


2017

# On Comparative Algorithmic Pathfinding in Complex Networks for Resource-Constrained Software Agents

Michael Moran  
*Walden University*

Follow this and additional works at: <https://scholarworks.waldenu.edu/dissertations>

 Part of the [Databases and Information Systems Commons](#), and the [Statistics and Probability Commons](#)

---

This Dissertation is brought to you for free and open access by the Walden Dissertations and Doctoral Studies Collection at ScholarWorks. It has been accepted for inclusion in Walden Dissertations and Doctoral Studies by an authorized administrator of ScholarWorks. For more information, please contact [ScholarWorks@waldenu.edu](mailto:ScholarWorks@waldenu.edu).

# Walden University

College of Management and Technology

This is to certify that the doctoral study by

Michael Moran

has been found to be complete and satisfactory in all respects,  
and that any and all revisions required by  
the review committee have been made.

## Review Committee

Dr. Timothy Perez, Committee Chairperson, Information Technology Faculty

Dr. Steven Case, Committee Member, Information Technology Faculty

Dr. Gail Miles, University Reviewer, Information Technology Faculty

Chief Academic Officer

Eric Riedel, Ph.D.

Walden University  
2017

Abstract

On Comparative Algorithmic Pathfinding in Complex Networks for

Resource-Constrained Software Agents

by

Michael J. Moran

MS, Columbus State University, 2013

BA, Thomas Edison State University, 2010

Doctoral Study Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Information Technology

Walden University

July 2017

## Abstract

Software engineering projects that utilize inappropriate pathfinding algorithms carry a significant risk of poor runtime performance for customers. Using social network theory, this experimental study examined the impact of algorithms, frameworks, and map complexity on elapsed time and computer memory consumption. The 1,800 2D map samples utilized were computer random generated and data were collected and processed using Python language scripts. Memory consumption and elapsed time results for each of the 12 experimental treatment groups were compared using factorial MANOVA to determine the impact of the 3 independent variables on elapsed time and computer memory consumption. The MANOVA indicated a significant factor interaction between algorithms, frameworks, and map complexity upon elapsed time and memory consumption,  $F(4, 3576) = 94.09, p < .001, \eta^2 = .095$ . The main effects of algorithms,  $F(4, 3576) = 885.68, p < .001, \eta^2 = .498$ ; and frameworks,  $F(2, 1787) = 720,360.01, p < .001, \eta^2 = .999$ ; and map complexity,  $F(2, 1787) = 112,736.40, p < .001, \eta^2 = .992$ , were also all significant. This study may contribute to positive social change by providing software engineers writing software for complex networks, such as analyzing terrorist social networks, with empirical pathfinding algorithm results. This is crucial to enabling selection of appropriately fast, memory-efficient algorithms that help analysts identify and apprehend criminal and terrorist suspects in complex networks before the next attack.

On Comparative Algorithmic Pathfinding in Complex Networks for Resource-  
Constrained Software Agents

by

Michael J. Moran

MS, Columbus State University, 2013

BA, Thomas Edison State University, 2010

Doctoral Study Submitted in Partial Fulfillment  
of the Requirements for the Degree of  
Doctor of Information Technology

Walden University

July 2017

## Dedication

I dedicate this to my late father, Robert, who I miss dearly, whose service to his country during WWII helped shape my values and encourage me to similarly support my country in the U.S. Army during the First Gulf War era. When I was young, he always tried to answer all my questions, and as I got older and the questions got tougher, he never wavered in his support, even when he no longer had answers (or, no longer had answers that my younger, unwise self, wanted to hear). I will always remember the kindness, wisdom and love he shared with me, my wife and two sons. Dad, we love and miss you.

I also dedicate this to my wife, Letty. I thank her for her unwavering and incalculable support over these last few years. I know it hasn't been easy with me in school, nor was our 2015 relocation across country for my new job easy, so I love you for patiently supporting me. And I thank my two boys, Michael and John, for putting up with a dad who was usually too busy writing or studying, to do anything cool. I missed a lot of family time, I know. I apologize for being so busy, and I love you both for your continued support. This degree is for you. May you learn from my experience; complete your future degrees *before* you have children!

*"Go where you are loved. People who see the best in you bring out the best in you."*

-- Lupita Nyong'o

## Acknowledgments

Many thanks to my committee members and instructors for guidance, especially my Chair, Dr. Tim Perez, my Second Chair, Dr. Steven Case, and my URR, Dr. Gail Miles.

Special thanks go out to my instructors Dr. Steve Case, and Dr. Bhanu Kapoor (my former Chair), for being inspirational mentors early in my doctoral study, during a time of confusion, darkness and strife. Remember, the DIT degree program was completely new at that time (late 2013 through 2015); it was constantly changing; many in my cohort dropped out for various reasons; and no one had yet graduated from the program. Your positive spirits and attitudes were greatly appreciated, as was the much-appreciated introduction to downtown Atlanta's finest Indian cuisine at Haveli's restaurant, during my second DIT residency in August 2015. The DIT program has grown from those early days, and now has graduates. Thank you.

Finally, my current employer and immediate managers deserve a big thank you affording me time to complete my research. Without your assistance, this endeavor would have taken me much more time. Thank you for all your support!

*"Do or do not. There is no try."*

*"Always pass on what you have learned."*

-- Yoda

## Table of Contents

List of Tables .....	v
List of Figures .....	vii
Section 1: Foundation of the Study.....	1
Background of the Problem .....	1
Problem Statement.....	2
Purpose Statement.....	3
Nature of the Study .....	3
Quantitative Research Question.....	5
Hypotheses.....	5
Theoretical Framework.....	5
Definition of Terms.....	7
Assumptions, Limitations, and Delimitations.....	9
Assumptions.....	9
Limitations .....	11
Delimitations.....	13
Significance of the Study .....	14
Contribution to Information Technology Practice.....	14
Implications for Social Change.....	15
A Review of the Professional and Academic Literature.....	16
Theoretical Framework: Social Network Theory .....	19
Modern Applications of Social Network Theory.....	43



Rival Theory to the Selected Theoretical Framework .....	51
Independent Variable: Pathfinding Algorithms .....	55
Independent Variable: Graph Analysis Frameworks .....	60
Independent Variable: Map Complexity.....	63
Dependent Variable: Elapsed Time .....	70
Dependent Variable: Memory Consumption .....	74
Computer Programming Languages: Python vs. Java .....	76
Modern Applications of Algorithmic Pathfinding.....	80
Transition and Summary.....	85
Section 2: The Project.....	88
Purpose Statement.....	88
Role of the Researcher .....	89
Participants.....	90
Research Method and Design .....	92
Method .....	92
Research Design.....	93
Population and Sampling .....	101
Ethical Research.....	112
Data Collection .....	113
Instruments.....	113
Data Collection Technique .....	127
Data Analysis Technique .....	131

Study Validity .....	136
Transition and Summary.....	141
Section 3: Application to Professional Practice and Implications for Change .....	143
Overview of Study .....	143
Presentation of the Findings.....	143
Pilot Test of the Algorithm Instrumentation.....	144
Results of the Algorithm Instrumentation Pilot Test .....	150
MANOVA and its Relationship to the Experimental Variables.....	150
Data Screening and Transformations.....	152
Descriptive Statistics.....	153
MANOVA Assumptions.....	156
MANOVA Statistical Output.....	165
Interpretation of Inferential Results .....	171
Summary and Theoretical Framework Implications.....	183
Applications to Professional Practice .....	191
Implications for Social Change.....	193
Recommendations for Action .....	195
Recommendations for Further Study .....	197
Reflections .....	201
Summary and Study Conclusions.....	203
References.....	205
Appendix A: Graph-Tool A* Algorithm Instrument.....	240

Appendix B: Graph-Tool Bellman-Ford Algorithm Instrument.....	243
Appendix C: Graph-Tool Dijkstra Algorithm Instrument .....	246
Appendix D: Network-X A* Algorithm Instrument.....	249
Appendix E: Network-X Bellman-Ford Algorithm Instrument.....	251
Appendix F: Network-X Dijkstra Algorithm Instrument .....	254
Appendix G: Python TimeIt Instrument .....	256
Appendix H: Python Memory_Profiler Instrument .....	258

## List of Tables

Table 1. The Categorical Variables and their Levels.....	96
Table 2. The Experiment: Randomized, Between Groups, Post-Test Only .....	97
Table 3. The 12-way Factorial Matrix, in Standard Research Design Notation.....	98
Table 4. List of Pathfinding Algorithms Analyzed in this Study (per Graph Framework) .....	99
Table 5. List of Graph Analysis Frameworks Analyzed in this Study .....	99
Table 6. Map Complexities Considered in this Study .....	100
Table 7. Dependent Variables Analyzed in this Study .....	100
Table 8. Summary List of Variables Used in this Study.....	100
Table 9. Small-World Network Properties of the 2D Map Samples .....	103
Table 10. Recommended Sample Sizes: Summary of G*Power Inputs and Results .....	107
Table 11. List of Instruments Used and their Validity and Reliability References .....	114
Table 12. The 8 Instruments Used in this Study and their Reference Locations.....	115
Table 13. Wilcoxon Signed Ranks -- Pilot Test Results for Graph-Tool A* Instrument	146
Table 14. Wilcoxon Signed Ranks -- Pilot Test Results for Graph-Tool Bellman-Ford Instrument .....	146
Table 15. Wilcoxon Signed Ranks -- Pilot Test Results for Graph-Tool Dijkstra Instrument .....	147
Table 16. Wilcoxon Signed Ranks -- Pilot Test Results for Network-X A* Instrument	148
Table 17. Wilcoxon Signed Ranks -- Pilot Test Results for Network-X Bellman-Ford Instrument .....	148

Table 18. Wilcoxon Signed Ranks -- Pilot Test Results for Network-X Dijkstra	
Instrument .....	149
Table 19. Sample Counts (N) per Between-Subject Factors .....	153
Table 20. Statistical Test, Assumptions, and Methods of Verifying Assumptions .....	156
Table 21. Mahalanobis Distances between Elapsed_Time and Memory_Consumption	158
Table 22. The General MANOVA Analysis Process (Mertler & Reinhart, 2017, p. 128)	
.....	165
Table 23. Homogeneous Subsets (Scheffe): Elapsed Time .....	169
Table 24. Homogeneous Subsets (Scheffe): Memory Consumed .....	170
Table 25. Summary of the Multivariate (Pillai's Trace) Tests <sup>a</sup> .....	172
Table 26. Means and Standard Deviations for the Dependent Variables for All Treatment	
Groups.....	182

## List of Figures

Figure 1. Milgram's small-world theory in a small social network. ....	6
Figure 2. References by peer review status. ....	19
Figure 3. References by year of publication. ....	19
Figure 4. An example graph.....	20
Figure 5. Sparse and dense graphs visually compared. ....	24
Figure 6. A high density (i.e., low occlusion ratio, few obstructions) grid map and corresponding graph.....	26
Figure 7. A low density (i.e., high occlusion ratio, many obstructions) grid map and its corresponding graph.....	27
Figure 8. Two regular 2D lattice networks: grid (left) and circular (right). ....	33
Figure 9. Two semi-random 2D lattice networks: grid (left) and circular (right).....	34
Figure 10. Regular, small-world & random networks (based on Watts & Strogatz, 1998). .....	37
Figure 11. (a) Shortest grid path, (b) the real shortest path (based on Nash & Koenig, 2013). ....	69
Figure 12. Grid-based autonomous rover algorithmic pathfinding (NASA, n.d.).....	84
Figure 13. High-level overview of this study's experimental process flow.....	96
Figure 14. This study's experimental 12-way factorial matrix. ....	97
Figure 15. Example two dimensional grid map.....	101
Figure 16. Population and sample stratification plan.....	104
Figure 17. MANOVA interaction effects: 12 Groups, 3 IVs, and 2 DVs.....	108

Figure 18. MANOVA main effects for IV algorithm (with three levels).....	108
Figure 19. MANOVA main effects for IVs framework and map complexity (each with two levels).....	108
Figure 20. An example of random assignment used in this study. ....	110
Figure 21. The relationships between this study's instrumentation. ....	116
Figure 22. A python example of time profiling using the TimeIt python module.....	121
Figure 23. A python example using the memory_profiler python module.....	124
Figure 24. A random generated grid map (left) and its abstracted genotype (right). ....	128
Figure 25. Outline of python program to collect experimental data. ....	130
Figure 26. Outline of python program to parse experimental data. ....	131
Figure 27. Descriptive statistics (part 1 of 2): elapsed time. ....	154
Figure 28. Descriptive statistics (part 2 of 2): computer memory consumption. ....	155
Figure 29. Graph-Tool, A* (a-star): scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right). ....	160
Figure 30. Graph-Tool, Bellman-Ford: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right). ....	161
Figure 31. Graph-Tool, Dijkstra: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right). ....	161

Figure 32. Network-X, A* (a-star): scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right). .....	162
Figure 33. Network-X, Bellman-Ford: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right). .....	162
Figure 34. Network-X, Dijkstra: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right). .....	163
Figure 35. Box's M-test for equality of covariance matrices. ....	163
Figure 36. MANOVA summary table of multivariate results. ....	166
Figure 37. Univariate ANOVA data summary. ....	167
Figure 38. Post hoc results (Scheffe test) for elapsed time and memory consumed, per pathfinding algorithm.....	168
Figure 39. Homogeneous subsets: mean elapsed time per pathfinding algorithm. ....	169
Figure 40. Homogeneous subsets: mean memory consumption per pathfinding algorithm. ....	170
Figure 41. Multivariate effect sizes (Pillai's trace). ....	174
Figure 42. Elapsed time (seconds) per framework and algorithm. ....	178
Figure 43. Memory consumed (megabytes) per framework and algorithm. ....	179
Figure 44. Elapsed time (seconds) per map complexity and algorithm.....	180
Figure 45. Memory consumed (megabytes) per map complexity and algorithm. ....	180



Figure 46. Elapsed time (seconds) per algorithm and map complexity.....	181
Figure 47. Memory consumed (megabytes) per algorithm and map complexity. ....	181
Figure A1. Abbreviated Graph-Tool A* (a-star) algorithm API demonstration. ....	241
Figure B1. Abbreviated Graph-Tool Bellman-Ford algorithm API demonstration. ....	244
Figure C1. Abbreviated Graph-Tool Dijkstra algorithm API demonstration. ....	247
Figure D1. Abbreviated Network-X A* (a-star) algorithm API demonstration. ....	250
Figure E1. Abbreviated Network-X Bellman-Ford algorithm API demonstration. ....	252
Figure F1. Abbreviated Network-X Dijkstra algorithm API demonstration. ....	255
Figure G1. Abbreviated python TimeIt API demonstration.....	257
Figure H1. Abbreviated python memory_profiler API demonstration.....	260

## Section 1: Foundation of the Study

Algorithms play an important role in computer science. In modern computing, algorithms are the rules and step-by-step instructions by which computer programs solve problems. Because algorithms are necessary to modern computing, it is important that software engineers choose appropriate algorithms. Poor algorithm selection may yield suboptimal computer program performance to the detriment of customers. For example, in the scenario of unmanned aerial vehicle (UAV) flight navigation, poor pathfinding algorithm choice can lead to tracking and navigation errors (Liu, Egan, & Santoso, 2015), which may result in costly accidents. In the scenario of semiautonomous robotic microsurgery, poor pathfinding algorithm choice may lead to permanent injury or paralysis (Gerber et al., 2014). This study does not cover every aspect of algorithm choice, design or implementation, but the intent of this study is to provide software engineers with information on applied pathfinding algorithm performance, so they can make better-informed algorithm choices when writing their own pathfinding software.

### **Background of the Problem**

Stakeholders face many challenges when creating good software, in part because engineering good, non-trivial software is not easy (Wohlin & Aurum, 2015). It is incumbent upon software engineers to appropriately select the algorithms used in the software they write. However, for some problem domains, such as robotic search and rescue, algorithm selection may be very complicated because there are so many algorithms from which to select and implement, thus creating an inconvenient gap between what is theoretically possible, and real-world physical limitations (Bazregar,

Piltan, Nabaee, & Ebrahimi, 2013). By sharing knowledge gained from applied algorithm experiments, effective algorithm selection may be made easier. By examining the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time and memory consumption, this study was specifically designed to provide software engineers writing pathfinding software with new insight on comparative pathfinding algorithm performance.

### **Problem Statement**

The shortest path problem is a critical issue in diverse domains like Internet packet routing, military, robotics, transportation, and social networking – Facebook for example manages a graph containing over 1 billion users (Balaguru, Nallathamby & Robin, 2015; Brooks, Hogan, Ellison, Lampe & Vitak, 2014). Peta-scale pathfinding problems are unsolvable within a human timescale when using poorly selected algorithms, but with appropriate algorithms it is possible to achieve a significant 80× factor improvement in performance (Franke & Ivanova, 2014). The general IT problem is software engineers sometimes select inappropriate algorithms, resulting in poor software performance. The specific IT problem is that some software engineers lack information on the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and memory consumption, in order to select appropriate pathfinding algorithms for resource-constrained software agents running in complex networks, network dead zones or GPS-denied environments.

### **Purpose Statement**

The purpose of this quantitative experimental study is to examine the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and memory consumption, in order to select appropriate pathfinding algorithms for resource-constrained software agents running in complex networks, network dead zones or GPS-denied environments. The targeted population consists of local computer random-generated two-dimensional (2D) maps. The three independent variables are (a) pathfinding algorithms; (b) graph analysis frameworks; and (c) map complexity (e.g., small vs. large maps; high random rewiring vs. low random rewiring). The two dependent variables are (a) elapsed time, and (b) computer memory consumption. Contributions to positive social change from efficient pathfinding algorithms are wide-ranging, from saving lives to saving money. Some recent examples include (a) fast robotic debris cleanup of airport runways to prevent fatal accidents during takeoff and landing (Öztürk & Kuzucuoğlu, 2016); (b) bounded-cost optimization of business expenses (Stern, et al., 2014); and (c) search and rescue missions in unmapped terrain (Liu & Lyons, 2015).

### **Nature of the Study**

This doctoral study follows a quantitative research method. Based on a positivist philosophy (Luft & Shields, 2014), the goal of this study is to examine potential causal relationships between these three independent variables: (a) pathfinding algorithms; (b) graph analysis frameworks; (c) map complexity (e.g., small vs. large maps; high random rewiring vs. low random rewiring); and these two dependent variables: (d) elapsed time; and (e) the amount of computer memory consumed during pathfinding operations.

Researchers employing qualitative methods may explore new problems by seeking open-ended *where* or *who* answers rather than statistically explain a cause-effect outcome (Balakrishnan & Penno, 2014). Because this study aims to identify cause-effect relationships between the aforementioned variables of interest, not to answer open-ended *where* or *who* questions, this renders qualitative research methods inappropriate. Mixed methods research involves combining both quantitative and qualitative approaches within a single research study (Daigneault & Jacob, 2014). Because this study does not use qualitative research methods, this renders the mixed methods approach inappropriate. Quantitative methods may use descriptive statistics to describe the sample population, and inferential statistics to infer the results to the broader population (Hoare & Hoe, 2013, p. 50). The quantitative method was selected over a qualitative approach (e.g., case study, ethnographic, phenomenological) because of my desire to statistically identify cause-effect between the variables of interest.

Experimental designs are considered strongest of all designs regarding internal validity, which itself is the center of cause-effect inferences (Gassen, 2014). An experimental design was selected for this study because of the desire to identify causal relationships between the variables of interest by intentional manipulation of the independent variables, sample stratification, and random assignment of samples to treatment groups. As indicated by Turner, Balmer, and Coverdale (2013), quasiexperimental designs do not permit random assignment of samples to treatment groups, and correlational designs do not permit control or manipulation of treatments (p. 305). Therefore, because of its lack of random sample assignment to treatment groups, a

quasiexperimental design is not appropriate. Because my research involves intentional manipulation of the independent variables in order to measure possible treatment effects on the dependent variables, the correlational design is therefore also rendered inappropriate.

### **Quantitative Research Question**

What is the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption?

### **Hypotheses**

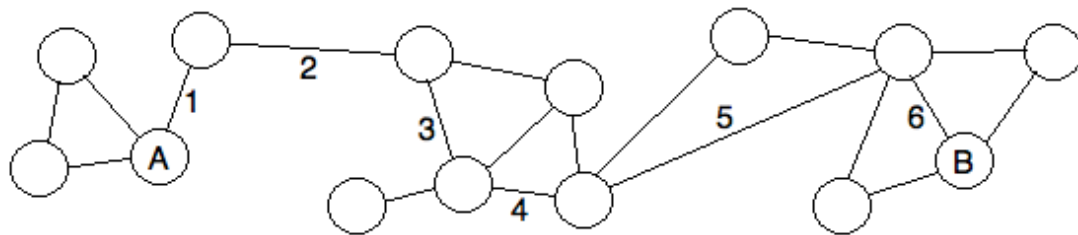
Null Hypothesis ( $H_0$ ): There is no relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption.

Alternative Hypothesis ( $H_a$ ): There is a relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption.

### **Theoretical Framework**

Social network theory grounded my study. Social network theory has roots in graph theory, which itself has its origin in the 18th century work of Leonard Euler (Albert & Barabási, 2002, p. 9). The grand premise of social network theory is that patterns of interaction among *nodes* in a network (i.e., the people or objects abstracted in a graph) are the building blocks of networks (Erikson, 2013; Krause, Croft, & James, 2007; Merchant, 2012). In 1954, social networks were first mentioned in a scientific context by the social anthropologist John A. Barnes during his anthropological research on the

population of a small fishing village in Bremnes, Norway (Barnes, 1954; Wasserman & Faust, 1994, p. 10). In 1965, Stanley Milgram developed his theory of small-world social networks (Korte & Milgram, 1970). His small-world experiments examined the average path length of social networks between people in the United States. Today, Milgram's seminal work is considered the basis of modern social network theory (Wang, 2015).



An example of the six degrees of separation between persons A and B. Circles represent people, and the lines denote social connections between people in their social networks. Notice the shortest path between persons A and B has a path length of 6.

*Figure 1.* Milgram's small-world theory in a small social network.

As applied to this doctoral study, social networks can be mathematically represented as 2D graphs (adjacency matrices), which can represent many different relationships (Kepner et al., 2015). In this study, social network theory is applied to gain an understanding of the relationship between the dependent and independent variables as applied to the shortest path problem in networks represented as 2D graphs. Social network theory drives this research because it addresses pathfinding, connectivity, and path lengths in networks, all of which are key concepts addressed in this pathfinding algorithm study.

Milgram's small-world social network theory (Korte & Milgram, 1970) explains the choice of independent and dependent variables because in this study the independent

variables are deliberately manipulated to see what impact (if any) such changes have on the dependent variables. The independent variable *map complexity* controls the overall maximum size of each graph to be searched by each of the pathfinding algorithms compared, per graph analysis framework, and it drives the complexity of the network structure by controlling the level of randomness exhibited in nodal connectivity patterns (e.g., a grid vs. a random network structure) in the 2D grid maps (and their underlying adjacency matrices). The interactions between pathfinding algorithms, graph analysis frameworks, and map complexity were measurable and noticeably impacted the dependent variables: elapsed time and computer memory consumption. These relationships were measured and statistically analyzed in this empirical study.

### **Definition of Terms**

The content of this study is graph theoretic and mathematical in nature, as such there may be terms that could be unfamiliar to readers. The following definitions provide context to what may be unfamiliar terms.

*Clustering coefficient*: A mathematical value, ranging from 0.0 to 1.0, representing the tendency for connected node communities to form in a graph (Albert & Barabási, 2002, p. 3). It is calculated by dividing the actual number of links (edges) in a graph, by the maximum possible number of links in that graph.

*Degree distribution*: The probability that a randomly selected node in a graph has exactly  $k$  edges, where  $k$  is a number  $\geq 0$  (Albert & Barabási, 2002, p. 3).

*Graph*: A mathematical and visual representation of a network, where the nodes (vertices) are represented by circles or dots, and the edges which connect the vertices are



represented by lines (or arcs) (Barnes, 1954, p. 43). A *complete graph* is one in which each node in a graph is connected to every other node within the same graph.

*Graph Analysis Framework:* These are source code libraries and software programs that enable graph theoretic network analyses and visualizations of complex networks (Nocke et al., 2015). Many are free or open source, while others are proprietary. The graph analysis frameworks used in this study were free or open source. Some graph analysis frameworks provide an application programmer's interface (API) which permits software engineers to programmatically utilize internal framework code within custom computer programs, thereby extending (customizing) the utility of the graph analysis framework.

*Path length:* The number of steps (links) in a graph between the starting object and the destination object (Barnes, 1954, p. 46). More specifically, the shortest path length would be the least number of steps (links) between two nodes of a graph (Albert, Jeong, & Barabási, 1999).

*Random network:* Graphs where the probability that any two vertices of a graph being connected is completely random (Barabási & Albert, 1999, p. 511). These networks exhibit little overt structure or pattern to the way the vertices are connected, yet tend to have short path lengths.

*Regular (i.e., grid) network:* Graphs where nodes and edges are constructed in an organized fashion, like a two-dimensional (2D) grid such as a chessboard. Nodes in regular networks are not randomly connected because connections in regular networks are structured (i.e., regularly positioned), unlike connections in random networks.

Regular networks have higher clustering coefficients (i.e., nodes tend to share similar connections with their neighbors) and longer path lengths, than comparable-sized random networks, or small-world networks (Albert & Barabási, 2002; Watts & Strogatz, 1998).

*Scale-free network:* A graph where the degree distribution follows a power law (Albert & Barabási, 2002, p. 27), not a Poisson (i.e., Bell-curve) distribution. These graphs are characterized by most nodes having few links, held together by a few super connected hub nodes. The *hub and spoke* network architecture of the air traffic system is a relevant example.

*Small-world network:* Graphs which are rich in structured short-range connections (i.e., high clustering coefficient), but also have a few, random, long-range connections (Kleinberg, 2000, p. 845; Watts & Strogatz, 1998). These few long-range connections give small-world networks overall shorter path lengths than corresponding regular networks (Zhang & Wang, 2013).

*Social network analysis:* The analysis of the relationship between network actors. The actors may be individual humans, or they could be organizations, nation states, animals, bank accounts, IP addresses, etc. The typical focus of social network analysis is on relationships (edges, arcs) between the actors, not on the individual actors (nodes, vertices) themselves (Erikson, 2013, pp. 219-221).

## **Assumptions, Limitations, and Delimitations**

### **Assumptions**

Assumptions are factors or beliefs that could be considered true, but may be difficult to verify (Kirkwood & Price, 2013). These beliefs may drive the approaches and

conduct of the research process itself, and the conclusions drawn afterwards (Kirkwood & Price, 2013). I assumed the computer language selected for implementation of this doctoral study's experimental framework, Python, was appropriate for scientific and research-oriented computing, as suggested by Day (2014, p. 88), and for data analysis, as suggested by Severance (2015, p. 10), and therefore was appropriate for this study. As a long time Java developer, this was a difficult choice to make since another computer language that is not interpreted, but rather is compiled, such as Java or C, could have been selected. Although Python is an interpreted language (Farooq, Khan, Ahmad, Islam, & Abid, 2014), Python has well documented and widespread support for scientific computing via the plethora of free or open source modules available, such as (but not limited to) NumPy for numerical computing, and the Graph-Tool and Network-X graph analysis frameworks for network analysis and visualization. The existence of many open source modules makes Python a flexible, quick to develop, easy to use language for scientific computing, prototyping, and rapid development (Orchard & Rice, 2014), in part because much of the code is already written for you in the form of freely available modules and frameworks.

Another assumption made in this study was that the instruments used to gather elapsed runtime (in seconds) and memory consumption (in megabytes) statistics in Python, were implemented well enough (possibly at the operating system kernel level) to generate reliable and valid results. These instruments are discussed in more detail in Section 2 of this study.

A final assumption made in this study was that the pseudo random number generator (PRNG) available in Python (and by extension, supported by Mac OS which is the operating system that ran the Python development environment on the Apple hardware used in this doctoral study), generated random numbers that were random enough for this study. Generating true random numbers on computers may be challenging (Nilsen, 2007; Thomas, Luk, Leong, & Villasenor, 2007), but testing the randomness of the Python random number generator running on the targeted Apple Mac hardware was beyond the scope of this study. It is assumed that the PRNG provided by Python, on the targeted Apple MacBook Air laptop hardware, generated random numbers which were truly random enough to not have negatively impacted the results of this study.

### **Limitations**

There were some noteworthy limitations in this study. According to Horga, Kaur, and Peterson (2014), limitations in experimental studies are shortcomings that may reduce the validity and the reproducibility of a study's findings (p. 4). Sometimes these limitations are beyond the control of the researcher, and other times they are self-imposed (pp. 3-4).

Much modern Internet software development depends on free or open source software, in part because it is cost effective (Zhang, Anzalone, Faria, & Pearce, 2013). This study specifically compared pathfinding algorithms supported by two popular free or open source graph analysis software frameworks (a) Graph-Tool, and (b) Network-X. Therefore, the first limitation with this study was a self-imposed limit to only compare pathfinding algorithms supported by two popular graph analysis frameworks, not to

compare all possible graph analysis frameworks that currently exist. Next, from a positive social change perspective, I deemed it more socially beneficial to the wider Internet software engineering community to compare pathfinding algorithm implementations already available in free or open source graph analysis frameworks, rather than to write pathfinding algorithm implementations myself (which I assumed would likely generate less world-wide social interest).

Each computer language (e.g., Java, Python) and software framework has pros and cons (Orchard & Rice, 2014). Some may be better implemented than others. This is simply a reality of professional software development. Another limitation with this study was that it did not delve into the reasons why one graph analysis framework was better than the other (although that could be a topic for further research). Instead, it measured the pathfinding algorithm performance of each graph analysis package and algorithm tested, and then statistically analyzed the outputs to answer the research questions and hypotheses. Thus, at a high level, this study has a self-imposed limit to benchmark several pathfinding algorithms commonly supported by two popular graph analysis frameworks, not to write and compare pathfinding algorithms I implemented myself. I felt it would be impossible to implement the most efficient pathfinding algorithms myself in Python because such implementations, if attempted, could suffer biased runtime performance, which therefore would have reduced the validity of this study. Furthermore, more people use the aforementioned free or open source graph analysis frameworks compared in this study, than would ever use pathfinding algorithm code written

specifically by me (and I have never contributed any source code to any open source project). Finally, I am relatively new to the Python computer language.

### **Delimitations**

There were several delimitations in this study. Delen, Kuzey, and Uyar (2013) suggested that some delimitations may restrict the ability to use or follow certain research approaches, but that these restrictions are sometimes made by choice. Ionel-Alin and Emil (2013) suggested that delimitations are self-imposed boundaries incurred, in part, by the reality that resources and capacities are limited, and that exceeding those self-imposed limits may cause research challenges that could compromise results.

Although each computer language has its own characteristics (Farooq et al., 2014), the first delimitation with this study was that it does not compare algorithmic pathfinding performance between programs written in different computer languages. This study uses the same computer language for implementation, Python, for consistency. Similarly, this study also did not compare pathfinding algorithm performance across different hardware brands or vendors, nor between different computer operating systems (OS).

A second delimitation was with the variables used in this study. Regarding map complexity, a deliberate choice was made to test only two categories of maps: (a) small and highly rewired; and (b) large with less random connectivity rewiring. This choice was deliberately made to maintain a low number of factorial treatment groups.

Additionally, while both elapsed time and memory consumption are relevant dependent variables in my study, this study did not measure the impact of path finding

algorithms upon external storage (e.g., solid state drive, and hard drive utilization). Some data structures, like B-trees, are designed to work well on external storage, and are used (either directly, or derivations thereof) in the Linux operating system today (Rodeh, Bacik, & Mason, 2013). Comparative benchmarking of the effect of pathfinding algorithms upon *external* storage devices was beyond the scope of this research (but could be a topic of further research).

Finally, it was deemed beyond the scope of this study to compare and analyze parallel computing algorithms. It was also beyond the scope of this study to directly, utilize the computational capability of graphics processing units (GPUs) to assist the central processing unit (CPU) with pathfinding algorithm computations. While these topics are interesting, and may indeed be worthy of further research, they were beyond the scope of this study.

### **Significance of the Study**

#### **Contribution to Information Technology Practice**

Software developers face challenges comparing algorithm analyses from disparate authors, which may impede the selection of appropriate algorithms. Some of these challenges include (a) authors might analyze only one algorithm; (b) authors may use incompatible comparison metrics; (c) samples used in one analysis might not relate to samples used in other analyses; (d) differences in computer hardware may yield different results; (e) differences between computer languages may yield different results; and (f) some authors may implement their own pathfinding algorithms, while other authors may instead use pathfinding algorithms already implemented (by someone else) in free, open

source, or proprietary software frameworks. These differences make it difficult to quantitatively compare algorithm analyses published by disparate researchers. In contrast, the results of this study may provide software engineers with empirical information related to pathfinding algorithm performance, by providing a single-source reference that compares several pathfinding algorithms and graph analysis frameworks at once, using the same computer language, using the same metrics, using comparable samples, in a clinical experimental setting, all implemented and tested on the same computer hardware. This single-source compilation of research results is intended for applied software developers who need help selecting appropriate pathfinding algorithms for the pathfinding computer software they write.

### **Implications for Social Change**

Contributions to positive social change from efficient pathfinding algorithms are wide-ranging: from saving lives to saving money (sometimes both). Some examples of the positive social benefits derived from efficient pathfinding algorithms include (a) fast robotic debris cleanup of airport runways to prevent fatal accidents during takeoff and landing (Öztürk & Kuzucuoğlu, 2016); (b) bounded-cost optimization of business expenses (Stern et al., 2014); (c) search and rescue missions in unmapped terrain (Liu & Lyons, 2015); and (d) terrorist social network analysis for the identification and apprehension of terror suspects and perpetrators (McBride & Hewitt, 2013). The last example is particularly important given the recent terrorist attacks that occurred in (a) Manchester, UK, concert arena bombing on May 22, 2017; (b) St. Petersburg, Russia, metro train station suicide bombing on April 4, 2017; (c) Istanbul, Turkey, nightclub



shooting on January 1, 2017; (d) Orlando, FL, nightclub shooting on June 12, 2016; (e) Brussels, Belgium, airport and rail station bombings on March 22, 2016; (f) San Bernardino, CA, shooting on December 2, 2015; (g) Paris, France, shootings and Bataclan theatre bombing on November 13, 2015; and (h) the Charlie Hebdo shooting in Paris, France on January 7, 2015; to name just a few recent examples whose perpetrators were suspected to be involved in terrorist social networks. By combining pathfinding algorithms with complex network analysis and information technology, links between terror suspects might be detected before deadly attacks occur, giving law enforcement the chance to apprehend the terrorists, thus preventing loss of life and thereby contributing to positive social change.

