of 12 options	Same-as: 15 Not-same-as: 10
	3	25	10	15, average of 10 options	Same-as: 15 Not-same-as: 15
Iranian WMD	4	35	25	28, average of 10 options	Same-as: 15 Not-same-as: 25
	5	50	25	25, average of 12 options	Same-as: 20 Not-same-as: 25

The outcome of the evaluation is that the algorithm for propagating the effects of changes in nodes by means of *same as*, *not same as* constraints and instance sets is able to handle scenarios and problems sizes of interest to the analysts. The algorithms can provide solutions to the larger problems described in scenarios 4 and 5 in less than a second. Providing VOI to the analyst on direct, indirect, forced and zero option decisions significantly improved their productivity. VOI allowed analysts to focus on the real key nodes and links and not be distracted by examining nodes with large numbers of direct dependencies. The evaluation of the probability algorithms was conducted as follows:

1. Twenty pairs of nodes were selected with a dependency chain of between 3 and 12 links and included a mixture of instance and node sets. The node pairs included all combinations of unknown and instantiated nodes (e.g., unknown → instantiated, unknown → unknown, etc.).

2. For each node pair the probability weightings were set to link preference only, equal preference and node preference only resulting in 60 different test cases (number of pairs * number of preference options) and each case evaluated. For each node pairing and preference setting the algorithm correctly identified the **Maximum Probability** value in less than 2 seconds.
3. Once a **Maximum Probability** value was identified for a given node pair/preference option, one or more probability values (*dependency* or *membership*) along its dependency path was decreased forcing a recalculation of the **Maximum Probability**.
4. In some cases, this resulted in the identification of a new higher probability dependency path between the node pairing. As the network is a graph there are potentially multiple dependency paths between the selected nodes. Again for each node pairing and preference setting the algorithm correctly identified the **Maximum Probability** value for the path.

The outcome of the evaluation was that the algorithm for calculating the **Max Probability** identified the correct value. The reduction in probability values was used to reflect the analyst's hypothesis that one or more of the probabilities in the path could have a lower value. Hence what would be the consequences to their analysis if they were and which nodes and links should they focus upon.

6. Summary and Future Work

Evaluation of the approach with the SMEs, identified the algorithms for constraint reasoning, probability reasoning and unmapped dependency reasoning are capable of handling problems of the size and complexity of interest to analysts. Analysts found the insights provided by VOI regarding the **direct** and **indirect** effects of their decisions extremely useful as its significantly increased their overall situational awareness. Particularly with respect to direct, indirect, forced and zero option decisions. The evaluations confirmed that the algorithms appear to provide greater analyst insights when applied in combination rather than singularly. For example, using dependency analysis to identify the most important unknown nodes and then reducing this list further via IN and RO analysis. Finally, using probability analysis on the top 2 or 3 candidates to identify their key dependee nodes/links. In some cases, this resulted in analysts selecting an unknown node that was 5^{th} or 6^{th} in the initial dependency rankings due to its high IN and RO scores and high probability scores for all its key dependees. Further evaluations are needed on the strengths of combining the algorithms, pairwise, triples, etc. and the improved VOI they would provide.

Evaluation of the probability algorithm did identify that the current model did not provide the fidelity required to capture certain nuances of the scenarios in some cases. For example, all options in an instance set have equal probability however geographic information available to Cassandra would identify that several options are not in the locale or that two unknowns constrained by a same-as constraint are 200 miles apart. The ability to associate probabilities with specific options within an instance set and to propagate these values to influence the *dependency* probabilities is an obvious extension. The evaluation resulted in the identification of additional probabilities to measure the confidence in the instantiation of an unknown node, probability that a node possesses a specified capability or dependency and the probability of a class relationship (e.g. familial) between nodes would significantly improve the algorithm's capabilities. Further discussions are needed with analysts to identify and prioritize which measures need to be added and the additional VOI they would bring.

References

[1] Drabble, B., (2015), *Dependency Based Analysis to Support Robust Schedule Generation*, Final Report, Naval Postgraduate School, Contract N00244-14-1-0056 NAVSUP, Monterey, CA, USA.

[2] Drabble, B., (2011), Dependency based collaboration: Ontology based information management, *Proceedings of the Collaborative Technologies and Systems Conference (CTS 2011)*, (pp. 579-586), Philadelphia, PA, USA: IEEE.

[3] Drabble, B., (2012), Information propagation through a dependency network model, *Proceedings of the Collaborative Technologies and Systems Conference (CTS 2012)*, (pp. 266-272). Denver, CO, USA: IEEE.

[4] Drabble, B. and Kinzig, C., (2010), The information triad: Collaborating across structured and non-structured information, *Proceedings of the Collaborative Technologies and Systems Conference (CTS 2010)*, (pp. 255-264), Chicago, IL, USA: IEEE

[5] Drabble, B., McCrabb, M. and Haq, N., (2006), *Dependency Based Vulnerability Assessment*, Final Report, Defense Advanced Research Projects Agency, DARPA Order U051-16, Contract No: W41P4Q-06-C-003, Washington DC, USA.

[6] Grant, T. J., R. H. P. Janssen, and H. Monsuur. "Network Topology in Command and Control: Organization, Operation, and Evolution." 1-320 (2014), accessed May 21, 2015. DOI: 10.4018/978-1-4666-6058-8.

[7] Joint Expeditionary Forces Experiment: https://en.wikipedia.org/wiki/JEFX

[8] Floyd Warshall: http://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[9] Dijkstra's Algorithm: http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm