



**The Use of Data Mining and Network
Algorithms for Chemical Warfare Agent
Interdiction**

THESIS

Travis L. Cox, CPT, USA
AFIT-ENS-MS-17-M-120

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, the United States Department of Defense or the United States Government. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

AFIT-ENS-MS-17-M-120

THE USE OF DATA MINING AND NETWORK ALGORITHMS FOR
CHEMICAL WARFARE AGENT INTERDICTION

THESIS

Presented to the Faculty
Department of Operational Sciences
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Operations Research

Travis L. Cox, B.S., M.S.

CPT, USA

March 2017

DISTRIBUTION STATEMENT A
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT-ENS-MS-17-M-120

THE USE OF DATA MINING AND NETWORK ALGORITHMS FOR
CHEMICAL WARFARE AGENT INTERDICTION

THESIS

Travis L. Cox, B.S., M.S.
CPT, USA

Committee Membership:

Dr. Richard F. Deckro
Chair

Dr. Brian J. Lunday
Member

Abstract

The clandestine production of chemical warfare agents is a clear threat to the security of all populations in a world where groups and individuals will use any means necessary to terrorize others and advance their agenda. Chemical synthesis routes for the production of chemical warfare agents from common household chemicals are known. This is troubling because the clandestine production of drugs from common household chemicals has been notoriously hard to stop. This research discusses methods for identifying key targets to interdict the clandestine production of dangerous chemicals through the use of both shortest path and frequent pattern search algorithms.

Acknowledgements

It is often hard to adequately express how deeply grateful one is. It is something deeply felt but hard to describe. I am extremely grateful to my adviser Dr. Richard Deckro. He has made this endeavor much better than it would have been had I journeyed alone. Dr. Lunday also deserves much praise. He is one of the most admirable, humble, and caring people I know.

Travis L. Cox

Table of Contents

	Page
Abstract	iv
Acknowledgements	v
List of Figures	viii
List of Tables	ix
I. Introduction	1
1.1 Background	1
1.2 Problem Statement	2
1.3 Problem Approach	3
1.4 Research Scope	5
1.5 Assumptions	5
1.6 Thesis Organization	6
II. Literature Review	8
2.1 Chapter Overview	8
2.2 Chemistry in Operations Research	8
2.3 Computer Aided Synthesis Design	10
2.4 Network of Chemistry	13
2.5 Cost and Risk Determination	14
2.6 Multi-Objective Programming	16
2.7 Interdiction	19
2.8 Data Mining	21
2.9 Network Algorithms	26
2.10 Methamphetamine Regulation Effectiveness	29
2.11 Summary	32
III. Mixed Integer Linear Programming Model Methodology	33
3.1 Chapter Overview	33
3.2 Methamphetamine Network	33
3.3 Linear Programming Model	37
3.4 Cost Determination	39
3.5 Value Determination	42
3.6 Value Percentage Designation	44
3.7 Value Percentage Analysis	46
3.8 Conclusion	49

	Page
IV. Generating Test Chemical Networks	52
4.1 Chapter Overview	52
4.2 Graph Construction	52
Distribution of Node Assignments	53
Node Category Assignment	54
Edge Weight Assignment	56
4.3 Graph Testing	58
4.4 Conclusion	62
V. Methodologies for Targeting Chemical Synthesis Networks	64
5.1 Chapter Overview	64
5.2 Network Analysis.....	64
5.3 Conclusion	70
VI. Conclusion	72
6.1 Overview	72
6.2 Study Objectives	73
6.3 Conclusion	75
6.4 Recommendations for Future Research	76
Appendix A. GAMS and Python Code	77
1.1 GAMS Base Model Code	77
1.2 GAMS Loop Model Code	81
1.3 GAMS Full Model Code	87
1.4 GAMS Probability Analysis.....	94
1.5 GAMS Weighted Sums Pareto Code	100
1.6 GAMS ε -Constraint Pareto Front Code.....	101
1.7 Python Chemical Network Tree Generation	102
1.8 Python Scale-Free Chemical Network Generation	111
1.9 Python Powerlaw Fit Test	123
1.10 Python All Paths and Frequent Pattern Tree Code.....	124
Bibliography	130

List of Figures

Figure	Page
1	Aniline Synthetic Map [4] 10
2	Aniline Reaction Network Digraph 10
3	Network of Chemistry [54] 12
4	Chemical Reaction Petri Network 13
5	Pareto Optimal Front 18
6	Frequent Pattern Tree 25
7	Floyd-Warshall example graph 28
8	Red Phosphorus Methamphetamine Synthetic Map 34
9	Network Graph of Synthetic Map 34
10	Modified Network Graph 35
11	Tested Methamphetamine Network 37
12	Full Methamphetamine Network with Probability Arcs 47
13	Optimal Solution at Default Probabilities 47
14	Pareto Points for Second Arc Value Method 50
15	Modified Normal Distribution 54
16	Example Representative Network 57
17	Bipartite Graph with Numerous Chemical in One Reaction 59
18	Example Representative Network 65
19	1000 Node Network 67
20	Target Chemical Area 68

List of Tables

Table		Page
1	Initial adjacency matrix	28
2	Predecessor matrix.....	28
3	Final adjacency matrix	29
4	Predecessor matrix.....	29
5	Bipartite Graph Powerlaw Fit	60
6	Frequent-Pattern Tree Results	68
7	Household Chemical Sources	69

THE USE OF DATA MINING AND NETWORK ALGORITHMS FOR CHEMICAL WARFARE AGENT INTERDICTION

I. Introduction

If I have indeed acquired these [nuclear and chemical weapons] weapons , then I thank God for enabling me to do so. And if I seek to acquire these weapons, I am carrying out a duty. It would be a sin for Muslims not to try to possess the weapons that would prevent the infidels from inflicting harm on Muslims.

Osama bin Laden

1.1 Background

The quote above was stated by Osama bin Laden in 1999 during an interview with *Time Magazine*. Unfortunately, the threat of terrorist groups using clandestine chemical agents did not die one dark night in Pakistan with the former al-Qaeda leader. Reports continue to surface that the Islamic State of Iraq and the Levant (ISIL) is using chemical weapons. One report from March 2016 notes that 1,500 people required treatment after ISIS employed rockets containing blister agents in Iraq [10]. Historically, the threat of covert agent use has extended beyond Islamic militant threats.

The Aum Shinrikyo cult in Japan, a little over 20 years ago, produced large amounts of these deadly chemicals. This group was exceptionally well funded with assets worth several hundred million dollars and properties extending from Japan to Australia [56]. This wealth allowed them to build five Hastelloy reactors and a

two-story distillation column capable of producing significant amounts of chemical warfare agents. Their scientists estimated that this facility could produce two tons of the nerve agent Sarin per day [53]. The group concentrated on Sarin production but was reported to have produced a pound of blister agent, 8.5 kilograms of sodium cyanide, a very lethal poison, and had on a few occasions produced the nerve agent VX, using it to assassinate one of the cult's opponents [53].

Secret production of chemical warfare (CW) agents by a group like Aum Shinrikyo has not yet happened, to our knowledge, in the United States. There is, however, current covert production of hazardous chemicals that occurs regularly in this country. Methamphetamine and other drugs are produced in small, unsophisticated labs throughout the country. The study of methamphetamine production parallels the hidden production of chemical weapons; studying how methamphetamine is produced and the effectiveness of interdiction methods provides valuable insights into how CW agents may be secretly produced. Most importantly, it provides insight into how the production can be detected and stopped.

1.2 Problem Statement

With the continued danger of terrorist threats in our country and the desires of such groups to produce terror by any means possible, it is necessary to develop better ways and means of thwarting these enemies. Highly toxic chemicals can be produced by any motivated individual in much the same way amphetamines are generated for the black market. Synthetic methods to produce sarin, sulfur mustard, tabun, and VX from unregulated and readily available household chemicals have been identified [30]. It is vital that clandestine production means and methods be studied to thwart this threat.

1.3 Problem Approach

In 1980, Grossman and Santibanez applied mixed integer linear programming to optimize chemical process synthesis [31]. The initial approach taken in this research uses a mixed integer shortest path network flow linear programming formulation. The production of a chemical can be represented using a network graph modeling framework. The source nodes of the network are the chemicals that can be combined to make the desired product. The sink node is the final chemical that an individual may seek to produce. Each synthesis pathway represents an assembly tree of the overall network, terminating at a sink node.

If a person wants to make water, they would have a hydrogen source node and an oxygen source node. However, this is not the easiest way to make water. Simply mixing the two gasses together does not create water. A spark or flame is required, which causes an explosion and the energy needed for them to combine. The expense of acquiring oxygen and hydrogen gas and the little yield produced from the reaction would be prohibitive. A better way would be to take a mixture that contains water already, for example, a can of soda. The soda could be boiled and the water vapor collected. The network graph could, in this case, be expanded from using only oxygen and hydrogen gas as source nodes, to have a source node of soda as well.

There is a cost associated with each step of the network. In the example above it may cost \$100 to make a liter of water from a tank of oxygen and a tank of hydrogen. There is also a cost associated with the danger of producing water this way. A person would have to make small quantities of water at a time to avoid blowing themselves up. It would cost much less to buy two liters of soda, boil the water off, and collect it.

In this research, the costs associated with the production of methamphetamine is first examined. This is done because clandestine amphetamine production parallels

the secret production of CW agents closely and helps answer questions about hidden costs that may be applied to CW agent production. A network graph is produced from the common methods of methamphetamine production. The cost to buy each chemical needed in the manufacture is determined. A minimum cost network flow problem is developed using the cost associated with acquiring each precursor chemical. The answer to the minimum cost flow problem provides the least expensive method for making methamphetamine using the chosen production methods and also helps indicate to law enforcement the key path to target and interdict the production.

In the case of methamphetamine production or any other production of an illegal substance, there is an additional cost associated with getting apprehended, as well as the availability of some precursors due to laws that have come into effect. These costs can be evaluated to help determine the monetization of the expenses. The value of not getting interdicted is incorporated into the objective function of the linear program. A multiobjective formulation can be used to evaluate the trade-offs between risk-taking on the part of the clandestine chemist and the physical cost of the chemicals. Building a synthesis network to use in the formulation of a multiobjective linear program provides a cost effective means of building a small network consisting of the synthetic routes that are most likely to be known to a clandestine chemist. Large databases with millions of known reactions are also available that offer a way to identify unknown synthetic routes. However, these databases are proprietary and cost tens of thousands of dollars to access.

Grzybowski [32] and numerous collaborators have looked at building a complex network of the reactions contained in these databases. The Network of Chemistry they developed has over ten million different chemicals and close to that many reactions, and it is available in a proprietary product called Chematica [54]. The network was not available for this research, so networks with similar properties were generated.

Using the generated network graphs, all shortest paths are determined for the production of a regulated chemical. An all shortest paths algorithm is used to identify each pathway of production for the target chemical, including each pathway from household chemicals. Each node along the shortest path from each molecule to the target chemical is then considered to be a set to be used in a set mining algorithm. The set mining algorithm helps to identify the most common chemicals and reactions used in the production of the clandestine chemical agent.

1.4 Research Scope

There will not be a focus on the large-scale or state-sponsored CW agent production. The interdiction of large-scale agent production could be conducted by tracking Hastelloy or other high-nickel and cobalt-containing metal alloy sales, as well as the through tracking the sales of large, glass-lined reactor and distillation columns. This large-scale production leaves much evidence and is easier to track than the small-scale clandestine manufacturing process, as shown by the continuing methamphetamine and clandestine drug manufacture epidemic that such scrutiny had been placed but with very little reduction in use. Focus is thus put on the interdiction of small-scale clandestine production efforts. Clandestine production is a more complex and harder-to-track endeavor that requires greater insight.

1.5 Assumptions

The main assumption is that a cell wishing to develop chemical warfare agents desires to do this in the least expensive way possible without getting apprehended. The premise that a terrorist cell is concerned with producing an agent in the most economical way possible may be invalid because the cell may not have the knowledge to do so and may not be concerned with the cost. The terrorists may only know one

way of making the agent and would not know how to adapt their methods or know about different methods. Additionally, they may not care about getting apprehended. It may be that they only desire to cause the fear and panic that would occur if they were trapped in the middle of producing a weapon of mass destruction in our back yard.

Another assumption is that the cheapest source of a chemical is the cost that can be found online. The main resource used to determine the cost of a certain chemical is the website of the chemical company Sigma-Aldrich. The Sigma-Aldrich website feeds the Chematica database when Chematica uses chemical costs in their algorithms [54]. Lower costs may be available from other manufactures. Walmart and Amazon were used as the primary sources to determine costs of household chemicals. It is also possible that local or other online sources are cheaper than these. Additionally, the black market may provide a viable, cheaper source than is found through a common Internet search. Avoiding the market all together is also possible through theft. The risk of getting apprehended would increase, but the materials would cost nothing.

The next assumption is that the entities who may desire to produce chemical weapons have the necessary, basic lab skills to perform common organic chemistry tasks. These tasks may include extractions, filtration, washing and drying agents, and distillations. No assumption is made as to the laboratory equipment that can be used. Common organic chemistry glassware could be utilized or some other makeshift means common to clandestine drug or explosive production.

1.6 Thesis Organization

Chapter 2 provides a review of literature supporting the methods and information relevant to the thesis. Chapter 3 provides an in-depth discussion of the methodology used in the thesis. A network model of methamphetamine production is developed,

and the minimum cost of production is determined including the cost of sourcing the chemicals; as well as the cost function for taking into account the risks of production being interdicted. Chapter 4 discusses the production of bipartite Petri network graphs for use as representative chemical synthesis networks. Chapter 5 discusses two synergistic methods to develop targets for interdiction of the clandestine production of dangerous chemicals. Chapter 6 provides the summary and conclusions of this research as well as suggestions for future research.

II. Literature Review

2.1 Chapter Overview

The purpose of this chapter is to establish the foundation of published literature supporting the methodology of this thesis. First, applications of operations research in chemistry will be discussed, and specifically, how optimization has been used in the design and synthesis of chemicals. Second, methods of determining the cost of chemical synthesis, as well as methods of quantifying and handling risk in optimization, will be shared. Next, the research involved in the interdiction of networks will be presented. Finally, a review of methamphetamine precursor regulations and studies on the effectiveness of the regulations will be examined.

2.2 Chemistry in Operations Research

Linear programming was first applied to chemistry in 1956 by Fenech and Acrivos [26]. The authors sought to introduce chemical engineers to the recently-developed mathematical procedure of linear programming (LP). To illustrate their point, the authors used an example of optimizing the profit of a chemical plant. Constraints on how much product could be produced each day, along with the cost of production and disposal of waste, were used [26]. Dantzig, one of the founders of linear programming, soon published a paper relating LP to chemistry. He used LP to minimize the free energy of a mixture of chemicals to determine what chemicals would be present, once the reaction was completed [20].

From the 1950s through the 1970s, much work went into answering the more complex questions in chemical engineering. However, due to the lack of computational power, it was difficult to answer chemical process design problems. Chemical engineers approached these problems by decomposing the entire process into smaller, more

manageable parts, through heuristics, and through incremental improvements made by experienced engineers [35]. Much work was completed in the particular process design problem of the heat exchanger network. This problem is nonlinear in nature, but Grossmann and Santibanez used mixed-integer linear programming (MILP) to approximate the problem with good results in 1980 [31]. Grossman and Santibanez, in the same paper, showed how MILP can be used to maximize the net present value of a chemical synthesis project. [31]

From these early beginnings, optimization has become commonly used in chemistry for design and synthesis, as well as process control and process identification purposes. Typical formulations continue to include heat exchanger networks, as well as chemical reactor networks, optimal distillation system design, thermodynamic optimization, and blending problems [24]. Studying reactor networks and heat exchanger networks is not the only area that network optimization has been used in operations research.

Network optimization and graph theory, as applied to chemistry, was first studied by Balaban, who depicted chemical reactions as graphs in 1966 [7]. Organic chemicals are commonly depicted as graphs with carbon atoms at the vertices and with edges of the graph representing bonds between the atoms. Graphs representing the whole chemical reaction process are depicted with chemicals being the vertices and the edges representing reaction steps [6]. After the networks are formulated, they can then be used to develop mathematical programming models allowing for the analysis and optimization of the synthesis process. An example of a chemical reaction graph, taken from *The Algebra of Organic Synthesis* by John Andraos [4], may be found in Figure 1. Figure 2 depicts the entire chemical reaction as a graph of intermediate nodes and a source and product node.

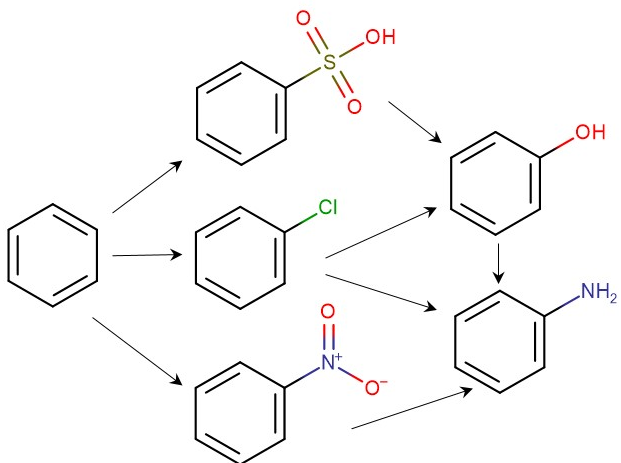


Figure 1. Aniline Synthetic Map [4]

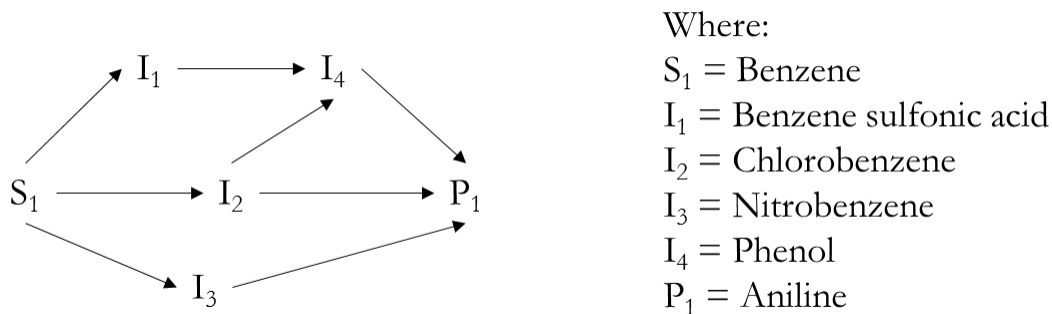


Figure 2. Aniline Reaction Network Digraph

The idea of using a network digraph to depict synthetic pathways has been extended to the mapping of entire databases of chemical reactions. This is part of the broader field of computer aided synthesis design (CASD). There are different approaches to using computers to design pathways from reactants to final products.

2.3 Computer Aided Synthesis Design

Generally, there are two different methods that are used in computer aided synthesis of chemicals. The first is by using rules-based methods, the second method involves precedents-based tools. Alternate products also exist based on reaction pre-

diction methods but have not been used in computer aided synthesis design [50]. Rules-based methods use algorithms to determine rules that govern favorable reactions. This approach dates back to Corey who was first to publish a set of systematic rules for retrosynthetic methods [14]. These methods focus on breaking a reaction down into fundamental structural features and working backward from the desired product to intermediate products until a desired starting material is reached. Previous to Coreys work in 1967, much of synthetic chemistry was focused on starting with the desired starting materials and searching for pathways that may lead to the desired product. Precedent-based CASD is a combination of the retrosynthetic methods and this older method.

The precedent-based method starts with the desired chemical and follow known pathways backward using databases of published reactions. SciPlanner and Reaxys are the two main tools that employ this method [50]. Using the Reaxys database of reactions, Grzybowski *et al.* have developed a network of organic chemistry NOC available through their product, Chematica. This is a step beyond the SciPlanner and Reaxys tools because it allows for the entire network of chemical reactions to be examined. SciPlanner and Reaxys use a greedy heuristic for path choosing. With SciPlanner, the user decides at each node, usually based on percent yield, what preceding chemical to use. Reaxys has this option as well, but also it provides an AutoPlan feature that generates alternative synthesis between which the user can choose [54].

If the user of these tools chooses a preceding chemical that has an 85% yield in lieu of a chemical that has a 75% yield at one node, it could lead to suboptimal results. Suppose the chemical with the 85% yield had a precessor with a 35% yield and the chemical with a 75% yield had precessors with 95% and 90% yields. Using a greedy approach and choosing the 85% yield chemical would make the pathway chosen suboptimal [54]. A knowledgeable user could avoid such pitfalls, but it would be a very

time consuming task checking each possible path. The number of possible synthesis to consider reaches on average 10^{19} options within five steps [39]. The Reaxys AutoPlan feature helps to remove some of the error introduced using a greedy method but is limited to 10 steps and does not evaluate all possibilities, only the "few best" [54].

Using a network model allows for all paths to be examined at once and allows for efficient network algorithms to be used. When examining a network of ten million nodes and close to ten million reactions [54], it is very important to be able to use efficient algorithms if an optimal path is desired. A graph of the NOC can be seen in Figure 3.

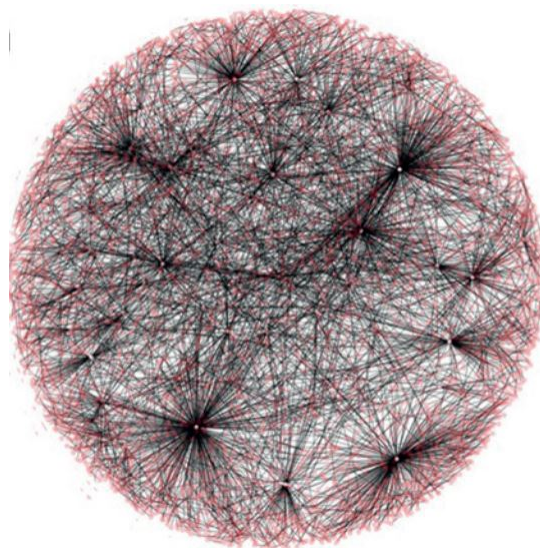


Figure 3. Network of Chemistry [54]

The Reaxys database claims to have 105 million organic, inorganic and organometallic compounds as well as greater than 42 million chemical reactions in the database. [25] Chematica does not have that many reactions and compounds. Grzybowski *et al.*, in the 2009 paper *The ‘Wired’ Universe of Organic Chemistry* [32], state that the Beilstein Database (the precursor to the Reaxys database) is used in the formation of the Chematica network. The researchers also pruned to remove catalysts, solvents and other substances that do not participate in reactions. At the time the

NOC consisted of six and a half million chemicals and seven million reactions [32]. In 2016 Szymku *et al.* published that the database used by Chematica had grown to 10 million chemicals and “a similar number of reactions” [54].

2.4 Network of Chemistry

Grzybowski’s Network of Chemistry has several defining characteristics. One characteristic is that it has been designed as a Petri net. Petri nets are networks consisting of state nodes and transition nodes. Petri first presented the idea of Petri nets in his dissertation in 1962. [49] This design is compatible for a chemical network because it allows the network to be constructed logically from a chemical point of view. The state nodes consist of the chemical substances and the transition nodes are made up of the reactions. For example, for the reaction $A + B \rightarrow C$ it does not make sense for an arc to go directly from A to C and from B to C because this seems to imply that A or B lead directly to C, when it is the combination that leads to C [54]. Adding a reaction node as shown in Figure 4 below represents the logical flow of the graph.

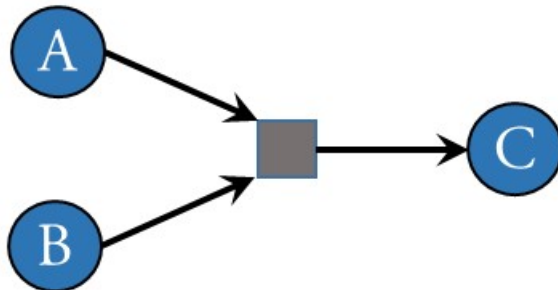


Figure 4. Chemical Reaction Petri Network

In addition to the NOC having a Petri net structure, the network has been shown to follow a powerlaw probability distribution [27]. This phenomenon was first described by Barabasi and Albert in 1999 as part of their effort to develop a graph generator that produced graphs more closely resembling real world networks than the

random graph generators being used at the time [8]. The powerlaw distribution is described as the probability $P(k)$ that a node interacts with k other nodes. The distribution decays following $P(k) \sim k^{-\alpha}$. Many real-world networks have been found to follow this distribution in the range $2 < \alpha < 3$ [12]. The NOC has $\alpha_{out} = 2.14$ for node interaction going out and $\alpha_{in} = 2.67$ for arcs coming in. [27]

Networks as diverse as protein interaction networks, the internet, social networks and movie actors are claimed to exhibit the powerlaw distribution. [12] Clauset *et al.* present the following steps to test power-law distribution:

1. Estimate the lowest value of k where powerlaw behavior could be expected to hold for the network as well as an estimate of α .
2. Find the goodness-of-fit between the power-law and the data.
3. Use a likelihood ratio test to compare alternative hypotheses.

Methods are also given by Clauset *et al.* [12] to conduct the above steps.

2.5 Cost and Risk Determination

For a deterministic network synthesis problem, the cost can be considered in various ways. This research sought to assign a monetary cost of production and was not as concerned with determining cost by the amount of waste produced or how efficient the reaction is in terms of percent conversion, percent selectivity, or percent yield. These percentages were not used because it was assumed that a clandestine manufacturer is more concerned with how much it costs to make a chemical while not getting detected making it. There is, most likely, little concern with the conversion percent from precursor materials as long as they are successful at obtaining the material. Similarly, a terrorist seeking to synthesize a chemical warfare agent is likely

more concerned with the cost of acquiring the materials to make the agent and not being interdicted before they can release it.

Determining the cost of chemical production is a large area of focus in the chemical industry. Recently much work has been done in the area in regards to green chemistry, the environmentally conscious use and production of chemicals. In *The Algebra of Organic Synthesis: Green Metrics, Design Strategy, Route Selection, and Optimization*, Andraos presented a formula of how to determine the cost of a chemical product [4]. This total cost is represented via Equation (1).

$$Cost = \sum \frac{\$reagents + \$solvents + \$work-up\ materials + \$auxiliary\ materials}{\text{mass of target product}} \quad (1)$$

Equation (1) was used to determine arc costs of the synthesis network by sourcing the different chemicals online. Of note, what is more difficult than this determination is calculating the hidden cost to a person seeking to produce a substance illegally.

These hidden costs are determined by incorporating risk into the model. The risk specifically considered is the risk of the clandestine chemist getting caught before the synthesis of the target substance is complete. The anticipated value of the chemist not being interdicted is worked into the objective function as a means of minimizing the cost of synthesis while maximizing the value of not getting apprehended. The objective is constructed from the viewpoint of the illegal chemist; if the optimal solution is identifiable from the clandestine chemist's prospective, the agency wishing to stop the synthesis will have a good idea of where to focus efforts to stop the production or make the production suboptimal.

Stochastic programming offers a method of incorporating risk into a linear programming problem. There is some debate as to when the field of stochastic programming started, but early papers include Dantzig's 1955 paper, *Linear Program-*

ming Under Uncertainty, [19] Beales’s 1955 paper, *On Minimizing a Convex Function Subject to Linear Inequalities*, [9] and Charnes, Cooper, and Symond’s 1958 paper, *Cost Horizons and Certainty Equivalents: An Approach to Stochastic Programming of Heating Oil* [11].

This paper draws on the idea of incorporating expectation and probabilities by incorporating a risk weighting method. The weights that should be assigned according to different risks are hard to quantify; because of this, parametric programming was used to examine the changes in solutions due to changing risk weight percentages. Parametric programming was introduced by Gass and Saaty in 1954 [51] and was later expanded to integer linear programming by Roodman in 1972, Bowman in 1974, and Nauss in 1975 [46]. Parametric programming is used as a means to examine changes in the objective function resulting from perturbations to the right-hand sides of the program’s constraints. Researchers vary these parameters to conduct sensitivity analyse, trade-off analysis, and shadow price analyse. In the area of multiobjective programming, parametric programming can be used as a more reliable means of examining shadow prices. In linear programming, the shadow prices obtained from the optimal solution represent the value of one more unit of a resource; in integer programming this is not necessarily the case [46].

2.6 Multi-Objective Programming

Multi-objective optimization does not typically yield a single optimal solution but instead yields multiple nondominated solutions referred to as Pareto optimal solutions. The minima on the boundary of the area defined by the possible Pareto optimal solutions forms a line known as the Pareto front.

Methods of finding the Pareto optimal front include the weighted sum approach. In this approach the objective functions are each weighted with different weighting

coefficients (w) where the sum of the weights usually equals one. The optimization is then stated as:

$$\begin{aligned} \text{Minimize: } & \sum_{i=1}^k w_i f_i(x) \\ \text{subject to: } & g_i(x) \leq 0, \quad i = 1, 2, \dots, m < n, \\ & h_j(x) \leq 0, \quad j = 1, 2, \dots, r < n, \\ & x_l \leq x \leq x_u. \end{aligned}$$

With two objective functions, the Pareto front can be graphed in two dimensions. The weight of the first function is set between some small value greater than zero and a value of one in steps of 0.01 (or some other element step size) with the weight of the second function equal to one minus the weight of the first function so that the objective is: $w_1 f_1(x) + (1 - w_1) f_2(x)$. The optimization is solved for each weight and the values of $f_1(x)$ and $f_2(x)$ are recorded for each iteration. Graphing $f_1(x)$ vs $f_2(x)$ shows the resulting Pareto optimal front [5]. Figure 5 shows the resulting Pareto optimal front from the following two objective functions:

$$\text{Minimize: } f_1(x) = 2x_1 + x_2$$

$$\text{Minimize: } f_2(x) = (x_1 - 3) + (x_2 - 3)$$

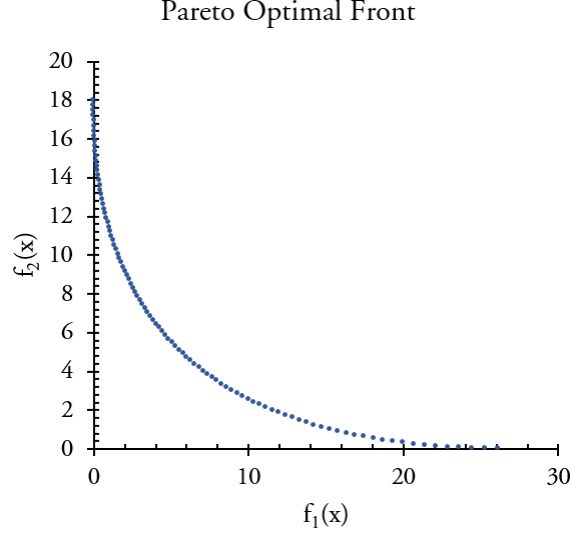


Figure 5. Pareto Optimal Front

The weighted sums approach is simple to implement but cannot be used for non-convex Pareto optimal fronts. Other methods of finding the Pareto optimal solutions include Goal Programming, the ε -constraint method, and meta-heuristic approaches. Of note, The multiobjective function for the model in this research is convex but produces a non-convex Pareto front.

Goal Programming involves setting a target goal for each objective function. The goal program's objective function then minimizes the deviation from the respective target goals. The optimization is then stated with $w_{1,k}$ and $w_{2,k}$ representing the weights for goal k , p_k representing the underachievement of goal k , n_k representing the over-achievement of goal k , and τ_k representing the desired goal [5]:

$$\begin{aligned} \text{Minimize: } & \sum_{i=1}^K w_{1,k}p_k + w_{2,k}p_k \\ \text{subject to: } & f_k(x) + p_k - n_k = 0, \quad k = 1, 2, \dots, K, \\ & p_k, n_k \geq 0, \quad k = 1, 2, \dots, K. \end{aligned}$$

The disadvantage to the goal programming approach is that the decision maker

must develop the weights and priorities without knowing how the objective functions may interact. Moreover, for non-linear objective functions, finding optimal solutions can be difficult [13].

For non-convex objective function, the ε -constraint method can be used to map a Pareto front. The ε -constraint method involves choosing one of the objective functions to be the primary objective function. The other objective functions are then set as constraints that are less than or equal to some value ε . As ε is varied, the solution to the mathematical program for each ε -value is recorded as a single Pareto point [5].

The advantages to the ε -constraint method is that it will produce a Pareto optimal front for non-convex objective functions and is relatively easy to execute. The disadvantage to the method is that it can produce weakly non-dominated Pareto solutions. Additionally, when the ε -constraint method is used as part of a meta-heuristic approach the ε -constraint method can be difficult to program when there are too many objectives [13].

2.7 Interdiction

Using the linear programming model to minimize cost and maximize the expected value of not getting caught from a potential terrorist's point of view, it is necessary to explore models to inhibit or stop the optimal production of an illicit material. As such, techniques were explored to interdict this production. R. Kevin Wood of the Naval Post Graduate School has completed much work in the field of network interdiction. In 1993, he formulated an interdiction problem as an enemy seeking to move something from a source node to a termination node [60]. The problem formulation is a directed network and is meant to consume no more than R total resources and leaving a network with a max-flow of no more than some value U . The formulation was based on the dual of a max-flow linear program. Each arc has a

capacity from node i to node j (u_{ij}), and an interdicator requiring a certain expenditure (r_{ij}) to interdict each arc.

Cuts are identified with the binary variable α_i representing nodes on the different sides of the cut, with $\alpha_i = 1$ for i if the node is on the terminal side of the cut, and $\alpha_i = 0$ for node i on the source side of the cut. $\beta_{ij} = 1$ if the arc from i to j is not cut and $\gamma = 1$ if the arc from i to j is broken. The total budget for interdiction is represented by R .

In order for this formulation to work for multiple sources some modifications must be made. A super-source and super-sink node are added with s' and t' nodes representing the original source and sink nodes. The first constraint is removed, all $\alpha_{s'} = 0$, and all $\alpha_{t'} = 1$. This is done to ensure that the super-sink and sink nodes are on the same side of the optimal cut and that the super-source and source nodes are on the opposite side [60].

The formulation is as follows:

$$\begin{aligned}
& \text{Minimize: } \sum_{(i,j) \in A} u_{ij} \beta_{ij} \\
& \text{subject to: } \alpha_i - \alpha_j + \beta_{ij} + \gamma_{ij} \geq 0, \quad \forall (i,j) \in A \\
& \qquad \qquad \alpha_t - \alpha_s \qquad \qquad \qquad \geq 1 \\
& \qquad \qquad \sum_{(i,j) \in A} r_{ij} \gamma_{ij} \qquad \leq R \\
& \qquad \qquad \alpha_i \in \{0, 1\}, \qquad \qquad \qquad \forall i \in N \\
& \qquad \qquad \beta_{ij} \in \{0, 1\}, \qquad \qquad \qquad \forall (i,j) \in A
\end{aligned}$$

The network interdiction problem (NIP) has been expanded upon many times. Kennedy *et al.* extended the Woods model to the interdiction of nodes, where the Woods model focuses on edge interdiction [37]. Cormican, Morton, and Wood developed a stochastic network interdiction problem (SNIP) formulation in 1996 [15]. Their formulation seeks to minimize the flow through a network when the success of

interdiction is uncertain. Lunday and Sherali have expanded the model to a dynamic network [41] and Morton, Pan, and Saeger developed a model for nuclear material interdiction using stochastic programming [44]. Their model involved an interdicator installing sensors to detect nuclear materials with a smuggler seeking to travel a transportation network undetected. Models were presented with the smuggler being aware of the sensor locations and another formulation where the smuggler is only aware of a subset of the locations [44].

One interesting extension of NIP was proposed by Zenklusen in 2010. He proposed a way of finding the minimum budget needed to remove arcs and vertices of a network so that the demands of a sink node cannot be met [61]. Another interesting extension was made by Israeli and Wood in 2002. They called the method the MXSP (Maximizing the Shortest Path) method. The assumption is made that an interdicator does not have the budget to fully disconnect the source node from the supply node. Instead, the interdicator seeks to make the path as long and expensive as possible for the path user [36]. Both of these principles could be used to good effect on the linear programming formulation of the chemical synthesis network.

For a very large program these interdiction models would be hard to formulate. For the 10 million node network of chemistry, quicker and more efficient methods were explored to identify targets for interdiction. The methods explored were data mining and network algorithms used for quickly identifying sets and paths of interest for targeting a chemical production synthesis.

2.8 Data Mining

As computer memory, storage, and processing capacity has grown, so has the need for efficient routines that are capable of sorting through the enormous amount of data that can now be collected and stored. Using data mining algorithms helps to sort

through this data and quickly arrive at valuable conclusions. In data mining, frequent item set techniques are used as a way to analyze items among a list of items that frequently occur together.

Frequent item set determination is often used in marketing and advertising. Purchase records can be mined to determine what products are frequently purchased together. This analysis helps to identify associations between purchased items for example to help determine the likelihood of a person purchasing Item B if they have purchased Item A or the likelihood that a person would purchase the items together. In this research this is applied to chemical synthesis to help determine what chemicals and sets of chemicals are involved most often in the production of a target chemical agent. If one chemical was unregulated and was involved in most of the synthesis pathways, a lawmaker may look at regulating the particular chemical to hinder the production of most of the pathways.

The first published frequent pattern mining technique is the Apriori algorithm. It was published in 1994 by Agrawal and Srikant [1]. The algorithm is not used in this research but is presented to provide a simple example of how set mining methods operate and as a comparison to the Frequent Pattern Tree algorithm used in this paper. The Apriori method is comprised of the following steps:

1. Count the number of times a particular item occurs in the original sets.
2. Prune the items that do not meet the minimum threshold number.
3. Create candidate itemsets of n items from the single items and count the number of occurrences of each set in the original sets.
4. Prune the itemsets that do not meet the minimum threshold number. (The threshold number reflects the minimum amount of occurrences a set must meet to be considered significant)

5. Create candidate itemsets of $n + 1$ items from the remaining n -itemsets.
6. Return to step 3 until no itemsets meet threshold criteria.

Consider the itemset consisting of the following sets of items:

- (1, 2, 3)
- (1, 3, 4)
- (1, 4)
- (1, 2, 3, 5)

Item 1 occurs 4 times; Item 2, twice; Item 3, three times; Item 4 occurs twice; and Item 5 occurs once. If the threshold criteria is a count of two, only Item 5 would be pruned; each of the other items occur at least twice. Next, candidate itemsets would be created from the possible combinations of the items remaining. The following sets would be created with the following occurrence counts:

Set	Count
(1, 2)	2
(1, 3)	3
(1, 4)	2
(2, 3)	2
~~(2, 4)~~	0
~~(3, 4)~~	1

The sets (2, 4) and (3, 4) are pruned, not meeting the selection criteria of 2. Three item sets are now created from the remaining possible items. The following set and occurrence count is obtained:

Set	Count
(1, 2, 3)	2

The set (1, 2, 4) cannot be formed because (2, 4) was pruned, and sets (1, 3, 4) and (2, 3, 4) cannot be formed because (3, 4) was pruned. The algorithm is now complete

Set	Count
(1, 2)	2
(1, 3)	3
(1, 4)	2
(2, 3)	2
(1, 2, 3)	2

because it is not possible to form a four item set meeting the selection criteria. The following sets are significant using the selection criteria of 2:

The sets (1, 2), (1, 3), and (2, 3) are included in the set (1, 2, 3) so the answer obtained from the Apriori algorithm is $\{(1, 4), (1, 2, 3)\}$.

The Apriori algorithm is often criticized because it can be slow for large data sets. This results from the fact that the algorithm must iterate over the whole dataset each iteration. Modifications have been made to the algorithm to improve this, and newer algorithms have been developed as well. One commonly used algorithm is the Frequent-Pattern Tree approach published by Han *et al.* [34].

The Frequent-Pattern tree approach involves building a frequency tree and analyzing the tree to obtain the most common sets. Using the same original itemset used in the example above, $\{(1, 2, 3), (1, 3, 4), (1, 4), (1, 2, 3, 5)\}$ the Frequent-Pattern tree algorithm would work as follows:

1. Count and reorder the items from greatest to least.

Item	Count
1	4
3	3
2	2
4	2
5	1

2. Add the first set, (1, 2, 3) to the tree, in the sorted order from above, counting the number of occurrences for each item.

3. Add the second set, (1, 3, 4) to the tree, tracking the number of occurrences.
4. Add the third set, (1, 4) to the tree.
5. Add the final set, (1, 2, 3, 5) to the tree.

Figure 6 below illustrates steps 2–5.

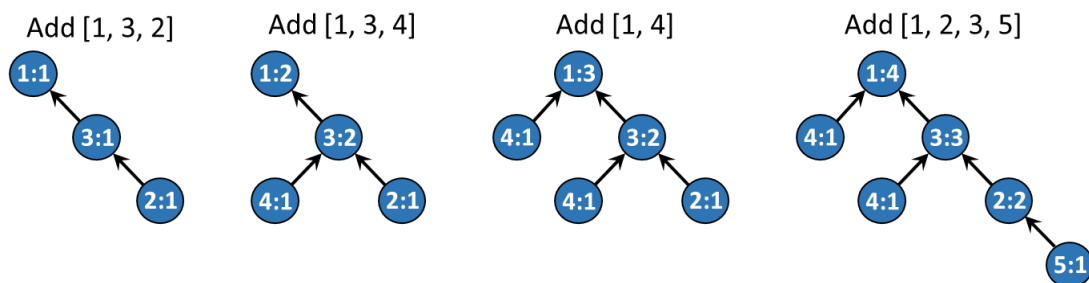


Figure 6. Frequent Pattern Tree

6. Analyze the graph starting with the last node, 5, and work up the tree. The following sets and counts are obtained:

Set	Count
~~(5, 2)~~	1
~~(5, 3)~~	1
~~(5, 1)~~	1
~~(5, 2, 1)~~	1
~~(5, 3, 1)~~	1
~~(5, 2, 3, 1)~~	1
~~(4, 3)~~	1
(4, 1)	2 (1 to 4 and 1 to 3 to 4)
~~(4, 3, 1)~~	1
(2, 3)	2
(2, 1)	2
(2, 3, 1)	2
(3, 1)	3

7. After pruning the sets that do not meet the selection criteria, the same results are obtained as above.

The Frequent-Pattern tree only iterates over the data set twice as opposed to each iteration for the Apriori algorithm. The first scan of Frequent-Pattern tree scans the database and collects the set of frequent items. The next scan builds the tree [34]. With very large data sets with large numbers of possible combinations this saves computation time. The frequent pattern tree algorithm is used in this research to gain insights into what chemicals are most commonly used in a synthesis process. A tree of all ancestors to a given chemical can be compiled and the shortest path from all household chemicals can be found using shortest path algorithms. If the chemicals in each path were considered as an itemset, the frequent pattern tree algorithm can be used to determine the sets of chemicals and reactions that are the most common in the synthesis from household agents. These sets can then be examined to gain insights into what chemicals would be valuable to watch or regulate in order to disrupt or detect and disrupt the process.

2.9 Network Algorithms

Shortest path problems are used commonly in network flow problems to find, as the name suggests, the shortest path from a source to a target node of the network. The algorithms used in solving the shortest path problem can be broken into two broad categories, label-setting and label correcting algorithms. The first label-setting algorithm was developed by Dijkstra in 1959 [23]. Shortly after, Dantzig [21] independently published a label setting algorithm in 1960 as well as Whiting and Hillier, independently the same year [48]. Dijkstra’s algorithm has since been modified to improve the running time and storage space required for implementation. Dial’s implementation, as well as various heap implementations introduce various storage structures in order to avoid scanning all temporary labeled nodes as required by Dijkstra’s algorithm [2].

The second broad category of shortest path algorithms are label-correcting algorithms. These algorithms label each node with a distance. This label is upheld and corrected in sequence until an optimal solution is obtained. Ford was the first to develop a label-correcting algorithm in 1956 [29]. This label correcting idea has been expanded and methods of finding the shortest path between each pair of nodes have been developed. The Floyd-Warshall algorithm, published by Floyd in 1962 is chief among them [2].

Algorithm 1 Floyd-Warshall [2]

Finds the shortest path between each node i and every other node j

```

for all node pairs  $[i, j] \in N$  do
     $d_{i,j} = \infty$ 
     $p_{i,j} = 0$ 
    for all nodes  $i \in N$  do
         $d_{i,i} = 0$ 
    end for
    for all node pairs  $d_{i,j} \in N$  do
         $d_{i,j} = w_{i,j}$ 
         $p_{i,j} = i$ 
    end for
    for all nodes  $k \in N$  do
        for all  $i \in N$  do
            for all  $j \in N$  do
                if  $d_{i,j} > d_{i,k} + d_{k,j}$  then
                     $d_{i,j} = d_{i,k} + d_{k,j}$ 
                     $p_{i,j} = p_{k,j}$ 
                end if
            end for
        end for
    end for
end for

```

Floyd and Warshall separately published what is essentially the same algorithm in 1962 [28, 59]. The algorithm is a matrix manipulating algorithm where an adjacency matrix is generated and then updated each step to keep track of the shortest distance. The algorithm begins with an adjacency matrix (d), arc weights (w), and a predecessor

matrix (p). Each iteration the matrices are updated subject to the condition that the shortest path from i to j must pass through node k . This is repeated until each node k in the set of all nodes, N , is passed through from each node i to every other node j , for all $i, j \in N$. The predecessor matrix is updated when a shorter path is found according to $p_{i,j} = p_{k,j}$. The pseudo code for the complete algorithm is found in Algorithm 1.

Table 1 shows an example initial adjacency matrix compiled from the information in the graph (Figure 7). Table 2 shows the initial predecessor matrix.

Table 1. Initial adjacency matrix

	A	B	C	D	E	F
A	0	4	∞	∞	∞	∞
B	∞	0	9	∞	∞	6
C	∞	∞	0	∞	10	∞
D	∞	∞	2	0	∞	∞
E	∞	∞	8	6	0	5
F	1	∞	∞	∞	6	0

Table 2. Predecessor matrix

	A	B	C	D	E	F
A	0	A	0	0	0	0
B	0	0	B	0	0	B
C	0	0	0	0	C	0
D	0	0	D	0	0	0
E	0	0	E	E	0	E
F	F	0	0	0	F	0

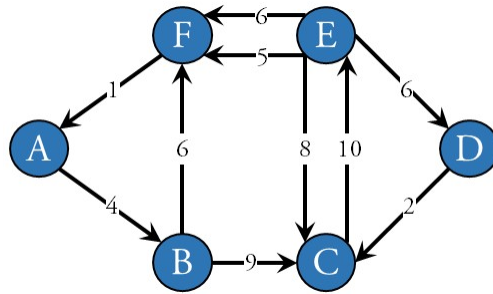


Figure 7. Floyd-Warshall example graph

The final adjacency matrix, Table 3, and predecessor matrix, Table 4, are displayed to show how the predecessor matrix can be used to determine the shortest path between each node. For example, the shortest path from A to D has a length of 22.

The predecessor matrix has node E as the predecessor to A-D. The predecessor to A-E is F, the predecessor to A-F is B, and finally the predecessor to A-B is A so the shortest path is $A \rightarrow B \rightarrow F \rightarrow E \rightarrow D$.

Table 3. Final adjacency matrix

	A	B	C	D	E	F
A	0	4	13	22	16	10
B	7	0	9	18	12	6
C	16	20	0	16	10	15
D	18	22	2	0	12	17
E	6	10	8	6	0	5
F	1	5	14	12	6	0

Table 4. Predecessor matrix

	A	B	C	D	E	F
A	0	A	B	E	F	B
B	F	0	B	E	F	B
C	F	F	0	E	C	E
D	F	A	D	0	C	E
E	F	A	E	E	0	E
F	F	A	B	E	F	0

2.10 Methamphetamine Regulation Effectiveness

Methamphetamine regulation effectiveness has been studied in some detail. For this reason, the study of methamphetamine interdiction was researched in order to gain key insights into the interdiction of the clandestine manufacture of chemical warfare agents.

Cunningham and Liu [16] reported on the impact of methamphetamine precursor regulations on arrests in California. They found that, after the first federal methamphetamine precursor regulation enacted in 1989, banning the purchase of bulk powder ephedrine and pseudoephedrine, arrests for methamphetamine transportation, possession, and sale leveled off at approximately 3,000 a month and then decreased over a two-year period to about 2,000 a month. After 1992, arrests surged to a peak of approximately 7,000 a month in 1995 [16]. The bulk powder law eventually lost its effect as drug makers started to source ephedrine and pseudoephedrine that had already been made into pill form [45]. It was noted by McKetin, Sutherland, Bright, and Norberg that this regulation came at a time of a wider effort to combat the drug

trade, and there was a general decrease in illegal drug indicators during this period [43].

In 1993, a second regulation was implemented requiring ephedrine tablet sellers to report sales of the drug [43]. Pseudoephedrine was left off the legislation to appease lobbyists from the pharmaceutical industry [45]. A slight decrease in arrests per month occurred in California, but this temporary decline did not notably affect the upward trajectory of the arrest rate [16].

The year 1995 brought a bill that required ephedrine sellers to acquire a DEA license and keep records of sales. [45] After passing this regulation, arrests in California decreased sharply for six months but quickly rebounded [16]. Methamphetamine supply and purity were reduced, and price increased as well. [17] A regulation requiring that pseudoephedrine product sales be tracked and reported came into effect in 1997. Arrests in California decreased from about 6,500 a month to close to 4,000 a month in 1999 but then began to increase once again [16]. Purity of methamphetamine on the street decreased, and price decreased sharply as well [17].

A law was passed on the federal level in 2000 that restricted the retail sales of pseudoephedrine to 9g from 24g. This law did not cause a measurable impact [43]. From 2000 to 2004, clandestine laboratory seizures increased along with per capita drug seizures. Price per pure gram has decreased, and average purity has increased along with per capita treatment episodes [47]. In 2005, Congress passed the most recent regulation at the national level. This law required ephedrine- and pseudoephedrine-containing products to be sold from behind the pharmacy counters and prohibited more than 3.6 grams to be purchased per day and 9 grams per month. Pharmacies are also required to track all sales [45].

The 2015 National Drug Threat Assessment Summary put out by the Drug Enforcement Administration (DEA) showed that there has been a 43.6% percent de-

crease from 2010 to 2014 in methamphetamine labs seized; however, availability continues to increase. Purity is increasing, and hospital methamphetamine-related treatment incidents indicate increased use [58]. The increase in hospital incidents could also be due to the increased purity and not due to increased use. The DEA also reported that methamphetamine seizures along the US/Mexican border have increased over the last four years and that “most of the methamphetamine seized in the United States is clandestinely produced in Mexico and smuggled across the southwest border.” [58] The decrease in labs seized in the United States could also be due to just a new production method being used stateside.

The popularity of “shake and bake” or “one-pot” methamphetamine has continued to rise [45]. This could also be a significant contributor to the availability of methamphetamine. This method allows methamphetamine cooks to produce the chemical in small containers such as a two-liter bottle; there is no need for a trailer or house to contain a lab and is much harder to be detected than previous methods. The DEA report indicated that, for the years 2010 through 2014, 83.7% of the labs seized were small labs only capable of producing two ounces or less of methamphetamine per batch [58].

Unfortunately, due to the influx of methamphetamine from the southern border of the United States, and the rise of more clandestine methods of production, it seems that for all measurable indicators the precursor regulations have ultimately been a failure. Most of the methamphetamine precursor regulations were able to reduce the availability of the drug for a time, as indicated by a reduction in arrests, reduced purity (indicating dealers cutting in other substances to increase their supply), and amplified street price (indicating a higher demand from decreased supply). This reduction has been temporary with methamphetamine now available in higher quantities and purity than it has been in the past.

2.11 Summary

Clandestine laboratory interdiction is a delicate operation. Methamphetamine production interdiction has been shown to be a challenging problem, but taking lessons from the history of efforts made to control methamphetamine production is helpful in providing an answer regarding how we might interdict the clandestine production of chemical warfare agents. There are significant tools available to produce a chemical reaction graph and apply deterministic operations research to provide insights into this complex problem. Using the Network of Chemistry to detect all published paths to chemical warfare agents and identifying key nodes to watch proves helpful in detecting the clandestine manufacture of warfare agents. Staying ahead of production methods should help to stop new methods of production before they being.

III. Mixed Integer Linear Programming Model Methodology

3.1 Chapter Overview

The methodology for the analysis of clandestine chemical warfare agents using a Mixed Integer Linear Program (MILP) is discussed in this chapter. This discussion begins with the analysis of clandestine methamphetamine production to help determine a suitable approach. A model is built using three methods of producing methamphetamine from ephedrine or pseudoephedrine. Clandestine methamphetamine analysis is done to first test the model and to draw insights into the clandestine chemical production process. The amount of information available on the clandestine production of drugs is far greater than that available on the clandestine production of warfare agents. The interdiction of drug production is well practiced and reviewed; whereas there has, fortunately, not been a lot of known activity on warfare agent production.

The methamphetamine chemical reaction network would be very large. During this research there was no available access to the large reaction databases so an attempt was made to construct a small portion of this network using select available publications. The mathematical model is formulated using an example of this small portion to illustrate the process and discuss the results. The positive aspects of constructing a model using a MILP are discussed as well as negative issues involved in using such a model.

3.2 Methamphetamine Network

A synthetic map of a portion of the Red Phosphorus production method [40, 52] for methamphetamine can be found in Figure 8. In the figure, a source of ephedrine or pseudoephedrine is used for methamphetamine production. Ephedrine or pseu-

doephedrine is then combined with red phosphorus and hydriodic acid to form iodoephedrine. Iodoephedrine is then processed via one of two ways to form methamphetamine hydrochloride, the final product.

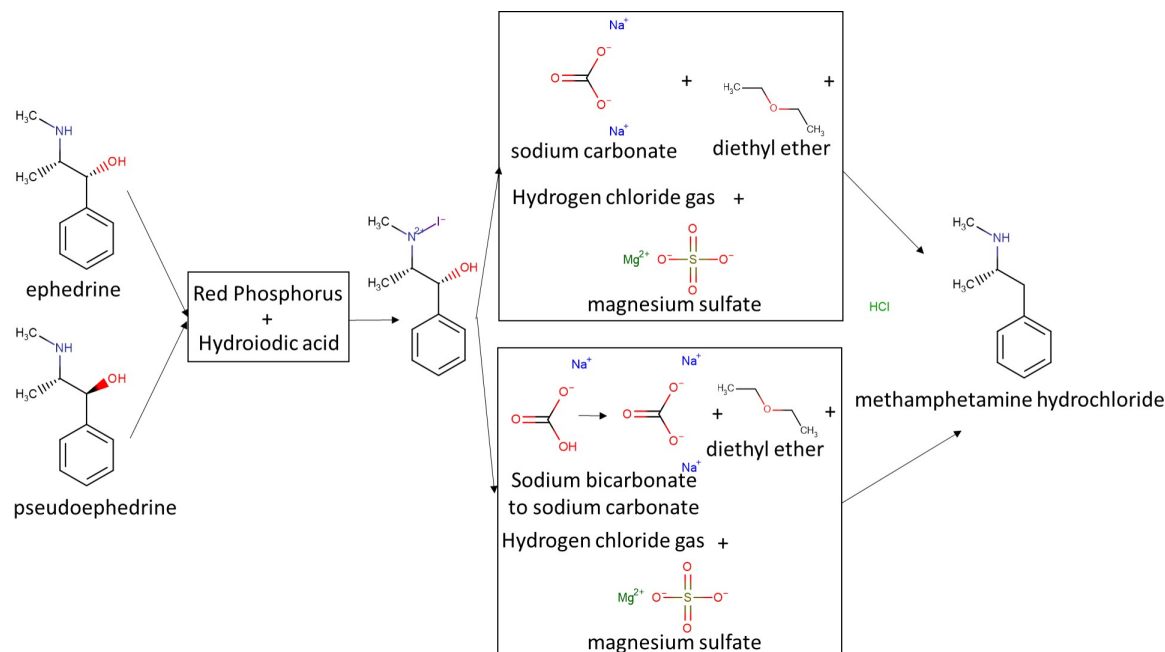


Figure 8. Red Phosphorus Methamphetamine Synthetic Map

The synthesis map in Figure 8 can be broken down into a simple graph as shown in Figure 9 with the following numbers representing the chemical nodes:

- | | | |
|-------------------------|----------------------|------------------------|
| 1- Ephedrine | 2- Pseudoephedrine | 3- Hydroiodic acid |
| 4- Red phosphorus | 5- Sodium carbonate | 6- Diethyl ether |
| 7- Hydrogen chloride(g) | 8- Magnesium sulfate | 9- Methamphetamine HCl |
| 10- Sodium bicarbonate | | |

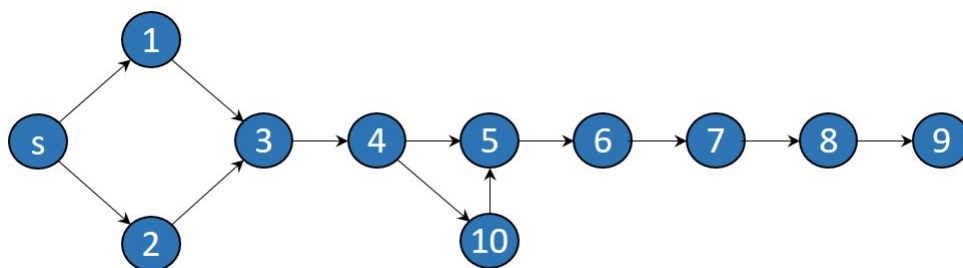


Figure 9. Network Graph of Synthetic Map

The order of the nodes represents an interaction or series of interactions that take place, leading to the product node. Each path from the source node to the product node represents all the chemicals needed to produce the product. Some of the arcs could represent reactant A combining with reactant B to form product C, whereas other arcs simply represent reactant A interacting with B. Of note, the model would still be valid if the order of the nodes were changed as long as the combination of nodes in the path remained the same. However, maintaining some logical flow is helpful for understanding the process. The order of some nodes remains arbitrary for the path or logical flow, for example, Nodes 3 and 4 could be interchanged with no consequence to the answer obtained from the model or the logical flow. If 3 and 4 combine to form iodoephedrine it is equivalent to 4 and 3 combining to form iodoephedrine.

The second modeling aspect to note is that, if a chemical cannot be purchased commercially, it is left out of the network. Iodoephedrine, the product of Nodes 3 and 4 (hydroiodic acid and red phosphorus) has not been included because it is not available commercially. If it were available commercially, it would be necessary to add another node. This node would be Iodoephedrine2 and would start a new pathway to methamphetamine. Figure 10 depicts this addition.

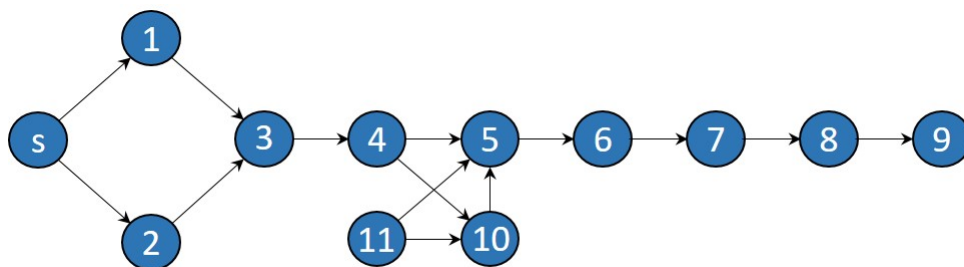


Figure 10. Modified Network Graph

One more additional note should be made about path $4 \rightarrow 10 \rightarrow 5 \rightarrow 6$ and path $4 \rightarrow 5 \rightarrow 6$. In path $4 \rightarrow 10 \rightarrow 5 \rightarrow 6$ sodium bicarbonate (Node 10) is used to make sodium carbonate (Node 5). In a case such as this wherein a chemical is synthesized

from another chemical, the weight of the arc from the chemical to the produced chemical is considered to be zero. Sodium bicarbonate forms sodium carbonate when heated in a dry heat source such as an oven. The power used to warm the oven has not been considered in the model. If desired, such costs could be incorporated into the model, but it was excluded from this research as a negligible cost because many of the other reactions also require heating and/or stirring but have been considered as insignificant compared to the cost and risk of being apprehended. The cost of power is most likely not a concern for an illegal chemist in a small clandestine lab, not drawing more power than a regular household, but it could be accounted for if the demand for power was high enough above normal levels that it could be used by authorities to identify unusual illegal activity. (Such identification is done with the power draw from grow lamps in marijuana production.)

Figure 11 shows a representation of the methamphetamine synthesis network that was used in the testing of the model. It depicts three ways of producing methamphetamine from ephedrine or pseudoephedrine. Various choices are shown with the use of different solvents and bases for the method depicted in the top portion of the network. The methods depicted in the middle and bottom of the graph could be similarly expanded as well additional methods of production. Allen and Cantrell discuss 22 different methods of methamphetamine and amphetamine production [3]. The network was not expanded beyond that depicted in Figure 11 for reasons discussed in the results section and in Chapter 4.

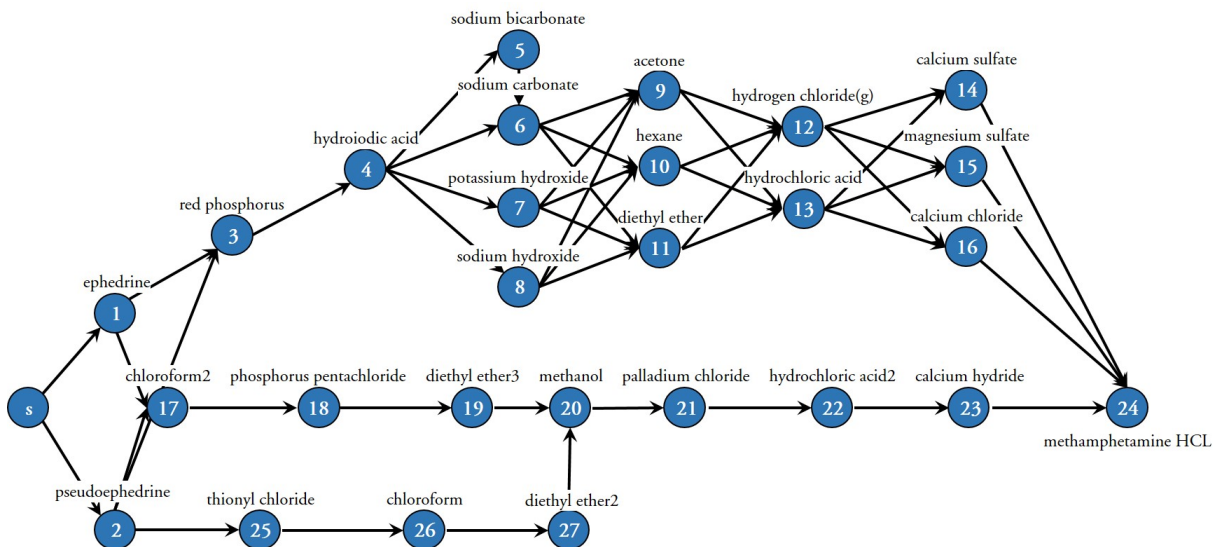


Figure 11. Tested Methamphetamine Network

3.3 Linear Programming Model

The linear program is based on a shortest path network flow model with some modifications. The variables and parameters are described as follows:

- Parameters
 - i is the node from which an arc begins. This represents a chemical in the synthesis pathway.
 - j is the node to which an arc terminates.
 - k is the cost number for a specific arc. This is used to choose between different possible arc cost from i to j .
 - t is the terminal (sink) node. This represents the final product.
 - s is the source node. This is used as a starting point and does not represent a chemical.
 - A is the set of all nodes.
 - K_{ij} is the set of all costs to connect node i to j .
 - C is the set of all sets K_{ij} .

- c_{ijk} is the cost of connecting an arc from node i to node j for cost number k . The cost is determine, in dollars per gram of the final product. c_{ijk} is determined by the cost of the chemical at node j . Cost determination is discussed in greater detail in the Cost Determination section.
- p_{ijk} is the percentage of an arc value that is assigned from node i to node j at cost number k (expressed in decimal form). This is assigned according to the risk of apprehension as discussed further in Section 3.6.
- v_{ijk} is the value of using arc i to j . The method of determining value is discussed further in Section 3.5.

- Variables

- z is the objective function solution value
- x_{ijk} is the connection of an arc from node i to node j for cost number k where x_{ijk} is binary with a value of 1 indicating the arc is utilized and a value of 0 indicating the contrary.

- Equations

$$\begin{aligned}
&\text{Minimize: } \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^o c_{ijk} x_{ijk} - \sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^o p_{ijk} v_{ijk} x_{ijk} \\
&\text{subject to: } \sum_{j=1}^m x_{sjk} - \sum_{j=1}^m x_{jsk} = 1, \\
&\quad \sum_{j=1}^m x_{ijk} - \sum_{j=1}^m x_{jik} = 0, \quad \forall i \in A \setminus \{s, t\}, \\
&\quad \sum_{j=1}^m x_{tjk} - \sum_{j=1}^m x_{jtk} = -1, \\
&\quad \sum_{k=1}^K x_{ijk} \leq 1, \quad \forall (i, j) \in A, \\
&\quad x_{ijk} \in \{0, 1\}, \quad \forall (i, j) \in A, k \in K_{ij}.
\end{aligned}$$

The objective function is made up of two objectives. The first objective is to minimize cost, the second is to maximize the use of the most valuable arcs. The arc percentages, p_{ijk} , are assigned according to how risky the arc is. If an arc is more risky, a lesser percentage of its value, v_{ijk} , is assigned as a weight to arc x_{ijk} in the second objective. A higher arc value serves to improve the overall objective to minimize the cost minus the value of each arc. The first constraint is used to ensure that there is one unit of flow from the source into the network. The second constraint establishes that if there is flow into a node, there is a matching flow out. Constraint three requires one unit of flow to reach the target chemical at the terminal node. The fourth constraint provides that only one arc is used into and out of a node when more than one arc exists. The final constraint is a binary restriction. When x_{ijk} is one, there is flow on the arc; when zero, there is no flow directed.

3.4 Cost Determination

The cost is determined using three main steps. The first step is determining the source of the chemical. The second is determining the price of the chemical per gram or per liter. The third step is determining the price per gram in terms of the final product.

During the first step, research must be conducted for each chemical to determine if the chemical is available from common household sources. The Household Products Database from the U.S. Department of Health and Human Services website [57] was used to determine whether a suitable common household source is available for a chemical. Additional online searches were conducted to verify that there was not a common source of a chemical if it was not listed in the Household Products Database. This information is found on many different websites and can be chemical dependent. These other online sources were verified using a product’s Safety Data Sheet (SDS),

the same source used to compile the Household Products Database.

If someone on an online drug making forum claimed you could get chemical x from product y , the SDS, typically available from a products webpage, is looked up for product y to verify the questionable online source is correct. If a common household source is available, online retailers are searched to determine the minimum cost during step two.

The second step is the pricing search. Amazon, Walmart, and Home Depot were used as the primary sources for non-laboratory-grade chemical pricing. When a common household, non-laboratory-grade, chemical could not be sourced, Sigma-Aldrich was used for laboratory-grade sourcing and pricing.

Step three involves determining the dollars per gram of the final product. This is determined by using the theoretical yield. The theoretical yield is used because it provided a consistent result. The literature provides an actual yield in some cases but these results are obtained by experienced chemists using quality laboratory equipment. This is most likely not the case for a clandestine chemical synthesis. The other factor in deciding to use the theoretical yield is that, in the literature, experimenters usually only report using a certain solvent or another chemical (i.e. a different acid or base for changing pH or a different drying agent) when it is possible to use multiple solvents or other acids or bases. If a different solvent or other chemical is used, the actual yield could be changed from the reported literature value. Using the theoretical yield as a baseline allows for a larger network to be modeled by incorporating different chemicals that could be used. Using the actual yield would be a more accurate way of expressing the cost in terms of dollars per gram of the final product but it was not possible during this research to conduct an experiment for each possible pathway to determine the actual yields. If such yields were known, it would be valuable to incorporate their use and would lead to more accurate results.

To illustrate how the cost in dollars per gram of the final product is obtained the following example is provided. If the literature calls for pseudoephedrine as a starting product, and a source of pseudoephedrine (PSE) sulphate is available from the Rite Aid pharmacy at a cost of \$14.29 per box of 10 tablets, then the following calculation would be made:

$$\frac{\$14.29}{10\text{tablets}} \times \frac{1\text{tablet}}{240\text{mgPSEsulphate}} \times \frac{1\text{mgPSEsulphate}}{0.823356\text{mgPSE}} = \frac{\$0.00723158}{\text{mgPSE}}$$

$$\frac{\$7.23158}{\text{gPSE}} \times \frac{1\text{molMethHCl}}{185.700\text{gMethHCl}} \times \frac{1\text{molPSE}}{1\text{molMethHCl}} \times \frac{165.236\text{gPSE}}{1\text{molPSE}} = \frac{\$6.435039}{\text{gMethHCl}}$$

The tablets per milligram (mg) of PSE is obtained from the packaging. The mg of PSE sulphate to mg of freebase PSE is obtained from the percent composition using the molecular weights as follows:

$$\frac{\text{MolecularWeightPSE}}{\text{MolecularWeightPSEHCl}} = \frac{165.236}{200.686} = 82.3356\%$$

The moles (mol) per mg and mg per mole of methamphetamine hydrochloride (HCl) and PSE are the molecular weights, and the theoretical yield is one mole of PSE to one mole of methamphetamine HCl.

The costs for the other chemicals used in the reaction that are not the main reagent, like PSE discussed previously, are determined in a similar manner. The price per gram of the chemical is found and then the grams required per gram of the main reagent is used to convert the price in terms of the main reagent and then to the price per gram in terms of the final product.

For liquids, the densities were used to convert to grams from milliliters. After converted to grams, the same procedure can then be followed. If a certain acid/base is used to neutralize a base/acid, theoretical acid-base chemistry was used. If an

experimenter used x grams of a certain base, the equivalent amount of another base was determined theoretically by determining the number of moles to be neutralized and the required amount of moles to neutralize the chemical. If a different solvent or drying agent was used, it was assumed to be an acceptable approximation that the same amount of a different solvent or drying agent would be used. Another item of consideration when determining cost is that the cost is considered to be zero if the chemical is produced from prior chemicals and is not required to be purchased, as discussed above.

3.5 Value Determination

Value is determined by using the stated linear programming model with only the cost portion of the multiobjective function. The objective function then becomes:

$$\sum_{i=1}^n \sum_{j=1}^m \sum_{k=1}^o c_{ijk} x_{ijk}$$

The percentage and value terms are removed. The model is solved in a loop with a constraint forcing each arc into the solution one at a time. The constraint $x_{ijk} = 1$ is implemented for each node one at a time until all nodes have been set equal to 1 and the solution for z is obtained with each arc in turn being forced into the solution. The same loop is then implemented, but with the constraint, $x_{ijk} = 0$, for each arc, one at a time, giving the value of z when each arc cannot be used.

The value is then determined by subtracting the value of z when the arc must be used from the maximum value of z obtained. For example, if the worst value obtained was $z = 13$, and when arc (i, j, k) was required to be part of the solution a value of $z = 3$ was obtained, the value of the arc would be $13 - 3 = 10$. The idea of this value determination is that, if the arc (i, j, k) is forced into the solution and it is the worst

solution, there is no value in using that arc over the others; and if the arc is part of the optimal solution then it has the largest possible value. The worst value is used as a baseline to determine how much better a solution that must use a particular arc is than the worst solution possible. This gives an approximate global value for each arc or how valuable it is to use that arc compared to all the other arcs. The most valuable arcs are the arcs that when forced into the solution produce the best solution. The least valuable arcs are the ones that when forced into the solution produce solutions that are furthest from the best solution.

This was implemented using GAMS modeling software (code is attached in Appendix 1). Two constraints were added to the model, as well as a binary variable y_{ijkn} . The parameter n was used to indicate whether there was a second cost arc or not for the given arc (i, j) and was implemented to ensure all cost arcs were forced into and out of the solution. The two added constraints then became:

$$x_{ijk} - y_{ijkn} \leq 1 \tag{2}$$

$$x_{ijk} + y_{ijkn} \geq 1 \tag{3}$$

The loop was then set to iterate for each arc (i, j, k) with $y_{ijkn} = 0$ and all other arcs set to $y_{ijkn} = 1$ and solving the system of equations. If $y = 0$, the two constraints enforce $x_{ijk} \leq 1$ and $x_{ijk} \geq 1$ which collectively enforce an equality constraint such that $x_{ijk} = 1$. If $y = 1$, the two constraints collectively require $0 \leq x_{ijk} \leq 2$, which allows x_{ijk} to be either zero or one because of the aforementioned binary restriction. This logical modeling technique was used to allow for the constraints to be changed within a loop in GAMS [42].

A second method of determining value was also examined by subtracting the z value obtained if you cannot use an arc from the z value if you must use the arc.

This yields negative values for arcs that are not part of the optimal solution and positive values for arcs that are. This was accomplished using Equations (2) and (3) set to be less than or equal to 0 and greater than or equal to 0, respectively, instead of bounding them on a value of 1. This allowed each arc to be forced out by constraining each x_{ijk} to be equal to zero and solved iteratively using the same loop.

3.6 Value Percentage Designation

The weight percentage is determined by the source of the chemical. The idea is that, if a chemical is purchased that has a regulation in place prohibiting the sale or restricting the sale above a certain level, there is a higher chance of detection if you use this chemical. The arcs to these chemicals should be assigned a smaller percentage of the arcs value. Beyond consideration of imposed regulations, the source of the chemical supplier was also used in the weight percentage determination. If a chemical is purchased that may be obtained for a legitimate household use from say Walmart there would be a low probability of detection so more of the arcs value should be assigned. If you are buying a box of baking soda, most outlets would assume you are making cookies, not methamphetamine.

The initial weight percentages are then set as follows:

1. High chance of not being interdicted, $p = 0.95$ (all percentages are expressed in decimal form). Defined as an unsuspecting common purchase. (i.e. sodium hydroxide purchased as Drano from Walmart)
2. Medium-High chance of avoiding interdiction, $p = 0.80$. Defined as an unregulated but suspicious purchase. This is a purchase of a common chemical in a way that requires some deception. (i.e. sodium hydroxide purchased as a laboratory grade chemical from Sigma-Aldrich, a vendor that is usually only used for businesses and educational institutions)

3. Medium-Low chance of not getting apprehended, $p = 0.3$. Defined as a regulated purchase that is common but tracked. (i.e. buying cold medicine containing pseudoephedrine from the Walmart Pharmacy)
4. Low chance of not being interdicted, $p = 0.1$. Defined as a regulated and suspicious purchase. This is a purchase of a regulated chemical in a way that raises suspicion. (i.e. purchasing laboratory grade pseudoephedrine from Sigma-Aldrich)

The initial percentages were chosen based on the estimated comparative suspicion that would be raised by procuring the chemical through the four main means of acquisition. If the chemical is unlawful to purchase or the quantity of purchase is regulated, trying to acquire the item would immediately raise suspicion and lead to a low percentage of the arc value being assigned. The estimates are only designed to provide a baseline for evaluating the model. Further discussion follows in the next section regarding the analysis of these weight percentages. These percentages are designed so the objective function can be formulated as an min and max multiobjective, minimize the cost and maximize the value of an arc when the chemical is acquired in an non-risky manner.

Setting weights as a percentage of arc values in this way is also an oversimplification. One thing to consider is that if someone is buying multiple chemicals it would be easier to determine what is being made even if each item individually is in the high percentage category. For example, if chocolate chips, flour, eggs, milk, and baking soda are all purchased together it is likely that chocolate chip cookies are being made. The arc value percents assigned are not independent of the other value percentages. If multiple items are purchased together, a lesser percentage of the arcs values should be assigned. Additionally, if someone is trying to stop another person from making chocolate chip cookies it may be the most useful to see if they buy chocolate chips,

ignoring the other ingredients. If the person is buying flour, eggs, milk, and baking soda they might just be making pancakes. If chocolate chips were being tracked in this manner there would be a higher chance of getting interdicted even if it is a common household purchase.

To overcome the difficulties of assigning a specific accurate percentage, analysis of the percentages is conducted to determine the ranges where the synthesis pathways remain the same and the points where they may force a change to another path. This is also helpful in determining what chemicals to regulate. If a regulator was considering the regulation of a certain low risk chemical, the model could be run with the percentage of that chemical reduced. If the elevated risk of getting apprehended did not force a change to another path it may not be beneficial to regulate that chemical.

3.7 Value Percentage Analysis

The value percentages were evaluated using the following method. Each risk category percentage was varied from zero to one in increments of 0.05. The model was solved using a GAMS model on the CPLEX solver available through the NEOS server. The NEOS server is hosted by the University of Wisconsin in Madison and offers the use of over 60 solvers [18]. The model was solved in a loop that incremented the percentages in one category from zero to one, solving the model at each percentage. The percentages were then placed back to the default value, and then the next category was solved incrementally using the same process until all four categories were solved for each percentage value.

Figure 12 shows the methamphetamine network with colored arcs corresponding to the probability value category of the arc.

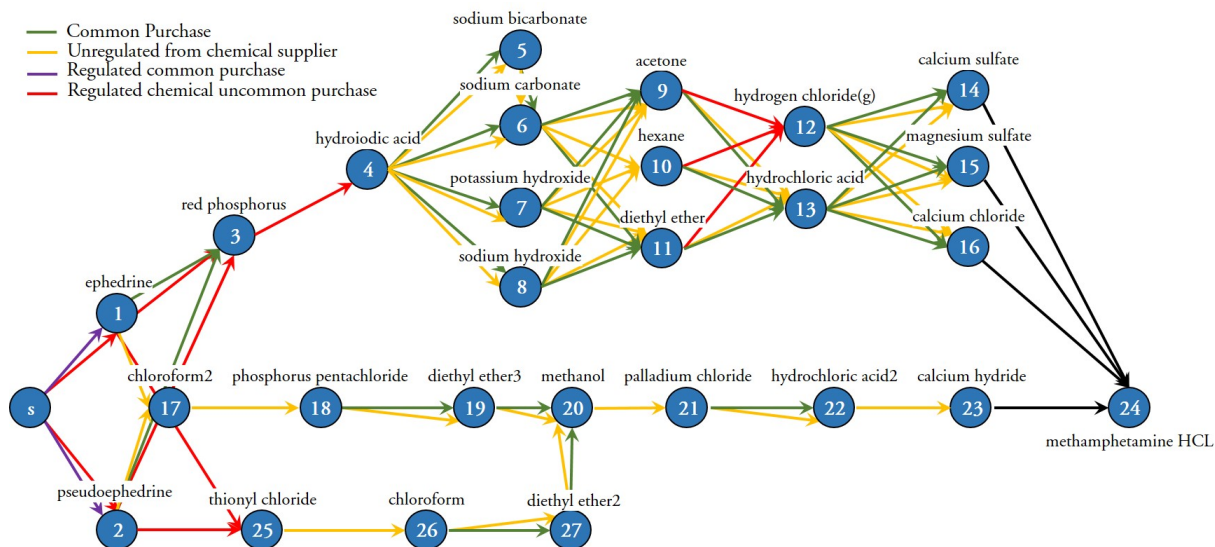


Figure 12. Full Methamphetamine Network with Probability Arcs

The most dominant solution at the default percentage values is shown in Figure 13. When solutions for the ranges of percentages are examined, there is very little variation from this solution. This is due to the dominance of the cost objective, this solution is the most cost effective route.

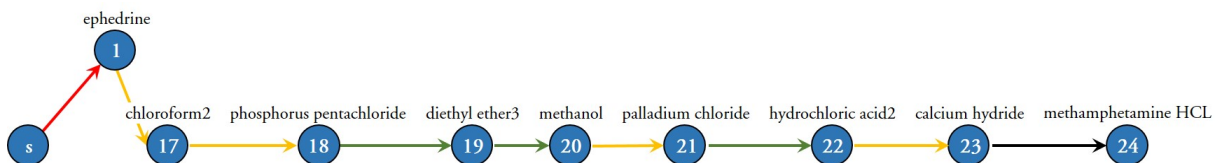


Figure 13. Optimal Solution at Default Probabilities

The upper pathway in Figure 12 going through red phosphorus is a far more common synthesis method for clandestine labs. The cost to acquire both red phosphorus and hydroiodic acid make this pathway uncommon in a model based on cost. The bias toward the results of the cost objective function indicate that cost has an overemphasis in the model.

This overemphasis is due to how arc values are determined for the arc value objective. Basing the arc values on the results from the cost objective reinforces the

optimal solution to the cost portion of the objective function. This was viewed as desirable during the initial formulation. It was hypothesized that it was desirable for the value function to reinforce the optimal solution to the cost function. The issue is that this valuation may reinforce it too much, not allowing for enough effect of the arc value portion of the objective because the value function is not independent of the cost function. This emphasis on cost leads to an overemphasis on cost and not enough emphasis on risk.

For all value percentages, except for when the medium-high category is above $p = 0.8$, the highest risk arc from source to ephedrine is chosen. This arc is chosen over the lesser risk arc of source to pseudoephedrine in the medium-low risk category. The difference in cost between these two arcs is \$4.31 per gram of methamphetamine so the less risky arc is not chosen until there is a value percentage of 0.8 assigned to that arc. It is more realistic for risk to have a greater effect on the path chosen. If a gram of methamphetamine were sold for \$100 it is likely that a criminal would easily take a \$4.31 cut in profit to reduce the risk of jail time. In this example there is little movement from the less risky path which would imply that this is not the case.

The unrealistic risk trade-offs are in part caused when the most costly arc is the less risky arc. In all but four of the cases (86.2% of the time) where two arcs exists between nodes, the more risky arc is also the more costly arc. It is when the less risky arc is the more costly arc that cost risk trade-off is a larger issue. Another point that may be causing an issue with unrealistic risk trade-offs is the range of costs themselves. Only 69.4% of the costs are more than \$2.00 per gram of methamphetamine. The costs have a mean of \$1.026, a standard deviation of \$2.021 and a median of \$0.1395.

It is also the case that the same four arcs that have a more expensive lower risk arc have a large discrepancy in cost. The prime examples are the two arcs from the source node to ephedrine. The first arc is part of the worst solution and the second arc is

part of the optimal solution. This is due to the \$11.00 cost difference and because the most risky arc is \$4.31 and \$1.75 cheaper than the only other possible arcs from the source node. When most of the arcs cost below \$0.14 that is a significant difference.

A model focusing on the risk without the emphasis on the cost is more beneficial from an interdiction standpoint. Later models developed in this research focus on a risk that increases with product availability and source without concern with cost. This same linear programming approach to a chemical synthesis network can be taken with weights based on risk of interdiction rather than cost. This would allow classic interdiction models, that are generally mathematical programming based, to be applied to a chemical synthesis network. The cost based model however, is still valuable from an economic standpoint.

From an economic viewpoint, the impact of the regulations to stop methamphetamine may have had a large impact in the cost of acquiring regulated chemicals. Further research into the costs of the chemicals before regulation would need to be conducted, but this model seems to indicate that the regulations have had a large impact on the cost of purchasing regulated precursor chemicals, pushing optimal paths, based on cost, to a more obscure synthesis pathways.

3.8 Conclusion

The dominant solutions are shown in the plot of the Pareto front, Figure 14. The plot was created using the ε -constraints method because of the non-convexity of the Pareto front.

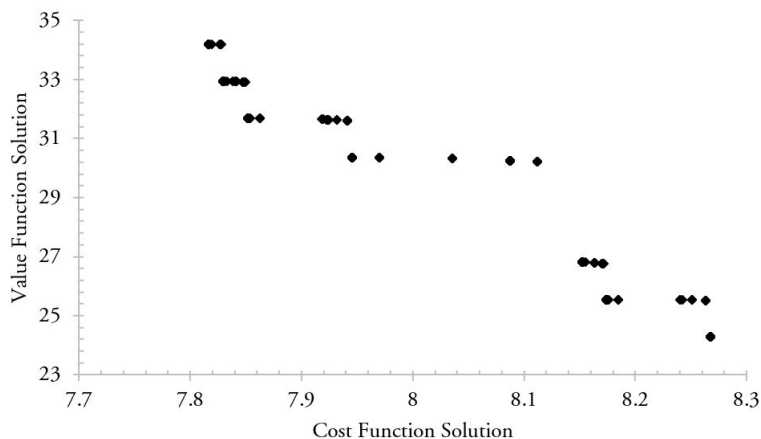


Figure 14. Pareto Points for Second Arc Value Method

The model was designed to determine the most cost efficient method of making a prohibited substance without detection. After the model was complete the goal was for an organization intent to stop the production of a chemical warfare agent to be able to change an arcs value percentage to gain insight into how effective a new regulation would be on that particular arc. If the arc was changed from an unregulated-common purchase arc to a regulated-common purchase arc, would it possibly change the most common method of production?

As the percentages were evaluated it became apparent that adjusting the percentages would not allow the model to be used in the way initially intended. The solutions for the example are too dominate for changes in the value percentage to reliably show the effects of a change to one arc in such a manner. Additionally, there are issues with unrealistic risk trade-offs that make it apparent that further adjustments must be made or a new approach taken. It may be that applying the model to a larger network may provide more variation and greater insight.

The model is valuable from an economic standpoint for a study of the costs of regulations. Additionally, the model proves very useful for a company concerned with the most economical synthesis pathways to a desired product. If the percentages were

based on market research and the chance that certain chemicals would rise or fall in cost, this methodology could provide valuable insight into the most cost effective synthesis pathways under market uncertainties.

This methodology has also shown how entrenched known methods of clandestine chemical synthesis can become. Synthesis pathways that are available from easily attainable materials and are widely known are more important than more cost effective pathways using harder to purchase materials. The optimal path primarily used laboratory grade chemical purchases with chemicals unavailable as household chemicals.

Identifying possible pathways from household chemicals becomes highly important for the prevention of synthetic routes that may become popular if the current pathways are shut down. Chapter 5 discusses a methodology to identify known and previously unknown synthesis pathways from common household chemicals. The next chapter briefly discusses bipartite Petri network generation methods to provide test networks for the methods used in Chapter 5.

IV. Generating Test Chemical Networks

4.1 Chapter Overview

The commercially available Chematica network of chemistry was not available for this research. The lack of access required that a representative network be created to test the methodology. Network trees and directed scale-free networks were created using the Networkx library. These networks were then modified to represent the Petri net type framework of the network of chemistry. The nodes designated as chemicals were then randomly assigned to be labeled as household, commercially available, non-commercially available, and regulated chemicals. The graph then received edge weights according to the chemical label assigned.

The Networkx graph generators used as starting graphs were:

- *barabasi_albert_graph*
- *gn_graph*
- *gnr_graph*
- *scale_free_graph*
- *directed_havel_hakimi_graph*
- bipartite *havel_hakimi_graph*

These were the main graph generators explored as starting graphs. Most of the other available random graph generators were explored and quickly eliminated as feasible options due to size constraints or lack of power-law degree distribution options.

4.2 Graph Construction

In Chapter 2 the network of chemistry was described in some detail. Representative graphs were generated in order to test the methodology used in Chapter 5. Graphs were first generated using graph generators from the Networkx Python

package [33]. The growing network graph generators were typically used as starting graphs for tree graphs because they attempt to generate a tree graph that follows the powerlaw. The scale-free directed graph generator was used as the starting point for non-tree graph testing. Binomial graph generators were also tested for use but were not used to test the weighted representative graphs.

The following general methodology was used to create the representative chemistry network.

1. Convert every other row of the graph to a chemical node
2. If a reaction node does not have a predecessor, assign a predecessor. This is assigned using a modified random normal distribution with a mean of 2.5 and a standard deviation of 1.8. See below for further discussion of distribution.
3. Assign nodes to sets of household, commercially available, regulated and non-commercially available. This is discussed below.
4. Assign weights to edges based on node sets.

Distribution of Node Assignments.

The normal distribution was chosen to determine the nodes assigned based on the desired number of chemicals involved in a typical reaction. A right tailed distribution with a mean of two was desired because the typical number of chemicals involved in a reaction is about two. As discussed in the literature review, there is an average of 2.67 inbound arcs and 2.14 outbound arcs in the network of chemistry. Using a random normal number generator with a mean of 2.5 and standard deviation of 1.8 lends to a right tailed distribution with a mean of 2.695 when all numbers are rounded and all numbers generated that are less than or equal to zero are set equal to 1. The

mean was measured when 500,000 numbers were generated. The modified normal distribution obtained is shown in Figure 15.

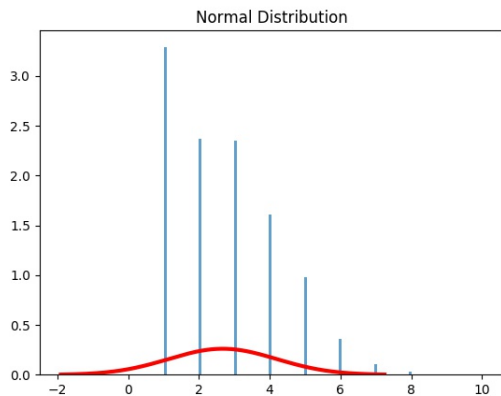


Figure 15. Modified Normal Distribution

A modified Poisson distribution provides a similar right tailed distribution when generated values of zero are set equal to one. The Poisson distribution with $\lambda = 2.5$ results in a mean of 2.574 and a shorter tail than is produced using the modified normal distribution. The modified Poisson produces a tail with 0.3914% of the values above five. The normal distribution produced 4.79% of the values above five. There was no indication found in the literature if a heavier tail was more indicative of reality. The normal distribution was however chosen because it provides more variety. This was viewed as desirable for testing purposes because it would provide for a greater probability that target nodes could be chosen from higher connected nodes for some testing iterations.

Node Category Assignment.

The number of possible household chemicals was determined to be 1,948, all of which are potetial household chemical sources. This number was obtained from compiling the number of household products listed on the Department of Health and

Human Services website [57]. A web extraction tool was used to compile a list of the chemicals and the CAS registry number. The JChem for Excel database was then used to compare CAS registry number and chemical names. This allowed substances like, asphalt and more ambiguous chemicals like “Water softener(s)”, with no CAS number to be screened from the results. These “chemicals” are not likely to be contained in the network of chemistry. After the chemicals were screened 1,948 entries remained of the initial 4,298 listings.

The number of regulated chemicals used was 161. Fuller *et al.* [30] mention over a thousand regulated chemicals in the network of chemistry but no specific number is stated. Szymkuc *et al.* [54] state that the Chematica database draws on lists published by the Australia Group, the Department of Homeland Security (DHS) and the Environmental Protection Agency (EPA) to compile the list of regulated chemicals used by Chematica. The EPA list is useful for designing an industrial synthesis but contains many chemicals that are environmental hazards. A further overview of the DHS list also reveals that the list contains chemicals that are regulated for toxic release and contamination thresholds [22]. For the scope of this model, the list was narrowed to the 161 chemicals regulated as precursors of chemical warfare agents or explosive substances, and regulated for theft threshold, the amount that must be reported, if stolen. The regulated substances not included were the substances that were solely regulated based on toxic release if the chemical fell into both categories it was included.

The amount of commercially available chemicals in the network was based on the desired ratio and could be set at 100% commercially available and zero non-commercially available chemicals. The ratio was typically set to 60% of the remaining chemicals after household and regulated chemicals were assigned. This percentage was chosen to help lower the computation time for the weighting algorithms and because

a majority of commercially available chemicals would be desired when planning a synthetic route.

The Chematica database gets the list of commercially available chemicals from the Sigma-Aldrich catalog. The catalog consists of 200,000 different chemicals [54]. 200,000 is a very small number compared to the 10 million total chemical nodes. A ratio of 60% was typically chosen because synthesis generally would take place from a commercially available molecule and a higher ratio helps to screen out non-commercially available chemicals that could not be used as a starting material unless it is naturally occurring.

Edge Weight Assignment.

The edges from chemical node to reaction node for these example networks were assigned a weight of one. This penalizes reactions that involve more steps. If there are two different pathways from a household chemical the pathway with less steps is generally viewed as desirable because there are less chemicals required, and therefore less chance of detection. Moreover, having to complete fewer steps generally means a less complicated synthesis process. The edges from reaction node to chemical node were assigned a weight based on the sum of the chemical values of the chemicals required for the reaction. The values were assigned an exponentially increasing weight based on the risk of using that type of chemical. Household chemicals are the least risky and were assigned a value of e^1 . The next most risky chemicals were the commercially available chemicals, assigned a value of e^2 . Regulated chemicals were assigned a value of e^3 . Non-commercially available chemicals were assigned a value based on the sum of the chemical values that are required to make the non-commercially available chemical.

Exponentially increasing values were chosen because it discouraged the use of reg-

ulated chemicals more than a linearly increasing function would, while still placing the value of commercially available chemicals closer in value to household chemicals. This designation was somewhat arbitrary and could have followed a functional curve of 2^x , or some other function and yielded similar results. Of course if the risk functions could be accurately determined by some other means, this would provide the most accurate results. The proposed weighting, however, does offer an analyst a usable proxy method of assigning weights that encourages pathways from household chemicals and highly discourages the use of pathways from regulated chemicals.

Figure 18 shows an example weighted graph. Node 8 to Node C-14 provides a good example of weight assignment. The edge from 8 to C-14 receives a weight of $e^2 + e^1$ based on the sum of the value from a commercial chemical, C-15, and a noncommercial chemical, C-16. The non-commercial chemical C-16 has a value of e^1 because the chemical used to make it is a household chemical, C-17.

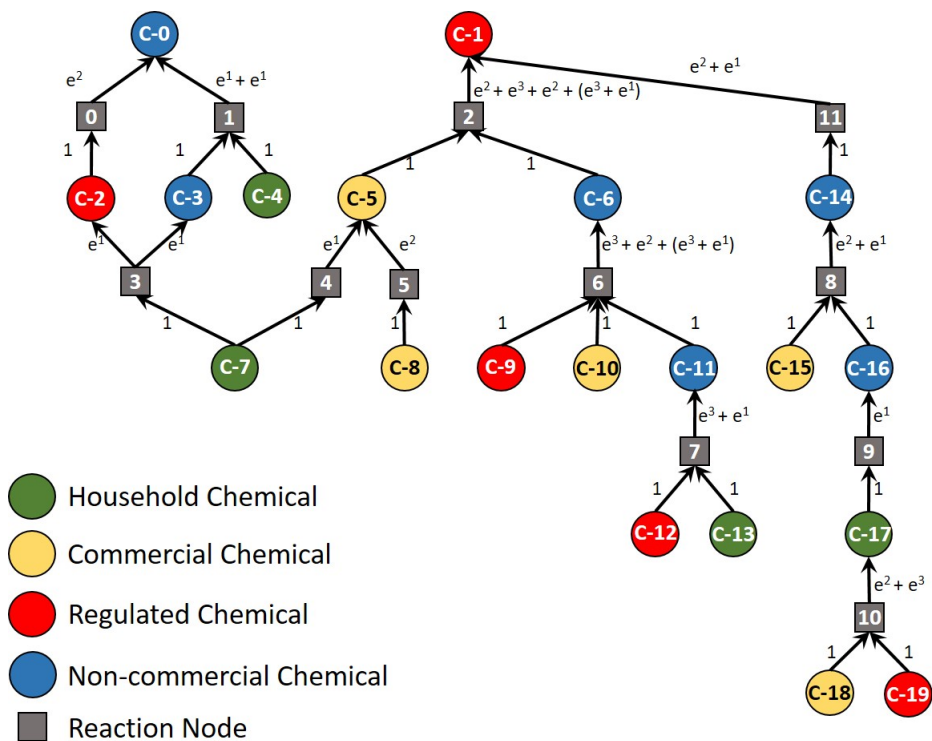


Figure 16. Example Representative Network

One assumption used in the weighting should be noted. The network of chemistry is compiled from known reactions. The assumption is made that a chemist would generally publish the synthesis from commercially available chemicals to possible non-commercially available chemicals and not the other way around. This assumption is necessary in assigning the non-commercial chemical values. The leaf nodes of the tree were not allowed to be assigned as non-commercial because it would then be impossible to assign a value based on the chemicals that formed it because no chemicals reacted to produce it. The coding of the graph generators could have been greatly simplified if a weight of $e^{1.5}$ were assigned to non-commercial chemicals. This is done for the testing of larger networks to save computational time in generating very large graphs as discussed in the following section.

4.3 Graph Testing

Testing was conducted to determine how well the generated graphs matched the known statistics of the network of chemistry. All testing was conducted on a computer with two Intel Xeon X5670 CPUs which operate @ 2.93 GHz and have six cores each. The system also had 96.0 GB of RAM.

Testing showed that the generated trees using the Networkx `gn_graph` generator typically did not follow a powerlaw distribution. The average alpha value for a sample of 100 different generated graphs of 8,000 nodes had an average alpha value of 3.14482. Using a loglikelihood ratio test, the power-law fit was tested against an exponential fit and a lognormal fit. The average R and p values obtained were $R = 62.351$ and $p = 0.011682$ when tested against the exponential fit and $R = -0.55724$ and $p = 0.249193$ when tested for a lognormal fit. This shows that the generated graphs were much more likely to follow a powerlaw distribution than an exponential distribution but were slightly more likely to follow a lognormal distribution than a

powerlaw distribution.

The network of chemistry is not a tree but is much more interconnected. However, tree structures were generated and examined in order to test the algorithms if a database other than the Chematica database were used. Using the Reaxys database would result in a tree structure because it works backwards from the target molecule. Algorithms that work backwards from a target chemical are also used by Chematica as discussed by Szymkuc *et al.* [54] as a means to speed up searches of the network of chemistry. Even the complete network of chemistry could be laid out as a tree by trimming self loops and creating sub-trees from the trimmed edges.

Bipartite network generators were also explored as an alternative to generate a representative network. A Petri network is a bipartite network. The network of chemistry is a Petri net with one node set consisting of the chemicals and the other node set made up of the reactions. The Networkx bipartite graph generators were investigated to generate starting graphs in the following manner but often led to unrepresentative results like can be seen in Figure 17 wherein 267 chemicals are involved in a single reaction. The green nodes, are the chemical nodes and the red node is the reaction node.

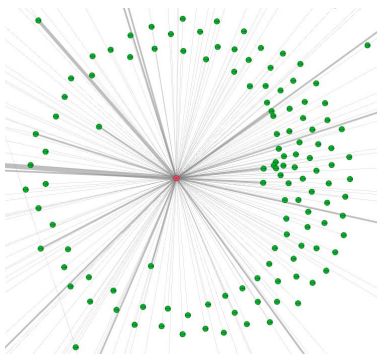


Figure 17. Bipartite Graph with Numerous Chemical in One Reaction

None of the bipartite graph generators claimed to generate a graph with a powerlaw distribution. Most of the bipartite generators required a degree sequence and

would generate edges from one set to the other based on the provided degree sequences. A random powerlaw sequence was desired to maintain the powerlaw degree distribution of the network of chemistry. Using two powerlaw sequences would generate large amounts of chemicals in one reaction, as in Figure 17. The generators also required that the sum of both sequences be equal. This made matching the powerlaw random number sequence with another sequence of another distribution that would provide a more realistic number of chemical reactants difficult.

Using a while loop to generate a powerlaw sequence and the same modified normal distribution discussed previously (mean of 2.5 and standard deviation of 1.8, rounded to integer values and all values generated ≤ 0 set equal to one) to generate a modified normal sequence. The loop continued until the sums of the distribution sequences were within two. Two was then subtracted from the maximum value in the powerlaw distribution or from a randomly chosen value above two from the normal distribution, whichever was larger. This provided better results when used to generate a bipartite graph than using two power law sequences. The base graph, when generated in that manner, had more realistic numbers of chemicals reacting but did not produce a good powerlaw fit. The average values obtained in testing 50 1,000-node graphs were:

Table 5. Bipartite Graph Powerlaw Fit

α	R (lognormal)	p (lognormal)	R (exponential)	p (exponential)
4.396	-0.4255	0.3667	149.9	0.0087

The graphs created are also undirected. An algorithm to assign the edges as directed one way or the other would have to be run. Because of the poor powerlaw match and the additional requirement of developing a node direction algorithm, the bipartite graph generators were set aside.

The scale-free directed graph generator provided to best powerlaw fit but these graphs were very resource intensive to generate. The powerlaw tree graph of 8,000

nodes was generated in an average of 15.789 seconds. The scale free directed graph generator averaged 130.637 seconds. When the growing network graphs were used as starting graphs, both the GN and GNR variations, the complete weighted bipartite, networks averaged seven minutes to generate. The scale-free graph had an advantage of taking less time total to generate the full graph, averaging only 5.67 minutes.

The reduced time for the scale-free starting graph was due to the method used to break loops of non-commercially available chemical nodes having non-commercially available predecessors that in-turn looped back to the original non-commercially available chemical. When this was the case the non-commercially available chemicals could not be assigned a weight because their weight was dependent on the weight of the predecessors. The scale-free algorithm tended to produce a significant number of such loops or loops of only non-commercially available chemicals. When such loops existed the first predecessor was reassigned to be a commercially available chemical. This made the weighting of the non-commercially available significantly less computational resource intensive. The full Python code is presented in Appendix A for the graph generators and testing code.

When scaled beyond 8,000 nodes, graph generation became very slow. An attempt was made to generate a representative chemical network from the growing network powerlaw tree graph of 100,000 nodes. After 72 hours the code was stopped to explore other options. The Barabási-Albert preferential attachment graph generator was decided upon to perform timing tests for 10,000,000 graphs. A Barabási-Albert graph of 10,000,000 nodes was generated using a preference of attaching 2 nodes from each new node. The average time to return a list of all paths to a target node was 279.074 seconds (4.65 minutes). On average, the Frequent Pattern Tree algorithm took 12.501 seconds on the same graphs. Attempts to generate paths and distances to all paths and not just a single target, maxed out the 96.0 GB of RAM and resulted

in memory errors.

The purpose of time testing was to gain an idea of how long these algorithms may take and if such methods may be feasible to use on the actual network of chemistry. The generated Barabási-Albert graph may not be the best representation of the network of chemistry because the graphs are undirected and is not bipartite when generated using the *barabasi_albert_graph* Networkx generator. Nevertheless, the Barabási-Albert generator was chosen because it is able to quickly generate a 10 million node, scale-free graph, with an average degree of two, to test on a large scale. The Barabási-Albert graph also has the characteristic of every node being reachable from every other node and provides a good idea of how long it takes an all paths algorithm to search and find the shortest path to every other node. This is a larger requirement than would be required for the network of chemistry where every node is not reachable for every other node.

4.4 Conclusion

Different methods of generating representative chemical networks were discussed, showing that 10,000 node networks matching the known network characteristics of the Chematica chemical network database are easily generated. Edge weights of the network were assigned according to the desirability of the use of a chemical in clandestine synthesis plans.

Furthermore, it was determined that for a 10,000,000 node network, finding all paths from every node to every other node is very resource intensive, but concentrating on one target node simplifies the resource requirements. On average, within five minutes a shortest path was determined between all ten million nodes and a single target node. The frequent pattern search of the sets of all ten million pathways averaged twelve and a half seconds. These result show that using these techniques is

feasible and that results are acquired in an efficient computational time. In Chapter 5, the results from the all paths and frequent pattern tree searches will be further explored.

One area of future research might be the generation of representative chemical networks beyond 10,000 nodes. Modifying the code for the Barabási-Albert Networkx algorithms to assign weights and direction as edges are assigned could be a simple means of quickly generating a scale-free representative network of chemistry on a large scale. The bipartite Networkx algorithms also hold promise for simple modifications leading to better representative results.

V. Methodologies for Targeting Chemical Synthesis Networks

5.1 Chapter Overview

The analysis of chemical networks in regards to the interdiction of clandestine synthesis of chemical warfare agents is explored in this chapter. In particular, the use of Dijkstra’s algorithm to identify all shortest paths that exist in a network database to synthesize regulated chemicals. Further, a means to identify the critical reactions and chemicals in all pathways of production is presented.

5.2 Network Analysis

The small network found in Figure 18 in Chapter 4 is first used to illustrate how the all shortest paths algorithm and the Frequent pattern tree set mining method can be used to draw insights into the chemicals that are most important over all synthesis process. The figure is presented again in this chapter for a reader’s convenience.

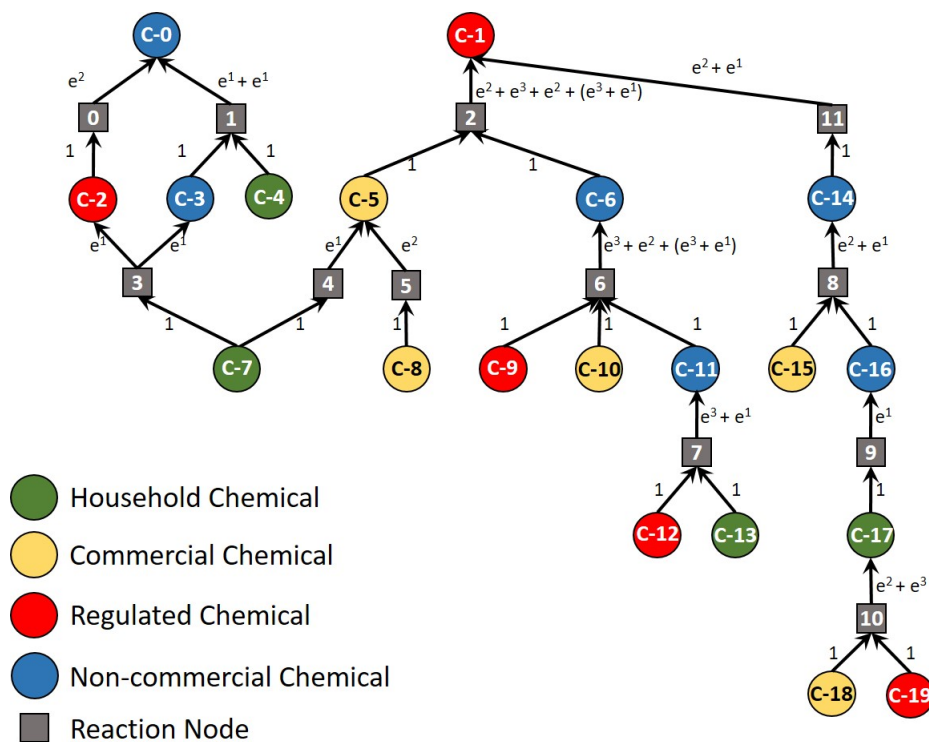


Figure 18. Example Representative Network

Dijkstra's Algorithm is used first to create a list of all paths to the target chemical. In Figure 18 the target chemical is C-1. The list of all shortest paths to the target chemical is used to determine the shortest path from a household chemical based on the weights developed in Chapter 4. This chemical is C-17 and the path length is 25.933. This implies that C-17 is the household chemical that is the most valuable starting point for the synthesis of the target regulated molecule without being interdicted. This is easily observed by examination of the graph as well. The path from C-7 appears promising on initial examination but has a much higher cost due to the additional chemicals required to form Chemical C-6. The only other chemical required to produce C-1 from C-17 is Chemical C-15, a commercially available chemical.

The shortest overall path is $C-14 \rightarrow 11 \rightarrow C-1$, with a length of 11.107. A list of the shortest path from every household chemical is also easily obtained by considering

only the paths from household chemicals, in this case there are production pathways from the household chemicals C-7, C-13, and C-17. Chemical C-13 however, does require a regulated chemical with which to react (reaction 7).

Each path to the target chemical from every reaction node is considered to be a set of chemicals and reactions. The paths from each reaction are used to avoid multiple counts caused by the number of chemicals reacting. If the sets consisted of the paths from each chemical, Node 6 would be counted as part of five different paths. This would lead to Node 6 being considered more important than it actually is. It is only involved in two different pathways of production, and really only one because it is only involved in the path from Node 7 because the non-commercial chemical, C-11, must be made from Chemicals C-12 and C-13.

The Frequent Pattern Tree algorithm applied with a set threshold of two occurrences yields the following results. The most frequent set is the single node [C-1] occurring nine times. It is easily noted that there are nine different paths from the reaction nodes to the target chemical. The next most frequent item is Node 2 with a frequency of five, the paths from the nodes preceding it and from itself to C-1. This would suggest that if Reaction 2 could be stopped this would disrupt the most production paths.

The information from the shortest paths and the frequent patterns work together to provide valuable insight. The information from the frequent pattern search indicates that Reaction 2 is a desirable target for interdiction, regulating or tracking Chemical C-5 may also be desirable (C-6 cannot be easily tracked because it is not commercially available). It should not be forgotten however, that the shortest path from a household chemical is not through Reaction 2 but through Reaction 11. Reaction 11 is the next most frequently occurring node with a count of 4. Caution should be used when applying insight from the information obtained, it is really a combina-

tion of the two algorithms that leads to more meaningful conclusions. All paths from household chemicals should be considered as well as the most frequent nodes because the combination of the two sets of information provides a more complete picture as shown in the preceding example.

Applying set mining to all shortest paths from household chemicals also helps to provide insight at the risk of overemphasizing some paths and nodes further up the tree. The frequent sets that occur in the network in Figure 18 are:

Set	Count
C-1	3
2	2
C-1, 2	2

A network of 1,000 nodes was chosen to further illustrate the insight that can be drawn from using an all-paths search and Frequent Pattern tree algorithm. For this particular problem the Networkx directed scale-free graph was chosen as the base graph. The the full graph, after labels and weights are assigned is depicted in Figure 19.

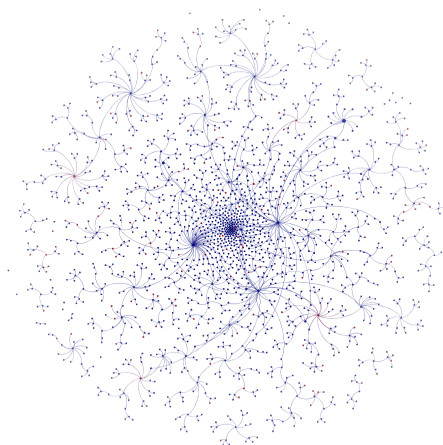


Figure 19. 1000 Node Network

Even at only 1,000 nodes the graph starts to become hard to visually interpret

and draw insight from. Figure 20 is a close up of the area around Chemical R-17, the red center-most node. This was the regulated chemical that was randomly chosen to represent the target molecule.