### **A Review of the Professional and Academic Literature**

This quantitative experimental study examined the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and memory consumption, in order to help software engineers, select appropriate pathfinding algorithms for resource-constrained software agents running in network dead zones or GPS-denied environments. The research question for this study addressed the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. The three independent variables are (a) pathfinding algorithms, (b) graph analysis frameworks and (c) map complexity. The two dependent variables are (a) elapsed time, and (b) computer memory consumption. The null hypothesis ( $H_0$ ) postulated there was no relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory

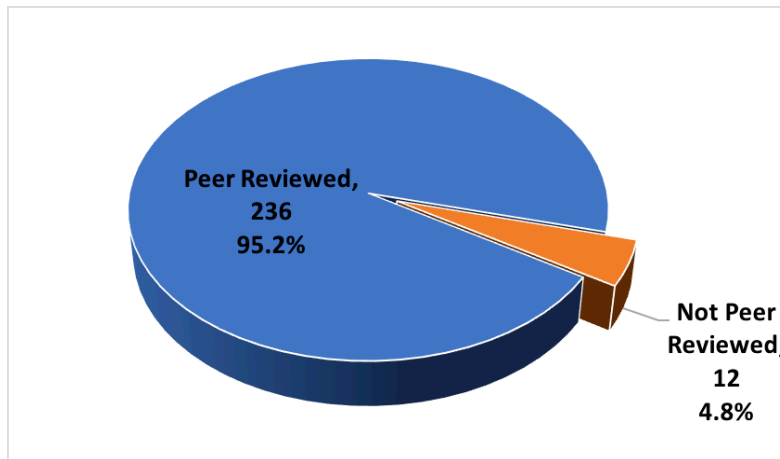
consumption. The alternative hypothesis ( $H_a$ ) postulated there was a relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption.

A hallmark of efficient computer algorithms is the ability to complete tasks while using the minimum computational resources (e.g., memory and CPU), in the minimum amount of elapsed time (Becton & Wang, 2015; Thakur & Guttman, 2016). In economic terms, finding the shortest, most efficient routes between entities of interests, such as (but not limited to) cities, cars, and people, has positive utility value. Knowledge gained from this study may be used by software engineers to write more efficient CPU, memory, and time-efficient pathfinding software. Although algorithmic pathfinding could be considered a mature field, in reality new and faster hardware will cause major changes in computer-assisted navigation, particularly in the area of augmented reality (Alfoor, Sunar, & Kolivand, 2015, p. 9). Therefore, although algorithmic pathfinding has a long history of scholarly research, next generation hardware, big data, complex software, and ever rising end-user expectations suggest increased future demand for more efficient algorithmic pathfinding software.

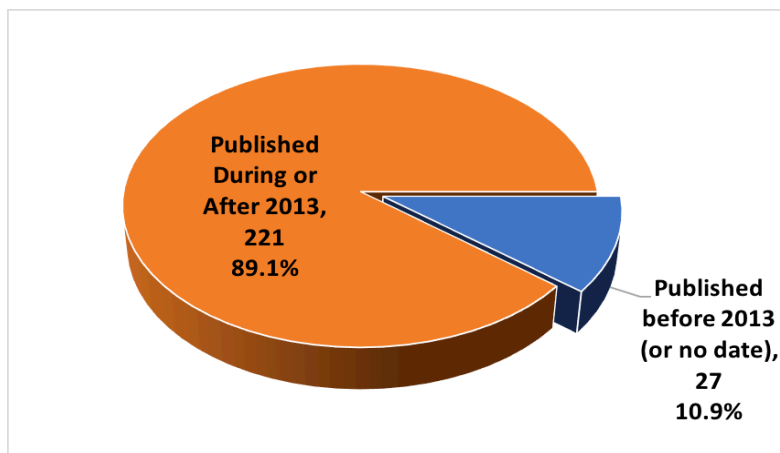
This literature review consists of 10 categories. The first involves an examination of the theoretical framework used in this study. The second involves a discussion of modern applications of social network theory. The third section discusses a rival theory to the selected theoretical framework. The fourth, fifth, and sixth categories review the independent variables used in this study (pathfinding algorithms, graph analysis frameworks, and map complexity, respectively). The seventh and eighth categories

review the two dependent variables (elapsed time, and computer memory consumption, respectively). The ninth category reviews implementation concerns related to computer languages. The tenth (last) category ends this section with a review of literature related to modern applications of algorithmic pathfinding.

A review of current literature to provide a framework and basis for this study was conducted, upon which gaps were identified in the literature that showcased the need for further empirical research, particularly for software developers actively writing modern pathfinding software. Peer-reviewed material was sourced from many online research databases including Academic Search Complete, Association for Computing Machinery (ACM), EBSCOhost, Elsevier, Emerald, Google Scholar, Institute of Electrical and Electronics Engineers (IEEE), ProQuest, SAGE, and ScienceDirect. Search terms included: *social network theory, social network analysis, complex networks, random network, small-world network, scale free network, algorithm performance, performance benchmarks, shortest path algorithm, graph theory, network theory, Dijkstra's algorithm, breadth first search, depth first search, Bellman-Ford algorithm, A\* (pronounced "A star") algorithm, memory consumption, elapsed time calculation, Java, Python, artificial intelligence, transport networks, epidemiological networks, terror networks, and criminal networks*. Only English-language or English-translated papers, articles, journals or books were used for all source material. A key word search for relevant literature for this literature review yielded 248 references, of which 236 (95.2%) were from peer-reviewed sources, and 221 (89.1%) were published within the last five years (2013 through 2017). A total of 113 (45.6%) of the references were used in the literature review.



*Figure 2.* References by peer review status.

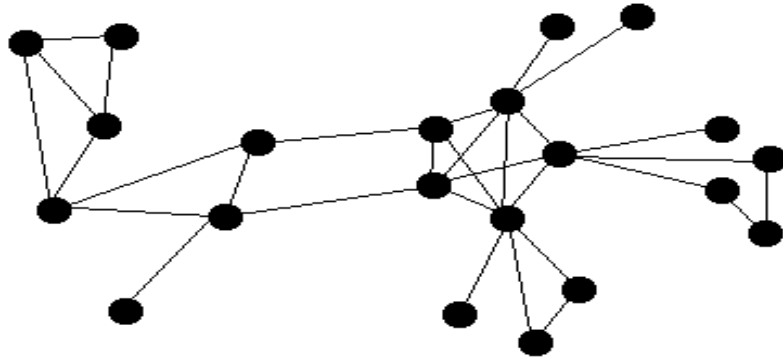


*Figure 3.* References by year of publication.

### **Theoretical Framework: Social Network Theory**

Social network theory grounded my study. The grand premise of social network theory is that patterns of interaction among nodes in a network graph (i.e., the people or objects abstracted in a graph) are the building blocks of networks (Erikson, 2013; Krause, Croft, & James, 2007; Merchant, 2012). Networks are frequently drawn as 2D mathematical graphs, with nodes (vertices) represented by circles or points, and

connections between the nodes represented as lines (arcs, edges), as discussed in Koujaku, Takigawa, Kudo, and Imai (2016).



*Figure 4.* An example graph.

Social network theory has roots in graph theory, which itself has its origin in the early 18th century work of Swiss mathematician Leonard Euler (Albert & Barabási, 2002, p. 9). Due to the lack of computers and large datasets in Euler's time, early graph theory focused on small, mostly regular graphs that could be hand drawn. In 1954, social networks were first mentioned in a scientific context by the English social anthropologist John A. Barnes, in his anthropological research of the small fishing village of Bremnes, in western Norway (Barnes, 1954; Wasserman & Faust, 1994, p. 10). In the Barnes study, concepts were first mentioned that are common in social network analyses today, such as social network stratification, network analysis, community membership (cliques), and the importance of the shortest path, such as the minimum number of connections between any two members of a given population (Barnes, 1954, pp. 45-46). Today, modern social network analysis is based on a structuralist interpretation of the foundational theoretical

works (Barnes, 1954; De Sola Pool & Kochen, 1979; and Korte & Milgram, 1970), combined with a neo-Kantian identification of *a priori* categories of relational types and patterns, making it flexible enough to operate outside the constraints of purely historical context or cultural settings (Erikson, 2013, p. 219). This implies social network theory may be applicable to problem domains beyond anthropology.

Small-world networks are a specific type of complex network that can be analyzed with social network theory using graph theoretic methods. In the late 1950s, social researchers Ithiel de Sola Pool and Manfred Kochen circulated an early manuscript describing the importance of influence, social contacts and social networks, calling it the *small-world* phenomenon. This manuscript was later formally published in 1979 (De Sola Pool & Kochen, 1979). In the mid 1960s, social psychologist Stanley Milgram read the manuscript, was intrigued by De Sola Pool and Kochen's concept of small-world networks, and began researching topic of human communication paths. While there were some theoretical models at that time that described *small-world* networks such as the aforementioned manuscript of De Sola Pool and Kochen, and the early work of Barnes (1954), there was little empirical evidence to describe the shortest path lengths connecting hypothetical friends and acquaintances in actual social networks (González-Bailón, 2013). Milgram ran his, now famous, small-world experiments and published several results in 1967, 1969, and 1970 (Korte & Milgram, 1970). Milgram's small-world experiments examined the average path lengths between random people in the United States. He discovered that random pairs of people in the U.S. were separated, on average, by six intermediary persons within their combined network of friends and

acquaintances (Korte & Milgram, 1970, p. 101). This surprisingly short path length later came to be known by others as the "six degrees of separation" (Kleinberg, 2000; Zhang & Wang, 2013). As described in Opsahl, Vernet, Alnuaimi, and George (2017), and in Watts and Strogatz (1998), the small-world network model was mathematically formalized by Watts and Strogatz (1998), whose work provided a framework and methodology that future researchers could use to detect small-world network characteristics in their networks of interest. While the early empirical research on small-world networks originated from Milgram's efforts of the mid-1960s, since then, as discussed by Erikson (2013), González-Bailón (2013), and Opsahl et al. (2017), small-world networks, and social network analysis more generally, have become very active areas of cross disciplinary science research.

Milgram's seminal small-world work is considered the basis of modern social network theory (Wang, 2015), and has been referenced in many subsequent peer-reviewed papers in diverse domains, such as computer science (Balaguru, Nallathamby, & Robin, 2015), economics (Wang, 2015), history (Mills et al., 2013), industrial supply chains (Capaldo & Giannoccaro, 2015), and social networks (Rezvanian & Meybodi, 2015), to name a few. Corporations that focus on social networking, like Facebook and Twitter, have also benefited from social network theory (Johnston, Tanner, Lalla, & Kawalski, 2013). This spread of social network theory across a broad spectrum of disciplines supports earlier claims by Erikson (2013), and González-Bailón (2013), that it has cross disciplinary appeal.

Social networks can be mathematically represented as graphs. Graphs can represent many different types of relationships (Kepner et al., 2015, p. 2455). In this doctoral study, Milgram's small-world social network theory is applied to gain an understanding of the relationship between the dependent and independent variables, as they relate to the shortest path problem in graphs, which today may be larger (in aggregate node and edge counts), and exhibit higher graph density (i.e., a high edge count to node ratio), than the personal networks studied and reported in the aforementioned seminal work of Korte and Milgram (1970).

Brooks, Hogan, Ellison, Lamp, and Vitak (2014) indicated that the average degree of separation between Facebook users is 3.74 persons (p. 12) which is much lower than Milgram's often cited six degrees of separation (Largeron, Mougel, Rabbany, & Zaïane, 2015, p. 6). In graph theory terminology, this may indicate that the vertex degree (the number of incoming and outgoing edges, per vertex) -- which in social network graphs represent connections with other people (and in 2D grid maps, can represent connected objects) -- might be larger now than the vertex degree for interconnected people during 1965 through 1970, when Milgram conducted and published his small-world social network experiments. This may be because if each intermediary person today has more connections to begin with (thanks, in part, due to technology, and social networking products like Facebook and LinkedIn) than people had in the mid-1960s, then the average path length between two random people may be shorter now, indicating potentially denser graph networks. The impact of dense graphs versus sparse graphs on algorithmic shortest path computation may be measurable.





On the left is an example of a sparse graph (the edge count is low). The graph on the right, by contrast, is denser (the edge count is noticeably greater).

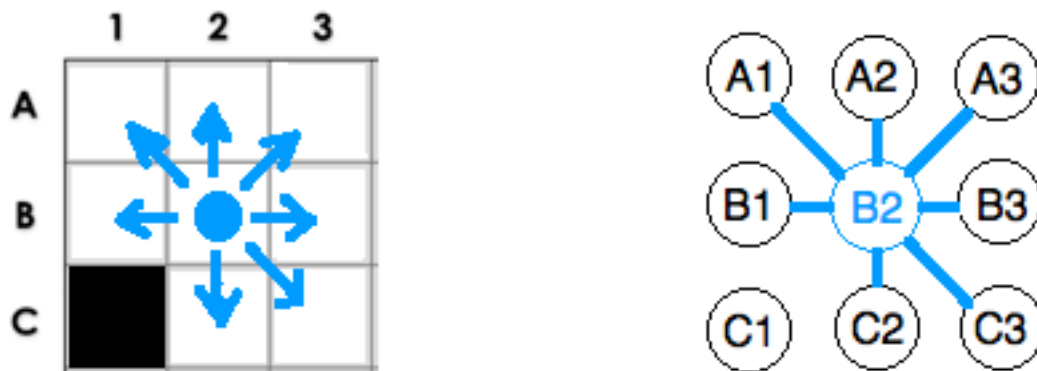
*Figure 5.* Sparse and dense graphs visually compared.

The impact of dense vs. sparse graphs on pathfinding, along with applied social network theory and Milgram's small-world social networks, drove this research, to help determine the nature of the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. More specifically, social network theory drives this research for two reasons. First, if the shortest path between two people today is less than the six degrees of separation noted by Milgram (Brooks, Hogan, Ellison, Lamp, & Vitak, 2014, p. 12), and if this is due to higher graph density, then measuring the relationship between graph density (i.e., graphs with a higher ratio of open connections per node than blocked connections) and pathfinding algorithm performance, particularly for algorithms designed to traverse sparse graphs but applied to dense graphs (and vice versa), may be worthy of further research. Xu, Liu, Li, & Ren (2014) suggested that the shortest path between nodes rapidly increases as the average vertex degree of the network decreases (p. 11), which may be interpreted as the average path lengths between two random nodes are generally longer in sparse networks, and shorter in dense networks. This may be measurable.

Second, today's software agents without native pathfinding capabilities may rely on other resources to find shortest paths, if they are Wi-Fi or radio frequency (RF) enabled, such as GPS, and/or network-enabled Web services (Huang, Zhang, Yuan, Zhang, & Ma, 2016). But, for software agents wholly dependent on pathfinding Web services for guidance, if or when GPS and network-enabled Web services are not available, pathfinding then may become an intractable problem. One possible contingency is adding onboard algorithmic pathfinding capability (Dean, 2013) to the software agent. My research compared pathfinding algorithms for software agents denied the benefits of GPS and network Web services, upon Watts and Strogatz (1998) style small-world graphs, which may be visually represented as 2D grid maps (and mathematically represented as 2D adjacency matrices).

Milgram's small-world theory (Korte & Milgram, 1970) explained my choice of independent variables, my dependent variables, and my aforementioned hypotheses, because in this study the independent variables are intentionally manipulated to see what impact (if any) this has on the dependent variables. More specifically, network graphs were abstracted and represented by random computer generated 2D grid maps. The independent variable "map complexity" controlled the overall maximum size of each graph to be searched by each of the pathfinding algorithms compared, per graph analysis framework, and it drove the network connectivity structure (i.e., a structured "grid" network vs. random network) by controlling the percentage of random connections made by each node in the 2D grid maps. The interactions between pathfinding algorithms, graph analysis frameworks, and map complexity, was measurable, and impacted the

dependent variables: elapsed time, and computer memory consumption. These relationships were measured and statistically analyzed in this study.



High density (low occlusion) 2D grid map on left, with corresponding graph representation on right. Note how only grid cell (vertex) C1 is not reachable from grid cell (vertex) B2.

*Figure 6.* A high density (i.e., low occlusion ratio, few obstructions) grid map and corresponding graph.

A high-density graph (also known as a "dense graph") means there are fewer obstructions (i.e., low occlusion ratio, fewer potential blockages) between adjacent vertices, which generally yields more possible connections per graph node, hence the high density. By contrast, a low-density graph (also known as a "sparse" graph) means there are fewer connections (i.e., high occlusion ratio, more potential blockages) per graph node. To visually see the impact of differing types of node connectivity in 2D graphs, compare and contrast the high-density (i.e., less obstructed) grid map images in Figure 6, against the low-density (i.e., sparsely connected, highly obstructed) grid maps in Figure 7.



Low density (high occlusion) grid map on left, with corresponding graph representation on right. Note how only grid cells (vertices) B1 and C2 are reachable from B2.

*Figure 7.* A low density (i.e., high occlusion ratio, many obstructions) grid map and its corresponding graph.

Visualizing graphs and node connections is important to understanding the underlying data represented in those graphs (Dawson, Munzner, & McGrenere, 2015; Wasserman & Faust, 1994). In a graph, interaction among node pairs is represented by a line connecting two nodes. This implies some form of communication or connectivity, or the possibility thereof, as shown in Figure 7. The existence of lines connecting two or more nodes in a graph provide the possibility for interaction, and, in some complex networks, present the possibility for nodes to try to influence each other (Chewning & Doerfel, 2013, p. 42). The study of complex networks is an active area of heavy cross-disciplinary research (Erikson, 2013). Merchant (2012) stated that relational networks create a sense of belonging and that the study of such networks allows one to trace the contours of existing divisions and conflicts between network entities (p. 4). These divisions may be wide ranging and varied, as are the modern networks one can join: (a) technological, (b) political, (c) economic, (d) class-based, (e) social, (f) epidemiological,

and (g) distance-oriented, to name a few. Krause, Croft and James (2007) suggested that experimental addition or removal of nodes in graphs during network analysis can have profound effects on the resulting descriptive statistics that describe the underlying network and entities studied (pp. 17, 27). Some factors related to social network theory and real world networks include (a) algorithmic pathfinding, (b) small-world networks, and (c) the path length, as mentioned by Lamprecht et al. (2015, pp. 3-4), all of which is discussed in detail, later in this quantitative study.

Social network theory provides a framework for understanding organizational structure, and how the entities modeled in a graph (e.g., people, robots, communities, corporations, nation-states, planetary bodies, galaxies, etc.) relate to other entities within their networks (Chewning & Doerfel, 2013, p. 41). In a graph, nodes may represent object that interact with each other (Albert & Barabási, 2002). According to Barnes-Mauthe, Gray, Arita, Lynham, and Leung (2015), from a resource acquisition perspective, one's position in a social network (i.e., proximity to other nodes and types of connections) determines the nature and extent of access to critical information and resources within that network (p. 3). The possibility of interaction between nodes in complex networks, and the desire to efficiently model such nodal interactions (e.g., path length calculations, graph traversal costs) makes social network theory relevant to this algorithm study.

Unlike some mathematical theorems which date back to ancient antiquity (e.g., the Pythagorean Theorem), formal social network theory was discovered between 1952-1953, during anthropologist John Barnes' study of the small fishing village of Bremnes,

Norway, later published in 1954, making social network theory a comparatively modern discovery (Barnes, 1954). In his seminal work (Barnes, 1954), Barnes did not specifically mention use of computational resources, and given the age of the publication (1954), what we know of the general state of computational technology at the time, and his field research location (a small, rural, Norwegian fishing village), together these factors may explain why Barnes did not specifically mention use of computational data or computational social science techniques in his seminal work. The population of Brednes, Norway at the time was just 4,600 people (Barnes, 1954, p. 40). From today's perspective of big data analytics and data mining, the Barnes's dataset from 1954 seems small (Balaguru, Nallathamby, & Robin, 2015). For social scientists engaged in qualitative research, interviewing and analyzing a few thousand people and their social networks, in person, may seem like a time-consuming endeavor, but as Barnes mentioned in his work, he spent two years (between 1952-1953) in the field, gathering his data through in-person interviews and observation (p. 39). By contrast, today's online social networks, such as Facebook with its over 1 billion active users alone (Balaguru et al, 2015; Brooks, Hogan, Ellison, Lampe, & Vitak, 2014), is several orders of magnitude larger than the dataset from the original Barnes study, and there are application programmer interfaces (APIs) available to help mine Facebook data (Brooks et al, 2014). These are modern computational tools that Barnes did not have back in 1954.

To better understand social network theory and how it applies to the research problem of pathfinding in complex networks for software agents running in network dead zones and GPS-denied environments, deeper research into the following constituent

aspects of social network theory literature must be addressed (a) graph theory, (b) random network theory, (c) small-world network theory, and (d) scale-free network theory; each of which is a constituent of social network theory, and is discussed in subsequent sections of this literature review.

Graph theory is a well-established branch of mathematics, and has influenced social network theory (Barnes-Mauthe et al., 2015; Wasserman & Faust, 1994). Graph theory originated in the 18th century work of mathematician Leonard Euler (Albert & Barabási, 2002, p. 9), and has been used in computer science since the mid-20th century (Phillips, Schwanghart, & Heckmann, 2015, p. 148). The findings of Phillips et al. (2015), suggested that graph theory was well suited to network analysis, and they considered graph theory to be a powerful tool for scientists (2015, p. 148). Due to the lack of computers and large, easily accessible datasets in Euler's time, early graph theory focused on small, highly regular graphs that could be hand drawn (Albert & Barabási, 2002). Additionally, according to Malliaros and Vazirgiannis (2013), graphs are an efficient way to represent a network, and are now a dominant structure used for analyses in many multidisciplinary problem domains, including (but not limited to) computer science, biology, neuroscience, physics, and sociology. Furthermore, in the 20th century, graph theory became more algorithmic and statistical, thanks in part to the development of computers, programming languages, and graph analysis software (Albert & Barabási, 2002, p. 9). This permitted graph theory to be more easily utilized across many disparate problem domains. In a study of complex networks, Mears and Pollard (2016) concluded that graph theory is flexible, enabling researchers to measure connectivity of individual

nodes within larger networks (p. 601). Mears and Pollard combined biology with computer science to solve neurological complex network problems. They advocated advancement of graph theoretic knowledge by way of longitudinal studies that identified possible temporal correlations between changes in network topology, or nodal characteristics, and the development of pathological conditions within the broader network graph (p. 602). They also acknowledged that variable results may occur due to differences in experimental design, subject cohort selection, sample size, network construction and analysis techniques (p. 602), but to be fair, these precautions (e.g., research design, subject selection, sample size, etc.) could generally apply to much experimental research anyway (Donaldson, Qiu, & Luo, 2013). Although this quantitative experimental study is not longitudinal, that could be an avenue for further research.

While analyzing and studying large networks in computer science with graph theory may be useful, in a study by Afuah (2013) the author's findings suggested that focusing primarily on network size as a sole determinant of a network's "value" would cause biased estimates of that network's worth, make research difficult to interpret, and is "tantamount" to the omission of important variables in network analysis (p. 271). Afuah recommended that network researchers also consider network structure (i.e., the layout of nodes and edges within a network graph), and network conduct (i.e., the behavior of nodes within the network graph) (p. 258). According to Afuah, only by considering all three variables (size, structure, and conduct) of a network graph, not just network size, may researchers reduce the likelihood of overlooking important information when conducting complex network analyses. Similarly, the work of Newman, Watts, and



Strogatz (2002), also discussed the importance of considering both graph structure, and graph size, when conducting complex network analysis. Taking network structure into consideration therefore leads to the topic of complex networks.

Complex networks abound in nature, as well as in the modern technology world (Mears & Pollard, 2016, p. 590). Some pertinent issues related to the study of complex networks include (a) estimating the maximum velocity objects may travel in their networks, (b) identifying the impact (if any) of network structure on object velocity, (c) identification of the shortest path between random source and destination nodes within a network, and (d) methods for calculation and quantification of shortest paths (Majeed & Rahman, 2015). What follows next is a discussion of current research on complex networks, the importance of social network theory as it relates to complex networks, several proposed social network-oriented answers to the above questions, and a discussion of several gaps in the literature.

Complex networks are not unique to computer science or social science. At a fundamental level these networks (whether social or technological) are comprised of entities called nodes, possibly connected to other nodes through one or more commonly shared characteristics (Barnes-Mauthe et al., 2015; Majeed & Rahman, 2015, p. 20). They occur in both computational and non-computational situations. Some example of complex networks include disease transmission networks (e.g., viruses, outbreaks of food-borne illnesses), the World Wide Web (WWW), the electrical power grid, social networks (e.g., Facebook, LinkedIn), financial networks (e.g., online banking and investing), volunteer networks, terrorist networks (e.g., Al Qaeda, Boko Haram, ISIS),

airline and highway transportation networks, political parties (e.g., Republican, Democrat), geopolitical networks (e.g., EU, NAFTA, NATO, United Nations), and even biological predator-prey (i.e., food chain) networks (Newman, Watts, & Strogatz, 2002; Traag, Drings, & Van Dooren, 2013; Watts & Strogatz, 1998). There are four main types of complex networks (a) regular (i.e., grid), (b) random, (c) small-world, and (d) scale-free. Each is discussed next.

Early research on random graphs was performed by Erdős and Rényi (1961). The classic Erdős and Rényi (ER) model of random graphs was their early attempt to explain the behavior of complex networks. The ER model of random graphs defined a random graph as having  $N$  random nodes, connected by  $M$  random edges (recall, each edge connects only two nodes). One way to generate a random graph is to start with a simple 2D rectilinear grid map of  $N$  by  $M$  points (also known as a regular, grid, or lattice network), as shown on the left in Figure 8. Note that researchers can also use circular regular graphs, as shown on the right in Figure 8.

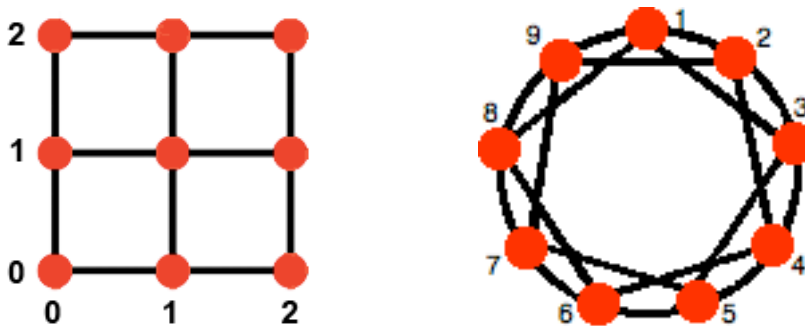
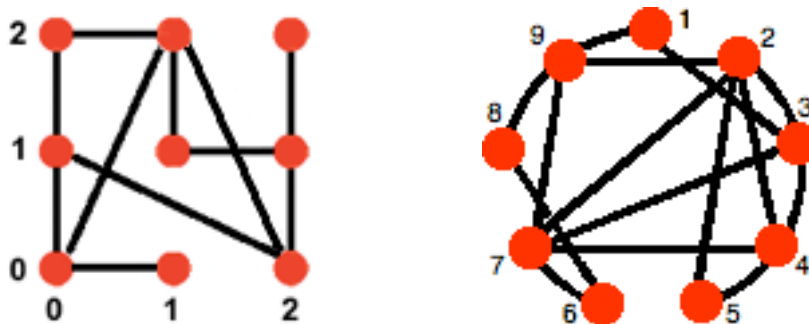


Figure 8. Two regular 2D lattice networks: grid (left) and circular (right).

Next, the probability  $P$ , with  $0 \leq P \leq 1$ , where  $P = 0$  means 0% randomization (i.e., the network is a pure regular network), and  $P = 1$  means each link has a 100%

chance of being randomly connected to another node (i.e., the network is the opposite of a regular network, nodal connections do not follow a structured "grid" or "lattice" pattern), determines the amount of randomization applied to the regular grid map. In Figure 9, the original example graphs have been semi-randomized (with  $P \approx 0.5$ ). Note how randomization of edge placement in a grid network changes the network, and therefore may change shortest paths among existing nodes within each network (Watts & Strogatz, 1998). For example, in the circular lattice network of Figure 8 (on the right), the shortest path between nodes 7 and 2, follows the path  $7 \rightarrow 9 \rightarrow 2$ , yielding a shortest path length of 2 between nodes 7 and 2. By contrast, in the circular semi-random network in Figure 9 (to the right), randomization has changed the shortest path between nodes 7 and 2. Now the shortest path between nodes 7 and 2 is simply  $7 \rightarrow 2$ , with a shortest path length of 1.



*Figure 9.* Two semi-random 2D lattice networks: grid (left) and circular (right).

The study of random graph theory was useful in that it provided a foundation for subsequent network research. But one weakness with the Erdős and Rényi (1961) paper was that although it did mention the importance of average path length, it did not mention which algorithms Erdős and Rényi used to calculate shortest path lengths in their test

networks, nor did they mention the performance of their random network generation algorithm, nor shortest path length statistics. My study may help fill those gaps in the literature.

The main goal of the ER model of random graph theory was to determine at what probability,  $P$ , would a desired property of a graph most likely arise (Albert & Barabási, 2002, p. 10). The greatest discovery of the ER random graph model is that many important properties of random graphs appear quite suddenly. That is, at a given probability  $P$  either most random graphs have some property  $Q$ , or most random graphs do not have that property. Two characteristics of regular (non-randomized) grid maps is that they feature high clustering (i.e., neighboring nodes tend to share many of the same connections), and high average path lengths (i.e., there are no short cuts from one edge of the grid to the other side) (Watts & Strogatz, 1998). By contrast, highly random graphs (i.e., graphs where probability  $P$  lies closer to 1 than to 0) are characterized by short path lengths due to the randomization effect on edge placement between nodes, and low clustering (Mears & Pollard, 2016, pp. 590-591). Furthermore, Newman, Watts, and Strogatz (2002) suggested that random graphs are well-studied in the discipline of discrete mathematics, with many published articles devoted to describing the properties of random graphs (p. 2567). Deficiencies with both the Newman, Watts, and Strogatz (2002), and the Mears and Pollard (2016) studies were (a) the lack of source code for analysis, (b) no indication if they used a graph analysis framework instead of implementing their own pathfinding algorithms, (c) no indication of which computer languages were used (if any), and (d) no indication which OS and hardware platforms

were used. These represent gaps in the literature that may be filled by my quantitative experimental study.

A drawback with random graphs is that they do not model certain complex real-world networks very well (Albert & Barabási, 2002; Barabási, 2016). For example, social (friendship) networks are characterized by mostly non-random development; in social networks, most people generally prefer connecting (non-randomly) to friends and acquaintances, not with random strangers. Alternately, regular graphs (e.g., grid maps) where nodes only connect with nearby neighbors as depicted in the 2D regular graph images earlier, do not model all real world complex networks (Kleinberg, 2000). Hence, as discussed by Albert and Barabási (2002), a new graph model was needed to explain some real-world networks.

In 1998, Watts and Strogatz published a paper on small-world networks, building on the earlier work on the six-degrees of separation by Korte and Milgram (1970), and the small-world networks theory postulated by De Sola Pool and Kochen (1979). The quantitative work by Milgram in 1965, 1967 and 1970 demonstrated the existence of the small-world phenomenon, meaning that, in theory, most people can be linked by short chains of acquaintances (Korte & Milgram, 1970). Furthermore, Milgram's experiments also showed that not only do short chains exist between people but that individuals are very good finding these chains by using primarily local information, like querying friends and acquaintances for the desired network knowledge (Fraigniaud & Giakkoupis, 2014, p. 231). On the scale of purely random to purely regular networks, small-world networks reside in the middle (Watts & Strogatz, 1998). Assume,  $P$ , equals the amount of

randomness in a network. Small-world networks are somewhere between pure regular networks (where  $P = 0$ ), and completely random networks (where  $P = 1$ ), that is, for small-world networks,  $0 < P < 1$  (see Watts & Strogatz, 1998). Figure 10 depicts three example networks. As the randomization coefficient,  $P$ , of each network grows, its effect on path length and graph structure becomes more noticeable. Visually it is possible to see how the additional randomization (i.e., an increase in  $P$ ) in the small-world network (center) can substantively shorten the path length from one side of the network to the other, compared to that of the (nonrandom) regular network on the left.

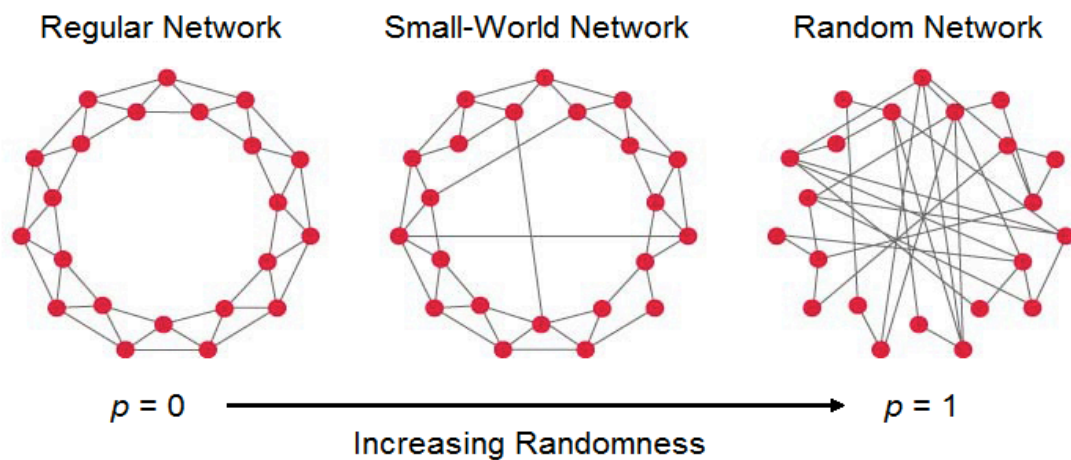


Figure 10. Regular, small-world & random networks (based on Watts & Strogatz, 1998).

Small-world networks are characterized by high clustering (neighbor nodes share most of the same connections), similar to regular (grid) networks, but have shorter path lengths than regular networks because of the potential "short-cut" path provided due to randomization (Watts & Strogatz, 1998). This means getting from one side of the graph to the other can be done in fewer hops in a small-world network than in a regular network, as described by Fraigniaud and Giakkoupis (2014). This has implications when

finding shortest paths in complex networks, as is discussed in more detail later in this literature review.

Small-world networks are useful at modeling (a) the working relationships between actors (i.e., the "six degrees of Kevin Bacon"); (b) the modern power grid; and (c) neural brain networks; among other real world networks (Kleinberg, 2000). Watts & Strogatz (1998) described small-world networks in much detail, but ultimately admitted that small-world networks do not model all real-world networks. Where small-world networks showed promise, however, was in the realm of modeling the behavior of supply chains. According to Hearnshaw and Wilson (2013), the Watts and Strogatz (WS) model can better model real world supply chains than does random networks, or regular (i.e., grid, lattice) networks, because the WS model optimally combines two conflicting goals of network management: (a) minimizing the high transaction costs of long distance connections due to decentralization; and (b) permitting efficient flow transfer throughout a complex network, again due to decentralization (p. 448). But there is an issue that the WS model does not answer, namely, how or why certain network connections form in the first place. According to Hearnshaw and Wilson (2013, p. 449), the WS small-world network model does not answer or predict if there is a preference for certain connections over others during network formation, nor if past network growth affects future network connectivity. One example network not explained by the WS small-world model is the growth of the World-Wide Web (Albert, Jeong, & Barabási, 1999). Yet again, a new graph model was needed. One weakness with the Watts and Strogatz (1998) paper was the lack of publically available source code describing their approach and methodology.

Another weakness was the lack of description about which pathfinding algorithm(s) they used in their software (if any), or how they benchmarked pathfinding algorithm performance. These deficiencies represent gaps in the literature that may be covered by this study.

In 1999, Albert, Jeong, and Barabási published a paper where they describe trying to measure the diameter of the World-Wide Web (WWW). Despite its increasing relevance to the modern world, and the fact that no one institution, entity or country controls it, its uncontrolled growth made it impossible to catalog all vertices (nodes) and edges (links) of the WWW (p. 130), furthermore although they tried, Albert, Jeong, and Barabási had difficulty matching their estimated diameter of the WWW, and its estimated structure, to the small-world network model. Albert, Jeong, and Barabási (1999) discovered that networks like the WWW are characterized by a vast majority of nodes having few connections, sprinkled with a few very highly connected "hub" nodes. This network topology did not correspond to small-world network topology promulgated by Watts and Strogatz (1998), or the earlier random networks of Erdős and Rényi (1961). Albert, Jeong, and Barabási called these new networks "scale-free" networks, because they are inhomogeneous, and connections are not made randomly, but instead are formed based on preferential attachment, with a degree distribution that follows a power-law statistical distribution, not a Poisson distribution like random and small-world networks (Albert & Barabási, 2002, p. 27). Poisson distributions are often depicted as bell curves. Barabási (2016, pp. 120, 123) describes the 2D graphical difference between Poisson and Power-Law degree distributions.



These two distinctly different degree distributions (Poisson vs. Power-Law) originate from distinctly different graphs. Random graphs follow the Poisson distribution, because most nodes have similar numbers of links (Erdős & Rényi, 1961). But scale-free networks follow a Power-Law degree distribution, characterized by most nodes having few links, but a few nodes have disproportionately many links (Albert & Barabási, 2002). Thus, this implies network structure may have implications in algorithmic shortest path calculations.

Researchers Albert, Jeong, and Barabási (1999) determined that two randomly chosen documents on the WWW were, on average, just 19 clicks (links) away from each other (1999, p. 130). Scale-free networks can be used to model traffic networks. In an urban transportation network study by Zou, Wu, Gao, and Xu (2014), they described how urban traffic flows could be modeled as scale-free networks. In this case, traffic flows toward the popular hub nodes due to the drivers' desire to take the shortest path to their destination, which in turn could convert those highly popular hub nodes into potential bottlenecks (i.e., virtual parking lots) under conditions of heavy traffic load, or a random node attack (i.e., a critical weather event), or intentional node attack (i.e., terrorism). One weakness with the Albert, Jeong, and Barabási (1999) study is that although they specifically mention creation and use of a software "bot," which crawled the Web, gathering data on URLs and links, from which the researchers derived their calculated diameter of the WWW, I could not find any publically available source code for the Web crawling "bot" for deeper analysis. Another weakness is the age of the study itself, since the Albert, Jeong, and Barabási experiment now is over 16 years old. Today's Internet is

larger and more complex (more links and users) than the Internet of 1999. Furthermore, they did not describe which graph algorithms (if any) they used in their bot software. The Zou, Wu, Gao, and Xu (2014) study showed similar weaknesses in not including source code used in their study (if any). By contrast, Franke and Ivanova (2014) did mention the influence of graph algorithms and frameworks on network traversal elapsed time, and recommended one framework (albeit theirs) for speedy pathfinding over the other frameworks they tested. A final weakness in the Albert, Jeong, and Barabási (1999) paper was the lack of raw statistical data to support their assertions. These deficiencies are gaps in the literature that are covered in my quantitative study, which includes source code, and numeric data.

Regular networks, random networks, small-world networks, and scale-free networks, may be considered further refinements to general graph theory (Barabási, 2016). Each network type provided a means to further understand networks, like social networks or traffic networks. In particular, concepts such as path length, path finding, clustering and vertex degree are relevant to graph theory, random network theory, small-world network theory and scale-free network theory, as they are relevant to modern social network theory (Barabási, 2016; Erikson, 2013), which is discussed next.

There is an issue with social network theory that must be discussed, notably its inconsistent theoretical foundation. According to Erikson (2013) in social network theory there are two major belief systems, relationalism, and formalism. Relationalism is aligned with inductive (i.e., qualitative) reasoning, and focuses on the experiences of the entities (i.e., the graph nodes) in their network, who derive their meaning, significance and

identity by the ever-changing roles they play and transactions which occur within their social milieu (p. 222). That is, it is the actions performed by these entities which, *a posteriori*, give form to the social network. So, distance and connectivity between nodes may change as nodes compete within their social milieu. This constant flux, ambiguity, and change in one's position require subjective interpretation for the relationalist (p. 235), with subjective analyses and interpretations leading toward qualitative analysis approaches. By contrast, in formalism, which is more aligned with deductive (i.e., quantitative) reasoning, it is the shape and form of the social network itself which, *a priori*, gives rise to the possibility of social interactions between entities (p. 228). For the formalist, it is the individual's position within the already existing network which generally dictates what actions the individual may take (p. 238). The formalist view that the *a priori* existence of networks drives behavior, not the other way around, was confirmed in a separate study by Chewning and Doerfel (2013) who stated that social network theory assumes the *a priori* existence of networks, without which social network theory would have little utility value (Chewning & Doerfel, 2013, p. 41).

This study, which is quantitative and experimental in design and methodology, aligns more with the aforementioned formalist approach to social network theory, due to reliance on deductive (quantitative), not inductive (qualitative) reasoning, thus best aligns with quantitative analysis and experimental research (Collins & Cooper, 2014). That is, the shape and structure of existing, non-changing networks (i.e., the static 2D maps used by the pathfinding algorithms in my study) were a focus of this research. In this experimental study, the independent variables are intentionally manipulated to infer

causal effects (if any) on the dependent variables. This is a deductive approach (Venkatesh, Brown, & Bala, 2013), which lends itself well to the formalist view to social network theory, because through post-positivist deduction this study intends to measure the effects of different 2D map samples on pathfinding algorithm output behavior, not the other way around (i.e., pathfinding algorithms did not modify any 2D sample maps in this study, but 2D maps may influence the output of algorithmic pathfinding results).

### **Modern Applications of Social Network Theory**

As discussed earlier, social network theory can be used in multiple problem domains, and itself utilizes graph theory. What follows is a discussion of the applications of social network theory (e.g., terrorist network detection), and several gaps in the literature are identified, which reinforce the need for this quantitative experimental study.

Community detection is a problem partially solved with algorithmic pathfinding. According to a social network theory study by Harenberg, Bello, Gjeltema, Ranshous, Harlalka, Seay, ... and Samatova (2014), community detection is a widely researched problem domain in the field of data analytics (p. 427). In their research, Harenberg et al, determined that it is possible to empirically compare community detection algorithms, both in terms of broad "goodness of fit" characteristics, as well as with quantitative performance metrics, but that these metrics are not equivalent. An algorithm that identifies social network communities "well," may exhibit poor runtime performance, and vice-versa (p. 438). In another study of social networks and community detection by Traag, Krings, and Van Dooren (2013), they analyzed a network based on votes from members of parliament (MEPs) of the European Parliament (EP) of the European Union

(EU). They used simple, unweighted Erdős and Rényi (ER) model random graphs. Their results indicated that the EP (i.e., the network of MEPs who themselves represent constituents of EU member nations) has become increasingly ideologically divided, with nationality playing little to no role (p. 6). From a graph theory perspective, one could view this as a situation where within the wider EP network comprised of MEPs, the ideologically oriented MEPs have formed communities which share similar viewpoints, and thus share shorter path lengths within their communities than outside their communities. One deficiency with the Traag, Krings and Van Dooren (2013) study was its heavy reliance on Erdős and Rényi (1961) networks, to the exclusion of small-world networks and scale-free networks. A deficiency with the Harenberg et al. (2014) study is that they inconsistently used three different computer languages (C++, Java, Python) in their algorithm implementations. Not surprisingly, the implementation language had an impact on run-time performance, as admitted by the authors (p. 437). This is a research gap that may be filled by this study which, in the interest of consistency, used one computer language, so as to avoid making incongruent "apples to oranges" comparisons between selected pathfinding algorithms implemented with different computer languages. By contrast, in a complex network study by Zhang and Wang (2013), they confirmed the Watts and Strogatz (1998) findings that small-world networks exhibit large clustering coefficients and short characteristic path lengths (p. 971). So, in the Traag et al. (2013) study, because the authors specifically used only ER style random networks, it is unknown whether use of additional Watts and Strogatz (WS) style small-world networks would have changed the results of the Traag et al. (2013) study. Additionally, the Traag

et al. (2013) study did not include computer source code, nor mention use of any graph analysis frameworks by their study. These gaps in the literature were filled by my quantitative study which includes source code and describes, in detail, the usage of several graph analysis frameworks.

Shorter paths may represent better-connected networks. Yang, Poon, Liu, and Bagchi-Sen (2015) performed a study of geographical information system (GIS) and complex networks from 1988 to 2013 (p. 534), with source data that originated from the United Nations commodity trade database. They confirmed the importance of measuring the shortest paths between network nodes of interest, and that the shorter the path length between partners, the better-connected the network (p. 536). Researchers Rohden, Witthaut, Timme, and Meyer-Ortmanns (2017) also described using shortest path calculations in power grid networks to seek bottleneck links, in order to increase the power transmission capacity of the identified weak links (p. 6). In a separate study of Internet geolocation techniques by Li, Chen, Guo, Liu, Zhang, Zhang, and Zhang (2013), they mentioned it may be possible to geo-locate devices based on Internet protocol (IP) addresses. So, although Yang, et al. (2015) did mention usage of the open source Gephi graph analysis framework, they did not take the next step and combine their GIS findings with geolocation techniques, such as those described by Li, et al. (2013). Geolocation and/or social network analysis may help detect social network cliques, like terrorist network cells, which is discussed at length in the works of Eiselt and Bhadury (2015) and Medina (2014). In a separate study of social networks by Zaglia (2013), the author's findings indicated that online social networks are "web based services" (p. 217), and that

the presence of the Internet further boosts user participation in virtual communities worldwide (p. 216). A question arises, could Zaglia's (2013) virtual communities work be geo-located in the Li et al. (2013) sense, and can this be combined with the GIS findings discovered by the aforementioned Yang et al. (2015) work? Perhaps it is possible that these three studies could be combined somehow, using techniques from social network analysis, to the broader benefit of society. There are some weaknesses with the studies, however. Both Yang, et al. (2015), and Rohden, et al. (2017) did not mention which computer language was used in their studies. Nor did Yang, et al. (2015) mention why they only used one graph analysis framework. While the Zaglia (2013) study mentioned use of inferential statistics, like the independent *t*-test, to determine if there was a statistically significant difference between the means of different population groups (p. 219), one weakness with that study was it did not show the underlying statistical data used by the authors in their statistical analyses. Lack of supporting data were also demonstrated in the Rohden, et al. (2017) study. These deficiencies represent gaps in the literature addressed by this quantitative study, which for completeness specifies the exact statistical methodology, provides underlying data, and includes the computer program source code.

Due to recent historical events, there is increased interest in using social network theory to analyze of terrorists and terrorist communication networks from a technological perspective (e.g., analysis of communication between suspected terrorists). In a peer-reviewed study of terrorist social network communication structure, researchers Eiselt and Bhadury (2015), investigated complex networks using communication metadata (i.e.,

the origin, destination, start time, and end time of the communication, not the actual contents of the communication) to identify and track membership in terrorist networks without the need for wiretapping. This involved calculating and monitoring the shortest paths and degrees of separation between suspected individuals to identify terrorist leaders (p. 2), similar in concept to Milgram's "six degrees of separation" mentioned earlier in this study (Korte & Milgram, 1970). Knowing the shortest path between network members is important as this helps researchers identify command nodes by identify key positions in the network structure. This research showcased the importance of determining the shortest path between terrorist leaders and their network's followers (each of whom can be represented as nodes in a graph structure). Their research also described that small-world networks are more sensitive to attacks on "bridge nodes" than on their corresponding hub nodes (p. 2). In graph theory, "bridge nodes" are nodes through which pass many shortest paths (they have high "between-ness"). But as described separately by Xu and Chen (2008, p. 84), hub nodes by contrast are nodes which have many links (they exhibit *high* degree") but not necessarily through which pass many shortest paths. The Xu and Chen (2008) paper studied dark networks, and their findings indicated that pure scale-free networks are susceptible to both hub and bridge node attacks, whereas small-world networks are more susceptible to bridge node attacks, than to hub node attacks (p. 64). Another finding from the Xu and Chen (2008) paper was the calculation that the length of the average shortest path between Osama Bin Laden and members of his Global Salafi Jihad (GSJ) network, was only 2.5 steps. This means the degrees of separation between Bin Laden and a typical member of his GSJ organization



was between just two to three people, and that the GSJ network was sparse (not dense), as sparseness combined with short path length helped to lower the risk of detection and enhance communication efficiency (p. 62). Thus, finding shortest paths in complex networks of interest can yield valuable intelligence information, and helps justify this quantitative study on algorithmic pathfinding. A weakness with the Xu and Chen (2008) study was the lack of mention of any graph analysis frameworks used, nor algorithms implemented. This weakness seemed to also be shared with the Eiselt and Bhadury (2015) study. If both studies had elucidated their graph analysis techniques, it would have benefited both papers. These weaknesses represent gaps in the literature which are filled by this quantitative experiment which, by contrast, specifically discussed graph pathfinding algorithms.

Terror networks are resilient even after removal of key network nodes. In another study of terrorist network communication using methodologies from social network theory, Medina (2014) studied the resiliency of terrorist network communication structures before and after the removal of key terrorists (e.g., Osama Bin Laden, and Abu Mussab al-Zarqawi). The fact that terrorist networks are not comprised of purely random members of society initially indicated to the author that terror networks were not random networks, but rather they could be classified, perhaps confusingly, as either small-world, or scale-free, or both (Medina, 2014, p. 108). The findings of Medina's analyses indicated that the Al Qaeda social network was indeed a scale-free network, not a small-world network, because the average path length between known terrorist members was too short, meaning this network did not meet the Watts and Strogatz (1998) definition of

small-world networks, which requires small-world networks to have average path lengths longer than, or equal to, random networks with the same number of nodes and edges, and that the clustering coefficient be much larger than that of a comparable random network (Medina, 2014, p. 109; Watts & Strogatz, 1998, p. 440). But despite repeated, targeted node attack to remove terrorist leaders, which should otherwise hurt small-world networks, as discussed in detail by Albert and Barabási (2002), and by Lordan, Sallan, and Simo (2014), Medina admitted that there is some network property, yet to be determined, that gave the Al Qaeda network its efficiency and resiliency (Medina, 2014, p. 109). Discovery of this "hidden network property" could be a further research opportunity. A deficiency with the Medina (2014) study was that although the author analyzed several communication networks, providing statistics on path lengths, clustering coefficients, network diameter and degree centrality, Medina did not mention which graph analysis tool or framework, or computer language was used to generate his results. These are gaps in the literature that were filled by this study.

According to a study of small-group 9/11 terrorist social networks by Lewis (2013), and confirmed by Watts and Strogatz (1998), while there are properties of small-world social networks (e.g., short radius, high between-ness centrality) that are shared by some physical networks, social network topologies do not need to be similar to topologies of physical networks, like the electric power grid, Internet, transportation systems, water and pipeline networks (Lewis, 2013, p. 7). One possible reason for the differences in the topological structures between social networks and physical networks, is that physical infrastructure is limited, in part, by economics, landscape, and regulations. By contrast,

social network topologies are much more resistant to the physical constraints of landscape, country, and economics (p. 18), thereby transcending boundaries of space, place and time. The Lewis (2013) paper shared the same deficiencies that the aforementioned Medina (2014) study had, namely, no graph analysis frameworks were discussed, and no computer language source code was provided. The Watts and Strogatz (1998) study lacked details on how to apply small-world network analysis to physical world problems, and also did not mention graph analysis frameworks used. This study addressed the graph analysis framework issue by describing, in detail, the graph analysis frameworks utilized.

Finally, while today's computational complex networks are large and sophisticated, such as the over 1 billion users of the Facebook social network (Balaguru et al, 2015; Brooks et al, 2014), a study by Poisot (2013) suggested that graph theory and social network theory -- while useful in performing complex network analysis -- may not be sufficient to analyze all aspects of complex networks. Poisot (2013) postulated that while measuring network structure is indeed important to understanding both the latent and emerging properties of complex networks, researchers lack an *a posteriori* measure that serves as a "goodness-of-fit" indicator for the results of complex network analyses. The author's proposed goodness-of-fit indicator is network modularity, which Poisot defined as the ratio of interactions established between members of the same module (i.e., community) vs. members of different modules within the same overall network (2013, p. 1). The Poisot network modularity factor is applied, *a posteriori* (that is, after other complex network analyses have been performed), to help choose the most appropriate

community partitioning scheme according to the desired network property the researcher wishes to study (p. 6). While the Poisot (2013) method could provide another way to analyze complex networks, one concern with the Poisot (2013) study was Poisot's use of "pseudo-random" networks (pp. 4, 11). Poisot did not elaborate why "pseudo-random" networks were selected, nor how Poisot style "pseudo-random" networks compare against the traditional ER style random networks and WS style small-world networks described earlier in this literature review. From a real-world network analysis perspective, Poisot did not describe how to apply the Poisot network modularity factor to very large social networks, such as Facebook with its over 1 billion users (Balaguru et al, 2015). Next, regarding Poisot's "pseudo-random" networks (pp 4, 11) in particular, two issues remain: (a) it is not clear if Poisot networks called "pseudo-random" due to usage of a computer pseudo random number generator (PRNG) during network creation; and (b) Poisot did not refer to the fact that WS style small-world networks can be derived from regular networks by manipulating the randomization probability coefficient,  $P$  (Watts & Strogatz, 1998). One gap in the Poisot (2013) study was the use of only the Network-X and iGraph graph analysis frameworks; no mention was made of the popular open source graph analysis framework Graph-Tool. Finally, while Poisot (2013) did mention use of Python (p. 3), it was not made clear which version of Python was used. These gaps were filled by this quantitative study.

### **Rival Theory to the Selected Theoretical Framework**

To ensure an exhaustive review of the professional and theoretical literature, a search for opposing theories that could have served as alternative core theoretical

frameworks by which this study's variables could be measured, examined, and interpreted, was conducted. One such rival theory was identified: chaos theory. Chaos theory was developed in 1963 by meteorologist and mathematician Edward Lorenz while he worked on weather systems and weather prediction algorithms (Hung & Tu, 2014, p. 1227). Lorenz conjectured that very small changes, such as a butterfly's wing-beat, may lead to radical consequences on a global scale. He coined this concept the "butterfly effect." Weather systems can be described as chaotic systems because they are aperiodic (they never repeat in the exact same way), yet are sensitive to initial conditions (Adewumi, Kagamba, & Alochukwu, 2016, p. 5). From this study's perspective, because network connections between nodes in dynamic graphs will change over time, dynamic graphs could also exhibit chaotic behavior. Such networks, with their ephemeral node and link structures that vary over time, would exhibit dynamic network topologies (Baingana & Giannakis, 2017). An example of this would be using a graph to model real-time urban traffic flow congestion, and then trying to find the shortest path through that traffic congestion (Adewumi, Kagamba, & Alochukwu, 2016, p. 5).

A small change can have a large, unintended (and unforeseen) effect on a complex system (Hung & Tu, 2014). Under chaos theory, the so-called "butterfly effect" of extreme local sensitivity leading to dynamic, global change, is due to the non-linear nature of the initial conditions because complex behavioral patterns may occur that are not proportional (and unpredictable) to their original causes, as discussed by Peters (2014). An example of this would be the spread of a vector-borne epidemic, where a small number of initially infected people within a dense network are sufficient to cause

the epidemic to broadly spread within the larger complex network, with an unpredictable end state (Dantas-Torres, 2015, p. 452). Upon initial inspection, chaos theory appeared to be a potentially viable theoretical framework for this study because removing nodes from network graphs in a chaotic (i.e., random) fashion may appear to lead to unintended consequences during shortest path calculations, particularly if a graph is small, connections are sparse, or the nodes removed are the major hub nodes (Barabási, 2016). However, depending on the type of network, randomly removing nodes does not necessarily irreparably "harm" all graphs to the same extent, especially for networks with many redundant connections, such as, but not limited to, ER style random networks (Barabási, 2016; Watts and Strogatz, 1998). For example, if randomly selected nodes and edges are both removed from, and added to a given graph, the damage to the network in terms of being able to find a shortest path between any two nodes may be partially mitigated, because according to Albert and Barabási (2002), random node removal will have different outcomes on different networks, not all of which are equally bad. However, Albert and Barabási do caution that, in general, node removal inflicts more damage to a network than edge removal (2002, p. 42).

Air transport networks are susceptible to chaos. In a study of air transport networks by Lordan, Sallan and Simo (2014), who used airports to represent nodes in their network graph, they determined that random "point to point" connections between airports, generally utilized by low cost air carriers, and follow the Erdős and Rényi (1961) random network model, are more robust and likely to survive both random and targeted node removal than the "hub-and-spoke" air transport networks generally used by

the full service air carriers, which more resemble scale-free networks, and which are quite vulnerable to targeted node removal (2014, p. 118), i.e., terrorist attack. Computer networks may be similarly vulnerable. The ultimate effect of chaotic (random) node removal depends on the underlying network. The structure of the network (e.g., regular, random, small-world, scale-free) may be impacted by chaos in different ways. This may be measurable. In separate research by Zou, Xiao, and Gao (2013) which studied the infliction of random chaos vs. intentional chaos on the urban transit system of the city of Foshan, China -- a network the authors claim had both small-world and scale-free network characteristics (p. 393) -- they discovered that random node removal from the Foshan urban transportation system noticeably degraded network performance, but not as quickly as the damage caused by intentional node removal (p. 390). Random vs. intentional node removal represented different levels or types of "chaos" in the Zou et al. (2013) study. A weakness with both the Lordan, Sallan and Simo (2014) study, and the Zou, Xiao, and Gao (2013) study, was the lack of quantified data describing the before and after effects of the application of chaos on their respective systems studied.

It seems evident from the aforementioned studies that network perturbations may have a negative impact on network routing and pathfinding performance. This quantitative study was not designed to research the *a posteriori* effects of chaotic node perturbations. Also, this study's randomly generated 2D network map samples are static throughout the algorithmic pathfinding phase, that is, the samples don't "grow" or change over time. Thus, chaos theory was deemed to be less relevant to this study than social network theory, because in this study's experimental design, there is no possibility of the

2D maps ever changing during the algorithmic pathfinding phase of the experiments. By contrast, if this study were designed to research dynamically changing network maps (i.e., 2D map samples that change during the algorithmic pathfinding phase), with the express intent to measure the impact of chaotic node perturbations on subsequent pathfinding results, then chaos theory would be a more appropriate theoretical framework for this study than the formalist approach of social network theory that I selected. Although this study does not use dynamic 2D maps (i.e., maps that change over time), research on dynamic maps and graphs, chaos theory, and implications thereof, are possible topics for further research, which were discussed in the recommendations for further research part of Section 3.

### **Independent Variable: Pathfinding Algorithms**

Many graph theoretic pathfinding algorithms have been discovered that perform the task of finding the shortest path between nodes in graphs, and therefore have many modern uses in a wide variety of problem domains (Boguchwal, 2015). In this section, pathfinding algorithms relevant to this study are discussed. These algorithms are Dijkstra, Bellman-Ford, and A\* (pronounced "A star").

One mid-20th century example of pathfinding algorithms is the now-famous Dijkstra shortest path algorithm. Edsger Dijkstra, a theoretical physicist by training, developed his pathfinding algorithm in the 1950s, and then published it in 1959 (Ammar, Bennaceur, Châari, Koubâa, & Alajlan, 2015). Dijkstra's algorithm is a graph search algorithm that finds the shortest path between nodes in non-negative weighted graphs, provided such a path exists, as discussed by Abdulkadir, Fadzli, Jamal, Makhtar, Awang,



Mohamad, and Susilawati (2015). It was a significant advancement beyond the breadth first search (BFS) and depth first search (DFS) pathfinding algorithms prevalent at that time. According to an algorithm study by Bohács, Gyimesi, and Rózsa (2015), both Dijkstra's algorithm, and BFS are guaranteed to find the shortest path between two nodes, provided a path exists, but Dijkstra's algorithm is more efficient than BFS in terms of memory consumption, and unlike BFS, Dijkstra's algorithm can handle different positive edge weights, since edge weights in BFS are not considered (p. 15). For completeness, it must be noted that while the depth first search (DFS) algorithm will also find a path between source and destination nodes, provided one exists, it is not guaranteed to be the shortest path (Bohács, Gyimesi, & Rózsa, 2015). As this study is focused on finding shortest paths, this rendered DFS to be of little relevance to this study. Additionally, researchers D'Angelo, D'Emidio, and Frigioni (2014) confirmed that, despite its age, Dijkstra's algorithm is still widely used today as part of the Open Shortest-Path First (OSPF) algorithm. Use of Dijkstra's algorithm in OSPF was also confirmed in a separate algorithm study by Vesović, Smiljanić, and Kostić (2016). The OSPF algorithm is an interior gateway protocol (IGP), used to exchange routing information between gateways (routers) over Internet Protocol (IP) networks. The fact that Dijkstra's algorithm is used in today's Internet, nearly sixty years after Dijkstra's algorithm was published, demonstrates how a well-designed algorithm may enjoy decades of longevity. This study compared Dijkstra's algorithm to other pathfinding algorithms, as is discussed in much more detail in Section 2.

One limitation with Dijkstra's algorithm is that for very large graphs it may exhaust all available memory when searching for shortest paths, according to Ammar, Bennaceur, Châari, Koubâa, and Alajlan (2015). A second limitation with Dijkstra's algorithm is that it cannot handle negative edge weights, that is, the weights assigned to the edges in a graph must not be negative, or Dijkstra's algorithm will fail (Vesović, Smiljanić, & Kostić, 2016). One algorithm that can handle negative edge weights (but not negative cycles) is the Bellman-Ford algorithm (Vesović et al, 2016). Additionally, Jukna and Schnitger (2016) confirm the utility of the Bellman-Ford algorithm in the areas of shortest path calculations, dynamic programming, and switching networks. However, there are weaknesses with the Bellman-Ford algorithm. While the algorithm can be distributed, according to Nanongkai (2014) the Bellman-Ford algorithm is not suitable for parallelization. But, since this study does not compare parallelized versions of pathfinding algorithms, this specific limitation of the Bellman-Ford algorithm was not a factor in this study. However, algorithm parallelization may be a fruitful avenue for further research, and is discussed in the further research part of Section 3.

Dijkstra's algorithm and the Bellman-Ford algorithm are not the only relevant algorithms to study. Another family of pathfinding algorithms to consider is the A\* (pronounced "A star") algorithm. In a study by Yoon, Yoon, Lee, and Shim (2015) who used the A\* algorithm, and several customized variants thereof, in car-like vehicles and robots running on grid maps, they discovered that not only is pathfinding algorithm choice important, but the kinematics of the robot or vehicle itself (e.g., turning radius, vehicle width, the ability to make 2-point and 3-point turns, the ability to drive in reverse

while turning, etc.) may affect the ability of these robotic vehicles to travel down tight corridors, collision free. The result of the Yoon et al. (2015) research suggested that when researchers think about autonomous vehicle pathfinding, in some cases researchers should also take the kinematics of the vehicle into consideration, not just the pathfinding algorithms alone. By contrast, in a separate study of robot pathfinding algorithms by Yang, Qi, Song, Xiao, Han, and Xia (2016), they compared over a dozen pathfinding algorithms for suitability in robots in autonomous path planning and navigation, not just the A\* algorithm. Interestingly, their first step was to model a terrain environment as a grid map (p. 2), which is an activity also performed in this study too (to be discussed in more detail in Section 2). Yang, et al. (2016), suggested that multifusion algorithm solutions (i.e., using more than one pathfinding algorithm at a time) may provide the best overall solution for complex network pathfinding scenarios. The authors did provide time complexity metrics, in mathematical big-Omega notation (p. 19), but they did not recommend which would be the best secondary algorithm to fuse with A\* despite being proponents of algorithm multifusion, nor did they discuss the impact of hardware on runtime algorithm performance. In a separate hardware-oriented algorithm study by Ediger, Jiang, Riedy, and Bader (2013), they tested multithreaded graph algorithms for massive graph analysis on synthetic (i.e., randomly generated) scale-free graphs. In their experiment involving graphs with 4.27 billion edges, they were able to detect all connected components in 2 minutes (p. 2227) using the GraphCT framework, customized for use with the 128 processor Cray XMT super computer. But in reality, few software developers have access to Cray XMT super computers, thereby limiting the

generalizability of Ediger et al. (2013) research results. This represents a gap in the literature that was filled by this study, because this study was not performed on expensive Cray XMT super computers. By comparison, the experiments performed in this study were performed on much less expensive commodity Apple hardware.

Comparing path finding algorithms can provide useful knowledge. In a 2014 study by Singh and Mishra, they compared four pathfinding algorithms in Erdős and Rényi (1961) *ER* style random networks. Their results showed that performance differed in sparse graphs vs. dense graphs (p. 26), and that Dijkstra's algorithm can achieve better runtime performance by using Fibonacci heaps vs. binary heaps (p. 23). This paper had a gap in the literature, however, in that it focused exclusively on *ER* style random networks. As discussed by Albert and Barabási (2002), and Watts and Strogatz (1998), most real world networks are not ER-style random networks. Close friendship networks are an example of this: these networks generally are not randomly generated (Albert & Barabási, 2002). Because the Singh and Mishra (2014) paper focused on only ER-style random networks, it loses applicability to other real world networks of the scale-free and small-world varieties discussed earlier. But to be fair, the seminal Watts and Strogatz (1998) paper also did not discuss the which pathfinding algorithms were the best for small-world network shortest path searches, which the Singh and Mishra (2014) study, to their credit, attempted to do, so both papers could have benefited from intellectual cross-pollination with each other. Additionally, while the experimental computer programs Singh and Mishra used were written in the C language, they did not provide source code. By using one language, at least they were consistent, but not including source code

represents a research gap. This gap was covered by my experimental study, which includes freely available source code (discussed in more detail in Section 2 of this study).

### **Independent Variable: Graph Analysis Frameworks**

To accomplish computational science endeavors, there are three main approaches software engineers may take: (a) write custom software; (b) use proprietary software; or (c) use free or open source software (Freeman, 2015). The open source software community benefits from open collaboration, ease of sharing and maintenance, and the testing and development efforts performed by others (p. 160). This quantitative experiment, for example, used free or open source graph analysis frameworks to save time and development effort. Many of the open source frameworks used specifically for graph analysis are also compatible with Python, and are discussed next.

Not all graph analysis frameworks, or computer languages, are equally liked by software engineers and researchers. In a comprehensive study of graph analysis frameworks by Nocke, Buschmann, Donges, Marwan, Schulz, and Tominski (2015), they compared 17 different graph analysis frameworks for complex networks analysis. Their findings indicated that 72% of researchers preferred using Python over other tools/languages like MATLAB, Mathematica, and even GIS systems (p. 549), in part because of Python's ease of use, and compatibility with a wide array of scientific and numerical libraries (e.g., NumPy, SciPy). In a separate study by Yang, Algesheimer, and Tessone (2016) of the widely used, open source (and Python compatible) iGraph graph analysis framework, they provided quantitative data on elapsed time consumed by eight different community detection algorithms supported by iGraph, after analyzing complex

networks of varying sizes and complexities. The results varied, depending on the type and size of network analyzed. There were no one-size fits all "best-of-fit" pathfinding algorithm answer to derive from their research results. One deficiency with the Yang, Algesheimer, and Tessone (2016) study was the lack of memory consumption as a dependent variable, even though the authors admitted that memory consumption would be a "crucial" issue when analyzing larger complex networks (p. 10). Additionally, they could have compared more graph analysis frameworks, like Nocke et al. (2015) did. Yang et al, did not quantitatively investigate other widely popular Python-compatible graph frameworks, such as Graph-Tool and Network-X. Similarly, the Nocke et al. (2015) paper is also not immune to critique. While the Nocke et al. (2015) paper is informative and a comprehensive resource for a high-level comparison of graph analysis frameworks, one significant weakness was the lack of numerical data quantifying the speed and efficiency of each framework against the others. Nocke et al. (2015) also did not specify if the same pathfinding algorithms were compared and tested by each graph analysis framework. These represent gaps in the literature which may be filled by this study.