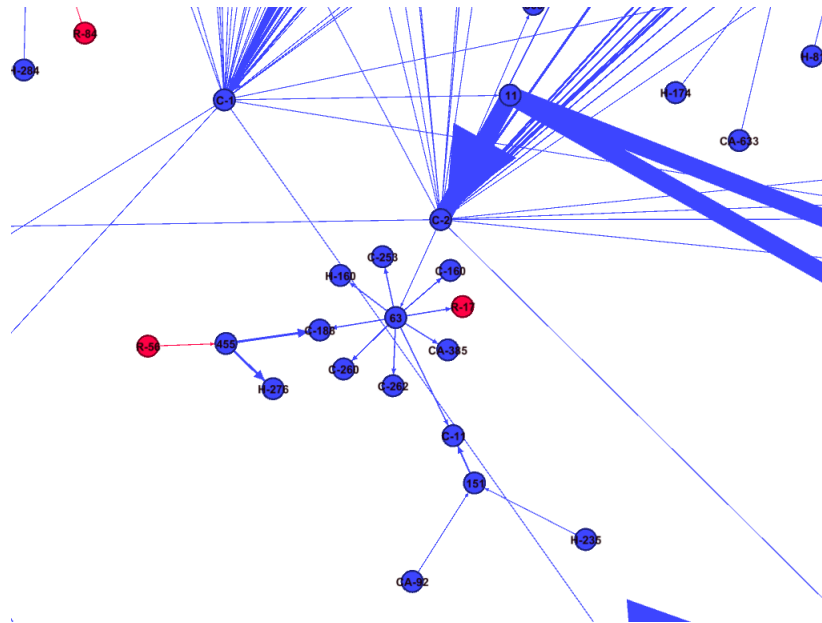


Figure 20. Target Chemical Area

Table 6 is a selection of the top patterns obtained from the Frequent Pattern tree search. Also listed is an example of some insight that can be drawn from the data.

Table 6. Frequent-Pattern Tree Results

Set	Count	Selected Insight
R-17	56	There are 56 different ways to make the target chemical
C-2	56	All 56 ways involve the non-commercially available chemical C-2
63	56	Reaction 56 is the most common reaction to produce the target chemical(in this case the only reaction)
C-2, 63, R-17	56	C-2 reacts to form the target chemical via reaction 63
11	34	The next most common reaction involved is reaction 11
C-1	34	The next most common chemical involved is C-1; involved in 34 paths
C-1, C-2	34	The two most common chemicals used to produce the target chemical

The counts from the frequent pattern tree results are somewhat like the criticality

index showing the number of times the chemicals or reactions are on the critical path, or how often they are used on all paths to make a target chemical. There were no household chemical that appeared frequently enough along the different synthesis pathways to show up in the frequent pattern search results. This does not indicate that no pathways exist from household chemicals, only that there is no household chemical that is involved in more than three pathways to the target molecule. The threshold criteria was set to three for this search. Searching the all paths results reveals that there are 12 different synthesis routes from household chemicals. These results are listed in Table 7. It can be noted that the two most frequent chemicals from the set search are involved in every path from a household chemical, no other intermediate chemical occurs in more than one path.

Table 7. Household Chemical Sources

Source	Pathway
H-173	[H-173, 909, C-1, 11, C-2, 63, R-17]
H-19	[H-19, 757, C-2, 63, R-17]
H-195	[H-195, 16, C-1, 11, C-2, 63, R-17]
H-2	[H-2, 880, C-2, 63, R-17]
H-226	[H-226, 772, C-1, 11, C-2, 63, R-17]
H-23	[H-23, 981, C-1, 11, C-2, 63, R-17]
H-230	[H-230, 880, C-2, 63, R-17]
H-257	[H-257, 27, C-1, 11, C-2, 63, R-17]
H-267	[H-267, 981, C-1, 11, C-2, 63, R-17]
H-52	[H-52, 444, C-1, 11, C-2, 63, R-17]
H-8	[H-8, 366, C-1, 11, C-2, 63, R-17]
H-81	[H-81, 18, C-2, 63, R-17]

In the example above it can be seen that Chemical C-2 is involved in every path and Chemical C-1 is also involved in a majority of the synthesis pathways. Interdiction efforts should be targeted at Chemicals C-1 and C-2 and their reaction pathways. If an interdictor could stop the production of those chemicals there would be no known way to produce the regulated substance. If the chemicals are not commercially available as is the case with this example, reactions could be the primary interdiction target.

Reactions can be tracked using the sales of special equipment used for those reactions. If the reaction required a unique piece of equipment the regulation and tracking of that equipment should be the focus of interdiction efforts.

The Networkx package provides a Floyd-Warshall algorithm and an all paths determination using Dijkstra’s algorithm. Networkx *single_source_dijkstra_path* and *single_source_dijkstra_path_length* implementations of Dijkstra’s algorithm were used. The original implementation of Dijkstra’s algorithm has a running time of $O(n^2)$ where Floyd-Warshall has a running time of $O(n^3)$ [2]. The Networkx Dijkstra’s all paths functions were used for path determination because the computation savings were significant. On a scale free network of 1,000 nodes, over 30 iterations, the *single_source_dijkstra_path* and *single_source_dijkstra_path_length* combined took an average of 1.533 seconds to run. The Floyd-Warshall function averaged 263.697 seconds (4.39 minutes). For a very large network there are significant computational cost savings using Dijkstra’s algorithm versus the Networkx Floyd-Warshall function.

5.3 Conclusion

The paths from one chemical to all other reachable chemicals can be acquired in a very efficient manner and frequent pattern mining was not found to be resource intensive on the tested networks. The Chematica software uses breath first searches to find synthesis pathways. Szymkuc *et al.* [54] mention the infeasibility of using the breath first searches for the optimal pathway for complex targets and mention depth first searches and Dijkstra’s algorithm being prone to infeasibility as well. It is unknown if the various implementations of Dijkstra’s algorithm had been tested, in particular, the single source all paths implementation used in the Python Networkx package. Testing on the actual network should be conducted but there is good indication using a ten million node graph that the algorithms are computationally

efficient. In the absence of the real-world network, generators were used to simulate the networks using the information available about the actual network characteristics. The information obtained proves to be valuable for gaining insight into chemical and reaction targets for interdiction planning.

The literature on the Chematica network of chemistry does not indicate that listings for household chemicals are incorporated into the network. Using the household products database would provide an updated list of these chemicals that could be assigned as attributes like the hazardous chemical information already incorporated. Weights would also have to be assigned as well, according to desirability of using common and readily available household chemicals.

In the Chematica network filtering criteria for commercial availability is incorporated as well as a cost for the commercially available chemicals. The specifics of how this is done is not available but the fact that screening criteria for optimal searches exists indicates that other criteria should be able to be added. Fuller *et al.* [30] indicate that alternative weighting methods are possible as well.

Using these techniques to find shortest paths and identify frequent occurrences on the network of chemistry would allow for a list to be compiled of the most common chemicals and reactions that could be used in the production of chemical warfare agents. Every path from unregulated and household chemicals would be known as well. The combination of this information provides a means to readily identify and track possible clandestine production of these agents.

VI. Conclusion

6.1 Overview

The clandestine manufacture of chemical warfare agents remains a real possibility. Known routes of production exist for these dangerous chemicals from common, unregulated, household chemicals. Identifying these routes and determining methods of intelligently targeting and detecting synthetic pathways could save many lives if a group or individual decided to release a chemical warfare agent on an unsuspecting populace.

A method of using a network mixed integer linear program to identify the most cost effective means of synthesis was discussed. This model incorporated a strategy to account for the expected value of avoiding detection. This model is detail oriented and accounts for nuances in chemical synthesis that the larger network of chemistry would not be able to include. These details, such as accounting for different solvents and drying agents, require an experienced chemist to accurately and efficiently compile but the information is easily compiled through publications available online.

Building a network from online resources and journal searches in this manner provides an inexpensive means of constructing a small network that has the additional benefit of being comprised of the methods that are likely to be known to potential terrorists intent on making a chemical weapon. Removing the expected value objective and applying exponential weights as was done with the synthesis of network graphs would remove the emphasis on cost and simply focus on weights based on the ease of acquiring the materials. As this network grows large, decomposition could be used to solve the linear program.

Using a large database provides a means of quickly accounting for known methods of production and speedily identifying new synthetic routes. Access to the large

commercial databases required to build a comprehensive database of all published reactions is however, expensive. A small Petri net can inexpensively be built from online sources and journal articles but loses the scope available by incorporating possible unknown pathways.

Efficient algorithms exist to identify all shortest paths from a single target node to all other reachable nodes. Within minutes, in a graph of 10 million nodes, the shortest path from every node to a target node can be identified. Less than a minute later, the most frequent nodes that occur along all those paths can be readily identified. This information, when gathered from a comprehensive network database, can then provide an intelligent starting point for interdiction efforts.

6.2 Study Objectives

The primary objective of this study was to provide a method of identifying critical synthetic pathways and chemicals that may be tracked or regulated to disrupt and prevent the clandestine production of chemical warfare agents. Identifying the key pathways and chemicals provides a focus area of the most likely avenues a terrorist may take in chemical agent synthesis. One of the reasons that methamphetamine has continued to persist is that it can be synthesized completely from common household chemicals. It is unrealistic to ban these household chemicals that are essential to normal day-to-day life. The ability to monitor the sales of household chemicals provides a means of preempting production when common sets of chemicals are known and can trigger an alert.

Fuller *et al.* [30] wrote of working with the U.S. Government on implementing such a scheme but indicated that algorithms to help narrow down searches to minimal sets of chemicals were of interest. Unfortunately, the approaches of narrowing down searches to minimal sets, if any, taken by the authors are not identified in their

paper, so it is not known whether schemes such as the Frequent Pattern Tree or other set mining algorithms have been applied to date. The authors also used what was called an inverse sums weighting system to weight edges according to availability and synthetic complexity using in-degree centrality. The idea of this weighting is that the nodes with the greatest in-degree centrality value are the most desirable because they are the most commonly used in reactions and therefore the least likely to draw suspicion.

It was more desirable for this study to use a ranking scheme based on the means of acquiring the chemical. The most desirable chemicals being common chemicals that can be picked up at a store like Walmart or Home Depot. Basing the weighting on in-degree centrality seemed to overemphasize chemicals that are studied often. This could mean that they are the most popular and common chemicals because they are commercially valuable and available. It could also mean that the chemical was studied and published often because of a reason that would make using it a negative for clandestine production, for example if it was studied often but is now unavailable because of environmental concerns or if it was studied and published often because of its value as a warfare agent.

This study sought to expand upon this previous work and provide ideas to improve upon it. Incorporating a database of common household chemicals and a weighting scheme with an emphasis on those pathways would help to identify the optimal paths from household chemicals to warfare agents. A means of identifying the most valuable sets of chemicals was also identified, providing a way of identifying minimal sets of the most commonly used molecules along all synthesis routes.

6.3 Conclusion

Chemical warfare agents are crude, imprecise instruments capable of harming many innocent men, women and children. The fear and resentment of chemical agents is not from the fact that they kill Soldiers. There are millions of different ways Soldiers are harmed and disfigured without the use of chemicals that do not receive the same outrage. The use of chemicals has never been decisive to victory, they are too prone to environmental factors, and too imprecise. It is when weapons loose precision and fall upon those who should not be harmed that fear and anger rise. The real fear of chemical agents is the harm and disfigurement they bring not to Soldiers, but to innocents.

The same can be said of terrorism in general: it is not the threat it brings to those trained in combat, it is the pain and violence it brings to the uncommitted civilians who have a right to safety. This research has provided a model for gaining insights into how to help prevent a dual chemical and terrorist threat. It is a step forward in aiding in the analysis of potential chemical warfare agent threats.

Finding all shortest paths in a network database provides a linked list of pathways to chemical agents, for both known and previously unknown paths. Further analyzing these paths to identify frequent patterns and sets provides a list of the most common chemicals and sets of chemicals in these paths. This knowledge can then be used to focus interdiction efforts. The fact that previously unknown pathways are identified is key. Reacting to how things are known to be done may stop them from being done that way again in the future, but there should be a means of staying ahead of future threats. The ability to identify and monitor synthesis pathways before they can be used provides a way to stay ahead of the enemy and protect those that terrorism and chemical weapons should not be allowed to touch.

6.4 Recommendations for Future Research

Access to a database with a chemical network is critical to further investigations, whether it is the Chematica network or if the network was built using access to the Reaxys or SciFinder databases. Building a network from the Reaxys or SciFinder databases may be preferential because of the significantly larger number of chemicals and reactions. SciFinder is a commercial product that provides access to 80.5 million reactions updated daily with 150,000 single and multi-step reactions added per week [55]. The Sci-finder or Reaxys databases contain many more reactions than the Chematica database and would provide extensive resources for building a new larger network with hazardous chemical interdiction as the focus.

Representative networks and graphs with similar qualities are useful for preliminary investigations but intricacies of the actual network may make such schemes unworkable. The Chematica network is the only reaction database found in this research that organizes reactions and chemicals into a network structure. This structure allows the interconnection of a reaction database to be explored as a whole, in a way that is not piecemeal.

One additional path of research involves incorporating network algorithm interdiction schemes. Khachiyan *et al.* [38], in 2006, proposed an extension to Dijkstra’s algorithm that incorporates node-wise arc interdiction . Even if access to the large chemical databases was not obtained, building an interdiction algorithm and testing the model on the representative graphs would be an interesting exploration into identifying crucial nodes and node sets. This may provide a means of further narrowing down the results obtained from those obtained by frequent set mining.

Appendix A. GAMS and Python Code

1.1 GAMS Base Model Code

```
$Title ThesisBase

Set      i   Nodes
        /s
        ephedrine
        psuedoephedrine
        redphosphorus
        hydroiodicacid
        sodiumcarbonate
        diethylether
        hydrogenchloride-g
        hydrochloricacid
        magnesiumsulfate
        methamphetamineHCl
        sodiumbicarbonate
        sodiumhydroxide
        potassiumhydroxide
        acetone
        hexane
        calciumchloride
        calciumsulfate
        thionylchloride
        chloroform
        diethylether2
        methanol
        palladiumchloride
        hydrochloricacid2
        calciumhydride
        chloroform2
        phosphoruspentachloride
        diethylether3/

alias(i,j)

Set
    k Cost selection /k1,k2/

    sa(i,j,k) source arcs
    /s      .   ephedrine      .(k1,k2)
    s      .   psuedoephedrine .(k1,k2)/

    ta(i,j,k) terminal arcs
```

/magnesiumsulfate.	methamphetamineHCl	.k1
methamphetamineHCl.	methamphetamineHCl	.k1
calciumchloride.	methamphetamineHCl	.k1
calciumsulfate.	methamphetamineHCl	.k1
calciumhydride.	methamphetamineHCl	.k1/

ia(i) source and terminal node

/s
methamphetamineHCl/

xa(i,j,k) node cost arcs

/s	. ephedrine	.(k1,k2)
s	. psuedoephedrine	.(k1,k2)
ephedrine	. redphosphorus	.(k1,k2)
ephedrine	. thionylchloride	.(k1)
ephedrine	. chloroform2	.(k1)
psuedoephedrine	. redphosphorus	.(k1,k2)
psuedoephedrine	. thionylchloride	.(k1)
psuedoephedrine	. chloroform2	.(k1)
redphosphorus	. hydroiodicacid	.(k1)
hydroiodicacid	. sodiumcarbonate	.(k1,k2)
hydroiodicacid	. sodiumbicarbonate	.(k1,k2)
hydroiodicacid	. sodiumhydroxide	.(k1,k2)
hydroiodicacid	. potassiumhydroxide	.(k1,k2)
sodiumcarbonate	. diethylether	.(k1,k2)
sodiumcarbonate	. acetone	.(k1,k2)
sodiumcarbonate	. hexane	.(k1)
diethylether	. hydrogenchloride-g	.(k1)
diethylether	. hydrochloricacid	.(k1,k2)
hydrogenchloride-g	. magnesiumsulfate	.(k1,k2)
hydrogenchloride-g	. calciumchloride	.(k1,k2)
hydrogenchloride-g	. calciumsulfate	.(k1,k2)
hydrochloricacid	. magnesiumsulfate	.(k1,k2)
hydrochloricacid	. calciumchloride	.(k1,k2)
hydrochloricacid	. calciumsulfate	.(k1,k2)
magnesiumsulfate	. methamphetamineHCl	.(k1)
methamphetamineHCl	. methamphetamineHCl	.(k1)
sodiumbicarbonate	. sodiumcarbonate	.(k1)
sodiumhydroxide	. diethylether	.(k1,k2)
sodiumhydroxide	. acetone	.(k1,k2)
sodiumhydroxide	. hexane	.(k1)
potassiumhydroxide	. diethylether	.(k1,k2)
potassiumhydroxide	. acetone	.(k1,k2)

potassiumhydroxide	. hexane	.(k1)
acetone	. hydrogenchloride-g	.(k1)
acetone	. hydrochloricacid	.(k1,k2)
hexane	. hydrogenchloride-g	.(k1)
hexane	. hydrochloricacid	.(k1,k2)
calciumchloride	. methamphetamineHCl	.(k1)
calciumsulfate	. methamphetamineHCl	.(k1)
thionylchloride	. chloroform	.(k1)
chloroform	. diethylether2	.(k1,k2)
diethylether2	. methanol	.(k1,k2)
methanol	. palladiumchloride	.(k1)
palladiumchloride	. hydrochloricacid2	.(k1,k2)
hydrochloricacid2	. calciumhydride	.(k1)
calciumhydride	. methamphetamineHCl	.(k1)
chloroform2	. phosphoruspentachloride	.(k1)
phosphoruspentachloride	. diethylether3	.(k1,k2)
diethylether3	. methanol	.(k1,k2)/;

Table c(i,j,k) node cost data

		k1	k2
s	. ephedrine	13.1234	2.12249
s	. psuedoephedrine	6.43467	3.87323
ephedrine	. redphosphorus	5.26762	0.036826
ephedrine	. thionylchloride	0.23116	
ephedrine	. chloroform2	0.13949	
psuedoephedrine	. redphosphorus	5.26762	0.036826
psuedoephedrine	. thionylchloride	0.23116	
psuedoephedrine	. chloroform2	0.13949	
redphosphorus	. hydroiodicacid	5.54706	
hydroiodicacid	. sodiumcarbonate	0.00953	0.441341
hydroiodicacid	. sodiumbicarbonate	0.01381	0.388361
hydroiodicacid	. sodiumhydroxide	0.02745	0.173631
hydroiodicacid	. potassiumhydroxide	0.01060	0.031654
sodiumcarbonate	. diethylether	0.40348	4.11
sodiumcarbonate	. acetone	0.09851	1.62567
sodiumcarbonate	. hexane	2.6961	
diethylether	. hydrogenchloride-g	0.33679	
diethylether	. hydrochloricacid	0.00086	0.014489
hydrogenchloride-g	. magnesiumsulfate	0.00297	0.118788
hydrogenchloride-g	. calciumchloride	0.00092	0.094529
hydrogenchloride-g	. calciumsulfate	0.01167	0.306890
hydrochloricacid	. magnesiumsulfate	0.00297	0.118788
hydrochloricacid	. calciumchloride	0.00092	0.094529
hydrochloricacid	. calciumsulfate	0.01167	0.306890
magnesiumsulfate	. methamphetamineHCl	0	
methamphetamineHCl	. methamphetamineHCl	0	

sodiumbicarbonate. sodiumcarbonate	0	
sodiumhydroxide. diethylether	0.40348	4.11
sodiumhydroxide. acetone	0.09851	1.62567
sodiumhydroxide. hexane	2.6961	
potassiumhydroxide. diethylether	0.40348	4.11
potassiumhydroxide. acetone	0.09851	1.62567
potassiumhydroxide. hexane	2.6961	
acetone. hydrogenchloride-g	0.33679	
acetone. hydrochloricacid	0.00086	0.014489
hexane. hydrogenchloride-g	0.33679	
hexane. hydrochloricacid	0.00086	0.014489
calciumchloride. methamphetamineHCl	0	
calciumsulfate. methamphetamineHCl	0	
thionylchloride. chloroform	0.13949	
chloroform. diethylether2	0.40348	4.11
diethylether2. methanol	0.13823	1.07525
methanol. palladiumchloride	0.89647	
palladiumchloride. hydrochloricacid2	0.01710	0.288156
hydrochloricacid2. calciumhydride	1.14027	
calciumhydride. methamphetamineHCl	0	
chloroform2. phosphoruspentachloride	0.91560	
phosphoruspentachloride. diethylether3	0.14959	1.523783
diethylether3. methanol	0.13823	1.07525;

Variable z total cost;

Binary Variables x(i,j,k) connection of node i to node j

Equations obj objective function
 source one unit of flow from source
 intermediate(i) flow in = flow out for intermediate nodes
 terminal one unit of flow into terminal node
 choosecost(i,j) only allow one cost value;

```
obj..
sum((i,j,k), x(i,j,k)$xa(i,j,k)*c(i,j,k)) =e= z;
source..
sum((i,j,k), x(i,j,k)$sa(i,j,k)) =e= 1;
intermediate(i)$ (not(ia(i)))..
sum((j,k), x(i,j,k)$xa(i,j,k))-sum((j,k),x(j,i,k)$xa(j,i,k))=e=0;
terminal..
sum((i,j,k), x(i,j,k)$ta(i,j,k)) =e= 1;
choosecost(i,j)..
sum(k, x(i,j,k)$xa(i,j,k)) =l= 1;
```

Options

```

limrow = 0,
limcol = 0,
solprint = off;

```

```

Model ThesisBase /All/;
Solve ThesisBase using MIP minimizing z;
Display x.l, z.l;

```

1.2 GAMS Loop Model Code

```

$title ThesisLoop
*Used to solve the cost only objective function with each arc forced into
*and out of the solution

```

```

Set      i  Nodes
         /s
         ephedrine
         psuedoephedrine
         redphosphorus
         hydroiodicacid
         sodiumcarbonate
         diethylether
         hydrogenchloride-g
         hydrochloricacid
         magnesiumsulfate
         methamphetamineHCl
         sodiumbicarbonate
         sodiumhydroxide
         potassiumhydroxide
         acetone
         hexane
         calciumchloride
         calciumsulfate
         thionylchloride
         chloroform
         diethylether2
         methanol
         palladiumchloride
         hydrochloricacid2
         calciumhydride
         chloroform2
         phosphoruspentachloride
         diethylether3/

```

```

alias(i,j,a,b)

```

```

Set

```

k Cost selection /k1,k2/

xa(i,j,k) node cost arcs

/s	. ephedrine	.(k1 ,k2)
s	. psuedoephedrine	.(k1 ,k2)
ephedrine	. redphosphorus	.(k1 ,k2)
ephedrine	. thionylchloride	.(k1)
ephedrine	. chloroform2	.(k1)
psuedoephedrine	. redphosphorus	.(k1 ,k2)
psuedoephedrine	. thionylchloride	.(k1)
psuedoephedrine	. chloroform2	.(k1)
redphosphorus	. hydroiodicacid	.(k1)
hydroiodicacid	. sodiumcarbonate	.(k1 ,k2)
hydroiodicacid	. sodiumbicarbonate	.(k1 ,k2)
hydroiodicacid	. sodiumhydroxide	.(k1 ,k2)
hydroiodicacid	. potassiumhydroxide	.(k1 ,k2)
sodiumcarbonate	. diethylether	.(k1 ,k2)
sodiumcarbonate	. acetone	.(k1 ,k2)
sodiumcarbonate	. hexane	.(k1)
diethylether	. hydrogenchloride-g	.(k1)
diethylether	. hydrochloricacid	.(k1 ,k2)
hydrogenchloride-g	. magnesiumsulfate	.(k1 ,k2)
hydrogenchloride-g	. calciumchloride	.(k1 ,k2)
hydrogenchloride-g	. calciumsulfate	.(k1 ,k2)
hydrochloricacid	. magnesiumsulfate	.(k1 ,k2)
hydrochloricacid	. calciumchloride	.(k1 ,k2)
hydrochloricacid	. calciumsulfate	.(k1 ,k2)
magnesiumsulfate	. methamphetamineHCl	.(k1)
methamphetamineHCl	. methamphetamineHCl	.(k1)
sodiumbicarbonate	. sodiumcarbonate	.(k1)
sodiumhydroxide	. diethylether	.(k1 ,k2)
sodiumhydroxide	. acetone	.(k1 ,k2)
sodiumhydroxide	. hexane	.(k1)
potassiumhydroxide	. diethylether	.(k1 ,k2)
potassiumhydroxide	. acetone	.(k1 ,k2)
potassiumhydroxide	. hexane	.(k1)
acetone	. hydrogenchloride-g	.(k1)
acetone	. hydrochloricacid	.(k1 ,k2)
hexane	. hydrogenchloride-g	.(k1)
hexane	. hydrochloricacid	.(k1 ,k2)
calciumchloride	. methamphetamineHCl	.(k1)
calciumsulfate	. methamphetamineHCl	.(k1)
thionylchloride	. chloroform	.(k1)
chloroform	. diethylether2	.(k1 ,k2)
diethylether2	. methanol	.(k1 ,k2)

```

methanol . palladiumchloride .(k1)
palladiumchloride . hydrochloricacid2 .(k1,k2)
hydrochloricacid2 . calciumhydride .(k1)
calciumhydride . methamphetamineHCl .(k1)
chloroform2 . phosphoruspentachloride .(k1)
phosphoruspentachloride . diethylether3 .(k1,k2)
diethylether3 . methanol .(k1,k2)/

```

```

sa(i,j,k) source arcs
/s . ephedrine .(k1,k2)
s . psuedoephedrine .(k1,k2)/

```

```

ta(i,j,k) terminal arcs

```

```

/magnesiumsulfate . methamphetamineHCl .k1
methamphetamineHCl . methamphetamineHCl .k1
calciumchloride . methamphetamineHCl .k1
calciumsulfate . methamphetamineHCl .k1
calciumhydride . methamphetamineHCl .k1/

```

```

st(i) source and terminal node

```

```

/s
methamphetamineHCl/;

```

Table c(i,j,k) node cost data

		k1	k2
s	. ephedrine	13.1234	2.12249
s	. psuedoephedrine	6.43467	2.18841
ephedrine	. redphosphorus	5.26762	0.036826
ephedrine	. thionylchloride	0.231164732	
ephedrine	. chloroform2	0.139491997	
psuedoephedrine	. redphosphorus	5.26762	0.036826
psuedoephedrine	. thionylchloride	0.231164732	
psuedoephedrine	. chloroform2	0.139491997	
redphosphorus	. hydroiodicacid	5.54706	
hydroiodicacid	. sodiumcarbonate	0.009534	0.441341
hydroiodicacid	. sodiumbicarbonate	0.013813	0.388361
hydroiodicacid	. sodiumhydroxide	0.027458	0.173631
hydroiodicacid	. potassiumhydroxide	0.010609	0.031654
sodiumcarbonate	. diethylether	0.403486	4.11
sodiumcarbonate	. acetone	0.098514	1.62567
sodiumcarbonate	. hexane	2.6961	
diethylether	. hydrogenchloride-g	0.336792	
diethylether	. hydrochloricacid	0.000860321	0.0144893

hydrogenchloride-g. magnesiumsulfate	0.002979	0.118788
hydrogenchloride-g. calciumchloride	0.000929917	0.09452937
hydrogenchloride-g. calciumsulfate	0.011678226	0.306890
hydrochloricacid. magnesiumsulfate	0.002979	0.118788
hydrochloricacid. calciumchloride	0.000929917	0.09452937
hydrochloricacid. calciumsulfate	0.011678226	0.306890
magnesiumsulfate. methamphetamineHCl	0	
methamphetamineHCl. methamphetamineHCl	0	
sodiumbicarbonate. sodiumcarbonate	0	
sodiumhydroxide . diethylether	0.403486	4.11
sodiumhydroxide . acetone	0.098514	1.62567
sodiumhydroxide . hexane	2.6961	
potassiumhydroxide. diethylether	0.403486	4.11
potassiumhydroxide. acetone	0.098514	1.62567
potassiumhydroxide. hexane	2.6961	
acetone . hydrogenchloride-g	0.336792	
acetone . hydrochloricacid	0.000860321	0.0144893
hexane . hydrogenchloride-g	0.336792	
hexane . hydrochloricacid	0.000860321	0.0144893
calciumchloride . methamphetamineHCl	0	
calciumsulfate . methamphetamineHCl	0	
thionylchloride . chloroform	0.139491997	
chloroform . diethylether2	0.403486	4.11
diethylether2 . methanol	0.138236225	1.07525021
methanol . palladiumchloride	0.89647426	
palladiumchloride. hydrochloricacid2	0.017109642	0.2881564
hydrochloricacid2. calciumhydride	1.140279666	
calciumhydride . methamphetamineHCl	0	
chloroform2 . phosphoruspentachloride	0.915604976	
phosphoruspentachloride. diethylether3	0.149592376	1.5237837
diethylether3 . methanol	0.138236225	1.075250;