Graph analysis frameworks can be used by researchers in disparate problem domains. Researchers Phillips, Schwanghart, and Heckmann (2015), who studied the applicability of graph theory to network analysis, determined that graph theory was well suited to network analysis, and recommended its use as a powerful tool in the sciences. Their findings also suggested that there are several free and open source software tools for graph analysis. One of these tools is iGraph (p. 149), which the authors found to be

Python compatible. Another Python-compatible graph analysis tool the authors discussed is Graph-Tool, and their findings suggested that Graph-Tool is one of the "most popular graph theory modules" for Python programmers" (p. 149). Although they mention two of the graph analysis frameworks used in this quantitative experimental study, they did not mention the Network-X graph analysis framework, which represents a gap in the literature to be filled by this study. The issue of "analyzing" a graph analysis framework, but not providing quantitative supporting data, was also noticed in the following graph analysis framework studies by Csardi and Nepusz (2006); Majeed and Rahman (2015); and Sayama, Pestov, Schmidt, Bush, Wong, Yamanoi and Gross (2013).

Although somewhat dated, the 2006 study by Csardi and Nepusz provided excellent information on the iGraph graph analysis framework, including actual Python source code examples showing how to use the framework. Unfortunately, the weakness with the Csardi and Nepusz (2006) paper was its focus on only the iGraph framework (no quantitative or qualitative cross comparisons with other graph analysis frameworks were provided). In the 2013 study by Sayama et al., they modeled and tested scale-free complex networks using Python and the Network-X graph analysis framework. Their results indicated researching network topologies is applicable not only to the social sciences (where social network theory originated), but also in other sciences (e.g., biology, physics). One deficiency with the Sayama et al. (2013) study was its primary focus on Albert and Barabási (AB) style scale-free networks. They did not provide an explanation why only AB-style networks were studied, nor why they neglected both ER

style random networks and WS style small-world networks. Another weakness was its focus on only the Network-X graph analysis framework.

Finally, in the Majeed and Rahman (2015) study of graph analysis frameworks, they studied four less popular graph analysis frameworks: (a) Gephi; (b) Pajek; (c) Cytoscape; and (d) Tulip. The authors admitted to having discovered memory leaks with Cytoscape, Tulip, and Pajek (p. 24). Memory leaks are bad in that they limit the ability of graph frameworks to handle large datasets (where graph visualization can be most useful), so why Majeed and Rahman (2015) continued to study memory leaking graph analysis frameworks was unclear. By contrast, the authors did not report similar memory leakage issues with the Gephi graph analysis framework. Interestingly, while the Majeed and Rahman (2015) study did use elapsed time as a metric in their tests, aside from noting memory leakage issues, they did not utilize memory consumption as a dependent variable in their study, which seemed odd, considering they discovered memory leaks in three of the four graph analysis frameworks they evaluated. Interestingly, the issue of memory consumption was also not mentioned in the seminal Watts and Strogatz (1998) study, nor in the seminal Albert, Jeong, and Barabási (1999) study. Not using memory consumption as a dependent variable represents a gap in the literature which was filled by this quantitative study.

### **Independent Variable: Map Complexity**

Not all terrain maps are equal. Some maps (or regions within maps) are more complex than others (Subarno, Siregar, Agus, & Sunuddin, 2016). One topic addressed in my quantitative study is the impact of map size on algorithmic pathfinding efforts.



Another topic addressed is the impact of network structure (e.g., small-world networks and random node connectivity) on algorithmic pathfinding efforts. Together, map size and nodal connectivity (or lack thereof) are factors that comprise the "map complexity" independent variable in this study. Discussed next are studies related specifically to map size and algorithmic pathfinding.

Map size has an impact on pathfinding algorithms. In a study of robot pathfinding simulators by Alotaibi and Al-Rawi (2016), their findings indicated that one critical factor which affected pathfinding algorithm performance was the size of the map (p. 147). As the grid maps they tested grew larger, more time was required for the selected pathfinding algorithm to find a shortest path solution (pp. 150-152). One weakness with their paper was the lack statistical test results for all algorithms and map sizes compared. This literature gap may be filled by the results of this quantitative study. By contrast, in a comparative study of four pathfinding algorithms by Lim, Seng, Yeong, Ang, and Ch'ng (2015), they used maps of length and width  $n$  (where  $n$  is a positive integer) to represent terrain grids (cells). These  $n \times n$  grid maps were used in their comparative algorithmic pathfinding experiments. They tested maps of sizes ranging from the smallest at dimensions of  $10 \times 10$  (i.e.,  $10^2 = 100$  grid cells), up to the largest dimensions of  $70 \times 70$  (i.e.,  $70^2 = 4900$  grid cells). Their findings suggested that maps with larger dimensions (i.e., more grid cells) took longer for their chosen pathfinding algorithms to process than the time needed for the same algorithms to process smaller terrain maps (p. 2727). While both the Alotaibi and Al-Rawi (2016) study, and the Lim, Seng, Yeong, Ang, and Ch'ng

(2015) study covered terrain maps, they both lacked detailed comparative statistical analyses. This is a gap that was covered by this quantitative study.

Large maps require more time to process when seeking shortest paths. In a comparative algorithm experiment by Zhu and Chiu (2015), which used grid maps of varying grid cell counts, they compared three maps of different grid cell counts (a) small: 2,888 grid map cells; (b) medium: 9,873 grid map cells; and (c) large: 13,108 grid map cells. Their results suggested that a map's total grid cell count is directly proportional to the computational time required to find the shortest path in the map, and in some cases as the map size grew linearly, the time required to find the shortest path also grew exponentially (p. 84). In a separate but supporting study of grid map-based comparative pathfinding algorithm experiments, by Sharon, Stern, Goldenberg, and Felner (2013) they compared three pathfinding algorithms in a multi-agent pathfinding context. They also used four different grid map dimensions as an independent variable: (a) 3x3 grid; (b) 4x4 grid; (c) 8x8 grid; and (d) 257x257 grid (p. 490). Their results suggested that grid map size had a direct impact on multi-agent pathfinding runtime performance, with larger maps requiring more time to process than smaller maps (p. 491). This is confirmed in similar findings by Zhu and Chiu (2015). Furthermore, these results concur with the results of a small-world network study of graph visualization frameworks, and network analysis community detection algorithms by Gibson and Vickers (2016). Gibson and Vickers (2016) confirmed the Watts and Strogatz (1998) findings that small-world networks are characterized by high clustering coefficients, and short average path lengths, but they also discovered that path lengths that grow logarithmically as more

nodes are added (Gibson & Vickers, 2016, p. 81). The study used maps which varied from 500 nodes with 5000 edges (their smallest network map), up to 5000 nodes with 25,000 edges (their largest network map). The Gibson and Vickers (2016) results suggested that the overall time required to calculate the shortest paths with their selected graph algorithms was greater when processing larger maps than when processing smaller maps (p. 83). These results correspond to the separate findings of Zhu and Chiu (2015), and Sharon, Stern, Goldenberg, and Felner (2013).

Two-dimensional terrain maps can be generated several different ways. One way is to abstract a 2D map of the Earth, or a portion of it, and convert that into a format readily accessible to pathfinding algorithm benchmark programs, a process known as map abstraction or map genotyping (Liapis, Yannakakis, & Togelius, 2015). A practical example of the benefits of map abstraction is described in Zhang, Su, Liu, Hu, and Zhu (2016) where they abstracted an aerial Earth map into a grid map, for subsequent aerial and land-based threat analyses for unmanned aerial vehicles (UAVs) using a more detailed 2D map grid for the actual pathfinding algorithm shortest path calculations.

Using map genotypes (i.e., map abstractions) aids in the algorithmic pathfinding process according to Zhang et al. (2016), who saved time and money using synthetically generated map genotypes to test various pathfinding routes without having to pilot a UAV, purchase fuel, or endanger civilian bystanders. Applications of this map abstraction (genotyping) technique can be applied to not just the military, but also for search and rescue operations (p. 27). In a separate but related study of robot pathfinding simulators by Alotaibi and Al-Rawi (2016), they tested maps from popular PC computer

games that were then abstracted into mathematical graphs, from which graph theoretic shortest path pathfinding algorithms could be benchmarked and performance results compared. The findings of Alotaibi and Al-Rawi (2016) suggested that it is possible to transform maps from popular PC computer games (e.g., Baldur's Gate II), into a numerical format for easy ingestion into an algorithm performance test harness for subsequent algorithm performance testing (p. 148). The Alotaibi and Al-Rawi (2016) map abstraction approach is similar to the Zhang et al. (2016) approach of abstracting maps of the Earth, in that each approach transforms existing complex terrain maps into simpler 2D grid maps for easier processing by pathfinding algorithms. One weakness with both the Alotaibi and Al-Rawi (2016) study and the Zhang et al. (2016) study, was the lack of mention of any sort of theoretical framework which may have guided their work.

Raw grid map size is not the only factor to influence algorithmic pathfinding performance. A second factor to consider in algorithmic pathfinding studies is the number of occluded (i.e., blocked or impassable) grids on a grid map, as this may affect the network structure and therefore nodal connectivity patterns (Zhang, Li, & Bi, 2016). In the real world, blockages (i.e., lack of connection between two nodes in a 2D grid map) may be due to several causes: (a) natural blocking terrain features (e.g., mountains, swamps, oceans); (b) other smart agents or vehicles or people already occupying a destination space (i.e., grid cell); or (c) unexpected hazardous conditions (e.g., fire). Together, map size and the network structure (including terrain obstructions, which may impact nodal connectivity) are the factors that comprise the "map complexity" variable

used in this study. Discussed next are studies that specifically involve terrain map obstacles (blockages), and their impact on algorithmic pathfinding.

More obstacles on a map means more time is required to algorithmically process that map. Zhang, Li, and Bi (2016) conducted a quantitative comparison of a custom variant of the A\* (pronounced "A star") algorithm versus the original A\* algorithm. The authors targeted the A\* algorithm for study because the popular A\* algorithm is considered the "gold standard" in some situations for shortest path search algorithms due to its overall effectiveness (p. 1). They tested their algorithms on 8-direction grid cell maps with obstacles consisting of blocked (impassible) cells, and traversable areas consisting of unblocked cells. Their findings indicated that A\* could be upgraded to improve its performance on maps with high blockage ratios (p. 9), especially where the blockages are irregularly shaped. In a separate, but similar, A\* algorithm study, researchers Ammar, Bennaceur, Châari, Koubâa, and Alajlan (2015) conducted a quantitative study on eight variant algorithms of the A\* pathfinding family. Their experiments involved grid maps with varying occlusion ratios. The findings by both Ammar et al. (2015) and Zhang et al. (2016), were that a high obstacle ratio broadly impeded pathfinding algorithm performance. Furthermore, Ammar et al. (2015) concluded that some algorithms are more affected than others during algorithmic shortest path calculations on occluded terrain maps, with the performances of both the popular Dijkstra's algorithm and the A\* algorithm noticeably degraded on highly occluded ratio grid maps. One limitation with both studies was the lack of details on statistical methods used, and no publicly available numeric data to support their assertions. These

deficiencies were covered by this study which includes a full description of the statistical methodology used, the numeric data, and results.

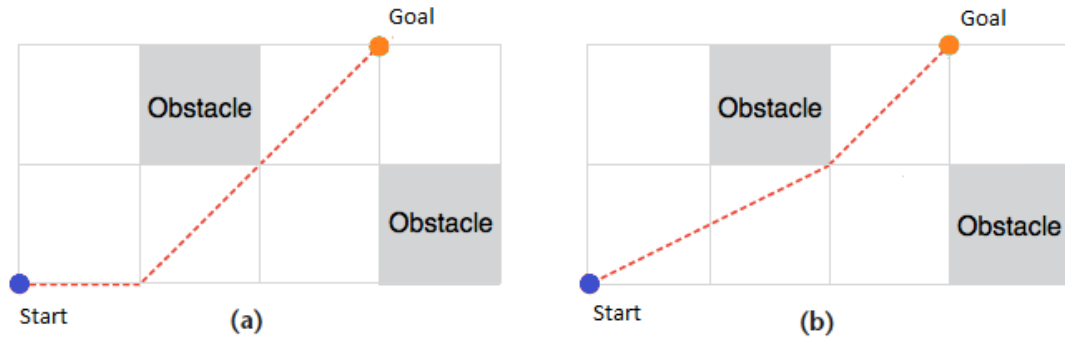


Figure 11. (a) Shortest grid path, (b) the real shortest path (based on Nash & Koenig, 2013).

Shortest paths on grid maps are not necessarily the true shortest paths. Nash and Koenig (2013) conducted a comparison study of path-planning methodologies for finding shortest paths in continuous terrain. Their findings indicated that in robotics and video games it is not unusual for software engineers to discretize continuous terrain into grid cells that are classified as either passable, or occluded (i.e., blocked, impassable), and then follow with a grid-based pathfinding algorithm to find the shortest non-occluded path in the resulting grid map (p. 85). However, occluded grid cells may cause distortions in the resulting calculated shortest paths, depending on the size or position of the occlusion vis-a-vis the size of the discretized grid cells (Nash & Koenig, 2013). The result is the shortest path may be short if one were to follow the edges of each discretized grid cell, but may not be the true shortest paths (pp. 90-91). This is demonstrated in Figure 11, which depicts a  $2 \times 4$  grid map (8 grid cells in total) with 2 blocked grid cells.

The occlusion ratio for the grid map in Figure 11, can be easily calculated as  $2 / 8 = 25\%$ , which means a quarter of the grid map is blocked due to impassible terrain.

Taking larger grid maps with differing occlusion ratios into consideration, there may be a divergence between the algorithmically calculated shortest grid path (which follows the grid cell edges) versus the true shortest path (which is free to short cut through the non-obstacle grids cells), making the occlusion ratio, and any possible impact on shortest path pathfinding, a worthy variable to study. The impact of map obstacles on algorithmic pathfinding was confirmed in a separate study of robot pathfinding simulators by Alotaibi and Al-Rawi (2016), whose findings indicated that one critical factor which affected pathfinding algorithm performance was the ratio of occupied vertices to unoccupied vertices (p. 147). One weakness with the Alotaibi and Al-Rawi (2016) paper was the lack statistical test results for all algorithms, map sizes, and map occupancy ratios they compared. Another limitation was although their simulation was written in C++, they did not indicate if any graph analysis frameworks were used, or if they implemented the pathfinding algorithms themselves. Similarly, the Nash and Koenig (2013) study, while comprehensive, did not include source code, nor did it mention which graph analysis frameworks were utilized in their study (if any). These represent gaps in the literature which may be covered by this quantitative experimental study.

### **Dependent Variable: Elapsed Time**

There is a common technique used when comparing algorithms. For quantitative algorithm comparisons, a vast majority of the pathfinding literature follows this general approach to algorithm analysis: (a) a set of benchmark problems are selected; (b)

mathematical constructs and algorithms to be compared are configured to operate on the selected problems; (c) the algorithms are then iteratively applied to each problem, and the results are collected from multiple trial runs; (d) the results are then statistically analyzed (McClymont, Keedwell, & Savic, 2015). This format is useful because it provides a deductive, quantitative, logical way to side-by-side compare algorithm performance. This approach was used with some variations (i.e., sample sizes, dependent variables, etc.) in the studies related to elapsed time that are described next.

As discussed by Freeman (2015), software engineers may write their own custom software, use proprietary software, or use free or open source software alternatives. In a quantitative comparative algorithm study by Franke and Ivanova (2014), they opted to create their own graph analysis framework. They measured time consumption during network navigation in complex, dynamically changing networks. They compared their own pathfinding framework, FALCON, written in C++, against several popular graphing libraries, running against dynamic graphs, and measured the elapsed time needed to find the shortest path. According to the authors, their framework, FALCON, was fastest compared to the others. As their framework was not tested against the graph analysis frameworks used in this study, the external validity of their results suffers, and this limits the applicability of Franke and Ivanova's (2014) research results to my study.

Additionally, as my experimental study compared popular free, or open source, Python-compatible graph analysis frameworks, not the proprietary FALCON framework, the external validity of my study should be greater than that of the Franke and Ivanova (2014) study. Similarly, in a comparative computer language and pathfinding algorithm



study by Klimaszewski (2014), the author studied the runtime efficiency of the A\* (pronounced "A star") algorithm, with programs written in C++, Java and Python. The findings of Klimaszewski (2014) suggested that in the most complex tests, Java was one order of magnitude slower (in elapsed seconds at runtime) than C++, and that Python was three orders of magnitude slower than C++ (p. 68). One deficiency with the Klimaszewski (2014) study is that only C++ source code was provided (no Java, nor Python source code). The second deficiency was that only elapsed seconds were measured (as a dependent variable), but not memory consumption. In both cases, Franke and Ivanova (2014) and Klimaszewski (2014) could have spent more time (a) describing their statistical methodologies, (b) providing supporting numeric data, and (c) discussing theoretical frameworks which directed their research efforts (if any). These deficiencies were addressed in this quantitative study.

The vehicle routing problem is a problem domain ripe for pathfinding algorithm research, but it is also a difficult problem domain to solve. Koç, Bektaş, Jabali, and Laporte (2016) compared several metaheuristic algorithms in the vehicle routing problem (VRP) domain. One dependent variable measured was elapsed time (p. 14), as this provided a way to benchmark algorithms against each other. The findings of their study suggest that while several highly accurate VRP algorithms have been developed, they suffer from high computation times, or they lack simplicity, or their results are difficult to reproduce (p. 16). So, while Koç, Bektaş, Jabali, and Laporte confirmed that some algorithms provided good compute times, the lack of simplicity makes implementation of their selected algorithms challenging, at best. Lack of implementations simplicity makes

it harder for others to replicate their research findings in different environments, thereby negatively impacting the external validity of the Koç, Bektaş, Jabali, and Laporte (2016) study. In a similar study of robot pathfinding simulators by Alotaibi and Al-Rawi (2016), they tested algorithmic pathfinding algorithms against computer game maps, while measuring execution time (i.e., elapsed time). Their findings suggested that, all else being equal, larger maps required more time to be processed by pathfinding algorithms than smaller maps (pp. 151-153). One weakness with the Alotaibi and Al-Rawi (2016) paper was the lack of mention of any sort of theoretical framework which guided their study, and no mention of the statistical methodology used, nor the resulting statistical output. The lack of mention of an overall theoretical framework which guided their research was a deficiency also shared by the Koç, Bektaş, Jabali, and Laporte (2016) study.

Genetic algorithms can also be used to solve pathfinding problems. Bezerra, Goldberg, Goldberg, and Buriol (2013) studied multiple variants of ant colony optimization (ACO) pathfinding algorithms on grid maps of varying sizes, and measured elapsed time as one of their dependent variables. They used the Kruskal and Wallis statistical test with a significance level of 95%, and the Wilcoxon one-tailed test at 97.5% significance level to determine if there was a statistically significant difference found between the groups they studied (p. 351). Their findings suggested that larger grid maps required more to complete pathfinding objectives than the time required on smaller maps (pp. 351, 353). While it was helpful that they described the statistical methods they used, they could have included more statistical output to support their conclusions. They could have also compared more algorithms. To be fair, the seminal Watts and Strogatz (1998)

paper, the Albert, Jeong, and Barabási (1999) study, and the Erdős, and Rényi (1961) study could have also included descriptions of comparison algorithms, or statistical methods and resulting output, but they also did not, therefore, the Bezerra, Goldbarg, Goldbarg, and Buriol (2013) study results are in similar company. Nonetheless, these deficiencies represent gaps in the literature which may be filled by this quantitative study.

### **Dependent Variable: Memory Consumption**

Available computer random access memory (RAM) has certainly grown over the last three decades, but it is not infinite, and out-of-memory warnings may still occur today. In a comparative Python implementation study by Redondo and Ortin (2015), one of the dependent variables they studied was computer memory consumption. Their findings suggested that not all implementations of Python are equal. Some Python implementations use more memory to complete the same programmatic task, than other Python implementations (pp. 82-83). This implies that if memory consumption is a concern to Python software engineers, then they must be cognizant which version(s) of Python they use. Similarly, in a graph theoretical study of De Bruijn graphs (DBG) authors Salmela and Rivals (2014) used Dijkstra's algorithm to analyze and calculate the shortest paths in specialized DBG networks (p. 3509). One of the metrics they used to compare data results included gigabytes of memory consumed during program execution (p. 3509). They obtained computer memory consumption results by periodically polling the Linux operating system of their computers. Their results suggested that it is possible to measure gigabytes of memory consumption (to the hundredths of a gigabyte of accuracy, e.g., "24.04 GB") by periodically polling the OS for memory consumption data

(Salmela & Rivals, 2014). This is similar to the methodology used for computer memory profiling, described by in a separate treatise by Gorelick and Ozsvald (2014). This study used similar OS polling techniques to gather memory usage statistics, which is described in more detail in Section 2.

Terrain features can have an impact on pathfinding algorithm performance in terms of memory consumption. In a study by Mora, Merelo, Castillo, and Arenas (2013), they compared 12 different variants of multi-objective ant colony optimization (MOACO) algorithms for shortest pathfinding efforts on terrain maps. One of the primary dependent variables they measured was memory consumption (in megabytes, MB). The other dependent variable was elapsed time. Each of the 12 MOACO algorithms followed different approaches (e.g., safety vs. speed vs. cost minimization) to achieve their pathfinding goals on hexagonal grid maps. Their results suggested that the characteristics of the grid map (e.g., the predominant terrain type: mountain, forest, river, etc.) and the pathfinding algorithm approach (e.g., safety, vs. speed, vs. cost minimization, etc.) had direct impacts on pathfinding algorithm performance in terms of memory consumption and the actual calculated paths from start to finish. The Mora et al. (2013) study was similar to the aforementioned Bezerra, Goldberg, Goldberg, and Buriol (2013) study in that both studied ant colony pathfinding algorithms on grid maps. One weakness with both the Mora et al. (2013) study, and the Bezerra et al. (2013) study, was lack of discussion of the exact statistical methods used and resulting data to support their assertions. This gap was filled in my study which includes all data and details of the statistical methods used.

There is more to consider than the shortest path, when it comes to pathfinding. Wen, Çatay, and Eglese (2014) performed a quantitative study of algorithmic solutions to network routing and scheduling problems. Their study used memory consumption and elapsed time as dependent variables (p. 920), and used two different heuristic variants to their chosen pathfinding algorithm (Dijkstra's algorithm) in order to determine the minimum cost path between a pair of nodes (p. 915). Their findings suggested that if time was the most desirable factor, then Dijkstra's algorithm would find the optimal path (p. 916). However, if cost minimization (excluding time) was the most desirable factor (i.e., minimization of fuel, labor, or avoidance of network congestion), then Dijkstra's algorithm was not guaranteed to always find the least cost path (p. 917). This implied that purely shortest paths and minimum cost paths may be significantly different, depending on current network traffic congestion. One weakness with this study was its lack of publically available source code describing the authors' implementation of Dijkstra's algorithm. A second weakness was no mention of the overarching theoretical framework which drove their research process and design. The lack of mention of a foundational theoretical framework was also shared by the Abdulkadir, Fadzli, Jamal, Makhtar, Awang, Mohamad, and Susilawati (2015) study which similarly discussed Dijkstra's algorithm. By contrast, the theoretical framework used in this experimental study was already mentioned earlier in Section 1.

### **Computer Programming Languages: Python vs. Java**

Today's software engineers have many computer languages available from which to choose. Over the last two decades, the popularity and features of Python, Java, C++,

and other languages, has yielded many software frameworks and libraries for use (Dierbach, 2014; Farooq, Khan, Ahmad, Islam, & Abid, 2014; Freeman, 2015), some free or open-source, others proprietary. Some of those frameworks were used and referenced in this doctoral study, as discussed in more detail in Section 2. Regardless of the specific computer language, there are broader issues of runtime efficiency, language simplicity and ease-of-use which must be considered, and are discussed next.

Shorter programs may be easier to understand than functionally equivalent longer programs written in another language. In a study of concurrency programming by Williamson and Olsson (2014), they explored language flexibility and the ease of writing highly parallelizable programs. Their findings suggested that Python's easy to learn, concise syntax allows developers to quickly create considerably shorter (in terms of lines of written code) and easier to read programs, than functionally equivalent programs written in other languages (p. 309). Similarly, in a discussion of Python by Dierbach (2014), the author's findings suggested that over the last decade, the popularity of Python has increased considerably, to the point where it has even become one of the first languages taught to undergraduate computer science (CS) students at some colleges. Java and Python are two popular language choices taught in colleges, and while the authors did not suggest that learning Java is a poor choice, they did note that there have been reports of significant improvement in student and instructor satisfaction after redesigning introductory CS courses to use Python rather than Java (Dierbach, 2014). A common deficiency of both the Williamson and Olsson (2014) study, and the Dierbach (2014)

study, was the lack of statistical methodology and output data to support their assertions. By contrast, this quantitative study includes a full statistical report.

The "simplicity" of a programming language has an impact on popularity and use. In a Java vs. Python language comparison by Hunt (2015), the author's findings suggested that the surge of interest in Python is due, in part, to the simplicity of Python, compared to the complexity inherent to languages like Java and C++. While, Java is an "excellent" language in many ways, Java was not designed or intended as a "teaching language" (p. 173). This implies that students may find learning Python easier than learning Java. Programs written in Python were 1/3 the size (in number of lines of code) than Java equivalents (p. 173). The author's findings suggested that by using Python one may end up writing fewer lines of code than by using Java, thereby saving the software engineer precious time during computer program implementation. A separate but related study by Muller, Bednar, Diesmann, Gewaltig, Hines, and Davison (2015), documented the surge in the popularity of Python among the sciences, due in part to its readability, modularity, and large, freely available standard library. Muller et al. pointed out that Python's popularity with scientists began with the emergence of Python's NumPy numerical analysis package in the late 1990s (2015, p. 1). There are also many other third-party, open source libraries and frameworks for graph analysis, easily usable with Python. However, a weakness with both the Hunt (2015) study, and the Muller et al. (2015) study, is that they did not specifically compare pathfinding algorithms or Python-compatible graph analysis frameworks. This is a research gap that was filled by this quantitative experimental study.

The Python language is not monolithic. There are many versions and implementations of Python available for use, as discussed next. Redondo and Ortin (2015) performed a comprehensive performance evaluation of seven common Python implementations: (a) CPython, (b) Cython, (c) WPython, (d) Stackless Python, (e) PyPy, (f) Jython, and (g) IronPython. Where available, they also compared Python version 2 and version 3 implementations of the aforementioned common Python implementations. CPython, being the reference implementation of Python, implemented in the C programming language (p. 79), was the standard to which the six other Python implementations were measured against. Their findings were mixed. Overall CPython did well, but it was not the fastest nor most memory efficient Python implementation for long running Python 2.x processes (p. 84). For long running Python 3.x processes, Cython performed better than CPython (p. 84). Two weaknesses with this study was its reliance on Windows only implementations of Python, and no performance results for Mac OS or Linux versions of Python were provided for comparison. As this study used Python on Mac OS, this limits the applicability of the Redondo and Ortin (2015) paper to this quantitative study. Nonetheless, the benefit of the Redondo and Ortin (2015) paper is that it should make Python developers aware that the version of Python used is important, as performance characteristics may differ between Python implementations and versions. To that end, in the interest of generating consistent results, where possible, this study limits its use of Python to one version only.



### **Modern Applications of Algorithmic Pathfinding.**

Autonomous robotic pathfinding can be aided with network support services, such as the Global Positioning System (GPS), and/or Wi-Fi (IEEE 802.11 protocol) networks (Sung, Kwak, & Park, 2015). With GPS or Wi-Fi-oriented Web services (that are GPS-enabled), the robot (i.e., the "bot," smart agent, or drone), or the human end user, may receive pathfinding guidance (Milner, 2016). Common everyday modern examples include using Google Maps on a mobile device, or using GPS in a personal automobile to map and locate a target destination. Some related topics of interest include (a) using pathfinding algorithms in emergency evacuation situations; (b) the formation of mobile ad-hoc networks (MANETs) for remote routing support; and (c) autonomous planetary surface navigation. These topics are described next.

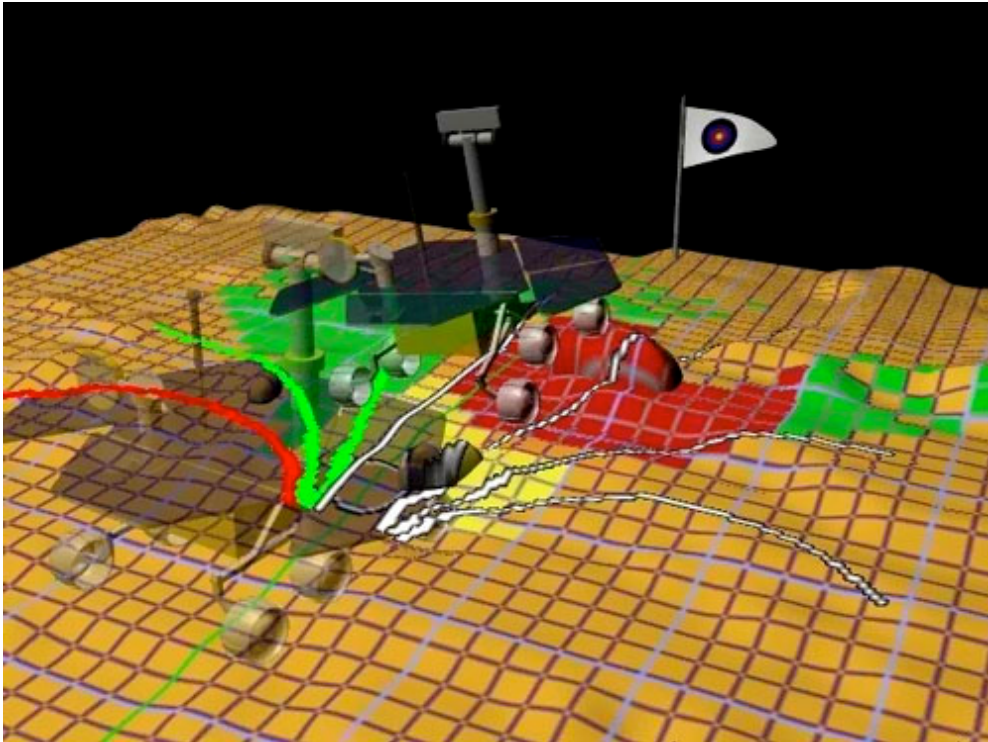
Natural occurring fires cause large amounts of socio-economic loss and create many victims. In a study of emergency escape route planning for forest fires by Wang, Zlatanova, Moreno, Van Oosterom, and Toro (2014), they researched the problem of how to get emergency relief vehicles to the affected areas as quickly as possible to fight forest fires. Their research used the A\* pathfinding algorithm to calculate escape routes and transit routes in both static and dynamic map environments. Their research also reviewed crowd-sourced data regarding the state of the area for calculating the shortest paths. Their findings implied they could be used not just for route planning during forest fires, but also for navigation in other types of disasters. In a related study by Kaur and Gangal (2015), they compared seven different mobile ad-hoc network (MANET) routing protocols. Their findings seemed to suggest that no specific protocol they compared was

the final "overall" best one, as each had their pros and cons. One finding from the Kaur and Gangal (2015) study was that using the "shortest path" as a routing metric (p. 26) is key from a resource efficiency perspective. Both studies, however, could have been combined to the benefit of each other. For example, in the interest of creating an ephemeral local communication network to enable emergency routing of traffic and personnel, Wang et al. (2014) could have discussed use of MANETs, in the manner described by Kaur and Gangal (2015), to provide peer-to-peer (P2P) message routing support. Conversely, Kaur and Gangal (2015) could have discussed ways to apply their MANET findings to the emergency support problem domain, or the benefits of algorithmic pathfinding approaches, in the manner described by Wang et al. (2014). One weakness with the Wang et al. (2014) findings was that they failed to address the topic of having secondary pathfinding support (whether GPS or some other method) if for some reason (e.g., computer memory exhaustion) their primary A\* algorithm approach failed to find a suitable path. Furthermore, although Wang et al. (2014), did measure elapsed time as a dependent variable, they did not evaluate the memory consumed by their software, nor did they describe if they tested their system on terrain maps of varying complexities. Finally, a weakness with the Kaur and Gangal (2015) study was the lack of statistical data to support their assertions, for each of the algorithms the authors compared. In both the Kaur and Gangal (2015) and the Wang et al. (2014) cases, the noted deficiencies represent gaps in the literature, some of which may be filled by this quantitative experimental study.

In a similar disaster evacuation study by Kang and Choo (2016), the authors compared various approaches to finding emergency evacuation routes, including graph theory approaches, and biological-inspired approaches. Their findings suggested that excessive local network communication overhead during an emergency would cause transmission interference and network communication congestion in the afflicted region. This suggested that if the local communication network were rendered inoperable, then the aforementioned A\* pathfinding algorithm-oriented emergency evacuation system described by Wang, Zlatanova, Moreno, Van Oosterom, and Toro (2014), could be a useful backup, as the Wang et al. (2014) method does not depend on GPS or Wi-Fi. This scenario could also make the MANET approach from the aforementioned Kaur and Gangal (2015) study useful, for if the Kaur and Gangal (2015) method could create an ad hoc ephemeral MANET to route communication traffic, independent from the overloaded local communication network, then that ephemeral MANET may supplant the original communication framework rendered inoperable due to emergency transmission overload, allowing emergency vehicle routing. One strength with the Kang and Choo (2016) study was the discovery that many evacuation algorithms focus on finding the safest paths, or the shortest paths, but do not consider route congestion. One weakness with this paper was the lack of statistical output, or mention of statistical methods used to compare results. Another weakness with the Kang and Choo (2016) study was that although they discuss many algorithms, they did not provide algorithm pseudo code or actual source code for analysis. These are gaps in the literature which were filled with this study which

will include source code, a full description of statistical methods, and the resulting quantitative output.

One problem with relying on GPS or Wi-Fi Web services is dependence on access to overhead satellite resources, or dependence on wireless network connectivity (e.g., IEEE 802.11 communication protocols) between sending and receiving devices (Milner, 2016). Underground or undersea settings may not have GPS or Wi-Fi support, nor would non-Earth planetary bodies, like the Moon or Mars. According to a study by Dean (2013), starting in 2004, and serially launched over the course of several subsequent years, three separate Martian surface rovers (named Spirit, Opportunity, and Curiosity, respectively) were sent to Mars to explore that planet's surface (see Figure 12). Although they supported manual control from Earth, they also employed autonomous pathfinding with onboard pathfinding software (Dean, 2013, p. 161). Due to the varying 3 to 22-minute delay in radio frequency (RF) transmission between Earth and Mars because of the distances involved, autonomous pathfinding can yield more responsive results for each rover than manual control from Earth. But autonomous pathfinding also runs the risk of the rovers getting mired in non-traversable terrain without Earth knowing about the situation for several minutes. To reduce the possibility of getting mired in bad terrain, the rovers were preloaded with Martian terrain maps to help reduce uncertainty during algorithmic pathfinding. But static maps are not always 100% accurate as they do not account for dynamic terrain changes.



*Figure 12.* Grid-based autonomous rover algorithmic pathfinding (NASA, n.d.).

The alternative, truly autonomous real time pathfinding using pathfinding algorithms, may better handle dynamic terrain changes and situations when direct manual control (via RF communication) from Earth is not possible (Dean, 2013). Findings from the Dean (2013) study confirmed that while an increase in terrain obstacles reduced the chance of pathfinding success, the biggest impact to successful algorithmic pathfinding was the interval at which the rovers' internal terrain maps were refreshed with updated terrain data (p. 177). One drawback with the Dean (2013) study was that it did not mention emergency alternatives to robotic algorithmic pathfinding, and it could have been enriched with concepts from the Kaur and Gangal (2015) study. For example, if a supporting mesh network of communication devices were scattered across the Martian

terrain, creating an ad hoc MANET network in the style of the aforementioned Kaur and Gangal (2015) study, then a Mars rover might be able to communicate with that local MANET to get supporting local navigational data for pathfinding purposes. Such a Kaur and Gangal (2015) style MANET network, if appropriately configured and equipped, in turn could communicate with Earth and vice-versa. This might solve the problem mentioned by Dean (2013) where a planetary rover needs real time pathfinding support, but cannot communicate directly with Earth. The Kaur and Gangal (2015) style MANET could be the intermediary between Earth and the Mars rover. Unfortunately, this was not discussed in the Dean (2013) study. Finally, although Dean (2013) did mention use of Python (pp. 163, 171), he did not provide any source code. Lack of publically available source code for analysis and review was also shared by the Kaur and Gangal (2015) study. This represents a gap in the literature filled by my study, which includes publicly available source code.

### **Transition and Summary**

Section 1 was an introduction to the study on the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and memory consumption. Section 1 also included the problem statement, the purpose statement, the nature of the research, the research hypotheses, the definition of terms, and the theoretical framework. The purpose of this quantitative experimental study was to examine whether there was a relationship between the study's variables of interest. While researching issues related to algorithmic pathfinding, social network theory seemed most directly relevant to this literature review due to its roots in graph theory, its interest in shortest

path calculations, and the importance of nodal interactions in complex networks. In investigating the plethora of applications where algorithmic pathfinding may be used, it was evident that the principles of social network theory -- strengthened by its deep roots in graph theory, random network theory, small-world social network theory, and scale-free network theory -- are significant and provide a theoretical framework by which researchers can quantifiably measure the impact of pathfinding algorithm selection on complex network navigation, while providing the lens by which algorithmic pathfinding research can be interpreted within the larger milieu of future applied pathfinding software problems. As discussed, software engineers may benefit from social network theory when it is applied to pathfinding problems because this knowledge may help software engineers solve future challenges in pathfinding software development while providing positive social change, such as terrorist network analysis and terrorist identification, and autonomous planetary surface exploration, both of which were discussed earlier. Pathfinding algorithm performance depends on a confluence of factors, each of which contributes to the end goal of writing efficient pathfinding software. The complexity, cost, and risks inherent in major software development projects (both in up-front development costs, and in later support/maintenance costs) make it essential that appropriate algorithms and software frameworks are evaluated, early, before significant time and money are spent on implementation and support. The lack of quantitative research on applied, comparative real world algorithm performance using Python and free or open source graph analysis frameworks, may hinder some pathfinding software development efforts. This literature

review attempted to identify existing knowledge on this topic, and also identified several gaps in the current algorithmic pathfinding research literature.

Section 2 contains a discussion of this study's chosen methodology. This includes elaboration of the research design, the setting and samples, all instrumentation, approaches to data collection, and analysis of the experimental data. Strategies to ensure reliability and validity of the proposed research are presented. Also discussed are issues related to test subject generation, selection, and privacy protection. Section 3 includes the results of this study and a discussion on how the research findings support or reject the null hypothesis, followed by a discussion of further research opportunities relevant to comparative algorithmic pathfinding.



## Section 2: The Project

This study examined the impact of pathfinding algorithms, graph analysis frameworks, and map complexity, on elapsed time and computer memory consumption. I used popular free or open source graph analysis frameworks (and their built-in pathfinding algorithms), instead of writing the pathfinding algorithms myself. This study helped elucidate how the chosen graph analysis frameworks performed at the task of algorithmic 2D grid map pathfinding, in terms of elapsed time and memory consumption. This section contains discussions of (a) my role as researcher; (b) the research method and design used by this study; (c) data collection methodology; (d) population, sampling and grouping issues; and (e) ethical concerns.

### **Purpose Statement**

The purpose of this quantitative experimental study was to examine the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and memory consumption, in order to select appropriate pathfinding algorithms for resource-constrained software agents running in complex networks, network dead zones or GPS-denied environments. The targeted population consisted of local computer random-generated two-dimensional (2D) grid maps. The three independent variables were (a) pathfinding algorithms; (b) graph analysis frameworks; and (c) map complexity (e.g., small vs. large maps; high random rewiring vs. low random rewiring). The two dependent variables are (a) elapsed time; and (b) computer memory consumption. Contributions to positive social change from efficient pathfinding algorithms are wide-ranging, from saving lives to saving money. Some recent

examples include (a) fast robotic debris cleanup of airport runways to prevent fatal accidents during takeoff and landing (Öztürk & Kuzucuoğlu, 2016); (b) bounded-cost optimization of business expenses (Stern et al., 2014); and (c) search and rescue missions in unmapped terrain (Liu & Lyons, 2015).

### **Role of the Researcher**

For this quantitative experimental study my high-level role was a combination of data collector, analyzer, statistician, and software developer of the computer programs used to gather grid map-oriented algorithmic pathfinding performance data. More specifically, my role in this study involved (a) creating measurable research questions and hypotheses; (b) finding gaps in the relevant literature; (c) locating software frameworks that perform algorithmic pathfinding; (d) writing short computer programs to instrument and automate data gathering; (e) collating the experimental results; and (f) applying appropriate statistical methodologies with rigor, trustworthiness, neutrality, and without bias, to confirm any findings, as recommended separately by Katz (2015); Lunde, Heggen, and Strand (2013); and Rutledge, Jones, Bailey, and Stewart, (2014).

Challenges with algorithm performance and memory consumption have interested me for over 20 years. To prevent my 23 years of experience as a professional software engineer from negatively biasing this research, this study relied on existing code frameworks, application programmer interfaces (APIs), and pathfinding algorithm implementations. Therefore, I only wrote "glue code" which connected the test harness to the graph analysis frameworks in order to collect algorithmic performance data, thereby

letting the graph analysis frameworks do the actual algorithmic pathfinding work, using their internal pathfinding algorithm implementations, not mine.

Because this study did not involve humans (living or deceased), nor their personally identifiable information (PII), the Belmont protocols established for the protection of vulnerable populations (Quinn, Kass, & Thomas, 2013) did not apply to this doctoral study.

### **Participants**

This research was conducted using local computer random-generated 2D grid maps, represented mathematically as 2D adjacency matrices. No human participants were required to collect this graph theory-related research data. The 2D maps used in this study were the participants, and were represented mathematically as graphs, thereby aligning with the main research question of this study. Researchers Shi and Weninger (2016) generated synthetic graphs which were then used as participants for their algorithm study with no human participants required. Similarly, Stevenson and Cordy (2014) utilized computer-generated graphs for their algorithm study, also without the need for human participants. Although the population and sampling methodology is discussed in much more detail later in the Population and Sampling part of Section 2, a brief summary follows.

A population of several thousand 2D maps were computer random-generated. This pool of 2D maps represented the initial population from which map samples were randomly selected. Randomization is required in true experimental designs (Maertens & Barrett, 2013). From the large initial population pool of random 2D maps, each map was

stratified into subgroups based on demographic characteristics. Demographic group membership was based on the *map complexity* independent variable: (a) *high complexity* maps; or (b) *low complexity* maps. Next, from each of the population demographic subgroups, random assignment was used to allocate map samples from the population subgroups to the targeted experimental treatment groups. This study sought to identify causal inference between the aforementioned variables of interest, so the random generation, stratification, and selection process used in this experiment was consistent with the 2D map characteristics and population that were the focus of this study, and represented by the *map complexity* independent variable. Stratification of the random samples into homogeneous groups was useful because some population demographic groups may behave differently to experimental treatments, as was noted in a quantitative experimental study by Krauss, et al., (2013), who compared treatment effects on different stratified sample groups. The aforementioned 2D map population generation strategy was similar to the graph generation concepts utilized in separate studies by Shi and Weninger (2016), and Stevenson and Cordy (2014). Furthermore, Qasem and Viswanathappa (2016) showcased the utility of sample stratification in experimental research. Finally, Almaghairbe and Roper (2016) also discussed use of stratified random sampling and random assignment in the domains of software engineering, software testing, and software anomaly detection. Again, the 2D map population and 2D map samples are discussed in more detail in the Population and Sampling part of Section 2.

## Research Method and Design

This doctoral study used a quantitative experiment research method to analyze the impact of pathfinding algorithms, graph analysis frameworks, and map complexity, on elapsed time and memory consumption. The research method and design were selected to align with the problem statement, purpose, and research question, in the interest of identifying causal relationships between my aforementioned variables of interest.

### Method

This doctoral study followed a quantitative research method. Based on a positivist philosophy (Tsang, 2014), the goal of this study was to examine potential causal relationships between these three independent variables: (a) pathfinding algorithms, (b) graph analysis frameworks, (c) map complexity, and these two dependent variables: (a) elapsed time, and (b) the amount of computer memory consumed during pathfinding operations.

Researchers employing qualitative methods may explore new problems by seeking open-ended *where* or *who* answers rather than statistically explain a cause-effect outcome (Ittner, 2014). Similarly, Barnham (2015) argued that qualitative research is generally focused on *why* questions, not on *what* questions. Other characteristics of qualitative research include interviews, observations, and a focus on the lived experience (Madill, 2015, p. 215). Since this study aimed to identify cause-effect relationships between the aforementioned variables of interest, not to answer open-ended *where* or *who* questions, this rendered qualitative research methods inappropriate.

Mixed methods research involves combining both quantitative and qualitative approaches within a single research study (Landrum & Garza, 2015). One benefit of mixed methods research is a more comprehensive understanding of the object under study (Riazi & Candlin, 2014). Another benefit to using mixed methods over single method studies is enhanced stimulation of theoretical imagination, permitting new ideas to flourish that otherwise would not in single method studies; however, the researcher pays a higher price with mixed methods in terms of time and resources consumed (Raich, Müller, & Abfalter, 2014). Since this study did not use qualitative research methods, this rendered the mixed methods approach inappropriate.

Quantitative methods generally use empirical data, often requiring descriptive statistics to describe the sample population under study (Bettany-Saltikov & Whittaker, 2014). Another indicator of quantitative methods is usage of inferential statistics, to infer results discovered in a small sample back to the wider population (Ersoy & Akbulut, 2014). Quantitative methods also focus less on interviews and open-ended *where* or *who* questions, but more on targeted *what* questions, using investigative techniques such as, but not limited to, experiments, multivariate statistics, and computer modeling (Jackson, 2015). For this study, the quantitative method was selected over a qualitative method (e.g., case study, ethnographic, phenomenological) because of my desire to statistically identify cause-effect relationships between the aforementioned variables of interest.

### **Research Design**

According to Cokley and Awad (2013), there are three main research design approaches available to quantitative researchers attempting to identify possible

relationships between dependent and independent variables: (a) correlational, (b) quasiexperimental, and (c) experimental.

Correlational designs do not permit control or manipulation of treatments; however, they can be used in surveys, and to assess relationships among variables (Cokley & Awad, 2013; Granato, Calado, & Jarvis, 2014). Another common use of correlational designs is in trend analysis (Groeneveld, Tummers, Bronkhorst, Ashikali, & Van Thiel, 2015). Since this study involves intentional manipulation of the aforementioned independent variables in order to measure treatment effects (if any) on the dependent variables, the correlational design is rendered inappropriate.

As described by Kumar, Nilsen, Abernethy, Atienza, Patrick, Pavel, ... and Hedeker (2013), quasiexperimental designs do not use random assignment of samples to treatment groups. A weakness with quasiexperimental designs is that making causal inference is more challenging than with experimental designs, as potential confounding variables may limit interpretation of effects (p. 14). But researchers Hancox, Quedsted, Thøgersen-Ntoumani, and Ntoumanis (2015), discussed how in some situations, due to the nature of the sample population under study, randomization may not be possible, nor even desired. Furthermore, quasiexperiments may implement certain study design features in order to rule out some plausible alternative associations between variables of interest (Donofrio, Class, Lahey, & Larsson, 2014). As this study intentionally used random assignment, and since quasiexperimental research does not support randomization, this rendered quasiexperimental designs inappropriate.

Experimental designs are considered strongest of all designs regarding internal validity, which itself is the center of cause-effect inferences, and are characterized by the introduction and intentional manipulation of one or more treatment variables (Quick & Hall, 2015). Furthermore, researchers Donaldson, Qiu and Luo (2013) suggested that experiments, particularly clinical laboratory experiments, are more rigorous than other types of research, and that such rigor aids in the detection of causal relationships. Experimental designs support use of random assignment, and this technique helps reduce threats to internal validity (Krishnan & Sitaraman, 2013). Randomized controlled trials (RCTs) are examples of experimental design, and due to their strong statistical support are considered by some to be the "gold standard" in causal inference (Clair, Cook, & Hallberg, 2014, p. 311). Additionally, experiments can give particular insight into algorithm performance issues, especially regarding asymptotic analyses where theorists might ignore constant factors of large orders of magnitude, though such factors may have dramatic performance impacts in the real world (Mitzenmacher, 2015). An experimental design was selected for this study, because of my desire to identify causal relationships between the aforementioned variables of interest, utilizing intentional manipulation of the independent variables and randomized assignment of samples to treatment groups.

In the interest of explaining this study's research design, it helps to understand its high-level process flow. Figure 13 depicts this study's high-level process flow, from the initial randomized inputs (left), to comparative performance tests and post-test measurements (middle), and then finally the statistical analyses (right).



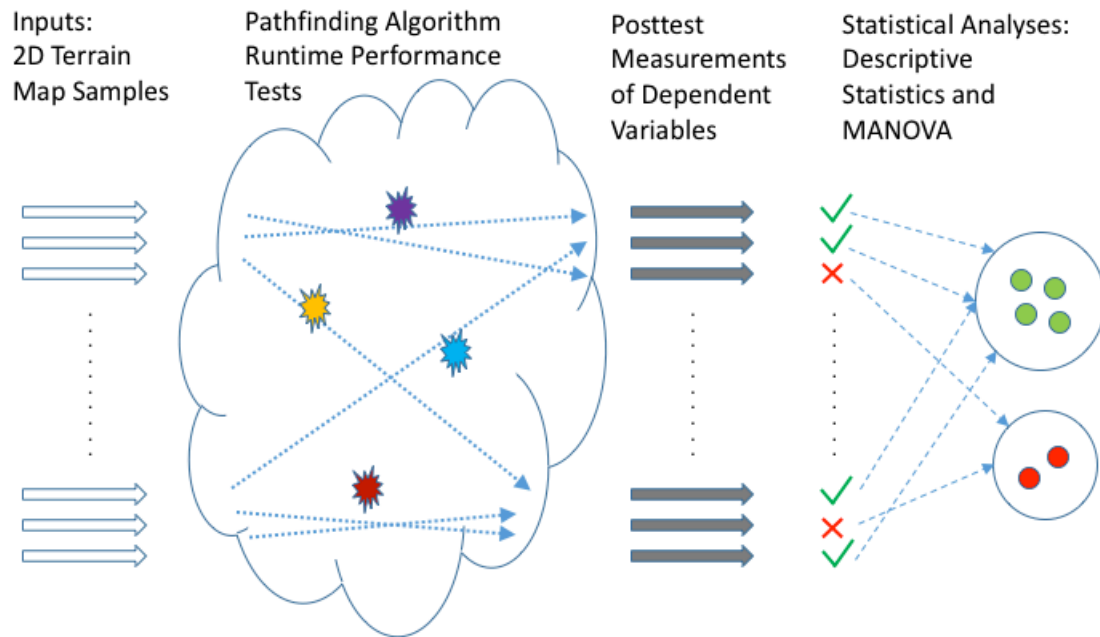


Figure 13. High-level overview of this study's experimental process flow.

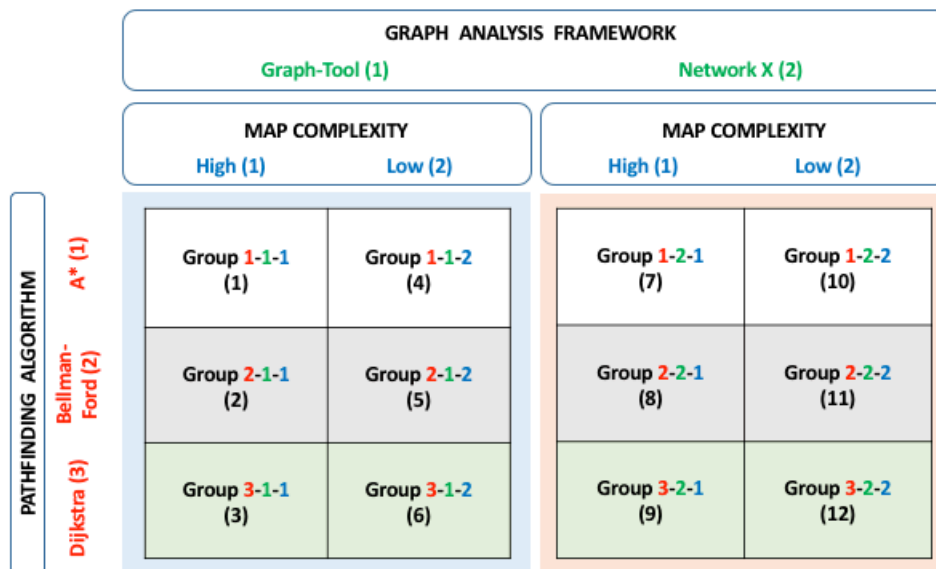
Knowing the number of treatment groups is important in factorial experimental research (Papaneophytou & Kontopidis, 2014), and this study's independent categorical variables and levels were: (a) three pathfinding algorithms; (b) two graph analysis frameworks; and (c) two map complexity types, which resulted in a  $3 \times 2 \times 2$  (i.e., 12-way) factorial experimental study. The combinations of the independent variables, yielding the experimental treatment groups, are summarized in Table 1.

Table 1

*The Categorical Variables and their Levels*

Categorical Variable Name	Number of Levels
Pathfinding Algorithm	3
Graph Analysis Framework	2
Map Complexity	2

To help visualize the aforementioned  $3 \times 2 \times 2$  factorial test matrix of treatment groups used in this study, all 12 treatment groups are depicted in Figure 14.



The treatment groups matrix is based on (a) pathfinding algorithms, (b) graph analysis frameworks, and (c) map complexity, yielding the  $3 \times 2 \times 2 = 12$  factorial treatment groups used in this experimental study.

Figure 14. This study's experimental 12-way factorial matrix.

In standard research design notation, where "R" = random assignment to a treatment group, "X" = treatment intervention, and "O" = observation and measurement, each treatment group in this experiment follows a randomized, between groups, post-test only experimental design, as depicted in Table 2.

Table 2

*The Experiment: Randomized, Between Groups, Post-Test Only*

R	X	O
---	---	---

Standard research design notation can be applied to all treatment groups of the experiment, as depicted next in Table 3.

Table 3

*The 12-way Factorial Matrix, in Standard Research Design Notation*

Groups and Demographics (by algorithm, graph analysis framework, and map complexity)	Random Sample Assignment	Algorithm Intervention ("Treatment")	Post Test Observation
Group 1: A*; Graph-Tool; Map Complexity: High	R	X	O
Group 2: Bellman-Ford; Graph-Tool; Map Complexity: High	R	X	O
Group 3: Dijkstra; Graph-Tool; Map Complexity: High	R	X	O
Group 4: A*; Graph-Tool; Map Complexity: Low	R	X	O
Group 5: Bellman-Ford, Graph-Tool, Map Complexity: Low	R	X	O
Group 6: Dijkstra, Graph-Tool, Map Complexity: Low	R	X	O
Group 7: A*, Network-X, Map Complexity: High	R	X	O
Group 8: Bellman-Ford, Network-X, Map Complexity: High	R	X	O
Group 9: Dijkstra, Network-X, Map Complexity: High	R	X	O
Group 10: A*, Network-X, Map Complexity: Low	R	X	O
Group 11: Bellman-Ford, Network-X, Map Complexity: Low	R	X	O
Group 12: Dijkstra, Network-X, Map Complexity: Low	R	X	O

As mentioned earlier, the pathfinding algorithm was the first categorical, independent variable. The pathfinding algorithms were discussed in more detail in the literature review in Section 1 of this study. Pathfinding algorithms represented the first categorical (independent) variable used in this study, and are summarized in the Table 4.

Table 4

*List of Pathfinding Algorithms Analyzed in this Study (per Graph Framework)*

Pathfinding Algorithm (independent, categorical variable)

- 
1. A\* Algorithm
  2. Bellman-Ford Algorithm
  3. Dijkstra's Algorithm
- 

The graph analysis framework was the second categorical, independent variable in this study. These were discussed in detail in Section 1. There were two different graph analysis frameworks compared in this study, and they are summarized in Table 5.

Table 5

*List of Graph Analysis Frameworks Analyzed in this Study*

Graph Analysis Framework (an independent, categorical variable)

- 
1. Graph-Tool
  2. Network X
- 

Map complexity was the third (and final) categorical, independent variable in this study. This factor was also discussed in the literature review in Section 1. There were two different map complexities utilized in this study, summarized in Table 6.

Table 6

*Map Complexities Considered in this Study*

Map Complexity (structure)	Adjacency Matrix Dimensions	Small-World Random Rewiring Coefficient	Number of Links per Node
1. High (circular lattice)	1000 × 1000	0.25 %	2
2. Low (circular lattice)	200 × 200	0.5 %	2

There were two dependent, quantitative variables used in this study. The first quantitative dependent variable was elapsed time. The second quantitative dependent variable was computer memory consumed. These are summarized in Table 7.

Table 7

*Dependent Variables Analyzed in this Study*

The Dependent, Quantitative Variables	Unit of Measurement
1. Elapsed Time	Seconds
2. Computer Memory Consumed	Megabytes

For ease of reference, a complete listing of the variables used in this study, by name, type, and level of measurement, are summarized in Table 8.

Table 8

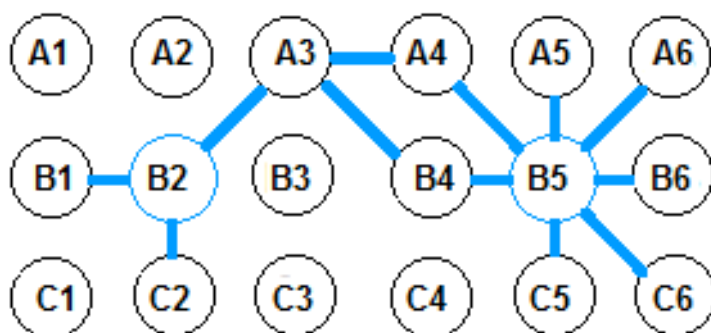
*Summary List of Variables Used in this Study*

Variable Name	Variable Type	Level of Measurement
1. Pathfinding Algorithm	Independent	Categorical (3 levels)
2. Graph Analysis Framework	Independent	Categorical (2 levels)
3. Map Complexity	Independent	Categorical (2 levels)

4. Elapsed Time	Dependent	Continuous (seconds)
5. Memory Consumed	Dependent	Continuous (megabytes)

### Population and Sampling

The target population for this study was local computer random-generated 2D maps, represented mathematically as adjacency matrices. No human participants were required in order to collect this research data. This is not unusual. In empirical studies of algorithms, the population and samples are usually limited by computational (not human) resources (Arcuri & Briand, 2014). Each member of the computer random-generated map population can be represented as a 2D grid map, where each grid represents a vertex (i.e., node), with some vertices connected to other vertices by edges (i.e., arcs, lines). There may be many vertices and edges per 2D grid map, as described by Maciejewski and Puleo (2014). Depicted in Figure 15 is an example 2D grid map with 18 vertices and 12 connecting edges.



*Figure 15.* Example two dimensional grid map.

Utilizing a population of 2D grid maps was important to this study because grid maps can represent terrain, and terrain can be traversed, algorithmically, to find the

shortest path from a starting vertex, to a destination vertex. This related to the overarching research question of this study, which was: "What is the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption?" With that research question in mind, in order to find a relationship between pathfinding algorithms and 2D maps, I first needed a population of 2D maps to draw samples from. It did not make sense to use a population of humans as 2D grid maps, because humans do not easily represent terrain. Since no humans were involved, the population of 2D maps were computer random-generated. There is a long history of using grid maps, combined with algorithmic pathfinding, for activities ranging from video games to planetary exploration with the Mars rovers (Spirit and Opportunity), as discussed in detail by Algfoor, Sunar, and Kolivand (2015), and by Dean (2013). Once a population of 2D grid maps was generated, I drew random samples from that 2D map population in order to experimentally test algorithms and graph analysis frameworks on those samples. The goal was to measure and compare the performances of the algorithms and graph analysis frameworks, on specific 2D map population demographics, as the pathfinding algorithms sought shortest paths in the 2D grid map samples, which helped me answer this study's main research question.

A population of two thousand 2D maps was computer random-generated, 1000 per each of the two desired demographic groups: (a) *high complexity* maps, and (b) *low complexity* maps. These 2D grid maps were generated via computer random number generation. Randomization is required in true experimental designs (Maertens & Barrett, 2013), and random sample selection reduces threats to internal validity by eliminating

sample selection bias (Rooney et al., 2016). Contrast this to quasiexperimental research which does not benefit from randomization (Krishnan & Sitaraman, 2013) and therefore is not as useful at making causal inference. Since this study sought causal inference between the aforementioned variables of interest, and since this was an experimental study, it was logical to use randomization in 2D map generation.

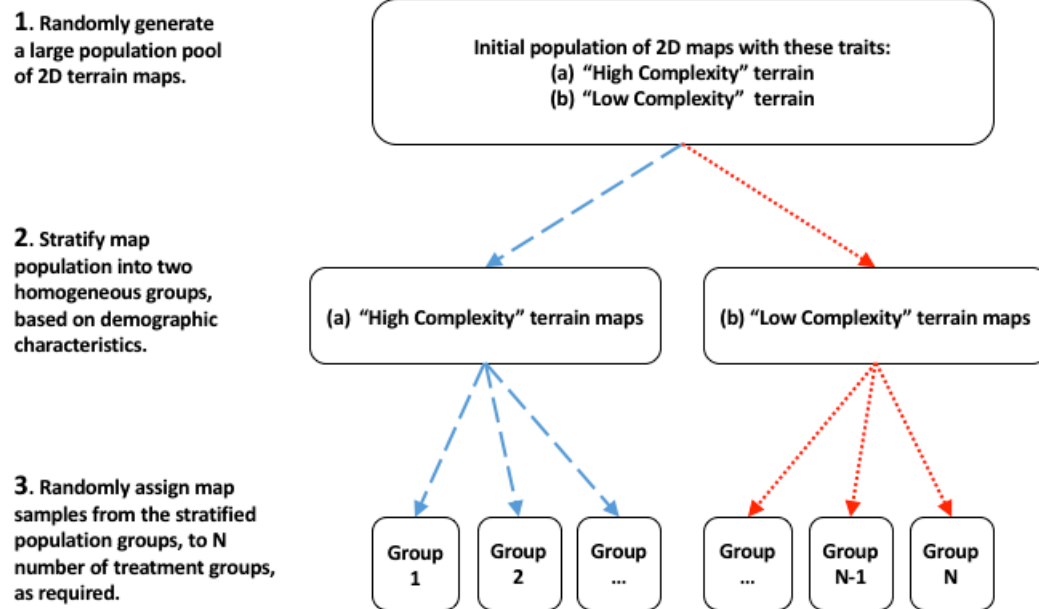
From the large initial population pool of random 2D maps, each map was stratified based on desired demographic characteristics. Stratification of samples into homogenous groups (e.g., by demography) for subsequent random sampling in randomized control trials, improves internal validity (Ariel et al., 2016). Demographic group membership was based on the map complexity property (i.e., *high* vs. *low* complexity) of the 2D map samples (which represented small-world networks), as depicted in Table 9.

Table 9

*Small-World Network Properties of the 2D Map Samples*

Map Complexity (structure)	Adjacency Matrix Dimensions	Random Rewiring Coefficient	Number of Links per Node
1. High (circular lattice)	1000 × 1000	0.25 %	2
2. Low (circular lattice)	200 × 200	0.5 %	2





*Figure 16.* Population and sample stratification plan.

Next, from each of the population demographic subgroups, random assignment was used to allocate map samples from the population subgroups to the target experimental treatment groups, as depicted in the sample stratification plan in Figure 16. Once the samples were assigned, each framework could then perform comparative algorithmic pathfinding performance tests on those map samples, using the relevant pathfinding algorithm. The selection criteria for this experiment was consistent with the 2D map characteristics and population that are the focus of this study. Stratification of the random samples into homogeneous groups was useful because some population demographic groups may behave differently to experimental treatments, as was noted in a quantitative experimental study by Krauss et al. (2013), who compared treatment effects on stratified sample groups.

An *a priori* power analysis using G\*Power 3.1 was conducted to determine the appropriate sample size. G\*Power is a freely available, statistical software program that can be used to conduct *a priori* sample size analyses (Faul, Erdfelder, Buchner, & Lang, 2009). G\*Power supports two different types of MANOVA *a priori* effect size calculations relevant to this study: (a) *main effects*, and (b) *interaction effects*. According to Fritz, Cox, and MacKinnon (2015) it is appropriate to calculate sample size, *a priori*, using multiple predictors, as this can reduce standard error. Both the MANOVA main effects *a priori* sample size, and MANOVA interaction effects *a priori* sample size calculations are described next.

For MANOVA interaction effects calculations, one may use the MANOVA "Special effects and interactions" option in the G\*Power GUI. Using this option, with an *a priori* medium effect size ( $ES$ ) = 0.1, power (i.e.  $1 - \beta$ ) = 0.8, with 12 groups (i.e., the number of experimental treatment groups), 3 predictor (i.e. independent) categorical variables, and 2 response (i.e., dependent) variables, tested at an alpha ( $p$ ) level = 0.05, would indicate *interaction effects* significance. The G\*Power analysis indicated a sample size of 72 (per treatment group) is sufficient to achieve the desired power level, given the above parameters. Increasing the sample size to 144 increases power to .99.

For MANOVA main effects calculations for the independent categorical variable *pathfinding algorithms*, one may use the MANOVA "Global effects" option in the G\*Power GUI. The independent variable pathfinding algorithms has three groups (i.e., each "level" of a categorical factor is called a "group" for this purpose), and its main effects sample size was calculated as follows. Using an *a priori* medium effect size ( $ES$ )

= 0.1, with power (i.e.  $1 - \beta$ ) = 0.8, with three groups (i.e., three "levels" of the categorical independent variable "pathfinding algorithms"), and two response (i.e., dependent) variables, tested at an alpha ( $p$ ) level = 0.05, would indicate *main effects* significance for the categorical independent variable pathfinding algorithm. The G\*Power analysis indicated that a sample size of 63 (per treatment group) would be sufficient to achieve the desired power level given the above parameters. Increasing the sample size to 129 increases power to .99.

Similarly, the categorical independent variables *graph analysis framework* and *map complexity* each have 2 groups (i.e., "levels"), and their main effects sample sizes were both calculated as follows. Using the "MANOVA: Global effects" option, an *a priori* medium effect size ( $ES$ ) = 0.1, with power (i.e.  $1 - \beta$ ) = 0.8, with two groups (i.e., two "levels" in both of the categorical independent variables of interest), and two response (i.e., dependent) variables, tested at an alpha ( $p$ ) level = 0.05, would indicate *main effects* significance for the categorical independent variables *graph analysis framework*, and *map complexity*, respectively. The G\*Power analysis indicated that a sample size of 100 (per treatment group) would be sufficient to achieve the desired power level, given the above parameters. Increasing the sample size to 218 increases power to .99. All G\*Power results are summarized next in Table 10.

Table 10

*Recommended Sample Sizes: Summary of G\*Power Inputs and Results*

G*Power Calculation	G*Power GUI Input Parameters	Minimum Sample Size Needed for Power = 0.8	Sample Size Required to Increase Power to 0.99
1. MANOVA interaction effects	Effect Size (ES) = 0.1, alpha ( $p$ ) = 0.05, Power (i.e. $1 - \beta$ ) = 0.8, groups = 12 predictors = 3 response variables = 2	72 (per group)	144 (per group)
2. MANOVA global effects for variable <i>Pathfinding algorithm</i>	Effect Size (ES) = 0.1 alpha ( $p$ ) = 0.05 Power (i.e. $1 - \beta$ ) = 0.8 groups (levels) = 3 response variables = 2	63 (per group)	129 (per group)
3. MANOVA global effects for variable <i>Graph Analysis Framework</i>	Effect Size (ES) = 0.1 alpha ( $p$ ) = 0.05 Power (i.e. $1 - \beta$ ) = 0.8 groups (levels) = 2 response variables = 2	100 (per group)	218 (per group)
4. MANOVA global effects for variable <i>Map Complexity</i>	Effect Size (ES) = 0.1 alpha ( $p$ ) = 0.05 Power (i.e. $1 - \beta$ ) = 0.8 groups (levels) = 2 response variables = 2	100 (per group)	218 (per group)

The resulting G\*Power output for MANOVA interaction effects and main effects calculations are depicted next in Figures 21, 22, and 23.