Set n /n1/;

alias(n,m)

alias(k,l)

Table y(i,j,k,n) node cost data

		k1.n1	k2.n1
s	. ephedrine	1	1
s	. psuedoephedrine	1	1
ephedrine	. redphosphorus	1	1
ephedrine	. thionylchloride	1	
ephedrine	. chloroform2	1	
psuedoephedrine	. redphosphorus	1	1
psuedoephedrine	. thionylchloride	1	

psuedoephedrine	. chloroform2	1	
redphosphorus	. hydroiodicacid	1	
hydroiodicacid	. sodiumcarbonate	1	1
hydroiodicacid	. sodiumbicarbonate	1	1
hydroiodicacid	. sodiumhydroxide	1	1
hydroiodicacid	. potassiumhydroxide	1	1
sodiumcarbonate	. diethylether	1	1
sodiumcarbonate	. acetone	1	1
sodiumcarbonate	. hexane	1	
diethylether	. hydrogenchloride-g	1	
diethylether	. hydrochloricacid	1	1
hydrogenchloride-g	. magnesiumsulfate	1	1
hydrogenchloride-g	. calciumchloride	1	1
hydrogenchloride-g	. calciumsulfate	1	1
hydrochloricacid	. magnesiumsulfate	1	1
hydrochloricacid	. calciumchloride	1	1
hydrochloricacid	. calciumsulfate	1	1
magnesiumsulfate	. methamphetamineHCl	1	
methamphetamineHCl	. methamphetamineHCl	1	
sodiumbicarbonate	. sodiumcarbonate	1	
sodiumhydroxide	. diethylether	1	1
sodiumhydroxide	. acetone	1	1
sodiumhydroxide	. hexane	1	
potassiumhydroxide	. diethylether	1	1
potassiumhydroxide	. acetone	1	1
potassiumhydroxide	. hexane	1	
acetone	. hydrogenchloride-g	1	
acetone	. hydrochloricacid	1	1
hexane	. hydrogenchloride-g	1	
hexane	. hydrochloricacid	1	1
calciumchloride	. methamphetamineHCl	1	
calciumsulfate	. methamphetamineHCl	1	
thionylchloride	. chloroform	1	
chloroform	. diethylether2	1	1
diethylether2	. methanol	1	1
methanol	. palladiumchloride	1	
palladiumchloride	. hydrochloricacid2	1	1
hydrochloricacid2	. calciumhydride	1	
calciumhydride	. methamphetamineHCl	1	
chloroform2	. phosphoruspentachloride	1	
phosphoruspentachloride	. diethylether3	1	1
diethylether3	. methanol	1	1;

Variable z total cost;

Binary Variables x(i,j,k) connection of node i to node j

Equations

obj	objective function
source	one unit of flow from source
intermediate(i)	flow in = flow out for intermediate nodes
terminal	one unit of flow into terminal node
choosecost(i,j)	only allow one cost value
include(i,j,k,n)	include or exclude arc
include2(i,j,k,n)	include or exclude arc;

```

obj..
sum((i,j,k), x(i,j,k)$xa(i,j,k)*c(i,j,k)) =e= z;
source..
sum((i,j,k), x(i,j,k)$sa(i,j,k)) =e= 1;
intermediate(i)$ (not(st(i)))..
sum((j,k), x(i,j,k)$xa(i,j,k))-sum((j,k),x(j,i,k)$xa(j,i,k))=e=0;
terminal..
sum((i,j,k), x(i,j,k)$ta(i,j,k)) =e= 1;
choosecost(i,j)..
sum(k, x(i,j,k)$xa(i,j,k)) =l= 1;
include(i,j,k,n)$ (xa(i,j,k))..
x(i,j,k)$xa(i,j,k) - y(i,j,k,n) =l= 1;
include2(i,j,k,n)$ (xa(i,j,k))..
x(i,j,k)$xa(i,j,k) + y(i,j,k,n) =g= 1;

*used to force arcs out of solution
*include(i,j,k,n)$ (xa(i,j,k))..
*x(i,j,k)$xa(i,j,k) - y(i,j,k,n) =l= 0;
*include2(i,j,k,n)$ (xa(i,j,k))..
*x(i,j,k)$xa(i,j,k) + y(i,j,k,n) =g= 0;

```

Model ThesisLoop /All/;

Parameter report (*,*,*,*,*);

Option report:5:3:1;

ThesisLoop.solprint=%Solprint.Quiet%;

loop((a,b,l,m)\$xa(a,b,l),

```

    y(a,b,l,m)$xa(a,b,l) = 0;
    Solve ThesisLoop using MIP minimizing z;
    y(a,b,l,m)$xa(a,b,l) = 1;
    report('z',a,b,l,m) = z.l;

```

);

Display report;

1.3 GAMS Full Model Code

\$Title ThesisFull

*Full Model with values determined by (worst z) minus
*(z with arc forced in base model)

Set i Nodes

/s, ephedrine, psuedoephedrine, redphosphorus, hydroiodicacid,
sodiumcarbonate, diethylether, hydrogenchloride-g,
hydrochloricacid, magnesiumsulfate, methamphetamineHCl,
sodiumbicarbonate, sodiumhydroxide, potassiumhydroxide,
acetone, hexane, calciumchloride, calciumsulfate,
thionylchloride, chloroform, diethylether2, methanol,
palladiumchloride, hydrochloricacid2, calciumhydride,
chloroform2, phosphoruspentachloride, diethylether3/

alias(i,j,a,b)

Set k Cost selection /k1,k2/

v /v1/

p /p1/

xa(i,j,k) node cost arcs

/s	. ephedrine	.(k1,k2)
s	. psuedoephedrine	.(k1,k2)
ephedrine	. redphosphorus	.(k1,k2)
ephedrine	. thionylchloride	.(k1)
ephedrine	. chloroform2	.(k1)
psuedoephedrine	. redphosphorus	.(k1,k2)
psuedoephedrine	. thionylchloride	.(k1)
psuedoephedrine	. chloroform2	.(k1)
redphosphorus	. hydroiodicacid	.(k1)
hydroiodicacid	. sodiumcarbonate	.(k1,k2)
hydroiodicacid	. sodiumbicarbonate	.(k1,k2)
hydroiodicacid	. sodiumhydroxide	.(k1,k2)
hydroiodicacid	. potassiumhydroxide	.(k1,k2)
sodiumcarbonate	. diethylether	.(k1,k2)
sodiumcarbonate	. acetone	.(k1,k2)
sodiumcarbonate	. hexane	.(k1)
diethylether	. hydrogenchloride-g	.(k1)

diethylether	.	hydrochloricacid	.(k1,k2)
hydrogenchloride-g	.	magnesiumsulfate	.(k1,k2)
hydrogenchloride-g	.	calciumchloride	.(k1,k2)
hydrogenchloride-g	.	calciumsulfate	.(k1,k2)
hydrochloricacid	.	magnesiumsulfate	.(k1,k2)
hydrochloricacid	.	calciumchloride	.(k1,k2)
hydrochloricacid	.	calciumsulfate	.(k1,k2)
magnesiumsulfate	.	methamphetamineHCl	.(k1)
methamphetamineHCl	.	methamphetamineHCl	.(k1)
sodiumbicarbonate	.	sodiumcarbonate	.(k1)
sodiumhydroxide	.	diethylether	.(k1,k2)
sodiumhydroxide	.	acetone	.(k1,k2)
sodiumhydroxide	.	hexane	.(k1)
potassiumhydroxide	.	diethylether	.(k1,k2)
potassiumhydroxide	.	acetone	.(k1,k2)
potassiumhydroxide	.	hexane	.(k1)
acetone	.	hydrogenchloride-g	.(k1)
acetone	.	hydrochloricacid	.(k1,k2)
hexane	.	hydrogenchloride-g	.(k1)
hexane	.	hydrochloricacid	.(k1,k2)
calciumchloride	.	methamphetamineHCl	.(k1)
calciumsulfate	.	methamphetamineHCl	.(k1)
thionylchloride	.	chloroform	.(k1)
chloroform	.	diethylether2	.(k1,k2)
diethylether2	.	methanol	.(k1,k2)
methanol	.	palladiumchloride	.(k1)
palladiumchloride	.	hydrochloricacid2	.(k1,k2)
hydrochloricacid2	.	calciumhydride	.(k1)
calciumhydride	.	methamphetamineHCl	.(k1)
chloroform2	.	phosphoruspentachloride	.(k1)
phosphoruspentachloride	.	diethylether3	.(k1,k2)
diethylether3	.	methanol	.(k1,k2)/

```

sa(i,j,k) source arcs
/s      . ephedrine      .(k1,k2)
s       . psuedoephedrine .(k1,k2)/

```

```
ta(i,j,k) terminal arcs
```

/magnesiumsulfate.	methamphetamineHCl	.k1
methamphetamineHCl.	methamphetamineHCl	.k1
calciumchloride.	methamphetamineHCl	.k1
calciumsulfate.	methamphetamineHCl	.k1
calciumhydride.	methamphetamineHCl	.k1/

```
st(i) source and terminal node
```

/s
methamphetamineHCl/;

Table c(i,j,k) node cost data

		k1	k2
s	. ephedrine	13.1234	2.12249
s	. psuedoephedrine	6.43467	2.18841
ephedrine	. redphosphorus	5.26762	0.036826
ephedrine	. thionylchloride	0.231164732	
ephedrine	. chloroform2	0.139491997	
psuedoephedrine	. redphosphorus	5.26762	0.036826
psuedoephedrine	. thionylchloride	0.231164732	
psuedoephedrine	. chloroform2	0.139491997	
redphosphorus	. hydroiodicacid	5.54706	
hydroiodicacid	. sodiumcarbonate	0.009534	0.441341
hydroiodicacid	. sodiumbicarbonate	0.013813	0.388361
hydroiodicacid	. sodiumhydroxide	0.027458	0.173631
hydroiodicacid	. potassiumhydroxide	0.010609	0.031654
sodiumcarbonate	. diethylether	0.403486	4.11
sodiumcarbonate	. acetone	0.098514	1.62567
sodiumcarbonate	. hexane	2.6961	
diethylether	. hydrogenchloride-g	0.336792	
diethylether	. hydrochloricacid	0.000860321	0.014489323
hydrogenchloride-g.	magnesiumsulfate	0.002979	0.118788
hydrogenchloride-g.	calciumchloride	0.000929917	0.09452937
hydrogenchloride-g.	calciumsulfate	0.011678226	0.306890
hydrochloricacid.	magnesiumsulfate	0.002979	0.118788
hydrochloricacid.	calciumchloride	0.000929917	0.09452937
hydrochloricacid.	calciumsulfate	0.011678226	0.306890
magnesiumsulfate.	methamphetamineHCl	0	
methamphetamineHCl.	methamphetamineHCl	0	
sodiumbicarbonate.	sodiumcarbonate	0	
sodiumhydroxide	. diethylether	0.403486	4.11
sodiumhydroxide	. acetone	0.098514	1.62567
sodiumhydroxide	. hexane	2.6961	
potassiumhydroxide.	diethylether	0.403486	4.11
potassiumhydroxide.	acetone	0.098514	1.62567
potassiumhydroxide.	hexane	2.6961	
acetone	. hydrogenchloride-g	0.336792	
acetone	. hydrochloricacid	0.000860321	0.014489323
hexane	. hydrogenchloride-g	0.336792	
hexane	. hydrochloricacid	0.000860321	0.014489323
calciumchloride	. methamphetamineHCl	0	
calciumsulfate	. methamphetamineHCl	0	

thionylchloride	. chloroform	0.139491997	
chloroform	. diethylether2	0.403486	4.11
diethylether2	. methanol	0.138236225	1.07525021
methanol	. palladiumchloride	0.89647426	
palladiumchloride	. hydrochloricacid2	0.017109642	0.288156413
hydrochloricacid2	. calciumhydride	1.140279666	
calciumhydride	. methamphetamineHCl	0	
chloroform2	. phosphoruspentachloride	0.915604976	
phosphoruspentachloride	. diethylether3	0.149592376	1.52378379
diethylether3	. methanol	0.138236225	1.07525021;

Table val(i,j,k,v) node value data

			v1
s	.ephedrine	.k1	0
s	.ephedrine	.k2	11.00091
s	.psuedoephedrine	.k1	6.68873
s	.psuedoephedrine	.k2	10.93499
ephedrine	.redphosphorus	.k1	3.04263
ephedrine	.redphosphorus	.k2	8.27343
ephedrine	.thionylchloride	.k1	11.00091
ephedrine	.chloroform2	.k1	10.57036
psuedoephedrine	.redphosphorus	.k1	2.97671
psuedoephedrine	.redphosphorus	.k2	8.20751
psuedoephedrine	.thionylchloride	.k1	10.93499
psuedoephedrine	.chloroform2	.k1	10.50444
redphosphorus	.hydroiodicacid	.k1	8.27343
hydroiodicacid	.sodiumcarbonate	.k1	8.27343
hydroiodicacid	.sodiumcarbonate	.k2	7.84162
hydroiodicacid	.sodiumbicarbonate	.k1	8.26915
hydroiodicacid	.sodiumbicarbonate	.k2	7.8946
hydroiodicacid	.sodiumhydroxide	.k1	8.2555
hydroiodicacid	.sodiumhydroxide	.k2	8.10933
hydroiodicacid	.potassiumhydroxide	.k1	8.27235
hydroiodicacid	.potassiumhydroxide	.k2	8.25131
sodiumcarbonate	.diethylether	.k1	7.96845
sodiumcarbonate	.diethylether	.k2	4.26194
sodiumcarbonate	.acetone	.k1	8.27343
sodiumcarbonate	.acetone	.k2	6.74627
sodiumcarbonate	.hexane	.k1	5.67584
diethylether	.hydrogenchloride-g	.k1	7.63252
diethylether	.hydrochloricacid	.k1	7.96845
diethylether	.hydrochloricacid	.k2	7.95482
hydrogenchloride-g	.magnesiumsulfate	.k1	7.93544
hydrogenchloride-g	.magnesiumsulfate	.k2	7.81964
hydrogenchloride-g	.calciumchloride	.k1	7.93749

hydrogenchloride-g	.calciumchloride	.k2	7.84389
hydrogenchloride-g	.calciumsulfate	.k1	7.92675
hydrogenchloride-g	.calciumsulfate	.k2	7.63153
hydrochloricacid	.magnesiumsulfate	.k1	8.27138
hydrochloricacid	.magnesiumsulfate	.k2	8.15557
hydrochloricacid	.calciumchloride	.k1	8.27343
hydrochloricacid	.calciumchloride	.k2	8.17983
hydrochloricacid	.calciumsulfate	.k1	8.26268
hydrochloricacid	.calciumsulfate	.k2	7.96747
magnesiumsulfate	.methamphetamineHCl	.k1	8.27138
methamphetamineHCl	.methamphetamineHCl	.k1	8.27138
sodiumbicarbonate	.sodiumcarbonate	.k1	8.26915
sodiumhydroxide	.diethylether	.k1	7.95053
sodiumhydroxide	.diethylether	.k2	4.24402
sodiumhydroxide	.acetone	.k1	8.2555
sodiumhydroxide	.acetone	.k2	6.72835
sodiumhydroxide	.hexane	.k1	5.65792
potassiumhydroxide	.diethylether	.k1	7.96738
potassiumhydroxide	.diethylether	.k2	4.26086
potassiumhydroxide	.acetone	.k1	8.27235
potassiumhydroxide	.acetone	.k2	6.74519
potassiumhydroxide	.hexane	.k1	5.67476
acetone	.hydrogenchloride-g	.k1	7.93749
acetone	.hydrochloricacid	.k1	8.27343
acetone	.hydrochloricacid	.k2	8.2598
hexane	.hydrogenchloride-g	.k1	5.33991
hexane	.hydrochloricacid	.k1	5.67584
hexane	.hydrochloricacid	.k2	5.66221
calciumchloride	.methamphetamineHCl	.k1	8.27343
calciumsulfate	.methamphetamineHCl	.k1	8.26268
thionylchloride	.chloroform	.k1	11.00091
chloroform	.diethylether2	.k1	11.00091
chloroform	.diethylether2	.k2	7.29439
diethylether2	.methanol	.k1	11.00091
diethylether2	.methanol	.k2	10.06389
methanol	.palladiumchloride	.k1	11.00091
palladiumchloride	.hydrochloricacid2	.k1	11.00091
palladiumchloride	.hydrochloricacid2	.k2	10.72986
hydrochloricacid2	.calciumhydride	.k1	11.00091
calciumhydride	.methamphetamineHCl	.k1	11.00091
chloroform2	.phosphoruspentachloride	.k1	10.57036
phosphoruspentachloride	.diethylether3	.k1	10.57036
phosphoruspentachloride	.diethylether3	.k2	9.19617
diethylether3	.methanol	.k1	10.57036
diethylether3	.methanol	.k2	9.63335 ;

Table prob(i,j,k,p)

			p1
s	.ephedrine	.k1	0.3
s	.ephedrine	.k2	0.1
s	.psuedoephedrine	.k1	0.3
s	.psuedoephedrine	.k2	0.1
ephedrine	.redphosphorus	.k1	0.95
ephedrine	.redphosphorus	.k2	0.1
ephedrine	.thionylchloride	.k1	0.1
ephedrine	.chloroform2	.k1	0.8
psuedoephedrine	.redphosphorus	.k1	0.95
psuedoephedrine	.redphosphorus	.k2	0.1
psuedoephedrine	.thionylchloride	.k1	0.1
psuedoephedrine	.chloroform2	.k1	0.8
redphosphorus	.hydroiodicacid	.k1	0.1
redphosphorus	.hydroiodicacid	.k2	0
hydroiodicacid	.sodiumcarbonate	.k1	0.95
hydroiodicacid	.sodiumcarbonate	.k2	0.8
hydroiodicacid	.sodiumbicarbonate	.k1	0.95
hydroiodicacid	.sodiumbicarbonate	.k2	0.8
hydroiodicacid	.sodiumhydroxide	.k1	0.95
hydroiodicacid	.sodiumhydroxide	.k2	0.8
hydroiodicacid	.potassiumhydroxide	.k1	0.95
hydroiodicacid	.potassiumhydroxide	.k2	0.8
sodiumcarbonate	.diethylether	.k1	0.95
sodiumcarbonate	.diethylether	.k2	0.8
sodiumcarbonate	.acetone	.k1	0.95
sodiumcarbonate	.acetone	.k2	0.8
sodiumcarbonate	.hexane	.k1	0.8
diethylether	.hydrogenchloride-g	.k1	0.1
diethylether	.hydrochloricacid	.k1	0.95
diethylether	.hydrochloricacid	.k2	0.8
hydrogenchloride-g	.magnesiumsulfate	.k1	0.95
hydrogenchloride-g	.magnesiumsulfate	.k2	0.8
hydrogenchloride-g	.calciumchloride	.k1	0.95
hydrogenchloride-g	.calciumchloride	.k2	0.8
hydrogenchloride-g	.calciumsulfate	.k1	0.95
hydrogenchloride-g	.calciumsulfate	.k2	0.8
hydrochloricacid	.magnesiumsulfate	.k1	0.95
hydrochloricacid	.magnesiumsulfate	.k2	0.8
hydrochloricacid	.calciumchloride	.k1	0.95
hydrochloricacid	.calciumchloride	.k2	0.8
hydrochloricacid	.calciumsulfate	.k1	0.95
hydrochloricacid	.calciumsulfate	.k2	0.8
magnesiumsulfate	.methamphetamineHCl	.k1	0

methamphetamineHCl	.methamphetamineHCl	.k1	0
sodiumbicarbonate	.sodiumcarbonate	.k1	0.95
sodiumbicarbonate	.sodiumcarbonate	.k2	0.8
sodiumhydroxide	.diethylether	.k1	0.95
sodiumhydroxide	.diethylether	.k2	0.8
sodiumhydroxide	.acetone	.k1	0.95
sodiumhydroxide	.acetone	.k2	0.8
sodiumhydroxide	.hexane	.k1	0.8
potassiumhydroxide	.diethylether	.k1	0.95
potassiumhydroxide	.diethylether	.k2	0.8
potassiumhydroxide	.acetone	.k1	0.95
potassiumhydroxide	.acetone	.k2	0.8
potassiumhydroxide	.hexane	.k1	0.8
acetone	.hydrogenchloride-g	.k1	0.1
acetone	.hydrochloricacid	.k1	0.95
acetone	.hydrochloricacid	.k2	0.8
hexane	.hydrogenchloride-g	.k1	0.1
hexane	.hydrochloricacid	.k1	0.95
hexane	.hydrochloricacid	.k2	0.8
calciumchloride	.methamphetamineHCl	.k1	0
calciumsulfate	.methamphetamineHCl	.k1	0
thionylchloride	.chloroform	.k1	0.8
chloroform	.diethylether2	.k1	0.95
chloroform	.diethylether2	.k2	0.8
diethylether2	.methanol	.k1	0.95
diethylether2	.methanol	.k2	0.8
methanol	.palladiumchloride	.k1	0.8
palladiumchloride	.hydrochloricacid2	.k1	0.95
palladiumchloride	.hydrochloricacid2	.k2	0.8
hydrochloricacid2	.calciumhydride	.k1	0.8
calciumhydride	.methamphetamineHCl	.k1	0
chloroform2	.phosphoruspentachloride	.k1	0.8
phosphoruspentachloride	.diethylether3	.k1	0.95
phosphoruspentachloride	.diethylether3	.k2	0.8
diethylether3	.methanol	.k1	0.95
diethylether3	.methanol	.k2	0.8

Set $n / n1 /;$

alias (n,m)

alias (k,l)

Variable z total cost;

Binary Variables $x(i,j,k)$ connection of node i to node j

Equations

obj objective function
 source one unit of flow from source
 intermediate(i) flow in = flow out for intermediate nodes
 terminal one unit of flow into terminal node
 choosecost(i,j) only allow one cost value;

```

obj..
  sum((i,j,k), x(i,j,k)$xa(i,j,k)*c(i,j,k)) -
  sum((i,j,k,p,v), x(i,j,k)$xa(i,j,k)*prob(i,j,k,p)*val(i,j,k,v))=e= z;
source..
  sum((i,j,k), x(i,j,k)$sa(i,j,k)) =e= 1;
intermediate(i)$ (not(st(i)))..
  sum((j,k), x(i,j,k)$xa(i,j,k))-sum((j,k), x(j,i,k)$xa(j,i,k)) =e= 0;
terminal..
  sum((i,j,k), x(i,j,k)$ta(i,j,k)) =e= 1;
choosecost(i,j)..
  sum(k, x(i,j,k)$xa(i,j,k)) =l= 1;
  
```

Options

```
optcr = 0;
```

```
Model ThesisFull /All/;
```

```
ThesisFull.solprint=%Solprint.Quiet%;
```

```
Solve ThesisFull using MIP minimizing z;
```

```
Display x.l, z.l;
```

1.4 GAMS Probability Analysis

Below is the additional code used to solve for separate probabilities in a loop. The other parameters are the same as in the full model above.

Sets high(i,j,k,p) set of nodes that are high p of not getting caught

/hydroiodicacid	.sodiumcarbonate	.k1	.p1
hydroiodicacid	.sodiumbicarbonate	.k1	.p1
hydroiodicacid	.sodiumhydroxide	.k1	.p1
hydroiodicacid	.potassiumhydroxide	.k1	.p1
sodiumcarbonate	.diethylether	.k1	.p1
sodiumcarbonate	.acetone	.k1	.p1
diethylether	.hydrochloricacid	.k1	.p1
hydrogenchloride-g	.magnesiumsulfate	.k1	.p1

hydrogenchloride-g	.calciumchloride	.k1	.p1
hydrogenchloride-g	.calciumsulfate	.k1	.p1
hydrochloricacid	.magnesiumsulfate	.k1	.p1
hydrochloricacid	.calciumchloride	.k1	.p1
hydrochloricacid	.calciumsulfate	.k1	.p1
sodiumbicarbonate	.sodiumcarbonate	.k1	.p1
sodiumhydroxide	.diethylether	.k1	.p1
sodiumhydroxide	.acetone	.k1	.p1
potassiumhydroxide	.diethylether	.k1	.p1
potassiumhydroxide	.acetone	.k1	.p1
acetone	.hydrochloricacid	.k1	.p1
hexane	.hydrochloricacid	.k1	.p1
chloroform	.diethylether2	.k1	.p1
diethylether2	.methanol	.k1	.p1
palladiumchloride	.hydrochloricacid2	.k1	.p1
phosphoruspentachloride	.diethylether3	.k1	.p1
diethylether3	.methanol	.k1	.p1

/

medHigh(i,j,k,p) set of nodes that are medium-high p of
not getting caught

/ephedrine	.chloroform2	.k1	.p1
psuedoephedrine	.chloroform2	.k1	.p1
hydroiodicacid	.sodiumcarbonate	.k2	.p1
hydroiodicacid	.sodiumbicarbonate	.k2	.p1
hydroiodicacid	.sodiumhydroxide	.k2	.p1
hydroiodicacid	.potassiumhydroxide	.k2	.p1
sodiumcarbonate	.diethylether	.k2	.p1
sodiumcarbonate	.acetone	.k2	.p1
sodiumcarbonate	.hexane	.k1	.p1
diethylether	.hydrochloricacid	.k2	.p1
hydrogenchloride-g	.magnesiumsulfate	.k2	.p1
hydrogenchloride-g	.calciumchloride	.k2	.p1
hydrogenchloride-g	.calciumsulfate	.k2	.p1
hydrochloricacid	.magnesiumsulfate	.k2	.p1
hydrochloricacid	.calciumchloride	.k2	.p1
hydrochloricacid	.calciumsulfate	.k2	.p1
sodiumhydroxide	.diethylether	.k2	.p1
sodiumhydroxide	.acetone	.k2	.p1
sodiumhydroxide	.hexane	.k1	.p1
potassiumhydroxide	.diethylether	.k2	.p1
potassiumhydroxide	.acetone	.k2	.p1
potassiumhydroxide	.hexane	.k1	.p1
acetone	.hydrochloricacid	.k2	.p1
hexane	.hydrochloricacid	.k2	.p1
thionylchloride	.chloroform	.k1	.p1

```

chloroform          .diethylether2      .k2 .p1
diethylether2       .methanol          .k2 .p1
methanol            .palladiumchloride   .k1 .p1
palladiumchloride   .hydrochloricacid2 .k2 .p1
hydrochloricacid2   .calciumhydride      .k1 .p1
chloroform2         .phosphoruspentachloride.k1 .p1
phosphoruspentachloride .diethylether3      .k2 .p1
diethylether3       .methanol          .k2 .p1 /

```

medLow(i,j,k,p) set of nodes that are medium-low p of
not getting caught

```

/s          .ephedrine      .k1 .p1
s           .psuedoephedrine .k1 .p1
ephedrine   .redphosphorus  .k1 .p1
psuedoephedrine .redphosphorus .k1 .p1 /

```

low(i,j,k,p) set of nodes that are low p of not getting caught

```

/s          .ephedrine      .k2 .p1
s           .psuedoephedrine .k2 .p1
ephedrine   .redphosphorus  .k2 .p1
ephedrine   .thionylchloride .k1 .p1
psuedoephedrine .redphosphorus .k2 .p1
psuedoephedrine .thionylchloride .k1 .p1
redphosphorus .hydroiodicacid .k1 .p1
diethylether .hydrogenchloride-g .k1 .p1
acetone      .hydrogenchloride-g .k1 .p1
hexane       .hydrogenchloride-g .k1 .p1 /

```

Variable z total cost;

Binary Variables x(i,j,k) connection of node i to node j

Equations obj objective function
 source one unit of flow from source
 intermediate(i) flow in = flow out for intermediate nodes
 terminal one unit of flow into terminal node
 choosecost(i,j) only allow one cost value;

```

obj..
sum((i,j,k),x(i,j,k)$xa(i,j,k)*c(i,j,k))-
sum((i,j,k,p,v),x(i,j,k)$xa(i,j,k)*prob(i,j,k,p)*val(i,j,k,v))=e= z;
source..
sum((i,j,k), x(i,j,k)$sa(i,j,k)) =e= 1;
intermediate(i)$ (not(st(i)))..