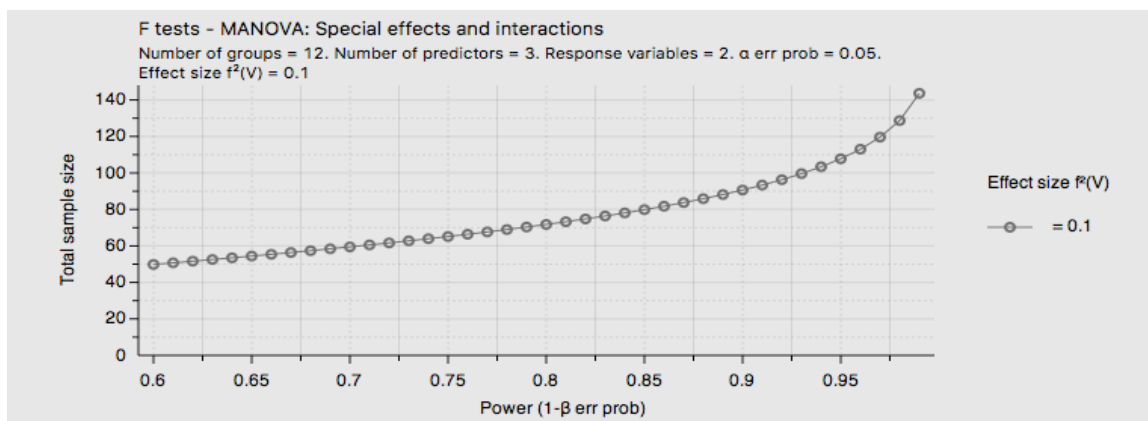


Figure 17. MANOVA interaction effects: 12 Groups, 3 IVs, and 2 DVs.

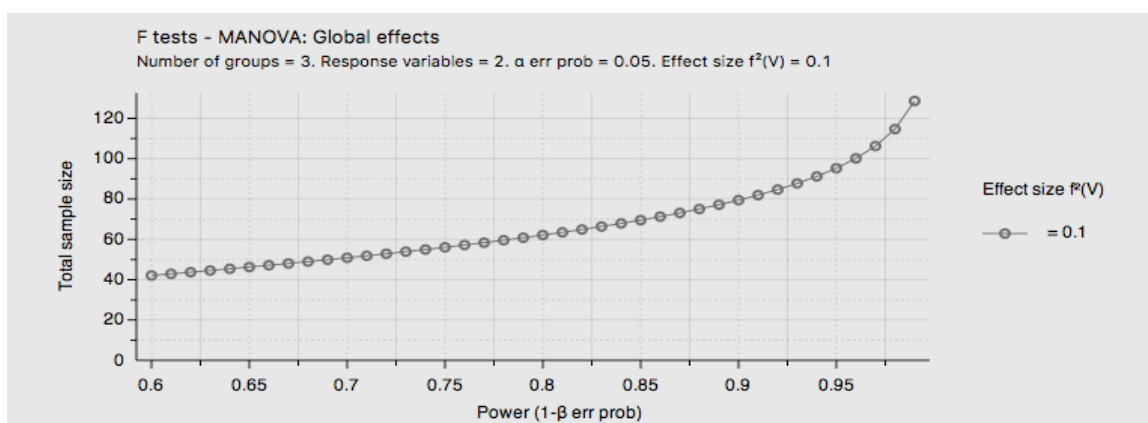


Figure 18. MANOVA main effects for IV algorithm (with three levels).

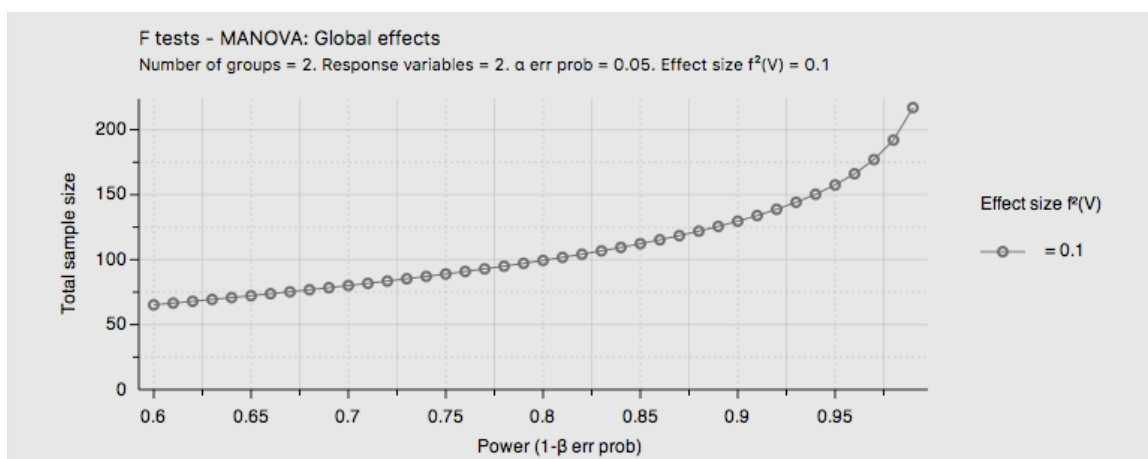


Figure 19. MANOVA main effects for IVs framework and map complexity (each with two levels).

In empirical software research, using *alpha* values of  $p = .05$  to guard against Type I error, and power = .8 (i.e.,  $1 - \beta$ ) to guard against Type II error, is not unusual, and in fact may even be considered "traditional" (Dybå, Kampenes, & Sjøberg, 2006, p. 746). According to Faul et al. (2009), Cohen's  $f^2$  serves as the effect size measure in F-tests, such as MANOVA, where  $f^2$  values of ".02, .15, and .35 can be called 'small,' 'medium,' and 'large' effects, respectively" (p. 1155). These same values were confirmed in the seminal work of Cohen (1992). The use of an *a priori* medium effect size for MANOVA analyses ( $f^2 = .1$ ) is appropriate for this study. The selected medium effect size was based on review of the following two complex network-related articles.

In a graph-theoretic study of small-world biological networks by Hwang, Hallquist, and Luna (2013), they successfully utilized Cohen-style small and medium effect sizes (pp. 2384, 2386), and their findings suggested they could successfully detect efficient communication network hubs of information transmission in biological small-world networks, initially present in childhood, that remain stable well into adulthood (p. 2391). In a study of emotional intelligence by Fernández-Berrocal, Cabello, Castillo, and Extremera (2012), they successfully utilized Cohen-style small and medium effects sizes (pp. 82, 83), and their findings suggested that age and gender did not play a large role emotional intelligence, even though women are generally more concerned with constructing satisfying social networks than men (p. 79).

As noted earlier, an initial population of several thousand computer random-generated 2D maps was created, and then evenly stratified into the aforementioned demographic groups: (a) *high complexity* maps; and (b) *low complexity* maps, thus

yielding many homogeneous sample 2D maps per demographic group, available for subsequent random assignment to treatment groups, as needed. A visual example of the random sample assignment process used in this study is depicted in Figure 20.

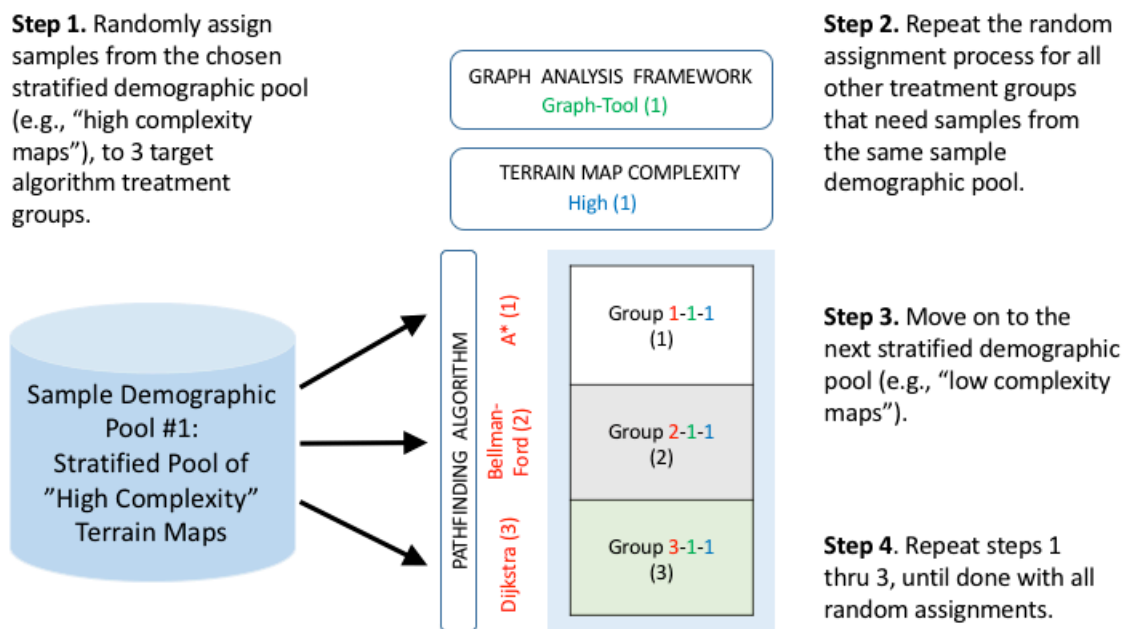


Figure 20. An example of random assignment used in this study.

There is precedent for using computer random map generation in comparative pathfinding algorithm research. In a study of robot pathfinding simulators by Alotaibi and Al-Rawi (2016), they used computer random map generation to create 2D maps of variable sizes and grid cell occupancy ratios, to compare runtime pathfinding algorithm performance (in elapsed seconds). Their findings suggested that writing a computer program to generate random maps, instead of the researchers creating maps by hand, saved the researchers time, allowing them to focus on writing or using pathfinding algorithms, not on creating maps for pathfinding tests (p. 145). Similarly, a study by

Arcuri and Briand (2014) confirmed that it is not unusual to use random sample generation in algorithm studies, since such studies generally do not involve human participants. One weakness with both papers was the lack of instructions on how to integrate test algorithms into a specific target test framework. Another weakness was no mention of the reality that many pathfinding algorithms already exist, in free or open source graph analysis frameworks. No mention of how to integrate their test tools with said graph analysis frameworks was provided, nor was there any statistical output provided, nor was there any mention of the statistical methods used to quantitatively compare algorithms. These statistical gaps may be filled by this quantitative study.

As mentioned earlier in the G\*Power calculations, between 63 to 218 samples were recommended by G\*Power to achieve the power levels from .8 to .99, depending on the specific MANOVA analysis required. To ensure that I had more than enough samples to maintain the power level of at least .8, this study used 150 randomly selected sample 2D maps, per each of the 12 treatment groups. This meant  $150 \times 12 = 1,800$  samples would be utilized. Half originated from the *high complexity* maps demographic group, and the other half originated from the *low complexity* maps demographic group. Several hundred samples remained, unselected, in the demographically stratified sample groups. This was by design. They were held in reserve in case extra samples were required (i.e., to replace missing data, or to replace data outliers, as appropriate).

A study by Nunn, Jordán, McCabe, Verdolin, and Fewell (2015) demonstrated that using random assignment was a valid technique to test and evaluate experimental treatment outcomes between different sample demographics in quantitative studies. As



the map samples used in this study were 100% computer random generated, if necessary it would not have been difficult to random-generate more 2D map samples, as I had full programmatic control over the 2D random map generator program.

### **Ethical Research**

Ethical research is an important part of experimental information technology research. Reasons for this include ethical issues related to data collection, data interpretation, patient consent, privacy and the de-identification (obfuscation) of private patient data (Slade & Prinsloo, 2013). This means researchers should seek to add to social welfare, not to detract from it by exploiting their test subjects' private data.

Twenty-first century data gathering techniques now include online questionnaires and surveys. Yet despite the advancements in data gathering technology, researchers still have ethical and legal issues to consider regarding research participation and data privacy that have not disappeared simply because of new and convenient data gathering techniques (Kaye, Whitley, Lund, Morrison, Teare, & Melham, 2015). Ethical concerns are not unique to research performed in the U.S. more broadly, or at U.S. universities more specifically. In a 2013 Canadian study of massive open online courses (MOOCs), researchers found that although many research studies used publically available data derived from MOOC research, only a small percentage considered the ethical issues of using such data (Liyanagunawardena, Adams, & Williams, 2013). In a 2014 South African study of geographical information sciences (GIS) education frameworks, ethics are taught to university students as part of the body of knowledge (BoK) in the interest of improving the future South African GIS workforce (du Plessis & Van Niekerk, 2014).

While there is the potential for bias between researchers and subjects by inadvertently influencing the responses of participants, or by not maintaining the data privacy of test subjects, since my study does not rely upon human subjects, nor data derived from human subjects, there is no possibility that private data from my test subjects would ever be lost or stolen, because there is no such data. The test samples are random computer generated 2D grid maps. Because these computer-generated maps are not human, and have no private data, there is nothing of personal value to be lost or stolen, therefore ethical concerns over private data use (or misuse) are prevented and mitigated. Finally, although this experiment did not rely on human subjects, or human-derived personal data, I still worked with the IRB to obtain IRB approval (approval number 03-10-17-0469285), prior to performing official data collection.

### **Data Collection**

#### **Instruments**

The research instruments used in this study gathered data on the two dependent variables (a) the elapsed time (in seconds), and (b) memory consumption (in megabytes). The data were gathered by the me, in person, while running the algorithm benchmark tests on my personal laptop computer, in a controlled experiment environment. Randomized controlled trials (RCTs) are often considered the "gold standard" of clinical trials since they provide the researcher the ability to assess the value and efficacy of multiple treatments (Wildiers et al., 2013). This study was an experiment and measured the efficacy of algorithmic treatments applied to random-generated 2D grid maps, in alignment with, and to answer the main research question of this study.

Eight instruments were used in this study (a) the pathfinding algorithms (three per graph analysis framework, for a total of six algorithm instruments); (b) one elapsed time counter; and (c) one memory profiler. See Table 11 for a summary of the eight instruments utilized in this study.

Table 11

*List of Instruments Used and their Validity and Reliability References*

Eight Experiment Instruments	Methods Used to Verify Validity and Reliability
Six Pathfinding Algorithms:	Instruments were verified by me in a pilot test, using the Wilcoxon "signed ranks" statistic.
<i>Graph-Tool</i>	
(1) A* algorithm	Scholarly literature supporting the chosen statistic:
(2) Bellman-Ford algorithm	(a) Dybå, Kampenes, and Sjøberg (2006)
(3) Dijkstra algorithm	(b) Bezerra, Goldbarg, Goldbarg, and Buriol (2013)
	(c) Arcuri and Briand (2014)
<i>Network X</i>	(d) Hric, Peixoto, and Fortunato (2016)
(4) A* algorithm	(e) Taylor et al. (2016)
(5) Bellman-Ford algorithm	(f) Vegas, Apa, and Juristo (2016)
(6) Dijkstra algorithm	
One Elapsed Time Counter:	Corroborating scholarly literature:
(7) Python: <i>TimeIt</i>	(a) Akeret, Gamper, Amara, and Refregier (2015)
	(b) Gorelick and Ozsvald (2014)
	(c) Pettengill et al. (2016)
	(d) Schreier (2017)
	(e) Steininger, Greiner, Beaujean, and Enßlin (2016)
One Memory Consumption Counter:	Corroborating scholarly literature:
(8) Python: <i>memory_profiler</i>	(a) Dunn and Weissman (2016)
	(b) Gorelick and Ozsvald (2014)
	(c) Li, Zhou, and Liu (2012)
	(d) Rossant and Harris (2013)
	(e) Murphy, O'Connell, Cox, and Schulz-Trieglaff (2015)

The algorithms themselves were already discussed in detail in Section 1 (Review of the Literature) and in Section 2 (Research Design). All eight instruments discussed next are also described in the appendices (A through H). Table 12 lists the instruments,

and their appendix reference locations. The subsequent paragraphs discuss how the validity and reliability of the six algorithm instruments were measured. Finally, a discussion of the other two test instruments used in this study concludes this section.

Table 12

*The 8 Instruments Used in this Study and their Reference Locations*

Instruments (8 total)	Appendix
1. Graph-Tool: A* algorithm	Appendix A
2. Graph-Tool: Bellman-Ford algorithm	Appendix B
3. Graph-Tool: Dijkstra algorithm	Appendix C
4. Network X: A* algorithm	Appendix D
5. Network X: Bellman-Ford algorithm	Appendix E
6. Network X: Dijkstra algorithm	Appendix F
7. Python: TimeIt	Appendix G
8. Python: memory_profiler	Appendix H

The instruments are related as follows. First, the six pathfinding algorithm instruments (described in Appendices A through F) seek the shortest paths for each of the input 2D map samples; however, they do not measure how long it takes, nor how much memory was consumed to find these paths. The elapsed time instrument, and the memory consumption instrument measured the time and memory required, respectively, by each pathfinding algorithm instrument, per input 2D map sample. Figure 21 visually depicts this relationship.

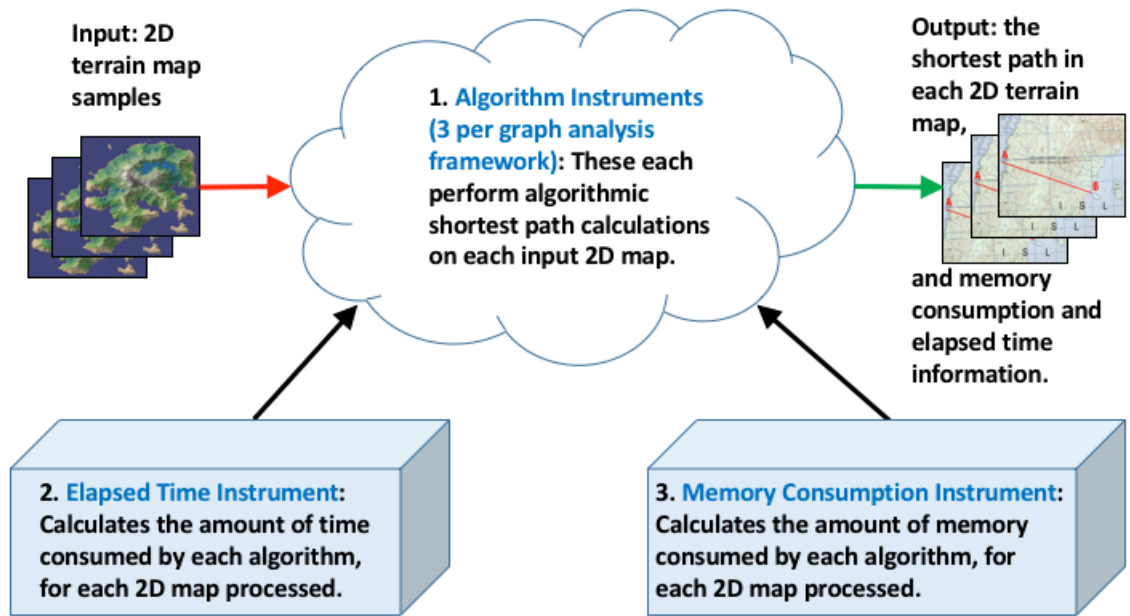


Figure 21. The relationships between this study's instrumentation.

Since no peer-reviewed literature could be found specifically describing the validity or reliability of the six pathfinding algorithms (i.e., three algorithms per graph analysis framework) used in this study, I verified the reliability and validity of the six algorithm instruments in a pilot test, using the Wilcoxon Signed Ranks statistical test, with sample size  $n=150$ , per algorithm instrument. One iteration of an algorithm instrument test was compared against another subsequent iteration, using the same input 2D map samples, to generate the "repeated measure" used by the Wilcoxon Signed Ranks statistic (with  $p = .05$  to guard against Type I error). If the results between groups were statistically similar for each tested algorithm instrument, then for this study that algorithm instrument was considered reliable and valid (i.e., with a reliable instrument, similar inputs should yield correspondingly similar outputs). This process was repeated for all six

algorithm instruments (i.e., three algorithms, per graph analysis framework). The six algorithm instruments are also described in Appendices A through F, and listed in Table 12. The next few paragraphs describe why this method was selected to test the reliability and validity of the six algorithm instruments.

Software validity, broadly speaking, is the ability of software to produce the result it was intended to produce (do Carmo Machado, McGregor, Cavalcanti, & De Almeida, 2014). Software reliability is the ability to consistently obtain similar outputs given similar inputs (do Carmo Machado, McGregor, Cavalcanti, & De Almeida, 2014), also known as test-retest reliability. As described in detail by Taylor et al. (2016), and by Hric, Peixoto, and Fortunato (2016), software validity and reliability can be verified by performing a statistical analysis on the output of the software under test. Furthermore, as recommended by Arcuri and Briand (2014, p. 220), statistical analyses are the preferred method for verifying algorithm validity and reliability. This is particularly important when performing empirical software engineering research (Dybå, Kampenes, & Sjøberg, 2006), such as the comparative algorithm research performed in this study.

Each of the six algorithms tested in this study were interventions (i.e., treatments) whose performances against the input 2D map samples were measured in terms of elapsed time and memory consumption. Researchers Hayes and Preacher (2014) described the utility of using multi-category independent variables in experiments aimed at inferencing causality. In this fashion, for this study, *pathfinding algorithm* was an independent, categorical variable, with 3 levels, where each level represented one of the

pathfinding algorithms commonly supported by the graph analysis frameworks compared in this study.

In a review of 92 different peer-reviewed, controlled software engineering experiments, authors Dybå, Kampenes, and Sjøberg (2006) discovered the four most popular statistical methods used to verify reliability and validity were (a) ANOVA, (b) *t*-test, (c) Wilcoxon, and (d) Mann-Whitney *U*-test (pp. 748-749). Per Hoare and Hoe (2013, p. 51), there are several varieties of the *t*-test, but its main purpose is to check for differences in the means between observations. The Mann-Whitney *U*-test is the nonparametric version of the independent (unrelated pairs) *t*-test. The Wilcoxon *signed ranks* test is the non-parametric version of the related pairs *t*-test. Using an appropriate sample size in statistical analyses is important. The mean sample sizes used in the aforementioned controlled experiments discussed by Dybå, Kampenes, and Sjøberg (2006) were: (a) ANOVA: 79 samples; (b) *t*-test: 34 samples; (c) Wilcoxon: 40 samples; and (d) Mann-Whitney: 34 samples (p. 749). The researchers Dybå et al. (2006) caution against using too many samples, because certain studies may, misleadingly, show significant results if the input sample sizes are too large (p. 752).

Regarding instrument reliability, per studies by Paiva et al. (2014), and by Bezerra, Goldbarg, Goldbarg, and Buriol (2013), using methods such as, but not limited to the paired *t*-test, ANOVA, Wilcoxon, and/or Mann-Whitney *U*-test, allows researchers to validate test-retest reliability. Similarly, in separate research by Raz, Bar-Haim, Sadeh, and Dan (2014, p. 112), and by Zaglia (2013), both papers also suggested using the *t*-test, Mann-Whitney *U*-Test, and/or Wilcoxon test as methods for assessing differences in

experiments, again to validate test-retest reliability. In a separate analysis of statistical tests used in algorithm research by Arcuri and Briand (2014), they reported that commonly used statistical methods in algorithm research are the *t*-test, Wilcoxon and the Mann-Whitney *U*-test (p. 228). One drawback to using parametric statistical tests such as *t*-test, or ANOVA, is the required distribution normal that the underlying data must meet in order to not violate those tests (Kitchenham et al., 2002). Nonparametric tests, like the Mann-Whitney *U*-Test and the Wilcoxon *signed ranks* test, are more flexible in this regard than the parametric versions since the input data need not be normally distributed; however, they generally require larger samples sizes (Arcuri & Briand, 2014). Fortunately for this study generating a large 2D map population (and samples thereof) was not problematic, since the 2D maps were local computer random-generated.

To test algorithm reliability, one recommended approach is to measure statistical differences between test runs (Arcuri & Briand, 2014; Taylor et al., 2016). For nonparametric statistical methods, recommended sample size, *n*, may range from  $n = 100$ , up to  $n = 1000$  (Arcuri & Briand, 2014, p. 244). Note this was larger than  $n = 34$  and  $n = 40$  for *t*-test and Wilcoxon tests, respectively, reported in the aforementioned study by Dybå, Kampenes, and Sjøberg (2006). If there are no significant statistical differences between test runs, then algorithm performance can be considered reliable and valid (Arcuri & Briand, 2014; Taylor et al., 2016). Successfully using the Mann-Whitney, Wilcoxon, and/or the *t*-test, for reliability and validity testing by comparing differences between iterations, was also separately confirmed by Sun, Ha, Teh, and Huang (2016), and by Vegas, Apa, and Juristo (2016, p. 128).



In this study, each pathfinding algorithm instrument accepted 2D map samples (one at a time) as input, and then was tasked with finding the shortest path in that map. The path length, and a list of the nodes comprising the shortest path, was provided by the algorithm. The pathfinding results included the elapsed time and memory consumption data, measured on ratio scales (i.e., continuous, quantitative results), using the memory and timing instruments described later (in much detail). Those algorithm instruments were statistically verified for reliability and validity in a pilot test. Reports from the instrumentation pilot tests were included with the final statistical output of this study in Section 3. The chosen algorithm instruments were appropriate for this study because this was an algorithm study, therefore logically, algorithm instruments were needed for analyses in an algorithm study. Administration of the algorithm instruments were performed by me.

Next is a discussion of the instruments used to collect elapsed time (see Appendix G), and memory consumption data (see Appendix H). Because there were two dependent variables for which data needed to be collected (a) elapsed time; and (b) memory consumption, two specialized instruments were used to collect this data. The first instrument, *TimeIt*, calculated elapsed time during pathfinding operations (described in Appendix G); the second instrument, *Memory\_Profiler*, calculated the memory consumption during pathfinding operations (described in Appendix H). Since the computer test programs for this study were written by me in the Python computer language, it was deemed logical to use Python-compatible test instrumentation. These instruments were administered by me, and are described next.

The data instrument used to gather runtime elapsed time results was the *TimeIt* module (Appendix G), which is built-in to Python (Akeret, Gamper, Amara, & Refregier, 2015), and is part of the Python Standard Library (Steininger, Greiner, Beaujean, & Enßlin, 2016). Capturing elapsed time data can be done by writing a few lines of Python code. The following Python code demonstrates the ease of using *TimeIt* to capture elapsed time for a hypothetical Python function that calculates factorials.

```

INPUT PYTHON SCRIPT "timertest.py":
import timeit
import operator

def iterative_factorial(n): return reduce(operator.mul, range(1, n+1))

def main():
    n = 30 # the integer to factorial
    start_time = timeit.default_timer() # get start time
    fact = iterative_factorial(n) # call the worker function
    end_time = timeit.default_timer() # get end time
    elapsed_time = end_time - start_time # calculate elapsed time
    print ("Factorial %d = %d" % (n, fact )) # print results
    print ("Elapsed time = %f seconds" % elapsed_time) # print results

if __name__ == '__main__':
    main() #call the function that does the work

COMMAND-LINE RUN-TIME EXAMPLE AND RESULTING OUTPUT:
docstudy$ python timertest.py
Factorial 30 = 265252859812191058636308480000000
Elapsed time = 0.000699 seconds

```

Figure 22. A python example of time profiling using the TimeIt python module.

The above Python example took 0.000699 seconds to complete a call to the iterative factorial function and return the result. In summary, there are four simple steps to follow when using *TimeIt*: (a) start the timer, (b) call a function whose elapsed time needs measurement, (c) stop the timer, and (d) subtract the start time from the end time to yield the elapsed time. This technique of using *TimeIt* to measure elapsed time was similarly used in this doctoral study to capture elapsed time data for pathfinding

operations performed on each input 2D map sample, per pathfinding algorithm, per graph analysis framework.

The *TimeIt* module was an appropriate instrument to use in this study because it was simple to use (only a few lines of Python code, as shown above), it was freely available (it comes with Python), it was reliable and well-supported in the Python community (Gorelick & Ozsvald, 2014). Administration of the instrument, *TimeIt*, was simple because the programmer has total control over when, and how frequently to use it (Gorelick & Ozsvald, 2014). The instrument, *TimeIt*, is popular with researchers and engineers, and has been widely used in a many problem domains. For example, Akeret, Gamper, Amara, and Refregier (2015) successfully used the *TimeIt* module, repetitively, to record elapsed time performance of a custom just-in-time compiler made for astronomical computations, running on Apple MacBook hardware (similar to the hardware used by the author of this doctoral study), allowing them to monitor and measure runtime performance areas of concern. In another case, Pettengill, Pightling, Baugher, Rand, and Strain (2016), used *TimeIt* to measure runtime performance of gene-distance calculations in their big data genomic study (pp. 3-5). Next, Schreier (2017) used the *TimeIt* module to quantify elapsed time performance of complex computations performed on multigrid matrices (pp. 12-13). Finally, Steininger, Greiner, Beaujean, and Enßlin (2016) used *TimeIt*, repetitively, to measure the runtime performance and scalability of a Python high-performance parallel computing framework. The above cases are real-world examples where *TimeIt* successfully measured elapsed time of computationally critical operations.

The results of *TimeIt* were quantitative, measured in seconds of elapsed time. This was appropriate for this study since quantitative elapsed seconds corresponded with the aforementioned dependent variable *Elapsed Time*. The larger the values reported by *TimeIt*, the more elapsed time has passed for the function or program under test. While I could have written my own elapsed time counter, doing so would have been far beyond the scope of this pathfinding algorithm study.

Finally, the data instrument used to gather memory consumption statistics is the *memory\_profiler* Python code module (described in Appendix H), freely available from the Python Software Foundation and published at the Python Package Index (PyPI) website: <https://pypi.python.org/pypi>. This instrument has been freely available for over a decade (Gorelick & Ozsvald, 2014). By programmatically using Python and *memory\_profiler*, a software engineer can calculate the amount of memory consumed by a Python script (Li, Zhou, & Liu, 2012). In a study of pathfinding algorithms and memory consumption, researchers Salmela and Rivals (2014) suggested that it was possible to measure megabytes of memory consumed during algorithm tests by periodically polling the operating system (OS) for memory consumption data, and this is what *memory\_profiler* does.

The *memory\_profiler* module (see Appendix H), available in Python, calculates memory consumption by querying the underlying OS. It can be called programmatically in Python scripts via its application programmer interface (API), or manually from the command line. Calculating the memory consumed by a Python script can be done by writing a few lines of Python code. In the following code snippet, a Python script named

"Network-X-pathfinding.py" was run using Python's *memory\_profiler* module. This hypothetical script was a test harness which called the Network-X graph analysis framework, instructing it to perform a shortest path test using Dijkstra's algorithm on a 2D map sample (map ID # 31). The example hypothetical Python script output follows.

```
python -m memory_profiler networkx-pathfinding.py "31"
```

Output:			
Line #	Mem usage	Increment	Line Contents
...			
331	9.336 MB	0.00 MB	@profile
332			def NetworkX_Dijkstra(mapId):
...			...
407	75.352 MB	66.016 MB	m = processMap(start,dest,mapId)
408			return m

Figure 23. A python example using the *memory\_profiler* python module.

Note in the above hypothetical example, the amount of memory consumed during the Dijkstra's algorithm test was 66.016 MB, as shown by the value in the *Increment* column. In summary, the memory increment values which resulted from usage of the *memory\_profiler* instrument were the values captured, parsed, summarized, and reported in this doctoral study, as they were relevant to the aforementioned *memory consumed* dependent variable utilized in this study.

Administration of the instrument, *memory\_profiler*, was simple because the programmer has total control over when to use it (Gorelick & Ozsvald, 2014). The instrument was valid and reliable, because it relied on underlying operating system kernel calls (Gorelick & Ozsvald, 2014) to gather the memory information, and has been thoroughly tested. In a quantitative performance study of graph analysis software by Rossant and Harris (2013), they reliably and successfully used *memory\_profiler* to

measure the memory consumed at runtime by an OpenGL-based graph analysis framework. Their *memory\_profiler* findings suggested that more memory efficiency could be gained by their software if it reduced unnecessary array copying during data load and transformation operations (p. 6). In a separate quantitative study, researchers Murphy, O'Connell, Cox, and Schulz-Trieglaff (2015) successfully used *memory\_profiler* to measure memory consumption of software running on a single core computer with 8 GB RAM available (p. 8). Their *memory\_profiler* findings indicated that the most memory-intensive portion of the software they tested occurred during the creation of tree-based data structures (pp. 6, 12-13). In a study that processed large genomic datasets in Python, the authors Dunn and Weissman (2016) also successfully used *memory\_profiler* to measure peak memory usage (pp. 2, 10, 11). Each of the above examples successfully showcased use of *memory\_profiler* to measure computer memory consumption with Python.

The results of *memory\_profiler* were quantitative, and represented megabytes of memory consumed. This output was appropriate for my study since this data type corresponded to the aforementioned quantitative *memory consumed* dependent variable. The larger the values reported by *memory\_profiler*, the more memory was consumed by the program under test. While I could have written operating system kernel-level code to gather memory statistics, doing so would have been far beyond the scope of this pathfinding algorithm study.

All data resulting from this study will be retained by the author of this study, and may be available upon request. Additionally, source code is available on GitHub

(<https://github.com>) for free download, in a new online GitHub project created by me, specifically for this doctoral study. See the instrument descriptions in the appendices for more details on where to find the downloadable source code.

### *Validity and Reliability*

As discussed earlier, I statistically tested the validity and reliability of each of the algorithm instruments, per graph analysis framework, used in this study, in a pilot test. The statistical output from the pilot test of the algorithm instruments is noted in Section 3 of this doctoral study.

The instruments *TimeIt*, and *memory\_profiler* (see Appendices G and H, respectively) are both valid and reliable, as already noted in their respective instrumentation descriptions above, and because they come with Python (now a 20-year-old computer language), or are official Python extensions, and rely on underlying operating system kernel calls to calculate elapsed time, or memory consumption, respectively (Gorelick & Ozsvald, 2014). This means that I did not need to write custom kernel-level code to measure elapsed time and memory consumption at the operating system (OS) level, as writing OS code was beyond the scope of this pathfinding algorithm study. Memory and time profilers are valid software engineering tools because they allow engineers to quickly identify performance problems and bottlenecks in complicated computing environments, especially considering that in some cases there are no other tools than could successfully perform this task (Yamamoto, Ono, Nakashima, & Hirai, 2016). Finally, since no human intervention or post data collection manipulation of results were manually performed (i.e., I merely recorded the results generated by the

aforementioned instruments), researcher bias, subject bias, and data coding interpretation bias did not impact results, as the results were straightforward. No manual hand-coding of responses or results, no interviews, and no subject-to-researcher human interaction was required nor was possible. The use of programmatically obtained data (instead of human interaction) eliminated the possibility for subject bias in data collection, which further enhanced the internal validity of this study.

### **Data Collection Technique**

In experimental research, using randomized controlled trials (RCTs) (also known as "clinical trials") can be a high-cost endeavor due to the time intensive nature of the testing process and the meticulous manner in which data must be collected and recorded in quantitative experiments (Dunn, Arslanian-Engoren, DeKoekkoek, Jadack, & Scott, 2015). However, writing computer programs to create computer random generated 2D map samples on demand, as was performed in this study, had several benefits. First, using local computer random-generated content helped reduce the financial and time burdens of data collection because no human interaction was required (no interviews needed, no time spent traveling to/from interview locales), and data privacy storage concerns did not exist (because no personally identifiable information was used). This liberated me from the burden of human interaction, and reduced the possibility of sample bias, thereby permitting me to focus more energy on the research study itself (Liapis, Yannakakis, & Togelius, 2015, p. 5), and less on administrative-oriented tasks.

A second benefit to using computer generated 2D map samples, such as the ones generated and used by this study, was the ease that such abstract 2D map samples can be



mapped directly to an array of integers equal to the number of grids on the map, where each integer represented or determined the contents of a single map tile, e.g., a forest, a mountain, a river, a plain, an obstacle, and so forth (Liapis, Yannakakis, & Togelius, 2015, p. 9). A hypothetical example map, and its abstraction (i.e., its numeric genotype) is depicted in Figure 24. The map terrain (highly abstracted) is on the left, and its numeric abstraction (the map's genotype) is shown on the right. In this example, note how the value of "0" indicates clear terrain, a "1" represents a diamond obstacle, a "2" represents a star obstacle, and "3" represents a black wall obstacle.

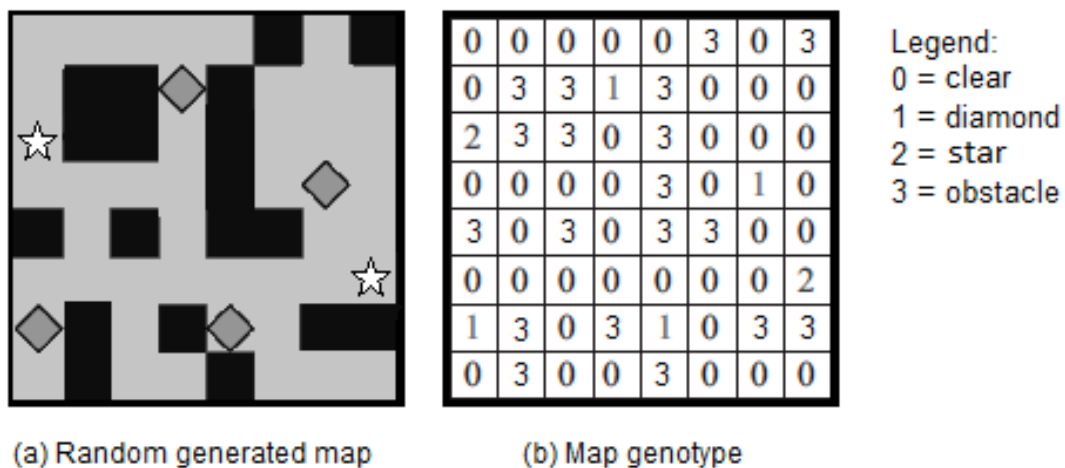


Figure 24. A random generated grid map (left) and its abstracted genotype (right).

One disadvantage with computer random generated 2D map samples is that they are not real world maps. However, in this study, the benefits outweighed the disadvantages because of the aforementioned convenience factor (low cost) and because random generated maps, while not 100% representative of reality, provided the opportunity to test pathfinding algorithms on simulated maps, at a fraction of the time

and cost that testing real pathfinding vehicles at hard-to-access locales (e.g. in enemy territory, under water, on Mars, etc.) would entail.

Once the map genotypes (i.e., map population) were randomly created, they were stratified into the demographic groups used in the study, as described earlier, and random selection was utilized to assign samples to the experimental treatment groups. More samples were available than were used. Testing pathfinding algorithms on randomly generated 2D map samples required installation of the aforementioned graph analysis frameworks on a test machine, which for this study was my MacBook Air laptop. A Python data collection program performed the steps shown in Figure 25, to collect experimental data, and then wrote the resulting data to file (for subsequent analysis).

The data collected were stored on file, in text format, for all 2D map samples tested, and included the following: (a) sample map ID, (b) the path length for the shortest path (if one exists) from source to destination nodes on the input 2D map, (c) pathfinding algorithm tested, (d) graph analysis framework tested, (e) elapsed time, and (f) memory consumed. Because the experimental data collected was stored in text file on the hard drive of my computer running the experiments, the experimental data were parsed using a second Python program to prepare it for subsequent statistical analyses. This data processing Python program performed the steps depicted in Figure 26.

Once the data were parsed and cleaned, it was collated into the final master text file for SPSS, and was later imported into SPSS for statistical analyses (e.g., MANOVA). All data resulting from this study will be retained by the author of this study, and may be available upon request.

1. Generate the 2D map population, with the desired demographic traits.
2. Stratify the population into demographic groups, based on traits.
3. Randomly pull a map sample from a target demographic group.
4. Determine which experimental treatment group gets the sample (based on the target graph analysis framework and pathfinding algorithm).
5. Start the elapsed time counter.
6. Send 2D map sample to Python script to perform pathfinding operations.
7. Wait for the Python tester script to complete pathfinding operations.
8. When the Python script completes, stop the elapsed time counter.
9. Write the pathfinding results and elapsed time data to file.
10. Start the memory consumption counter.
11. Send the same 2D map to Python script to perform pathfinding operations.
12. Wait for the Python tester script to complete pathfinding operations.
13. When the Python script completes, stop the memory consumption counter.
14. Write the pathfinding results and memory consumption data to file.
15. Tear down test iteration, clean up system, and prepare for next iteration.
16. Go back to step 3, repeat until finished processing all map samples.|

*Figure 25.* Outline of python program to collect experimental data.

1. Open output file(s) created by the prior Python data collection script
2. Parse the independent and dependent variables and their data
3. Save and append the data to a master CSV file for SPSS input
4. Repeat steps 1 through 3 until all data files have been processed

*Figure 26.* Outline of python program to parse experimental data.

In all maps, for simplicity, all edges (arcs) were each assigned a uniform weight of 1.0. This was because, as noted in Vesović, Smiljanić, and Kostić (2016), Dijkstra's algorithm cannot handle negative edge weights. Therefore, to be able to fairly test all three pathfinding algorithms, all random generated maps used the same positive, equal, edge weight of 1.0. Use of varying edge weights could be a topic for further research.

### **Data Analysis Technique**

The data analysis for this quantitative study focused on determining statistical significance regarding the research question: What is the relationship between pathfinding algorithms, graph analysis frameworks, 2D map complexity, elapsed time, and computer memory consumed?

The hypotheses of this study were tested to identify causal inference between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumed. Next are my hypotheses for this study.

Null Hypothesis ( $H_0$ ): There is no relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumed.

Alternative Hypothesis ( $H_a$ ): There is a relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time and computer memory consumed.

The sample data consisted of random computer generated, 2D grid maps (mathematically represented as adjacency matrices) that were stratified into the demographic categories mentioned in Section 2. Randomized trials are the "gold standard" in causal inference due to their strong statistical supporting evidence (Clair, Cook, & Hallberg, 2014, p. 311).

The collected data came from the aforementioned experimental treatment groups, as discussed earlier. The computer random-generated 2D map samples were stratified into demographic groups based on map complexity, and from these demographically homogenous groups, random sample assignment was used to allocate samples to each of the 12 groups for subsequent algorithm testing. According to Ariel et al. (2016), stratification of samples into homogenous groups (by demographic traits) for subsequent random sampling in randomized control trials, significantly improves internal validity.

Statistical analysis began with descriptive statistical analyses on the population of 2D map samples in order to verify homogeneity and calculate other descriptive characteristics of the input population. In a comparative study of shortest paths used for school route travel in urban vs. non-urban environments, researchers Buliung, Larsen,

Faulkner, and Stone (2013), demonstrated the benefits of using descriptive statistics in quantitative research. Additionally, output data were cleaned and transformed so that outliers which did not fit the desired demographics were modified accordingly. The samples removed or modified in this fashion were replaced (or modified) with the desired demographic characteristics to maintain equal sized groups. Using strict selection criterion and grouping demographics aids the researcher when testing hypotheses on samples that may exhibit clinically significant reactions to experimentally applied treatments (Drislane et al., 2014). Additionally, as discussed by Marozzi (2016, p. 42), removal of outliers improves the robustness of MANOVA calculations.

Following the descriptive statistical analysis, the factorial multivariate analysis of variance (MANOVA) statistical procedure was conducted. MANOVA is a standard statistical procedure to use when multiple dependent variables are present (Tonidandel & LeBreton, 2013). In a study of pathfinding in complex graphs, researchers Dawson, Munzner, and McGrenere (2015) successfully used multiple regression to evaluate the impact of multiple factors on their dependent variable. However, multiple regression is not used in this study because multiple regression supports only one dependent variable (Dawson, Munzner, & McGrenere, 2015; Mertler & Reinhart, 2017). Since this study intentionally uses multiple dependent variables, multiple regression was clearly rejected in favor of MANOVA, as MANOVA supports multiple dependent variables (Blasco-Arcas, Hernandez-Ortega, & Jimenez-Martinez, 2013; Puckett, Eggleston, Kerr, & Luetlich, 2014).

One weakness with MANOVA, however, is that when there is a significant correlation between variables, MANOVA has limits on its ability to discriminate the effects between multiple dependent variables (Tonidandel & LeBreton, 2013). To address this weakness in MANOVA, when significant MANOVA effects were detected, subsequent follow-up analyses were conducted using univariate ANOVAs with Scheffe's post hoc test (Marsh-Hunkin, Gochfeld, & Slattery, 2013).

Regarding missing data, there was no missing data in this study. This study was a controlled experiment, and if results were missing or was somehow inappropriate, it could be removed and a new one either (a) pulled from its demographic group surplus, or (b) a new one could be programmatically created with the desired demographic characteristics, as needed. This helped maintain equal sized groups. More practically, keeping group sizes equal, while not strictly required for MANOVA, was recommended because this helped avoid problems in the statistical analysis if assumptions related to the equality of the covariance matrices (i.e., homoscedasticity) were not met, as described in Field (2013, p. 194), and by Howitt and Cramer (2014, p. 291).

MANOVA analysis assumes the following are satisfied: (a) dependent variables must be continuous data types, not discrete or categorical; and independent variables must be categorical, not continuous -- this was handled during organization and test setup; (b) there is at a minimum at least one independent variable with at least two categories -- this was handled during organization and test setup; (c) the sample size is adequate -- as discussed in detail in Section 2, this experiment uses 150 samples ( $n = 150$ ) per treatment group (12 groups total), yielding total  $n = 150 \times 12 = 1,800$  samples;

(d) independence of observations and use of random sampling (Mertler & Reinhart, 2017, p. 129) -- which was resolved early, by utilizing proper theory and study design (Tabachnick & Fidell, 2014, p. 291); (e) univariate normality -- which was detected through boxplots, histograms, P-P plots, Q-Q plots, or normal curve inspection (Korkmaz, Goksuluk, & Zararsiz, 2014, p. 10), and then mitigated by data transformations or outlier removal (Tabachnick & Fidell, 2014, pp. 110, 117); (f) multivariate normality -- which was detected using Mahalanobis distance (Korkmaz, Goksuluk, & Zararsiz, 2014, p. 10; Mertler & Reinhart, 2017, p. 52; Tabachnick & Fidell, 2014, pp. 108-109), and mitigated through data transformations or outlier removal (Tabachnick & Fidell, 2014, pp. 110, 117); (g) linearity between the dependent variables within each treatment group (Gaston, Wilson, Mack, Elliot, & Prapavessis, 2013) -- which was verified with bivariate scatter plots (Hair, Anderson, Babin, & Black, 2010, p. 76; Mayorga & Gleicher, 2013, p. 1526; Veletsianos & Kimmons, 2016, p. 4), or statistical bivariate correlation (Amin, Malik, Kamel, Chooi, & Hussain, 2015, pp. 8-9; White & Perrone-McGovern, 2017, p. 42), and mitigated through data transformations or outlier removal (Tabachnick & Fidell, 2014, pp. 110, 117); and (h) homoscedasticity (Bird & Hadzi-Pavlovic, 2014), also known as *homogeneity of variance* (Tabachnick & Fidell, 2014, p. 120) -- which was detected with bivariate scatter plots, or Box's M test for equality of variance-covariance matrices (Mertler & Reinhart, 2017, p. 36). Finally, while transformations usually mitigate most violations of homoscedasticity (Tabachnick & Fidell, 2014, p. 120), violations of homoscedasticity are *not* fatal to multivariate statistical analyses (Mertler & Reinhart, 2017, p. 130). Tabachnick and Fidell (2014, p.



293) noted that when using equal sample sizes per treatment group, with at least  $n = 100$  per group (i.e., a *large* sample size, p. 114), robustness of multivariate significance tests are to be expected and one may disregard results of Box's M Test (p. 294). Nonetheless, if homoscedasticity were violated, MANOVA is generally resistant to assumptions violations (Rosa et al., 2016, p. 4), and usage of the more robust Pillai's Trace can be employed when interpreting the MANOVA results, as suggested by Mertler and Reinhart (2017, p. 132); Rosa et al. (2016, p. 4); Tabachnick and Fidell (2014, p. 311); and Warne (2014, p. 6). As a precaution, this study followed the process of removing or transforming outliers early (upstream) during the data screening phase, and always used equal numbers of samples per treatment group, as was strongly recommended by Tabachnick and Fidell (2014, p. 316) to simplify and improve later (downstream) multivariate statistical analyses and inferential results (p. 316).

The data were analyzed using IBM SPSS version 23. Using an *a priori* medium effect size ( $ES = 0.1$ ), power = 0.8, 12 treatment groups, three predictor (i.e. independent) variables, two response (i.e., dependent) variables, and tested at  $p = 0.05$ , would indicate *main effects* and *interaction effects* significance, as mentioned earlier in the Population and Sampling section.

### **Study Validity**

In the tradition of quantitative science research, research instruments and methods of data collection are tested, controlled and examined for validity (Collins & Cooper, 2014). Validity in the context quantitative research refers to how accurately do the results represent the objective truth. Regarding causal inference, bias influences the validity of

experimental, quantitative studies (Pluye & Hong, 2014). According to Venkatesh, Brown, and Bala (2013) in quantitative research there are three broad categories of validity: (a) content and construct validity (i.e., measurement validity); (b) internal and external validity (i.e., design validity); and (c) statistical conclusion validity (i.e., inferential validity).

Content validity refers to extent that questions posed actually measure the intended construct of research interest (Drost, 2013). The intent of this study was to specifically measure the impact of pathfinding algorithms, graph analysis frameworks, and map complexity on elapsed time and memory consumption, in order to make causal inference. For example, this study does not measure computer central processing unit (CPU) temperature, nor does it measure the refresh rate of computer monitors, as those concerns have no relevance to this study. Instead, the content validity is high in this study because the study only uses tools that specifically measure elapsed time and memory consumption. These tools have existed for over a decade so they have been well tested by the Python development community (Gorelick & Ozsvald, 2014), with newer versions (with bug fixes, enhancements) made available to the public, as needed.

Construct validity refers to the ability of the instruments to measure what they claim to measure (Drost, 2013). This study used instrumentation specifically geared to running algorithm pathfinding tests, measure computer timing and memory consumption. This is because elapsed time and memory consumption are my dependent variables of interest. Reliability is the degree to which the measurements are free from error and are consistent (Lakshmi & Mohideen, 2013). The aforementioned instruments used to

measure elapsed time and memory consumption are reliable because they reliably and repeatedly produce consistent output for a given consistent input (Gorelick & Ozsvald, 2014). Because the instrumentation is reliable, and because the instruments measure what is intended, no more, no less, high construct validity is maintained with the selected instrumentation. The pathfinding algorithm instruments were checked for validity and reliability by me, as mentioned earlier.

Internal validity refers to the ability to draw causal inferences from the data (Neall & Tuckey, 2014). When internal validity is high, one can make a strong case that one variable directly impacts another, hence the importance of internal validity in experimental studies. Internal validity can be increased reducing sample attrition and sample mortality. Additionally, history bias is a threat to internal validity, in that natural life historical events (e.g., death of family member) can cause human subjects to behave in unexpected ways, thereby potentially causing confounding effects. In this study, my samples are not alive, they do not mature, and they do not die, so they did not suffer from the effects of selection mortality or selection history. Selection bias is another threat to internal validity, but this can be reduced or eliminated by using random sampling and sample stratification, which were both utilized in this study.

External validity refers to the ability to generalize the results to other populations and other settings (Henderson, Kimmelman, Fergusson, Grimshaw, & Hackam, 2013; Zohrabi, 2013). Sample bias is a threat to external validity. One can more easily generalize study results if the samples are diverse. This study uses several different sample demographics, thereby providing a heterogeneous population from which to draw

samples. One drawback with using too much sample variety is that if the samples are too varied, yes external validity increases, but this also increases threats to statistical conclusion validity (Luft & Shields, 2014). Therefore a balance must be made between narrow vs. wide sampling strategies, which I followed in this study by utilization of a targeted population, stratified by the demographic criteria of interest.

Statistical conclusion validity is the degree to which conclusions drawn from the data are correct and reasonable (Neill & Tuckey, 2014). Threats to statistical conclusion validity include using samples that exhibit too much or too little heterogeneity, as this may create confounding results. Too much heterogeneity can occur if the sample population is too wide, as discussed earlier. Using variable (inconsistent) experimental procedures presents another threat to statistical conclusion validity because the treatment implementation would be unreliable, and therefore the data derived may be unreliable. This study did not suffer from these threats to statistical conclusion validity because (a) the use of randomly generated, demographically stratified samples; and (b) the experimental procedures were written in Python computer scripts which repeatedly and reliably ran the tests, in an automated fashion, one by one, until all samples were processed.

To prevent the my experience as a professional software engineer from negatively biasing this research, this study relied on existing code frameworks, application programmer interfaces (APIs), and pathfinding algorithm implementations. Therefore, I wrote only *glue code* which connected the test harness to the graph analysis frameworks to collect algorithmic performance data, and therefore let the graph analysis frameworks

do the actual algorithmic pathfinding work using their internal pathfinding algorithm implementations. This approach meant I did not implement the pathfinding algorithms used in this study. This is because, as already mentioned and documented in the literature review in Section 1, many Python developers world-wide use the graph analysis frameworks I quantitatively compared in this study, so a study researching and comparing those popular graph analysis frameworks may be of greater interest to them, than research on pathfinding algorithms specifically implemented by me (especially considering the fact that I have never contributed pathfinding code to any open source projects). To be clear, this empirical, applied study was a quantitative experiment comparing open source pathfinding algorithm code already written and published by others. This study did not compare pathfinding algorithm code implemented by me.

Using and comparing popular algorithm code written by others, thus not limiting this research to algorithm code written specifically by me, reduced author bias, and thereby increased internal validity. By not being the author of the pathfinding code, I was less likely to be biased when collecting and recording the test results. Methods to widen the potential audience for this study can improve its clinical generalization (Henderson, Kimmelman, Fergusson, Grimshaw, & Hackam, 2013). This improvement in generalizability (i.e., external validity) was accomplished by using open source code frameworks, since I assumed more people use the aforementioned open source graph analysis frameworks I compared than would ever use pathfinding algorithm code implemented specifically by me.

The samples used in the study consisted of local computer random-generated 2D maps (adjacency matrices). The sample size was limited to maps that can easily be processed on my laptop. However, these map samples are not representative of all the possible complex maps in the information technology world. Therefore, as is sometimes the case with clinical laboratory tests, any findings from this study may only be limited to similar settings (Zohrabi, 2013). This is due to the limitations of the variety of random computer-generated maps used. However, using more map variety and/or more pathfinding algorithms to increase external validity are valid avenues for further research and is discussed in detail in Section 3 of this study.

### **Transition and Summary**

Section 2 included details of my role as the researcher (and software engineer) of this study, and justification for the quantitative method and chosen experimental design. Furthermore, it described how this study did not require human subjects. Section 2 described this study's use of computer random generated 2D map samples, stratification of those samples based on demographic traits, and subsequent random sample assignment to experimental treatment groups. Random assignment is a hallmark of experimental research, and randomized trials are the "gold standard" in causal inference due to their strong statistical backing (Clair, Cook, & Hallberg, 2014, p. 311). This study is a quantitative experiment with the goal of identifying a causal nexus between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. Section 2 concluded with a presentation of the post data collection and analysis procedures, including a discussion of validity and reliability.

Next, Section 3 consists of a presentation of my findings, a discussion of the applicability and practicality of these findings for software engineers specifically, and to the wider information technology community more broadly, and it concludes with a discussion of implications for positive social change that may emerge from this study's results.

### Section 3: Application to Professional Practice and Implications for Change

Section 3 contains the results of the analysis presented in Section 2. This section includes (a) a brief overview of the study, (b) presentation of findings, (c) discussion of applications to professional practice, (d) discussion social change implications, (e) recommendations for action, (f) recommendations for further study, and (g) personal reflections. I then close the section with a summary and my conclusions.

#### **Overview of Study**

The purpose of this quantitative experimental study was to examine the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and memory consumption, to help software engineers select appropriate pathfinding algorithms for resource-constrained software agents running in complex networks, network dead zones or GPS-denied environments. The target population consisted of local computer random-generated two-dimensional (2D) maps (i.e., adjacency matrices). The three independent variables were (a) pathfinding algorithms; (b) graph analysis frameworks; and (c) map complexity. The two dependent variables were (a) elapsed time; and (b) computer memory consumption. The null hypothesis was rejected, and the alternative hypothesis was accepted. Elapsed time and computer memory consumption are both significantly affected by pathfinding algorithms, graph analysis frameworks, and map complexity.

#### **Presentation of the Findings**

I first reintroduce my quantitative research question, followed by my two hypotheses.



Research Question (RQ): What is the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption?

Null Hypothesis ( $H_0$ ): There is no relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption.

Alternative Hypothesis ( $H_a$ ): There is a relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption.

Analysis of the research question and hypotheses using MANOVA lead me to reject the null hypothesis. There was strong statistical evidence to support a relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. Before I discuss the main effects and interaction effects results of the MANOVA statistic, I first discuss the results of the pilot test (first mentioned in Section 2) that I used to statistically verify the reliability of my algorithm instruments.

### **Pilot Test of the Algorithm Instrumentation**

As I discussed in Section 2, since no peer-reviewed literature could be found specifically describing the validity or reliability of the six pathfinding algorithms from the selected graph analysis frameworks used in this study, I verified the reliability and validity of the six algorithm instruments in a pilot test, using the Wilcoxon Signed Ranks statistic, with 150 map samples per algorithm instrument. As there were six algorithm

instruments, the total sample size  $n$  utilized in the pilot test = (150 samples  $\times$  6 instruments) = 900. The pilot test was a repeated-measures design without intervention. There were six treatment groups (one per algorithm instrument). The population pool consisted of 1000 random generated map samples of the same size (200  $\times$  200 adjacency matrix), all with uniform edge weights of 1.0. Next, each group was assigned 150 randomly selected samples. Each group was tested against one of the six pathfinding algorithms, to find the shortest paths in those 2D map samples, while measuring the elapsed time and memory consumption results. Later, in a separate, second iteration (without intervention), the same map samples, per group, were again tested against the same pathfinding algorithm they were tested with the first time (hence the *repeated-measures* and *no intervention* aspects of the pilot test), and the resulting elapsed time and memory consumption results from the second iteration were also recorded. Afterwards, the results from both iterations, per algorithm, 2D map sample, and experimental group, were compared using the Wilcoxon Signed Ranks statistic. The Wilcoxon Signed Ranks statistic was used with  $p = .05$  to guard against Type I error. Usage of the Wilcoxon Signed Ranks test to verify the validity and reliability of software was recommended separately by Arcuri and Briand (2014); Bezerra, Goldberg, Goldberg, and Buriol (2013); Dybå, Kampenes, and Sjøberg (2006); Hric, Peixoto, and Fortunato (2016); Taylor, et al., (2016); and by Vegas, Apa, and Juristo (2016). The results from the Wilcoxon Signed Ranks pilot test are depicted in Table 13 through Table 18.

Table 13

*Wilcoxon Signed Ranks -- Pilot Test Results for Graph-Tool A\* Instrument*

Graph-Tool: A*	ELAPSED_TIME_2 -	MEMORY_CONSUMED_2 -
	ELAPSED_TIME	MEMORY_CONSUMED
Z	-.433 <sup>a</sup>	-.929 <sup>a</sup>
Asymp. Sig. (2-tailed)	.665	.353

a. Based on negative ranks.

A Wilcoxon Signed Ranks test was conducted to evaluate whether the Graph-Tool A\* algorithm instrument showed statistically different results between test runs, using a repeated measures design (150 samples) with no intervention. As shown in Table 13 above, for elapsed time,  $z(n = 150) = -.433$ , with two-tailed  $p = .665$ , which indicated no significant difference in the amount of elapsed time between either instrument test iteration. For memory consumption,  $z(n = 150) = -.929$ , with two-tailed  $p = .353$ , which indicated no significant difference in the amount of memory consumption between either instrument test iteration.

Table 14

*Wilcoxon Signed Ranks -- Pilot Test Results for Graph-Tool Bellman-Ford Instrument*

Graph-Tool: Bellman-Ford	ELAPSED_TIME_2 -	MEMORY_CONSUMED_2 -
	ELAPSED_TIME	MEMORY_CONSUMED
Z	-.627 <sup>a</sup>	-1.048 <sup>b</sup>
Asymp. Sig. (2-tailed)	.531	.295

a. Based on positive ranks.

b. Based on negative ranks.

A Wilcoxon Signed Ranks test was conducted to evaluate whether the Graph-Tool Bellman-Ford algorithm instrument showed statistically different results between test runs, using a repeated measures design (150 samples) with no intervention. As shown in Table 14 above, for elapsed time,  $z(n = 150) = -.627$ , with two-tailed  $p = .531$ , which indicated no significant difference in the amount of elapsed time between either instrument test iteration. For memory consumption,  $z(n = 150) = -1.048$ , with two-tailed  $p = .295$ , which indicated no significant difference in the amount of memory consumption between either instrument test iteration.

Table 15

<i>Wilcoxon Signed Ranks -- Pilot Test Results for Graph-Tool Dijkstra Instrument</i>		
Graph-Tool: Dijkstra	ELAPSED_TIME_2 - ELAPSED_TIME	MEMORY_CONSUMED_2 - MEMORY_CONSUMED
Z	-.743 <sup>a</sup>	-.784 <sup>b</sup>
Asymp. Sig. (2-tailed)	.458	.433
a. Based on positive ranks.	b. Based on negative ranks.	

A Wilcoxon Signed Ranks test was conducted to evaluate whether the Graph-Tool Dijkstra algorithm instrument showed statistically different results between test runs, using a repeated measures design (150 samples) with no intervention. As shown in Table 15 above, for elapsed time,  $z(n = 150) = -.743$ , with two-tailed  $p = .458$ , which indicated no significant difference in the amount of elapsed time between either instrument test iteration. For memory consumption,  $z(n = 150) = -.784$ , with two-tailed  $p = .433$ , which indicated no significant difference in the amount of memory consumption between either instrument test iteration.

Table 16

*Wilcoxon Signed Ranks -- Pilot Test Results for Network-X A\* Instrument*

Network-X: A*	ELAPSED_TIME_2 - ELAPSED_TIME	MEMORY_CONSUMED_2 - MEMORY_CONSUMED
Z	-.428 <sup>a</sup>	-.357 <sup>b</sup>
Asymp. Sig. (2-tailed)	.669	.721

a. Based on positive ranks.      b. Based on negative ranks.

A Wilcoxon Signed Ranks test was conducted to evaluate whether the Network-X A\* algorithm instrument showed statistically different results between test runs, using a repeated measures design (150 samples) with no intervention. As shown in Table 16 above, for elapsed time,  $z(n = 150) = -.428$ , with two-tailed  $p = .669$ , which indicated no significant difference in the amount of elapsed time between either instrument test iteration. For memory consumption,  $z(n = 150) = -.357$ , with two-tailed  $p = .721$ , which indicated no significant difference in the amount of memory consumption between either instrument test iteration.

Table 17

*Wilcoxon Signed Ranks -- Pilot Test Results for Network-X Bellman-Ford Instrument*

Network-X: Bellman-Ford	ELAPSED_TIME_2 - ELAPSED_TIME	MEMORY_CONSUMED_2 - MEMORY_CONSUMED
Z	-.943 <sup>a</sup>	-1.724 <sup>b</sup>
Asymp. Sig. (2-tailed)	.346	.085

a. Based on positive ranks.      b. Based on negative ranks.

A Wilcoxon Signed Ranks test was conducted to evaluate whether the Network-X Bellman-Ford algorithm instrument showed statistically different results between test runs, using a repeated measures design (150 samples) with no intervention. As shown in Table 17 above, for elapsed time,  $z(n = 150) = -0.943$ , with two-tailed  $p = .346$ , which indicated no significant difference in the amount of elapsed time between either instrument test iteration. For memory consumption,  $z(n = 150) = -1.724$ , with two-tailed  $p = .085$ , which indicated no significant difference in the amount of memory consumption between either instrument test iteration.

Table 18

<i>Wilcoxon Signed Ranks -- Pilot Test Results for Network-X Dijkstra Instrument</i>		
Network-X: Dijkstra	ELAPSED_TIME_2 - ELAPSED_TIME	MEMORY_CONSUMED_2 - MEMORY_CONSUMED
Z	-1.692 <sup>a</sup>	-.639 <sup>b</sup>
Asymp. Sig. (2-tailed)	.091	.523

a. Based on negative ranks.      b. Based on positive ranks.

A Wilcoxon Signed Ranks test was conducted to evaluate whether the Network-X Dijkstra algorithm instrument showed statistically different results between test runs, using a repeated measures design (150 samples) with no intervention. As shown in Table 18 above, for elapsed time,  $z(n = 150) = -1.692$ , with two-tailed  $p = .091$ , which indicated no significant difference in the amount of elapsed time between either instrument test iteration. For memory consumption,  $z(n = 150) = -.639$ , with two-tailed  $p = .523$ , which indicated no significant difference in the amount of memory consumption between either instrument test iteration.

### **Results of the Algorithm Instrumentation Pilot Test**

The results of the pilot test on the six algorithm instruments demonstrated that the algorithm instruments were statistically valid and reliable, as no Wilcoxon Signed Ranks tests, in a repeated-measures without intervention design, yielded  $p$  values  $\leq 0.05$ . Per Howitt and Cramer (2014), a Wilcoxon Signed Ranks value of  $p \leq 0.05$  indicates statistical significance, thus indicating significant differences between test iterations (pp. 186-191). In the pilot test, all Wilcoxon results were not significant ( $p > .05$ ), meaning each pair of iterations tested, per algorithm, generated statistically similar results. In conclusion, the results of the pilot test indicated the six algorithm instruments were both statistically valid and reliable enough to be used in this specific doctoral study.

### **MANOVA and its Relationship to the Experimental Variables**

The multivariate analysis of variance (MANOVA) is a multivariate version of the ANOVA, and supports two or more dependent variables, whereas ANOVA only supports a single dependent variable (Tabachnick & Fidell, 2014). Like ANOVA, the MANOVA is used to test the significance of group differences (Mertler & Reinhart, 2017), but unlike ANOVA, by design the MANOVA will support multiple, continuous (non-categorical) dependent variables. A factorial MANOVA is a MANOVA that involves two or more categorical independent variables (each with at least two categories or levels), and two or more continuous dependent variables. For MANOVA (whether one-way, multi-way, or factorial) the independent variables must always be categorical (e.g., gender, political party affiliation, marital status, etc.) each with at least two levels, and the dependent variables must be continuous (e.g., age in years, salary, bank balance, etc.)

As discussed in Section 2, this quantitative experimental study utilized a between-groups, post-test only,  $3 \times 2 \times 2$  factorial MANOVA design, with three independent categorical variables: (a) pathfinding algorithm (with 3-levels); (b) graph analysis framework (with 2-levels); and (c) map complexity (with 2-levels); and two continuous (i.e., quantitative) dependent variables: (a) elapsed time (measured in seconds); and (b) computer memory consumed (measured in megabytes).

There were 12 experimental treatment groups, and each was provided with an equal number ( $n = 150$ ) of pre-stratified and random-selected 2D sample maps from a 2D map population pool that was computer random-generated, as discussed in Section 2.

The reason for using MANOVA in this experiment is that I intentionally manipulated the three categorical independent variables (pathfinding algorithm, graph analysis framework, and map complexity), to detect and measure the impact of those various treatment manipulations upon two dependent variables (elapsed time, and computer memory consumption) in 12 experimental treatment groups. As discussed in detail below, MANOVA results indicated the mean differences between groups, due to the experimental treatments, were statistically significant, and therefore did not occur by chance. The ability to detect the significance of treatment group differences is a main feature of MANOVA (Mertler & Reinhart, 2017, p. 125). In this study, MANOVA results indicated which independent and dependent variables were statistically affected by the treatments, and the extent of the statistical relationship. When significant MANOVA effects were detected, follow-up analyses were conducted using univariate ANOVAs



with Scheffe's post hoc test, as recommended by Marsh-Hunakin, Gochfeld, and Slattery (2013), and by Tabachnick and Fidell (2014).

Experimental manipulations of the independent variables had a statistically significant impact on the dependent variables, so I rejected the null hypothesis ( $H_0$ ) and accepted the alternate hypothesis ( $H_a$ ).

### **Data Screening and Transformations**

SPSS version 23.0 was utilized to conduct the data analysis for this study. Before I conducted inferential statistical analyses, the data were screened to ensure they were reliable and valid for this study. The initial screen checked for missing data to ensure enough data existed for the MANOVA statistic. There was no missing data. The second screening checked for outliers in the dependent variables, as this could limit the accuracy of MANOVA results. This was performed by analyzing the boxplots, stem and leaf plots, and histograms generated by the SPSS Descriptive Statistics *Explore* feature, as recommended by Field (2013), Howitt and Cramer (2014), and by Mertler and Reinhart (2017). Further analyses of the data distributions (normality) of the dependent variables was verified by reviewing the shape of the distributions, seeking skewness and kurtosis. When outliers were found, they were transformed to fit within at least +/- 2.50 standard deviations of the mean for that variable, as recommended by Field (2013, p. 198), and to keep skewness and kurtosis both within +/- 1.0, as recommended by Mertler and Reinhart (2017, p. 45). Two new variables in SPSS were created to contain the transformed results of "elapsed time" and "memory consumed", and were named "Elapsed\_Time\_2" and "Memory\_Consumed\_2" respectively. Finally, I verified the homoscedasticity and

linearity of the dependent variables by using SPSS software. After data transformations, all subsequent inferential statistical analyses were performed using the transformed variables, as recommended by Mertler and Reinhart (2017, p. 44).