```

```

sum((j,k), x(i,j,k)$xa(i,j,k))-sum((j,k), x(j,i,k)$xa(j,i,k)) =e= 0;
terminal..
sum((i,j,k), x(i,j,k)$ta(i,j,k)) =e= 1;
choosecost(i,j)..
sum(k, x(i,j,k)$xa(i,j,k)) =l= 1;

Model ThesisProb /All/;

ThesisProb.solprint=%Solprint.Quiet%;

Parameter reportHigh(*,*,*,*,*)
           reportLow(*,*,*,*,*)
           reportMedHigh(*,*,*,*,*)
           reportMedLow(*,*,*,*,*);

Set y /High, Med-High, Med-Low, Low/
    t /px1*px21/;

Parameter probabilities(t);
probabilities("px1") = 0;

Option reportHigh:5:4:1
       reportLow:5:4:1
       reportMedHigh:5:4:1
       reportMedLow:5:4:1
       prob:2:3:1;

Option solveopt=replace, optcr=0;
Option clear=z, clear=x;

loop(t,
    loop((i,j,k,p),
        if(prob(i,j,k,p)$low(i,j,k,p) = 0.1,
            prob(i,j,k,p)$low(i,j,k,p)=0.05*probabilities(t);
        );
    );

    Solve ThesisProb using MIP minimizing z;
    reportLow('x',i,j,k,t) = x.l(i,j,k);
    reportLow('z','-', '-', '-', '-',t) = z.l;

    loop((i,j,k,p)$low(i,j,k,p),
        if( prob(i,j,k,p)$low(i,j,k,p) = 0.05*probabilities(t),
            prob(i,j,k,p)$low(i,j,k,p) = 0.1;
        );

```

```

);

probabilities(t+1) = probabilities(t)+1;
Option clear=z, clear=x;

);

Option clear = probabilities;
probabilities("px1") = 0;

loop(t,

    loop((i,j,k,p)$medLow(i,j,k,p),
        if( prob(i,j,k,p)$medLow(i,j,k,p) = 0.3,
            prob(i,j,k,p)$medLow(i,j,k,p)=
                0.05*probabilities(t);
        );
    );

    Solve ThesisProb using MIP minimizing z;
    reportMedLow('x',i,j,k,t) = x.l(i,j,k);
    reportMedLow('z','-', '-', '-', '-',t) = z.l;

    loop((i,j,k,p)$medLow(i,j,k,p),
        if( prob(i,j,k,p)$medLow(i,j,k,p)=
            0.05*probabilities(t),
            prob(i,j,k,p)$medLow(i,j,k,p) = 0.3;
        );
    );

    probabilities(t+1) = probabilities(t)+1;
    Option clear=z, clear=x;

);

Option clear = probabilities;
probabilities("px1") = 0;

loop(t,

    loop((i,j,k,p)$medHigh(i,j,k,p),
        if( prob(i,j,k,p)$medHigh(i,j,k,p) = 0.8,
            prob(i,j,k,p)$medHigh(i,j,k,p)=
                0.05*probabilities(t);
        );
    );

```

```

Solve ThesisProb using MIP minimizing z;
reportMedHigh('x',i,j,k,t) = x.l(i,j,k);
reportMedHigh('z','-', '-', '-', '-',t) = z.l;

loop((i,j,k,p)$medHigh(i,j,k,p),
      if( prob(i,j,k,p)$medHigh(i,j,k,p)=
          0.05*probabilities(t),
          prob(i,j,k,p)$medHigh(i,j,k,p) = 0.8;
      );
);

probabilities(t+1) = probabilities(t)+1;
Option clear=z, clear=x;

);

Option clear = probabilities;
probabilities("px1") = 0;

loop(t,

      loop((i,j,k,p)$high(i,j,k,p),
            if( prob(i,j,k,p)$high(i,j,k,p) = 0.95,
                prob(i,j,k,p)$low(i,j,k,p)=
                0.05*probabilities(t);
            );
      );

      Solve ThesisProb using MIP minimizing z;
      reportHigh('x',i,j,k,t) = x.l(i,j,k);
      reportHigh('z','-', '-', '-', '-',t) = z.l;

      loop((i,j,k,p)$high(i,j,k,p),
            if( prob(i,j,k,p)$high(i,j,k,p)=
                0.05*probabilities(t),
                prob(i,j,k,p)$high(i,j,k,p) = 0.95;
            );
      );

      probabilities(t+1) = probabilities(t)+1;
      Option clear=z, clear=x;

);

Display

```

```

reportHigh ,
reportMedHigh ,
reportMedLow ,
reportLow ;

```

1.5 GAMS Weighted Sums Pareto Code

Below is the portion of the GAMS code that was modified to use the weighted sums approach to find the Pareto optimal front.

```

Set n /n1/;
scalar w /1/;
alias(n,m)
alias(k,l)

Variable          z                      total cost;

Binary Variables x(i,j,k)                connection of node i to node j

Equations  obj                          objective function
           source                        one unit of flow from source
           intermediate(i)              flow in = flow out for intermediate nodes
           terminal                      one unit of flow into terminal node
           choosecost(i,j)              only allow one cost value;

obj..
w*sum((i,j,k), x(i,j,k)$xa(i,j,k)*c(i,j,k)) -
(1-w)*sum((i,j,k,p,v), x(i,j,k)$xa(i,j,k)*
prob(i,j,k,p)*val(i,j,k,v))=e= z;
source..
sum((i,j,k), x(i,j,k)$sa(i,j,k)) =e= 1;
intermediate(i)$ (not(st(i)))..
sum((j,k), x(i,j,k)$xa(i,j,k)) -
sum((j,k), x(j,i,k)$xa(j,i,k)) =e= 0;
terminal..
sum((i,j,k), x(i,j,k)$ta(i,j,k)) =e= 1;
choosecost(i,j)..
sum(k, x(i,j,k)$xa(i,j,k)) =l= 1;

Options
optcr = 0;

Model ThesisFullP /All/;

```

```

ThesisFullP.solprint=%Solprint.Quiet%;

Parameter report(*,*);

Set theataset /t-1*t-1000/;
Parameter thetapar(theataset)
           ob_values1(theataset)
           ob_values2(theataset);

scalar ob1 /0/
        ob2 /0/;
thetapar("t-1")=1

option ob_values1:8:0:1
       ob_values2:8:0:1
       report:8:0:1;

loop (theataset ,
      thetapar(theataset+1)=thetapar(theataset)+1;
      w = 0.001*thetapar(theataset);
      Solve ThesisFullP using MIP minimizing z;
      ob_values1(theataset) =
        sum((i,j,k), x.l(i,j,k)$xa(i,j,k)*c(i,j,k));
      ob_values2(theataset) =
        sum((i,j,k,p,v), x.l(i,j,k)$xa(i,j,k)*
          prob(i,j,k,p)*val(i,j,k,v));
      report('z',theataset) = z.l;
);

Display report , ob_values1 , ob_values2;

```

1.6 GAMS ε -Constraint Pareto Front Code

Below is the portion of the GAMS code that was modified to use the ε -constraint method of finding the Pareto optimal front.

Equations	obj	objective function
	obj2	expected value objective function
	source	one unit of flow from source
	intermediate(i)	flow in = flow out for intermediate nodes
	terminal	one unit of flow into terminal node
	choosecost(i,j)	only allow one cost value;


```

obj..
  sum((i,j,k), x(i,j,k)$xa(i,j,k)*c(i,j,k)) =e=z;
obj2..

```

```

sum((i,j,k,p,v), x(i,j,k)$xa(i,j,k)*prob(i,j,k,p)*val(i,j,k,v))=l= w;
source..
sum((i,j,k), x(i,j,k)$sa(i,j,k)) =e= 1;
intermediate(i)$ (not(st(i)))..
sum((j,k), x(i,j,k)$xa(i,j,k))-sum((j,k), x(j,i,k)$xa(j,i,k)) =e= 0;
terminal..
sum((i,j,k), x(i,j,k)$ta(i,j,k)) =e= 1;
choosecost(i,j)..
sum(k, x(i,j,k)$xa(i,j,k)) =l= 1;

Options
    optcr = 0;

Model ThesisFullP /All/;

ThesisFullP.solprint=%Solprint.Quiet%;

Parameter report(*,*);

Set theataset /t-1*t-420/;
Parameter thetapar(theataset)
    ob_values1(theataset)
    ob_values2(theataset);

scalar ob1 /0/
    ob2 /0/;
thetapar("t-1")=2450

option ob_values1:8:0:1
    ob_values2:8:0:1
    report:8:0:1;

loop (theataset,
    thetapar(theataset+1)=thetapar(theataset)+5;
    w = 0.01*thetapar(theataset);
    Solve ThesisFullP using MIP minimizing z;
    ob_values1(theataset) = sum((i,j,k), x.l(i,j,k)$xa(i,j,k)*c(i,j,k));
    ob_values2(theataset) = sum((i,j,k,p,v), x.l(i,j,k)$xa(i,j,k)*prob(i,j,
    report('z',theataset) = z.l;
);

Display report, ob_values1, ob_values2;

```

1.7 Python Chemical Network Tree Generation

```

import networkx as nx
import random as rand
import numpy as np
import time
from random import choice

start_time = time.time()

GN=nx.gn_graph(8000) #used for GN starting graph
#GN=nx.gnr_graph(8000,0.667) # un-comment for GNR starting graph

ndlist = list(GN.nodes())
z = 0 #used to track node numbers for labeling
rnd1 = [] #used to test modified random normal distribution

# convert every other node to c node if ... node 0 then its
# predecessors...then next node after 0 that was not a
# predecessor etc.
for x in ndlist:

    if GN.has_node(x):

        list1a = list(GN.predecessors(x))
        list1 = list(GN.predecessors(x))
        list2 = list(range(len(list1)))

        for y in list2:
            list1[y]='C-'+ str(z)
            z = z + 1

        dictGN = dict(zip(list1a , list1))
        GN = nx.relabel_nodes(GN, dictGN)

# function to check if v is an integer
def isInt(v):
    try: int(v)
    except: return False
    return True

# sort nodes into a list of chemical nodes and a list of reaction nodes
def sortNodeList(GN):

    ndlist = list(GN.nodes())
    codelist = []
    nodelist = []

```

```

# compile a list of the C nodes and non-C (reaction) nodes
for x in range(len(ndlist)):

    thestring = ndlist[x]
    if isInt(ndlist[x])==True:
        nodelist.append(ndlist[x])
        continue
    if thestring[0]=='C'or'H'or'R':
        codelist.append(ndlist[x])

return codelist , nodelist , ndlist

(codelist , nodelist , ndlist) = sortNodeList(GN)

# if node is a reaction node and has no C preds add norm random num of
# C nodes; node number beginning where z left off above
for n in GN.nodes():
    if n in codelist:
        continue
    list1 = list(GN.predecessors(n))
    if len(list1)==0:
        r = round(rand.normalvariate(2, .8))
        if r<=0:
            r = 1
        rnd1.append(r)
        for x in range(r):
            GN.add_node('C-' + str(z))
            GN.add_edge('C-' + str(z), n)
            z = z + 1

(codelist , nodelist , ndlist) = sortNodeList(GN)

# if node is a react node and has no C successors add norm random num
# of C nodes; node number beginning where z left off above
for n in GN.nodes():
    if n in codelist:
        continue
    list1 = list(GN.successors(n))
    if len(list1)==0:
        r = round(rand.normalvariate(1, 0.4))
        if r<=0:
            r = 1
        for x in range(r):
            GN.add_node('C-' + str(z))
            GN.add_edge(n, 'C-' + str(z))

```

```

        z = z + 1

(codelist, nodelist, ndlist) = sortNodeList(GN)

# A function to make a list of all nodes with no predecessors
def nopredlistmaker(GN):
    nopredlist=[]
    for x in GN.nodes():
        preds=GN.predecessors(x)
        if preds == []:
            nopredlist.append(x)
    return nopredlist

#Returns number of household, commercial, and regulated nodes
# according to graph size and desired ratio of commercial to
# noncommercial nodes
def get_h_r_c(GN, caRatio):
    h = min(round(len(codelist)*.2),1948)
    r = min(round(len(codelist)*.1),161)
    c = round((len(codelist)-h-r)*caRatio)
    return h, r, c

caRatio = .6
(h,r,c)=get_h_r_c(GN, caRatio)

#assign nodes to be household, commercial, regulated and
# noncommercial, h is the number of household, r = regulated,
# and c = commercial. The remainder become the noncommercial nodes.
# Lists of corresponding weights are also produced for each
# type of node.
def assignnodes(h, r, c):

    (codelist, nodelist, ndlist) = sortNodeList(GN)
    allnodes = codelist[:]
    canodes = []
    caweight = []
    regnodes = []
    regweight = []
    ncanodes = []
    hhnodes = []
    hhweight = []

    #pick a chem node to become a r node and one of its r node preds
    # or preds to be r node also if there are more than one pred
    while len(regnodes)<r:
        y = choice(allnodes)

```

```

regnodes.append(y)
regweight.append(np.e**3)
allnodes.remove(y)
preds = GN.predecessors(y)
if len(preds)>1:
    w = choice(preds)
    predsw = GN.predecessors(w)
    if len(predsw)>1:
        u = choice(predsw)
        if u in allnodes:
            regnodes.append(u)
            regweight.append(np.e**3)
            allnodes.remove(u)

for x in range(h):
    y = choice(allnodes)
    hhnodes.append(y)
    hhweight.append(np.e**1)
    allnodes.remove(y)

for x in range(c):
    y = choice(allnodes)
    canodes.append(y)
    caweight.append(np.e**2)
    allnodes.remove(y)
ncanodes = allnodes

(nopredlist) = nopredlistmaker(GN)

for x in nopredlist:
    if x in ncanodes:
        y = choice(canodes)
        z = 0
        while y in nopredlist:
            y = choice(canodes)
            z = z + 1
            if z >10:
                break
        if z>10:
            ncanodes.remove(x)
            canodes.append(x)
            caweight.append(np.e**2)
        else:
            canodes.remove(y)
            ncanodes.append(y)
            ncanodes.remove(x)

```

```

        canodes.append(x)

    return hhnodes, regnodes, canodes, ncanodes, hhweight, regweight, caweight

(hhnodes, regnodes, canodes, ncanodes, hhweight, regweight, caweight,
 )=assignnodes(h,r,c)

#removes duplicates from list(values)
def remove_duplicates(values):
    output = []
    seen = set()
    for value in values:
        # If value has not been encountered yet,
        # add it to both list and set.
        if value not in seen:
            output.append(value)
            seen.add(value)
    return output

#produces list of weights for the noncommercial nodes
# the weights are obtained by the sum of the pred weights
# of the reaction node that produces the noncommercial chemical
def getncaweights(G, ncanodes):
    #create dictionary with a weight of zero for each node
    ncaweightdict = {}
    for x in ncanodes:
        ncaweightdict[x]=0
    (nopredlist) = nopredlistmaker(G)
    succnopredlist = []
    ncasuccnopredlist = []
    foundweights = {}
    for x in nopredlist:
        succx = G.successors(x)
        for y in succx:
            succnopredlist.append(y)
    for x in succnopredlist:
        foundweights[x]=0
    for x in succnopredlist:
        succs = G.successors(x)
        for y in succs:
            if y in ncanodes:
                ncasuccnopredlist.append(y)

#get weights for react nodes that are successors of nodes on nopredlist
# the nopredlist should have no nca nodes on it per nopredlistmaker

```

```

for x in nopredlist:
    succx = G.successors(x)
    for y in succx:
        if x in hhnodes:
            foundweights[y] = foundweights[y] + np.e**1
        if x in canodes:
            foundweights[y] = foundweights[y] + np.e**2
        if x in regnodes:
            foundweights[y] = foundweights[y] + np.e**3

for x in succnopredlist:
    preds = G.predecessors(x)
    countn = 0
    for y in preds:
        if y in ncanodes:
            countn += 1
    if countn == 0:
        foundweights[x] = foundweights[x]
    else:
        foundweights[x] = 0

for x in ncasuccnopredlist:
    countr = 0
    preds = G.predecessors(x)
    fwlist = []
    for y in preds:
        if y not in succnopredlist:
            countr = countr + 1
        elif foundweights[y]==0:
            countr += 1
        else:
            fwlist.append(foundweights[y])
    if countr == 0:
        ncaweightdict[x]=sum(fwlist)

count = 0
for x in ncanodes:
    if ncaweightdict[x] !=0:
        count += 1

def getno_nca_predval(ncanodes):
    no_ncapredvalue = []
    for x in ncanodes:
        if ncaweightdict[x]==0:
            no_ncapredvalue.append(x)

```

```

    return no_ncapredvalue

(no_ncapredvalue)=getno_nca_predval(ncanodes)
while no_ncapredvalue != []:
    for x in no_ncapredvalue:
        preds = G.predecessors(x)
        ylist = []
        no_ncadict = {}
        for y in preds:
            predy = G.predecessors(y)

            ylist.append(y)
            b = 0
            for z in predy:
                if z in no_ncapredvalue:
                    no_ncadict[str(y)+'-'+str(b)]=0
                else:
                    if z in ncanodes:
                        no_ncadict[str(y)+'-'+str(b)]=
                            ((ncaweightdict[x] + ncaweightdict[z]))
                    if z in hhnodes:
                        no_ncadict[str(y)+'-'+str(b)]=
                            ((ncaweightdict[x] + np.e**1))
                    if z in canodes:
                        no_ncadict[str(y)+'-'+str(b)]=
                            ((ncaweightdict[x] + np.e**2))
                    if z in regnodes:
                        no_ncadict[str(y)+'-'+str(b)]=
                            ((ncaweightdict[x] + np.e**3 ))
            b = b + 1

q = min(no_ncadict, key = lambda x: no_ncadict.get(x))
if no_ncadict[q] == 0:
    ncaweightdict[x]=no_ncadict[q]
    continue
else:

    valuelist = []
    allvalues = []
    for t in ylist:
        for k, v in no_ncadict.items():
            if k[0:len(str(t))]==str(t):
                valuelist.append(v)
    value = sum(valuelist)
    allvalues.append(value)
    valuelist = []

```

```

ncaweightdict[x]=min(allvalues)

(no_ncapredvalue)=getno_nca_predval(ncanodes)

return ncaweightdict

ncaweightdict = getncaweights(GN, ncanodes)

#produces a dict with the node and corresponding weight for the node
def assignweights(G):
    (cnodelist, nnodelist, ndlist) = sortNodeList(G)
    weightdict = {}
    for i in range(len(hhnodes)):
        weightdict[hhnodes[i]] = hhweight[i]
    for i in range(len(canodes)):
        weightdict[canodes[i]] = caweight[i]
    for i in range(len(regnodes)):
        weightdict[regnodes[i]] = regweight[i]
    weightdict.update(ncaweightdict)

    for u, v in G.edges():
        if u in nnodelist:
            preds = G.predecessors(u)
            s = 0
            for y in preds:
                s = weightdict[y] + s
            G.edge[u][v]['weight'] = s
        else:
            G.edge[u][v]['weight'] = 1

    return weightdict

weightdict=assignweights(GN)

#Relabel all the node...H for household R for regulated CA comm avail
# remaining C nodes are Non-commercially available
relabel_list_h = []
for x in range(len(hhnodes)):
    relabel_list_h.append(hhnodes[x])
    hhnodes[x]='H-' + str(x)

label_dict = dict(zip(relabel_list_h, hhnodes))
GN = nx.relabel_nodes(GN, label_dict)

relabel_list_r = []
for x in range(len(regnodes)):

```

```

    relabel_list_r.append(regnodes[x])
    regnodes[x]='R-' + str(x)

del label_dict
label_dict = dict(zip(relabel_list_r , regnodes))
GN = nx.relabel_nodes(GN,label_dict)

relabel_list_ca = []
for x in range(len(canodes)):
    relabel_list_ca.append(canodes[x])
    canodes[x]='CA-' + str(x)

del label_dict
label_dict = dict(zip(relabel_list_ca , canodes))
GN = nx.relabel_nodes(GN,label_dict)

#nx.write_gexf(GN, 'mydir ')

final_time = time.time() - start_time
print(str(final_time) + ' seconds')
```

1.8 Python Scale-Free Chemical Network Generation

Python code for generating the non-tree chemical network from the starting point of the Networks scale free network generator. This is defined as a function for testing, so it can be ran in a loop and timed for measuring statistics.

```

import networkx as nx
import os as os
import random as rand
import numpy as np
import time
from random import choice

def scale_free_generator(number):

    start_time = time.time()

    def get_pred_dict(GN):
        pred_dict = {}
        for x in GN.nodes():
            preds = GN.predecessors(x)
            pred_dict[x] = preds
        return pred_dict

    def get_succ_dict(GN):
```

```

succ_dict = {}
for x in GN.nodes():
    succs = GN.successors(x)
    succ_dict[x] = succs
return succ_dict

def isInt(v):
    try: int(v)
    except: return False
    return True

def sortNodeList(GN):

    ndlist = list(GN.nodes())
    codelist = []
    nodelist = []

    # compile a list of the C nodes and non-C (reaction) nodes
    for x in range(len(ndlist)):

        thestring = ndlist[x]
        if isInt(ndlist[x])==True:
            nodelist.append(ndlist[x])
            continue
        if thestring[0]=='C' or 'H' or 'R':
            codelist.append(ndlist[x])

    return codelist, nodelist, ndlist

def get_has_n_dict():
    has_n_dict = {}
    for k,v in succ_dict.items():
        w = v
        for x in w:
            if x in codelist:
                has_n_dict[k]=True
                break
            else:
                has_n_dict[k]=False
    return has_n_dict

def nopredlistmaker(GN):
    nopredlist=[]
    for x in GN.nodes():
        preds=GN.predecessors(x)
        if preds == []:

```

```

        nopredlist.append(x)
    return nopredlist

#Returns number of household, commercial, and regulated nodes
# according to graph size and desired ratio of commercial
# to noncommercial nodes
def get_h_r_c(GN, caRatio):
    if len(codelist)>=10000:
        h = 1948
        r = 161
        c = round(len(codelist)*((1-((h/len(GN.nodes()))+
            (r/len(GN.nodes()))))*caRatio))
    else:
        h = round(len(codelist)*.2)
        r = round(len(codelist)*.1)
        c = round(len(codelist)*(1-(.3))*caRatio)
    return h, r, c

def assignnodes(h, r, c):

    (codelist, nodelist, ndlist) = sortNodeList(GN)
    allnodes = codelist[:]
    canodes = []
    caweight = []
    regnodes = []
    regweight = []
    ncanodes = []
    hhnodes = []
    hhweight = []

    while len(regnodes)<r:
        y = choice(allnodes)
        regnodes.append(y)
        regweight.append(np.e**3)
        allnodes.remove(y)
        preds = GN.predecessors(y)
        if len(preds)>1:
            w = choice(preds)
            predsw = GN.predecessors(w)
            if len(predsw)>=2:
                u = choice(predsw)
                if u in allnodes:
                    regnodes.append(u)
                    regweight.append(np.e**3)
                    allnodes.remove(u)

```

```

for x in range(h):
    y = choice(allnodes)
    hhnodes.append(y)
    hhweight.append(np.e**1)
    allnodes.remove(y)

for x in range(c):
    y = choice(allnodes)
    canodes.append(y)
    caweight.append(np.e**2)
    allnodes.remove(y)
ncanodes = allnodes

(nopredlist) = nopredlistmaker(GN)

for x in nopredlist:
    if x in ncanodes:
        y = choice(canodes)
        z = 0
        while y in nopredlist:
            y = choice(canodes)
            z = z + 1
            if z > 10:
                break
        if z > 10:
            ncanodes.remove(x)
            canodes.append(x)
            caweight.append(np.e**2)
        else:
            canodes.remove(y)
            ncanodes.append(y)
            ncanodes.remove(x)
            canodes.append(x)

return (hhnodes, regnodes, canodes, ncanodes,
hhweight, regweight, caweight)

def remove_duplicates(values):
    output = []
    seen = set()
    for value in values:
        # If value has not been encountered yet,
        # add it to both list and set.
        if value not in seen:
            output.append(value)
            seen.add(value)

```

```

    return output

#make a list of nca nodes that have a nca predecessor
#with no value assigned
def getno_nca_predval(ncanodes):
    no_ncapredvalue = []
    for x in ncanodes:
        if ncaweightedict[x]==0:
            no_ncapredvalue.append(x)
    return no_ncapredvalue

def getncaweights(GN, ncanodes):

    (no_ncapredvalue)=getno_nca_predval(ncanodes)

    #make dict of nca node keys and values of nca predecessors
    nca_preds_of_preds = {}
    for x in ncanodes:
        preds = GN.predecessors(x)
        for y in preds:
            predy = GN.predecessors(y)
            ncalist=[]
            for z in predy:
                if z in ncanodes:
                    ncalist.append(z)
            nca_preds_of_preds[x]=ncalist

    #remove any loops of nca to react to nca back to the same nca
    for x in ncanodes:
        q = nca_preds_of_preds[x]
        for y in q:
            if y==x:
                ncanodes.remove(q)
                del ncaweightedict[q]
                canodes.append(q)
                caweight.append(np.e**2)
                no_ncapredvalue.remove(q)

    while no_ncapredvalue != []:
        for x in no_ncapredvalue:
            preds = GN.predecessors(x)
            ylist = []
            no_ncadict = {}
            for y in preds:
                predy = GN.predecessors(y)

```

```

ylist.append(y)

b = 0
for z in predy:
    if z in no_ncapredvalue:
        no_ncadict[str(y)+'-'+str(b)]=0
    else:
        if z in ncanodes:
            no_ncadict[str(y)+'-'+str(b)]=
                (ncaweightdict[x] + ncaweightdict[z])
        if z in hhnodes:
            no_ncadict[str(y)+'-'+str(b)]=
                (ncaweightdict[x] + np.e**1)
        if z in canodes:
            no_ncadict[str(y)+'-'+str(b)]=
                (ncaweightdict[x] + np.e**2)
        if z in regnodes:
            no_ncadict[str(y)+'-'+str(b)]=
                (ncaweightdict[x] + np.e**3)
        b = b + 1

q = min(no_ncadict, key = lambda x: no_ncadict.get(x))
if no_ncadict[q] == 0:
    ncaweightdict[x]=no_ncadict[q]
    continue
else:

    valuelist = []
    allvalues = []
    for t in ylist:
        for k, v in no_ncadict.items():
            if k[0:len(str(t))]==str(t):
                valuelist.append(v)
        value = sum(valuelist)
        allvalues.append(value)
        valuelist = []
    ncaweightdict[x]=min(allvalues)

(no_ncapredvalue)=getno_nca_predval(ncanodes)

return ncaweightdict

#assign the weights from the weight dictionaries
def assignweights(G):
    (cnodelist, nnodelist, ndlist) = sortNodeList(G)

```

```

weightdict = {}
for i in range(len(hhnodes)):
    weightdict[hhnodes[i]] = hhweight[i]
for i in range(len(canodes)):
    weightdict[canodes[i]] = caweight[i]
for i in range(len(regnodes)):
    weightdict[regnodes[i]] = regweight[i]
weightdict.update(ncaweightdict)

for u, v in G.edges():
    if u in nodelist:
        preds = G.predecessors(u)
        s = 0
        for y in preds:
            s = weightdict[y] + s
        G.edge[u][v]['weight'] = s
    else:
        G.edge[u][v]['weight'] = 1

return weightdict

```

```
#####
```

```
GN=nx.scale_free_graph(number)
```

```
GN=nx.DiGraph(GN)
```

```
GN.remove_edges_from(GN.selfloop_edges())
```

```

start_time = time.time()
ndlist = list(GN.nodes())
z = 0

```

```

#create dict with nodes as keys and predecessors as values
pred_dict = get_pred_dict(GN)
#create dict with nodes as keys and successors as values
succ_dict = get_succ_dict(GN)

```

```

# if a node has predecessors make it a chemical node
oldlist = []
newlist = []
for x in pred_dict:
    if pred_dict[x] != []:
        oldlist.append(x)
        chem = 'C-' + str(z)
        newlist.append(chem)

```

```

z += 1

# relable to graph with old node values as new chem node values
dictGN = dict(zip(oldlist , newlist))
GN = nx.relabel_nodes(GN, dictGN)

#recreate dictionaries with new labels
pred_dict = get_pred_dict(GN)
succ_dict = get_succ_dict(GN)

#recreate chemical node and reaction node lists with new labels
(codelist , nodelist , ndlist) = sortNodeList(GN)

#write a dict with nodes as keys and boolean true
# if it has react node preds
has_n_dict = get_has_n_dict()

#if a chem node has chem successors remove the edge and
# add one to a r node
modlist = []
for x in succ_dict:
    if x in codelist:
        try:
            if has_n_dict[x]==True:
                modlist = succ_dict[x]
                for y in modlist:
                    if y in codelist:
                        GN.remove_edge(x,y)
                        pick = choice(nodelist)
                        GN.add_edge(x,pick)
        except KeyError:
            continue

#update dictionaries
pred_dict = get_pred_dict(GN)
succ_dict = get_succ_dict(GN)

#this removes arc from r to c if another arc exists form c to r
for x in succ_dict:
    q = succ_dict[x]
    for y in q:
        r = succ_dict[y]
        try:
            if x in r:
                if x in codelist:
                    GN.remove_edge(y,x)

```

```

        else :
            GN.remove_edge(x,y)
    except :
        continue

(codelist , nodelist , ndlist) = sortNodeList(GN)

#if node is a reaction node and has no C predecesors add
# norm random num of C nodes
# node number beginning where z left off above
for n in GN.nodes():
    if n in codelist:
        continue
    list1 = GN.predecessors(n)
    if len(list1)==0:
        r = round(rand.normalvariate(2, .8))
        if r<=0:
            r = 1
        for x in range(r):
            GN.add_node('C-' + str(z))
            GN.add_edge('C-' + str(z), n)
            z = z + 1

(codelist , nodelist , ndlist) = sortNodeList(GN)

#if node is a react node and has no C sucessors
# add norm random num of C nodes
# node number beginning where z left off above
for n in GN.nodes():
    if n in codelist:
        continue
    list1 = list(GN.successors(n))
    if len(list1)==0:
        r = round(rand.normalvariate(1, 0.4))
        if r<=0:
            r = 1
        for x in range(r):
            GN.add_node('C-' + str(z))
            GN.add_edge(n, 'C-' + str(z))
            z = z + 1

(codelist , nodelist , ndlist) = sortNodeList(GN)

caRatio = .6
(h,r,c)=get_h_r_c(GN, caRatio)

```

```

(hhnodes, regnodes, canodes, ncanodes, hhweight,
regweight, caweight)=assignnodes(h,r,c)

#create dictionary with a weight of zero for each node
ncaweightdict = {}
for x in ncanodes:
    ncaweightdict[x]=0
(nopredlist) = nopredlistmaker(GN)
succnopredlist = []
ncasuccnopredlist = []
foundweights = {}
for x in nopredlist:
    succx = GN.successors(x)
    for y in succx:
        succnopredlist.append(y)
for x in succnopredlist:
    foundweights[x]=0
for x in succnopredlist:
    succs = GN.successors(x)
    for y in succs:
        if y in ncanodes:
            ncasuccnopredlist.append(y)

#get weights for react nodes that are successors of
# nodes on nopredlist. the nopredlist should have no
# nca nodes on it per nopredlistmaker
for x in nopredlist:
    succx = GN.successors(x)
    for y in succx:
        if x in hhnodes:
            foundweights[y] = foundweights[y] + np.e**1
        if x in canodes:
            foundweights[y] = foundweights[y] + np.e**2
        if x in regnodes:
            foundweights[y] = foundweights[y] + np.e**3

succnopredlist = remove_duplicates(succnopredlist)

for x in succnopredlist:
    preds = GN.predecessors(x)
    countn = 0
    for y in preds:
        if y in ncanodes:
            countn += 1
    if countn == 0:
        foundweights[x] = foundweights[x]

```

```

    else:
        foundweights[x] = 0

for x in ncasuccnopredlist:
    countr = 0
    preds = GN.predecessors(x)
    fwlist = []
    for y in preds:
        if y not in succnopredlist:
            countr += 1
        elif foundweights[y]==0:
            countr += 1
        else:
            fwlist.append(foundweights[y])
    if countr == 0:
        ncaweightdict[x]=sum(fwlist)

(no_ncapredvalue)=getno_nca_predval(ncanodes)

#if all no_nca value nodes have a predecessor that
# has no_nca value the loop will run forever;
# this checks if this is the case and reassigns one of the
# nodes to a ca node so the code will complete

# check if nca predecessors with nca predecessors have common nca
# nca predecessor causing a loop- if so make the loop
# nca node a ca node
maxpred=[]
for x in no_ncapredvalue:
    preds=GN.predecessors(x)
    for y in preds:
        predy=GN.predecessors(y)
        for z in predy:
            if z in no_ncapredvalue:
                maxpred.append(z)

for x in maxpred:
    if maxpred.count(x)>1:
        canodes.append(x)
        caweight.append(np.e**2)
    try:
        ncanodes.remove(x)
    except:
        continue

ncaweightdict = getncaweights(GN, ncanodes)

```

```

weightdict=assignweights(GN)

relabel_list_h = []
for x in range(len(hhnodes)):
    relabel_list_h.append(hhnodes[x])
    hhnodes[x]='H-' + str(x)

label_dict = dict(zip(relabel_list_h , hhnodes))
GN = nx.relabel_nodes(GN,label_dict)

relabel_list_r = []
for x in range(len(regnodes)):
    relabel_list_r.append(regnodes[x])
    regnodes[x]='R-' + str(x)

del label_dict
label_dict = dict(zip(relabel_list_r , regnodes))
GN = nx.relabel_nodes(GN,label_dict)

relabel_list_ca = []
for x in range(len(canodes)):
    relabel_list_ca.append(canodes[x])
    canodes[x]='CA-' + str(x)

del label_dict
label_dict = dict(zip(relabel_list_ca , canodes))
GN = nx.relabel_nodes(GN,label_dict)

#plt.clf()
#nx.draw_networkx(GN)

#    mydir = r'E:\Documents\Thesis '
#    os.chdir(mydir)

#nx.write_gexf(GN, 'mydir ')

final_time = time.time() - start_time
return final_time , GN, regnodes , hhnodes , canodes

final_time , G2, regnodes , hhnodes , canodes = scale_free_generator(500)

print(str(final_time) + '_seconds')
```

1.9 Python Powerlaw Fit Test

```
import powerlaw as pl
import networkx as nx
import matplotlib.pyplot as plt

def powerlawfit(G):

    plt.clf()
    dictdegree=G.degree()
    degree=sorted(dictdegree.values())
    degree=sorted(degree)
    results=pl.Fit(degree)

    fit = pl.Fit(degree)
    fig1 = fit.plot_pdf(color='b', label='Empirical_pdf', linewidth=1)
    fit.power_law.plot_pdf(color='g', ls='dashdot', ax = fig1 ,
                           label = "Power_Law_Fit_pdf")
    fit.exponential.plot_pdf(color='r', ls='dotted', ax=fig1 ,
                             label='Exponential_Fit_pdf')
    fit.lognormal.plot_pdf(color='m', ls='--', ax=fig1 ,
                            label='Lognormal_Fit_pdf')

    plt.ylabel(u'p(X)')
    plt.xlabel(r'Average_Degree')
    plt.title('Network_Powerlaw_Fit')
    from matplotlib.legend_handler import HandlerLine2D
    leg = plt.legend(handler_map={fig1: HandlerLine2D(numpoints=4)},loc=3)
    leg.draw_frame(False)

    R, p = results.distribution_compare('power_law', 'lognormal')
    R2,p2, = results.distribution_compare('power_law', 'exponential')
    a = results.alpha
    b = results.xmin
    c = results.sigma
    return a, b, c, R, p, R2, p2

(alpha, xmin, sigma, R, p, R2, p2) = powerlawfit(B)

print('alpha_is_' + str(alpha))
print('xmin_is_' + str(xmin))
print('sigma_is_' + str(sigma))
print('R_is_' + str(R) + '_for_lognormal_comparisson')
print('p_is_' + str(p) + '_for_lognormal_comparisson')
print('R2_is_' + str(R2) + '_for_exponential_comparisson')
print('p2_is_' + str(p2) + '_for_exponential_comparisson')
```

1.10 Python All Paths and Frequent Pattern Tree Code

Python code to find all shortest paths and find frequent item sets. Finds paths and items according to chemical node type.

```
import networkx as nx
from random import choice
import time
from orangecontrib.associate.fpgrowth import *

def remove_duplicates1(inlist):
    outlist = []
    seen = []
    for x in inlist:
        # If value has not been encountered yet,
        # add it to both list and set.
        if x not in seen:
            outlist.append(x)
            seen.append(x)
    return outlist

# sort nodes into a list of chemical nodes and a list of reaction nodes
def sortNodeList(GN):

    ndlist = list(GN.nodes())
    codelist = []
    nodelist = []

    # compile a list of the C nodes and non-C (reaction) nodes
    for x in range(len(ndlist)):

        thestring = ndlist[x]
        if isinstance(ndlist[x],int)==True:
            nodelist.append(ndlist[x])
            continue
        if thestring[0]=='C' or 'H' or 'R':
            codelist.append(ndlist[x])

    return codelist, nodelist, ndlist

def isInt(v):
    try: int(v)
    except: return False
    return True
```

```
#####
```

```

def find_solutions(GN):

    start_time1=time.time()

    (cnodelist , nnodelist , ndlist) = sortNodeList(GN)

    #choose a node from the reg chemical nodes at random and make sure
    # that it has predecesors
    solution = nx.all_pairs_dijkstra_path(GN)
    targetnode = choice(regnodes)
    hpreds = True
    while GN.predecessors(targetnode)==[] or hpreds == True:
        targetnode = choice(regnodes)
        for x in GN.predecessors(targetnode):
            predx = GN.predecessors(x)
            for y in predx:
                if y in hhnodes:
                    hpreds = True
                    break
            else:
                hpreds = False
        if hpreds == True:
            break

    soldict = solution[targetnode]

    # makes a dict of solutions starting with hh nodes as keys and the
    # solution path as values
    # also makes a list of the hh nodes ...also for c, ca, and r nodes
    hhsoldict = {}
    hhsollist = []
    for x in hhnodes:
        sol = solution[x]
        if targetnode in sol:
            hhsoldict[x]=sol[targetnode]
            hhsollist.append(x)

    csoldict = {}
    csollist = []
    for x in cnodelist:
        sol = solution[x]
        if targetnode in sol:
            csoldict[x]=sol[targetnode]
            csollist.append(x)

    casoldict = {}

```

```

casollist = []
for x in canodes:
    sol = solution[x]
    if targetnode in sol:
        casoldict[x]=sol[targetnode]
        casollist.append(x)

rsoldict = {}
rsollist = []
for x in nodelist:
    sol = solution[x]
    if targetnode in sol:
        rsoldict[x]=sol[targetnode]
        rsollist.append(x)

#makes a list of the solution lengths and finds the
# shortest solution
#creates a dictionary with the lengths as keys and
# the source node as value
csollist.remove(targetnode)

sollengthlist = []
sollengthdict = {}
sollengthdict_full = {}
hhsollengthlist = []
hhsollengthdict = {}
hhsollengthdict_full = {}
casollengthlist = []
casollengthdict = {}
casollengthdict_full = {}
for x in csollist:
    sollength = nx.dijkstra_path_length(GN,x,targetnode)
    sollengthlist.append(sollength)
    sollengthdict[sollength]=x
    sollengthdict_full[x] = sollength
    if x in hhnodes:
        hhsollengthlist.append(sollength)
        hhsollengthdict[sollength]=x
        hhsollengthdict_full[x]=sollength
    if x in canodes:
        casollengthlist.append(sollength)
        casollengthdict[sollength]=x
        casollengthdict_full[x] = sollength

#prints the length and path for the shortest path(s) any chem type
slist = []

```

```

for x in sollengthdict_full:
    if sollengthdict_full[x] == min(sollengthlist):
        slist.append(x)
answer = []
for x in slist:
    ssuccs = GN.successors(x)
    for y in ssuccs:
        if y in solution[x]:
            answer.append(GN.predecessors(y))

answer_s=remove_duplicates1(answer)

print( 'The_shortest_path_is_' + str(min(sollengthlist)))
print( 'This_path_is_from_node/nodes_' + str(answer_s))

#prints the length and path for the shortest path(s) from hh nodes
hlist = []
for x in hhsollengthdict_full:
    if hhsollengthdict_full[x] == min(hhsollengthlist):
        hlist.append(x)
answer = []
for x in hlist:
    hsuccs = GN.successors(x)
    for y in hsuccs:
        if y in solution[x]:
            answer.append(GN.predecessors(y))

answer_h=remove_duplicates1(answer)

try:
    print( 'The_shortest_path_from_household_chemical_is_' +
        str(min(hhsollengthlist)))
    print( 'This_path_is_from_node/nodes_' + str(answer_h))
except:
    print( 'No_path_from_household_chemicals')

#prints the length and path for the shortest path(s) from ca nodes
calist = []
for x in casollengthdict_full:
    if casollengthdict_full[x] == min(casollengthlist):
        calist.append(x)
answer = []
for x in calist:
    csuccs = GN.successors(x)
    for y in csuccs:
        if y in solution[x]:

```

```

        answer.append(GN.predecessors(y))

answer_ca=remove_duplicates1(answer)

try:
    print('The shortest path from commercial chemical is ' +
          str(min(casollengthlist)))
    print('This path is from node/nodes ' + str(answer_ca))
except:
    print('No path from commercial chemicals')

end_time1=time.time()-start_time1

#####
start_time2 = time.time()
#finds all ancestors from the source node
target_ancestors = list(nx.ancestors(GN,targetnode))

(codelist , nodelist , ndlist) = sortNodeList(GN)

#makes a list of ancestors that are react nodes and
# a list of chem nodes
ctarget_ancestors = []
rtarget_ancestors = []
for x in target_ancestors:
    if x in codelist:
        ctarget_ancestors.append(x)
    else:
        rtarget_ancestors.append(x)

#get path for all chem node predecessors of target node
cT = []
for x in ctarget_ancestors:
    cT.append(csoldict[x])

#get path from reaction node predecessors of target
rT = []
for x in rtarget_ancestors:
    rT.append(rsoldict[x])

hT = []
for x in hhnodes:
    if x in hhsoldict:
        hT.append(hhsoldict[x])

itemsets_c = frequent_itemsets(cT, 2)

```

```

itemsets_c = list(itemsets_c)
itemsets_h = frequent_itemsets(hT, 2)
itemsets_h = list(itemsets_h)
itemsets = frequent_itemsets(rT,2)
itemsets_r = list(itemsets)

end_time2 = time.time()-start_time2
final_time = time.time()-start_time1

itemsets_c = sorted(itemsets_c ,key=lambda x:x[1] , reverse=True)
itemsets_h = sorted(itemsets_h ,key=lambda x:x[1] , reverse=True)
itemsets_r = sorted(itemsets_r ,key=lambda x:x[1] , reverse=True)

return (end_time1 , end_time2 , final_time , sollengthlist , hhsollengthlist ,
        casollengthlist , answer_s , answer_h , answer_ca , itemsets_c ,
        itemsets_h , itemsets_r , targetnode)

(end_time1 , end_time2 , final_time , sollengthlist , hhsollengthlist ,
casollengthlist , answer_s , answer_h , answer_ca ,
itemsets_c , itemsets_h ,
itemsets_r , targetnode) = find_solutions(G)

final_time , G7 , regnodes , hhnodes , canodes = easy_tree_generator(50)

```

Bibliography

1. Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, 1215:487–499, 1994.
2. Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.
3. Andrew Allen and Thomas S. Cantrell. Synthetic Reductions in Clandestine Amphetamine and Methamphetamine Laboratories: A Review. *Forensic Science International*, 42(3):183–199, 1989.
4. John Andraos. *The Algebra of Organic Synthesis: Green Metrics, Design Strategy, Route Selection, and Optimization*. CRC Press, Boca Raton, 2012.
5. Rajesh Kumar Arora. *Optimization: Algorithms and Applications*. CRC Press, Boca Raton, 1 edition, 2015.
6. Alexandru T Balaban. Applications of Graph Theory in Chemistry. *Journal of Chemical Information and Computer Sciences*, 25(3):334–343, 1985.
7. Alexandru T Balaban, D Farcasiu, and R Banica. Graphs of multiple 1, 2-shifts in carbonium ions and related systems. *Revue Roumaine de Chimie*, 11(10):1205–+, 1966.
8. Albert-Laszlo Barabasi and Reka Albert. Emergence of Scaling in Random Networks. *Science*, 286(15 OCTOBER):509–512, 1999.
9. E. M. L. Beale. On Minimizing a Convex Function Subject to Linear Inequalities. *Journal of the Royal Statistical Society*1, 17(2):173–184, 1955.
10. Darren Boyle. Tiny Victims of Islamic State’s Chemical Arsenal: 1,500 People Need Treatment after Mustard Gas Attack on Iraq Village which Killed Girl, Three, 2016.
11. A Charnes, W W Cooper, and G H Symonds. Cost Horizons and Certainty Equivalents: An Approach to Stochastic Programming of Heating Oil. *Management Science*, 4(3):235–263, 1958.
12. Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-Law Distributions in Empirical Data. *SIAM review*, 51(4):661–703, 2009.
13. Carlos A Coello Coello. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems*, 1(3):269–308, 1999.

14. Elias J. Corey. General Methods for the Construction fo Complex Molecules. *Pure and Applied Chemistry*, 14(1):19–38, 1967.
15. Kelly J. Cormican, David P. Morton, and R. Kevin Wood. Stochastic Network Interdiction. *Operations Research*, 46(2):187–197, 1998.
16. James K. Cunningham and Lon Mu Liu. Impacts of Federal Precursor Chemical Regulations on Methamphetamine Arrests. *Addiction*, 100(4):479–488, 2005.
17. James K. Cunningham, Lon Mu Liu, and Russell C. Callaghan. Impact of US and Canadian Precursor Regulation on Methamphetamine Purity in the United States. *Addiction*, 104(3):441–53, 2009.
18. Joseph Czyzyk, Michael P. Mesnier, and Jorge J. More. The NEOS Server. *IEEE Computational Science and Engineering*, 5(3):68–75, 1998.
19. George Dantzig. Linear Programming Under Uncertainty. *Management Science*, 1(3-4):197–206, 1955.
20. George Dantzig, Selmer Johnson, and Wayne White. A Linear Programming Approach to the Chemical Equilibrium Problem. *Management Science*, 5(1):38–43, 1958.
21. George B. Dantzig. On the shortest route through a netwo. *Management Science*, 6(2):187–190, 1960.
22. Department of Homeland Security. Appendix to Chemical Facility Anti-Terrorism Standards; Final Rule, 2007.
23. Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Nemerisch Mathematik*, 1(1):269–271, 1959.
24. Suman Dutta. *Optimization in Chemical Engineering*. Cambride University Press, New Delhi, 2016.
25. Elsevier R&D Solutions. Reaxys Fact Sheet. Technical report, Elsevier Research and Development Solutions, 2016.
26. Eugene J. Fenech and Andrew Acrivos. The Application of Linear Programming to Design Problems. *Chemical Engineering Science*, 5(2):93–98, 1956.
27. Marcin Fialkowski, Kyle J M Bishop, Victor A. Chubukov, Christopher J. Campbell, and Bartosz A. Grzybowski. Architecture and Evolution of Organic Chemistry. *Angewandte Chemie - International Edition*, 44:7263–7269, 2005.
28. Robert W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

29. Lester R. Ford and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(3):399–404, 1956.
30. Patrick E. Fuller, Chris M. Gothard, Nosheen A. Gothard, Alex Weckiewicz, and Bartosz A. Grzybowski. Chemical Network Algorithms for the Risk Assessment and Management of Chemical Threats. *Angewandte Chemie - International Edition*, 51(32):7933–7937, 2012.
31. Ignacio E. Grossmann and Jorge Santibanez. Applications of Mixed-Integer Linear Programming in Process Synthesis. *Computers & Chemical Engineering*, 4(4):205–214, 1980.
32. Bartosz A Grzybowski, Kyle J M Bishop, Bartłomiej Kowalczyk, and Christopher E Wilmer. The 'Wired' Universe of Organic Chemistry. *Nature Chemistry*, 1(1):31–36, 2009.
33. Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, 2008:11–16, 2008.
34. Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, 2004.
35. V Hlaváček. Synthesis in the Design of Chemical Processes. *Computers & Chemical Engineering*, 2(1):67–75, 1978.
36. Eitan Israeli and R. Kevin Wood. Shortest-Path Network Interdiction. *Networks*, 40(2):97–111, 2002.
37. Kevin T. Kennedy, Richard F. Deckro, James T. Moore, and Kenneth M. Hopkinson. Nodal interdiction. *Mathematical and Computer Modelling*, 54(11):3116–3125, 2011.
38. Leonid Khachiyan, Vladimir Gurvich, and Jihui Ahao. Extending Dijkstra’s Algorithm to Maximize the Shortest Path by Node-wise Limited Arc Interdiction. In *International Computer Science Symposium in Russia*, chapter 24, pages 221–234. Springer, 2006.
39. Mikołaj Kowalik, Chris M. Gothard, Aaron M. Drews, Nosheen A. Gothard, Alex Weckiewicz, Patrick E. Fuller, Bartosz A. Grzybowski, and Kyle J M Bishop. Parallel Optimization of Synthetic Pathways within the Network of Organic Chemistry. *Angewandte Chemie - International Edition*, 51(32):7928–7932, 2012.
40. Jared Ledgard. *A Laboratory History of Narcotics, Vol. 1 Amphetamines and Derivatives*. Lulu.com, 2007.

41. Brian J. Lunday and Hanif D. Sherali. A dynamic network interdiction problem. *Informatica*, 21(4):553–574, 2010.
42. Bruce A. McCarl. McCarl (Expanded) GAMS User’s Guide Version 24.6, 2016.
43. Rebecca McKetin, Rachel Sutherland, David A. Bright, and Melissa M. Norberg. A Systematic Review of Methamphetamine Precursor Regulations. *Addiction*, 106(11):1911–1924, 2011.
44. David P. Morton, Feng Pan, and Kevin J. Saeger. Models for nuclear smuggling interdiction. *IIE Transactions*, 39(1):3–14, 2007.
45. National Alliance for Model State Drug Laws. Controlling Methamphetamine Precursors Ephedrine and Pseudoephedrine: A Brief History of Controls and Current Initiatives. Technical report, National Alliance for Model State Drug Laws, Santa Fe, 2011.
46. Robert M. Nauss. *Parametric Integer Programming*. University of Missouri Press, 1975.
47. James Nonnemaker, Mark Engelen, and Daniel Shive. Are Methamphetamine Precursor Control Laws Effective Tools to Fight the Methamphetamine Epidemic? *Health economics*, 20:519–531, 2010.
48. JA Hillier PD Whiting. A method for finding the shortest route through a road network. *Journal of the Operational Research Society*, 11(1-2):37–40, 1960.
49. Carl Adam Petri. Communicating with Automata: Volume 1 Supplement 1. Technical report, DTIC Document, 1966.
50. Orr Ravitz. Data-driven Computer Aided Synthesis Design. *Drug Discovery Today: Technologies*, 10(3):e443–e449, 2013.
51. Thomas Saaty and Saul Gass. Parametric Objective Function (Part 1). *Journal of the Operations Research Society of America*, 2(3):316–319, 1954.
52. Harry F. Skinner. Methamphetamine Synthesis via Hydriodic Acid/Red Phosphorus Reduction of Ephedrine. *Forensic Science International*, 48(2):123–134, 1990.
53. Amy Smithson. Rethinking the Lessons of Tokyo. Technical report, Stimson Center, Washington, DC, 2000.
54. Sara Szymkuc, Ewa P. Gajewska, Tomasz Klucznik, Karol Molga, Piotr Dittwald, Michal Startek, Michal Bajczyk, and Bartosz A. Grzybowski. Computer-Assisted Synthetic Planning: The End of the Beginning. *Angewandte Chemie - International Edition*, 55:5904–5937, 2016.

55. The Americal Chemical Society. Reactions - CASREACT - Answers to your Chemical Reaction Questions., 2017.
56. Jonathan B. Tucker. *War of Nerves: Chermical Warfare from World War I to al-Qaeda*. Anchor Books, New York, 1 edition, 2006.
57. US Department fo Health and Human Services. Household Products Database, 2016.
58. US Department of Justice Drug Enforcement Administration. 2015 National Drug Threat Assessment Summary. Technical report, US Department of Justice Drug Enforcement Administration, 2015.
59. Stephen Warshall. A Theorem on Boolean Matrices. *Journal of the ACM*, 9(1):11–12, 1962.
60. R. Kevin Wood. Deterministic Network Interdiction. *Mathematical and Computer Modelling*, 17(2):1–18, 1993.
61. R. Zenklusen. Network Flow Interdiction on Planar Graphs. *Discrete Applied Mathematics*, 158(13):1441–1455, 2010.

REPORT DOCUMENTATION PAGE					Form Approved OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY)		2. REPORT TYPE		3. DATES COVERED (From — To)		
23-03-2017		Master's Thesis		Sep 2015 — Mar 2017		
4. TITLE AND SUBTITLE THE USE OF DATA MINING AND NETWORK ALGORITHMS FOR CHEMICAL WARFARE AGENT INTERDICTION				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) Cox, Travis L., CPT, USA				5d. PROJECT NUMBER		
				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT-ENS-MS-17-M-120		
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) USSOCOM CBD Program Coordinator J53-CBRN Branch Macdill Air Force Base, Florida 33621 DSN 299-0028, COMM 813-826-0028 Email: ronald.mussone.civ@socom.mil				10. SPONSOR/MONITOR'S ACRONYM(S) USSOCOM		
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION / AVAILABILITY STATEMENT DISTRIBUTION STATEMENT A: APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.						
13. SUPPLEMENTARY NOTES This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.						
14. ABSTRACT The clandestine production of chemical warfare agents is a clear threat to the security of all populations in a world where groups and individuals will use any means necessary to terrorize others and advance their agenda. Chemical synthesis routes for the production of chemical warfare agents from common household chemicals are known. This is troubling because the clandestine production of drugs from common household chemicals has been notoriously hard to stop. This research discusses methods for identifying key targets to interdict the clandestine production of dangerous chemicals through the use of both shortest path and frequent pattern search algorithms.						
15. SUBJECT TERMS Chemical Warfare Agents,Interdiction,Network Chemical Synthesis Model,Data Mining,Frequent Pattern Search, Chemical Agent Interdiction,Drug Interdiction						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON	
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Richard F. Deckro, AFIT/ENS	
U	U	U	U	145	19b. TELEPHONE NUMBER (include area code) 937-255-6565 x4325; richard.deckro@afit.edu	