### **Descriptive Statistics**

A total of 1,800 samples were used in this study, utilized in various between-subject factors, as depicted below in Table 19, Figure 27, Figure 28, and Table 26. Figures 31 and 32, depict the SPSS results of the descriptive statistics for each combination of the three independent variables ("Algorithm", "Framework", and "Map Complexity"), and the two (transformed) dependent variables (i.e., "Elapsed Time \_2", and "Memory\_Consumed\_2").

Table 19

*Sample Counts (N) per Between-Subject Factors*

Between-Subject Factors		N	% of total
Algorithm	A* (A-star)	600	33.3
	Bellman-Ford	600	33.3
	Dijkstra	600	33.3
Framework	Graph-Tool	900	50.0
	Network-X	900	50.0
Map Complexity	High	900	50.0
	Low	900	50.0

Each of the 1,800 samples were tested for Elapsed Time and Computer Memory Consumption. Samples were evenly distributed among the three independent variables, as described earlier in Table 3, and Figure 14, of Section 2. The means and standard deviations for the samples are depicted in Figure 27, Figure 28, and Table 26.

	ALGORITHM	FRAMEWORK	MAP_COMPLEXITY	Mean	Std. Deviation	N
ELAPSED_TIME_2	A-star	GRAPH-TOOL	HIGH	.07229	.002777	150
			LOW	.03955	.002005	150
			Total	.05592	.016571	300
		NETWORK-X	HIGH	.28334	.013120	150
			LOW	.04033	.001084	150
			Total	.16183	.122064	300
		Total	HIGH	.17781	.106126	300
			LOW	.03994	.001655	300
			Total	.10888	.101900	600
	Bellman-Ford	GRAPH-TOOL	HIGH	.05830	.002605	150
			LOW	.03955	.002005	150
			Total	.04893	.009672	300
		NETWORK-X	HIGH	.27201	.006368	150
			LOW	.04121	.001583	150
			Total	.15661	.115682	300
		Total	HIGH	.16515	.107142	300
			LOW	.04038	.001986	300
			Total	.10277	.098135	600
	Dijkstra	GRAPH-TOOL	HIGH	.07070	.003266	150
			LOW	.03999	.001272	150
			Total	.05535	.015577	300
		NETWORK-X	HIGH	.27970	.012954	150
			LOW	.04163	.001422	150
			Total	.16066	.119590	300
Total		HIGH	.17520	.105099	300	
		LOW	.04081	.001575	300	
		Total	.10801	.100188	600	
Total	GRAPH-TOOL	HIGH	.06710	.006895	450	
		LOW	.03970	.001802	450	
		Total	.05340	.014601	900	
	NETWORK-X	HIGH	.27835	.012192	450	
		LOW	.04106	.001479	450	
		Total	.15970	.119029	900	
	Total	HIGH	.17272	.106148	900	
		LOW	.04038	.001782	900	
		Total	.10655	.100067	1800	

Figure 27. Descriptive statistics (part 1 of 2): elapsed time.

The Elapsed Time mean and standard deviation for the A\* group (from Figure 27):  $M = .10888$ ,  $SD = .101900$ ,  $N = 600$ . For the Bellman-Ford group:  $M = .10277$ ,  $SD = .098135$ ,  $N = 600$ . For the Dijkstra group:  $M = .10801$ ,  $SD = .100188$ ,  $N = 600$ . Overall Elapsed Time descriptive statistics for all groups:  $M = .10655$ ,  $SD = .100067$ ,  $N = 1800$ .

	ALGORITHM	FRAMEWORK	MAP_COMPLEXITY	Mean	Std. Deviation	N
MEMORY_CONSUMED_2	A-star	GRAPH-TOOL	HIGH	2.07385	.013595	150
			LOW	2.04584	.005779	150
			Total	2.05985	.017481	300
		NETWORK-X	HIGH	.12747	.042949	150
			LOW	.07529	.013415	150
			Total	.10138	.041132	300
		Total	HIGH	1.10066	.975335	300
			LOW	1.06057	.986973	300
			Total	1.08061	.980557	600
	Bellman-Ford	GRAPH-TOOL	HIGH	4.94191	.027200	150
			LOW	5.01006	.024955	150
			Total	4.97598	.042943	300
		NETWORK-X	HIGH	.30269	.045527	150
			LOW	.16747	.015574	150
			Total	.23508	.075761	300
		Total	HIGH	2.62230	2.323787	300
			LOW	2.58877	2.425428	300
			Total	2.60553	2.373227	600
	Dijkstra	GRAPH-TOOL	HIGH	2.05013	.011777	150
			LOW	2.06542	.006590	150
			Total	2.05778	.012222	300
		NETWORK-X	HIGH	.40697	.171557	150
			LOW	.15487	.031189	150
			Total	.28092	.176333	300
Total		HIGH	1.22855	.831858	300	
		LOW	1.11015	.957134	300	
		Total	1.16935	.897895	600	
Total	GRAPH-TOOL	HIGH	3.02196	1.359280	450	
		LOW	3.04044	1.394388	450	
		Total	3.03120	1.376211	900	
	NETWORK-X	HIGH	.27904	.156192	450	
		LOW	.13255	.046175	450	
		Total	.20580	.136458	900	
	Total	HIGH	1.65050	1.678682	900	
		LOW	1.58649	1.757400	900	
		Total	1.61850	1.718312	1800	

Figure 28. Descriptive statistics (part 2 of 2): computer memory consumption.

Memory consumption mean and standard deviation for the A\* group (from Figure 28):  $M = 1.08061$ ,  $SD = .980557$ ,  $N = 600$ . For the Bellman-Ford group:  $M = 2.60553$ ,  $SD = 2.373227$ ,  $N = 600$ . For the Dijkstra group:  $M = 1.16935$ ,  $SD = .897895$ ,  $N = 600$ . Overall Memory Consumption descriptive statistics for all groups:  $M = 1.61850$ ,  $SD = 1.718312$ ,  $N = 1800$ .

Table 20

*Statistical Test, Assumptions, and Methods of Verifying Assumptions*

Statistical Test	Assumptions	Verifying Methods
Factorial- MANOVA	1. At least two dependent, continuous (i.e., quantitative) variables.	<i>A priori</i> study design choice
	2. At least two independent (categorical) variables, each with at least two levels (i.e., categories)	<i>A priori</i> study design choice
	3. Independence of Observations	<i>A priori</i> study design choice
	4. Random sampling	<i>A priori</i> study design choice
	5. Sample size	<i>A priori</i> study design choice
	6. Univariate normality	Boxplots; Histograms; P-P plots; Q-Q plots
	7. Multivariate normality	Mahalanobis distance; Bivariate scatter plots
	8. Linearity	Bivariate scatter plots
	9. Homoscedasticity (i.e., "homogeneity of variance")	Box's M-Test for equality of variance-covariance matrices; Bivariate scatter plots

**MANOVA Assumptions**

There are nine factorial MANOVA statistical assumptions, although Field (2013), Mertler and Reinhart (2017), and Tabachnick and Fidell (2014), discussed several situations where MANOVA is quite robust to violations of several of these assumptions. A summary of the MANOVA assumptions is depicted in Table 20.

The results of my MANOVA assumptions verifications were as follows.

1. *Dependent variables.* As discussed in Section 2, I designed this study to use two continuous dependent variables: elapsed time (measured in seconds), and computer memory consumption (measured in megabytes). This was by design.

2. *Independent variables.* As discussed in Section 2, I designed this study to use three categorical independent variables, all of which have at least two categories: (a) pathfinding algorithm (with three categories); (b) graph analysis framework (with two categories); and (c) map complexity (with two categories). This was by design.

3. *Independence of observations:* As discussed in Section 2, I designed this study to ensure each 2D map sample was used only once per graph analysis framework. Also, no dependent variable results derived from any 2D map sample was dependent on prior results derived from any other 2D map sample. This was by design.

4. *Random sampling.* As discussed in detail in Section 2, I designed this study around the fact that the entire 2D map population pool of 2,000 maps was local computer random generated. Furthermore, this population pool was stratified based on the "map complexity" independent variable (i.e., *high* vs. *low* map complexity), thereby forming two stratified population subgroups, from which random assignment of samples to experimental treatment groups was possible. This was by design.

5. *Sample size.* As discussed in Section 2, my *a priori* G\*Power analyses for MANOVA sample sizes yielded a minimum recommended sample size of 63 per treatment group. I used 150 samples ( $n = 150$ ), for each of the 12 treatment groups ( $150 \times 12 = 1800$  samples total). This was by design. Using a large and equal number of samples per treatment group was recommended by Tabachnick and Fidell (2014) to ensure a

robust MANOVA, and provided the option to disregard Box's *M*-test (p. 294). The freedom to ignore Box's *M*-test by using a large total sample size and equal-sized treatment groups was also separately confirmed by Field (2013, p. 652), Howitt and Cramer (2014, p. 291), Mertler and Reinhart (2017, p. 130), and by Tabachnick and Fidell (2014, p. 294). Note that my choice of  $n = 150$  samples per treatment group was considered a *large* sample size according to Korkmaz, Goksuluk, and Zararsiz (2014, p. 11), Mertler and Reinhart (2017, p. 130), Tabachnick and Fidell (2014, p. 114), and by White and Perrone-McGovern (2017, pp. 39-40). A final indicator that I utilized a *large* sample size was the fact that my selected per group sample size of  $n = 150$  was far larger than the G\*Power *a priori* minimum recommended sample size of 63 samples per treatment group, described in Section 2.

6. *Univariate normality*. This assumption is best tested through boxplots, P-P plots, and/or histograms, as recommended separately by Amin, Malik, Kamel, Chooi, and Hussain (2015, pp. 9-11); and by Tabachnick and Fidell (2014). As discussed above, the few univariate outliers discovered were manually transformed to have skewness and kurtosis scores  $\pm 1.0$ , as recommended by Field (2013), and by Mertler and Reinhart (2017, p. 45). Subsequent tests of the transformed variables (using boxplots, histograms, and/or P-P plots) showed no violations of univariate normality.

Table 21

<i>Mahalanobis Distances between Elapsed Time and Memory Consumption</i>			
Framework	Map Complexity	Algorithm	Mahalanobis Distance
Graph-Tool	High (1000 × 1000)	A* (A-star)	9.027
		Bellman-Ford	8.267

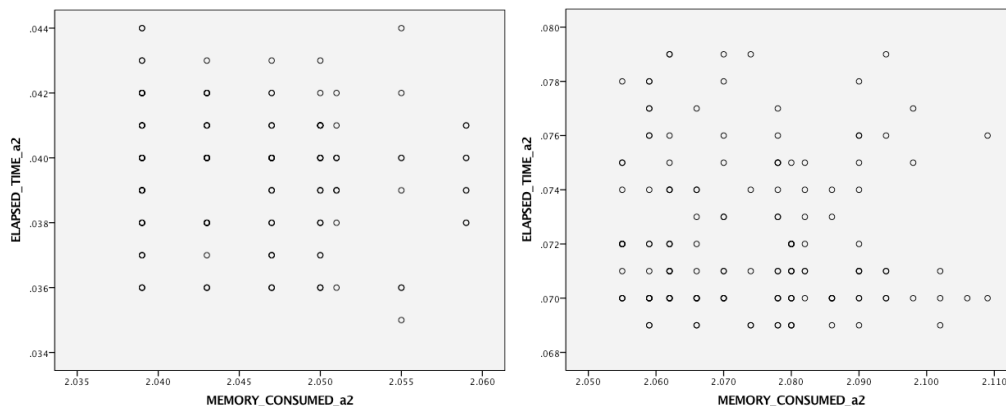
		Dijkstra	6.483
	Low (200 × 200)	A* (A-star)	8.321
		Bellman-Ford	10.109
		Dijkstra	8.602
Network-X	High (1000 × 1000)	A* (A-star)	8.111
		Bellman-Ford	7.305
		Dijkstra	6.304
	Low (200 × 200)	A* (A-star)	11.369
		Bellman-Ford	6.987
		Dijkstra	9.231

7. *Multivariate normality.* This assumption was tested by calculating the Mahalanobis distance of the transformed dependent variables, and comparing that to the permitted  $\chi^2$  critical value based on the degrees of freedom (i.e., number of dependent variables), with  $p < .001$ , as discussed by Korkmaz, Goksuluk, and Zararsiz (2014, p. 10), Mertler and Reinhart (2017, p. 52), and by Tabachnick and Fidell (2014, pp. 108-109). In this study, no Mahalanobis distance for any of the multivariates exceeded 13.816, which was maximum permitted  $\chi^2$  critical value, for  $df = 2$ , at  $p < .001$  (per Mertler & Reinhart, 2017, pp. 53, 357). Therefore, based on tests of the Mahalanobis Distances, there were no violations of multivariate normality. A summary of the Mahalanobis distances between the transformed dependent variables is depicted in Table 21.

8. *Linearity.* Linearity between dependent variables within each treatment group can be tested with bivariate scatter plots, as recommended by Field (2013, p. 192); Hair, Anderson, Babin, and Black (2010, pp. 76, 366); Mayorga and Gleicher (2013, p. 1526);



Mertler and Reinhart (2017, pp. 34, 55-56, 148); Tabachnick and Fidell (2014, p. 117); and by Veletsianos and Kimmons (2016, p. 4). Based on the 12 bivariate scatter plots (i.e., one per treatment group) depicted in Figure 29 through Figure 34 below, no linearity violations occurred. The shapes displayed in each of the 12 bivariate scatter plots are approximately elliptical (i.e., roughly oval). An approximate elliptical shape is indicative of a linear relationship between the dependent variables, as described by Field (2013, p. 192); Lampis, Díaz-Emparanza, and Banerjee (2015, p. 236); Mertler and Reinhart (2017, pp. 34, 55-56, 148); and by Tabachnick and Fidell (2014, pp. 117-118). It was clear to me, upon inspection of the dependent variable scatter plots, that no non-linear (e.g., curvilinear) relationship between dependent variables existed.



*Figure 29.* Graph-Tool, A\* (a-star): scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right).

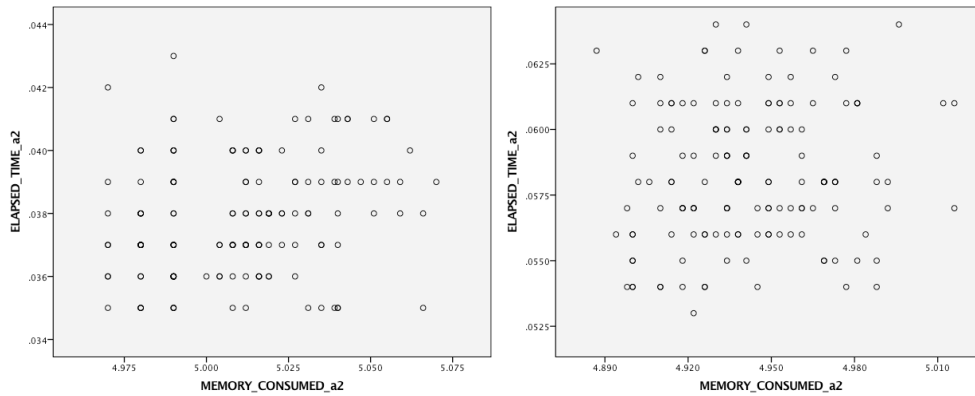


Figure 30. Graph-Tool, Bellman-Ford: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right).

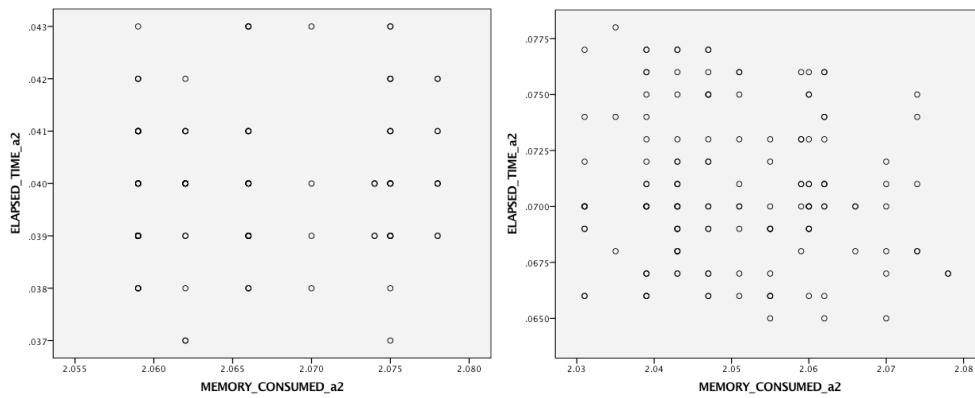


Figure 31. Graph-Tool, Dijkstra: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right).

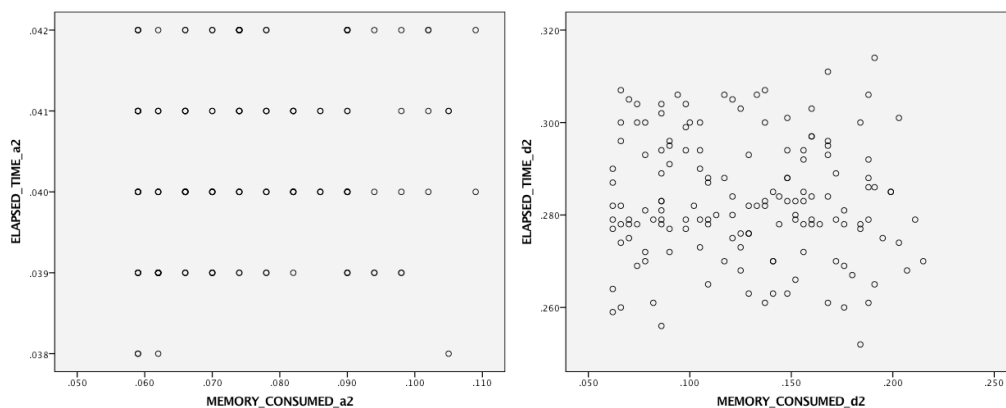


Figure 32. Network-X, A\* (a-star): scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right).

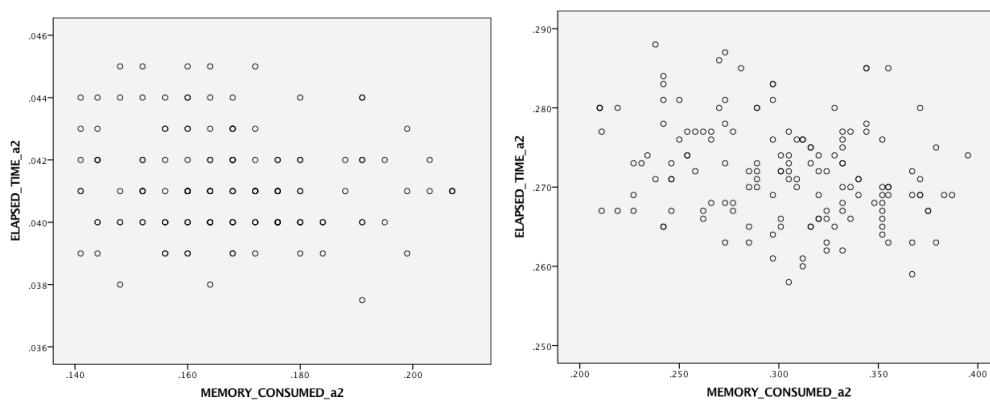


Figure 33. Network-X, Bellman-Ford: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right).

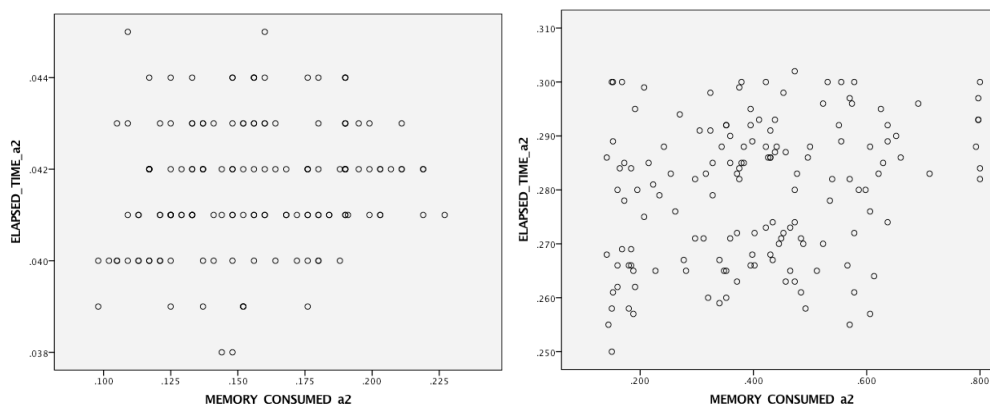


Figure 34. Network-X, Dijkstra: scatter plot of (transformed) elapsed time (in sec) vs. memory consumed (in MB) (low-complexity map group on left; high-complexity map group on right).

**Box's Test of Equality of Covariance Matrices<sup>a</sup>**

Box's M	5993.854
F	180.677
df1	33
df2	6775059.794
Sig.	.000

Tests the null hypothesis that the observed covariance matrices of the dependent variables are equal across groups.

a. Design: Intercept + ALGORITHM + FRAMEWORK + MAP\_COMPLEXITY + ALGORITHM \* FRAMEWORK + ALGORITHM \* MAP\_COMPLEXITY + FRAMEWORK \* MAP\_COMPLEXITY + ALGORITHM \* FRAMEWORK \* MAP\_COMPLEXITY

Figure 35. Box's M-test for equality of covariance matrices.

9. *Homoscedasticity*. This assumption was tested with Box's *M*-test (i.e., Box's Test of Equality of Covariance Matrices), using  $p > .001$  as the criterion, as recommended by Howitt and Cramer (2014, p. 269); and Mertler and Reinhart (2017, p. 36). As shown in Figure 35, Box's *M*-test yielded  $F(33, 6775059.794)$ ,  $p \leq 0.001$ , which was significant, therefore there were significant differences between covariance matrices,

so the assumption of homoscedasticity was not met. However, Tabachnick and Fidell (2014) contend that IBM's SPSS Box's *M*-test is "too strict" when using the large sample sizes often required in MANOVA analyses (2014, p. 120). Additionally, if samples sizes are both large and equal sized (for all treatment groups), one may disregard a statistically significant Box's *M*-test and use the stricter Pillai's Trace statistic in all subsequent MANOVA analyses, instead of the traditional Wilks' Lambda ( $\Lambda$ ) for subsequent interpretation of the MANOVA multivariate *F* results (2014, p. 294). With equal numbers of samples per treatment group robustness of significance tests can be expected (2014, p. 294). Additionally, Field (2013, pp. 643, 652) and Howitt and Cramer (2014, pp. 291, 305) also noted that violations of homoscedasticity due to results of a significant Box's *M*-test are not a concern if group sizes are equal. Field (2013, p. 643) even indicated that Box's *M*-test is "unstable" when per group sample sizes are equal, hence a major reason why Field (and others) recommended Box's *M*-Test be ignored when using equal sample sizes (I used equal sized groups). Marozzi (2016, p. 42) also discussed use of equal-sized groups to improve MANOVA robustness. Utilizing  $n = 150$  samples per treatment group is considered a "large" sample size according to Korkmaz, Goksuluk, and Zararsiz (2014, p. 11), Mertler and Reinhart (2017, p. 130), Tabachnick and Fidell (2014, p. 114), and by White and Perrone-McGovern (2017, pp. 39-40). Therefore, although the homoscedasticity assumption was not met, I analyzed the collected data anyway because: (a) I used a large sample size ( $n = 150$  per group; total  $n = 12 \times 150 = 1,800$ ); (b) I used equal sized treatment groups ( $n = 150$  each); and (c) I utilized the

stricter Pillai's Trace statistic (instead of Wilks'  $\Lambda$ ) to interpret the MANOVA multivariate  $F$  results; as recommended in the aforementioned references.

Table 22

*The General MANOVA Analysis Process (Mertler & Reinhart, 2017, p. 128)*

---

1. Examine the overall multivariate test of significance. If the overall MANOVA results are significant, proceed to the next step. Else stop.
  2. Examine the univariate tests of each of the individual dependent variables. If any ANOVAs are significant, proceed to the next step. Else stop.
  3. Examine the post hoc tests (e.g., Scheffe's Test) for significance, and (if available) examine the homogeneous subsets.
- 

### **MANOVA Statistical Output**

The MANOVA statistical analyses occurred in three steps, listed in Table 22. As discussed earlier in Section 3, the stricter Pillai's Trace statistic was used to interpret the MANOVA results. The multivariate MANOVA test results generated by SPSS are presented in Figure 36. The MANOVA results, interpreted with the strict Pillai's Trace statistic, showed significant factor interaction between (a) Algorithm  $\times$  Framework; (b) Algorithm  $\times$  Map Complexity; (c) Framework  $\times$  Map Complexity; and (d) the combined Algorithm  $\times$  Framework  $\times$  Map Complexity; on *both* dependent variables (elapsed time, and memory consumption).

MANOVA results also indicated significant main effects for (a) Algorithm; (b) Framework; and (c) Map Complexity; on *both* dependent variables (elapsed time, and memory consumption). A summary of the means and standard deviations for each

dependent variable utilized in this study is depicted in Table 26. I discuss results of the MANOVA in the "Interpretation of Inferential Results" part of Section 3.

**Multivariate Tests<sup>a</sup>**

Effect		Value	F	Hypothesis df	Error df	Sig.	Partial Eta Squared	Noncent. Parameter	Observed Power <sup>d</sup>
Intercept	Pillai's Trace	.999	968199.04 <sup>b</sup>	2.000	1787.000	.000	.999	1936398.08	1.000
	Wilks' Lambda	.001	968199.04 <sup>b</sup>	2.000	1787.000	.000	.999	1936398.08	1.000
	Hotelling's Trace	1083.603	968199.04 <sup>b</sup>	2.000	1787.000	.000	.999	1936398.08	1.000
	Roy's Largest Root	1083.603	968199.04 <sup>b</sup>	2.000	1787.000	.000	.999	1936398.08	1.000
ALGORITHM	Pillai's Trace	.995	885.680	4.000	3576.000	.000	.498	3542.718	1.000
	Wilks' Lambda	.006	10640.655 <sup>b</sup>	4.000	3574.000	.000	.923	42562.622	1.000
	Hotelling's Trace	165.420	73860.175	4.000	3572.000	.000	.988	295440.699	1.000
	Roy's Largest Root	165.419	147884.58 <sup>c</sup>	2.000	1788.000	.000	.994	295769.152	1.000
FRAMEWORK	Pillai's Trace	.999	720360.01 <sup>b</sup>	2.000	1787.000	.000	.999	1440720.02	1.000
	Wilks' Lambda	.001	720360.01 <sup>b</sup>	2.000	1787.000	.000	.999	1440720.02	1.000
	Hotelling's Trace	806.223	720360.01 <sup>b</sup>	2.000	1787.000	.000	.999	1440720.02	1.000
	Roy's Largest Root	806.223	720360.01 <sup>b</sup>	2.000	1787.000	.000	.999	1440720.02	1.000
MAP_COMPLEXITY	Pillai's Trace	.992	112736.40 <sup>b</sup>	2.000	1787.000	.000	.992	225472.806	1.000
	Wilks' Lambda	.008	112736.40 <sup>b</sup>	2.000	1787.000	.000	.992	225472.806	1.000
	Hotelling's Trace	126.174	112736.40 <sup>b</sup>	2.000	1787.000	.000	.992	225472.806	1.000
	Roy's Largest Root	126.174	112736.40 <sup>b</sup>	2.000	1787.000	.000	.992	225472.806	1.000
ALGORITHM * FRAMEWORK	Pillai's Trace	.994	883.036	4.000	3576.000	.000	.497	3532.143	1.000
	Wilks' Lambda	.006	10252.060 <sup>b</sup>	4.000	3574.000	.000	.920	41008.240	1.000
	Hotelling's Trace	154.562	69011.882	4.000	3572.000	.000	.987	276047.530	1.000
	Roy's Largest Root	154.562	138178.10 <sup>c</sup>	2.000	1788.000	.000	.994	276356.192	1.000
ALGORITHM * MAP_COMPLEXITY	Pillai's Trace	.283	147.121	4.000	3576.000	.000	.141	588.485	1.000
	Wilks' Lambda	.735	148.409 <sup>b</sup>	4.000	3574.000	.000	.142	593.635	1.000
	Hotelling's Trace	.335	149.697	4.000	3572.000	.000	.144	598.787	1.000
	Roy's Largest Root	.227	203.376 <sup>c</sup>	2.000	1788.000	.000	.185	406.756	1.000
FRAMEWORK * MAP_COMPLEXITY	Pillai's Trace	.988	70669.976 <sup>b</sup>	2.000	1787.000	.000	.988	141339.951	1.000
	Wilks' Lambda	.012	70669.976 <sup>b</sup>	2.000	1787.000	.000	.988	141339.951	1.000
	Hotelling's Trace	79.093	70669.976 <sup>b</sup>	2.000	1787.000	.000	.988	141339.951	1.000
	Roy's Largest Root	79.093	70669.976 <sup>b</sup>	2.000	1787.000	.000	.988	141339.951	1.000
ALGORITHM * FRAMEWORK * MAP_COMPLEXITY	Pillai's Trace	.190	94.085	4.000	3576.000	.000	.095	376.341	1.000
	Wilks' Lambda	.811	98.921 <sup>b</sup>	4.000	3574.000	.000	.100	395.682	1.000
	Hotelling's Trace	.232	103.775	4.000	3572.000	.000	.104	415.099	1.000
	Roy's Largest Root	.227	202.811 <sup>c</sup>	2.000	1788.000	.000	.185	405.622	1.000

a. Design: Intercept + ALGORITHM + FRAMEWORK + MAP\_COMPLEXITY + ALGORITHM \* FRAMEWORK + ALGORITHM \* MAP\_COMPLEXITY + FRAMEWORK \* MAP\_COMPLEXITY + ALGORITHM \* FRAMEWORK \* MAP\_COMPLEXITY

b. Exact statistic

c. The statistic is an upper bound on F that yields a lower bound on the significance level.

d. Computed using alpha =

Figure 36. MANOVA summary table of multivariate results.

The second step assessed the univariate ANOVAs for the transformed dependent variables "Elapsed\_Time\_2" and "Memory\_Consumption\_2", on each of the independent

variables (singly, and in combination), with results depicted in Figure 37. I discuss results of the univariate ANOVAs in the "Interpretation of Inferential Results" part of Section 3.

Tests of Between-Subjects Effects

Source	Dependent Variable	Type III Sum of Squares	df	Mean Square	F	Sig.	Partial Eta Squared	Noncent. Parameter	Observed Power <sup>c</sup>
Corrected Model	ELAPSED_TIME_2	17.951 <sup>a</sup>	11	1.632	46515.017	.000	.997	511665.185	1.000
	MEMORY_CONSUMED_2	5306.282 <sup>b</sup>	11	482.389	158561.932	.000	.999	1744181.26	1.000
Intercept	ELAPSED_TIME_2	20.435	1	20.435	582465.318	.000	.997	582465.318	1.000
	MEMORY_CONSUMED_2	4715.170	1	4715.170	1549881.93	.000	.999	1549881.93	1.000
ALGORITHM	ELAPSED_TIME_2	.013	2	.007	186.678	.000	.173	373.356	1.000
	MEMORY_CONSUMED_2	879.173	2	439.586	144492.532	.000	.994	288985.064	1.000
FRAMEWORK	ELAPSED_TIME_2	5.085	1	5.085	144945.927	.000	.988	144945.927	1.000
	MEMORY_CONSUMED_2	3592.315	1	3592.315	1180798.37	.000	.998	1180798.37	1.000
MAP_COMPLEXITY	ELAPSED_TIME_2	7.882	1	7.882	224654.231	.000	.992	224654.231	1.000
	MEMORY_CONSUMED_2	1.844	1	1.844	606.072	.000	.253	606.072	1.000
ALGORITHM * FRAMEWORK	ELAPSED_TIME_2	.000	2	.000	6.477	.002	.007	12.955	.851
	MEMORY_CONSUMED_2	828.027	2	414.014	136086.795	.000	.993	272173.589	1.000
ALGORITHM * MAP_COMPLEXITY	ELAPSED_TIME_2	.014	2	.007	196.938	.000	.181	393.876	1.000
	MEMORY_CONSUMED_2	.669	2	.334	109.947	.000	.110	219.895	1.000
FRAMEWORK * MAP_COMPLEXITY	ELAPSED_TIME_2	4.956	1	4.956	141272.900	.000	.988	141272.900	1.000
	MEMORY_CONSUMED_2	3.062	1	3.062	1006.425	.000	.360	1006.425	1.000
ALGORITHM * FRAMEWORK * MAP_COMPLEXITY	ELAPSED_TIME_2	.000	2	.000	5.970	.003	.007	11.940	.816
	MEMORY_CONSUMED_2	1.192	2	.596	195.921	.000	.180	391.841	1.000
Error	ELAPSED_TIME_2	.063	1788	3.508E-5					
	MEMORY_CONSUMED_2	5.440	1788	.003					
Total	ELAPSED_TIME_2	38.449	1800						
	MEMORY_CONSUMED_2	10026.891	1800						
Corrected Total	ELAPSED_TIME_2	18.014	1799						
	MEMORY_CONSUMED_2	5311.721	1799						

a. R Squared = .997 (Adjusted R Squared = .996)

b. R Squared = .999 (Adjusted R Squared = .999)

c. Computed using alpha =

Figure 37. Univariate ANOVA data summary.

The third step in MANOVA analysis was the assessment of pair-wise comparisons using Scheffe's post-hoc statistical test. Results of the SPSS Post-Hoc analyses on the Pathfinding Algorithm independent variable are depicted in Figure 38. Note, post-hoc analysis could only be performed on the independent variable *Pathfinding Algorithm* because only it had three levels (i.e., categories). The other two independent



variables, *Graph Analysis Framework* and *Map Complexity* each had only two levels (i.e., categories), and therefore SPSS did not perform Post-Hoc Scheffe's statistical analyses on those independent variables. Because the univariate ANOVA scores for the dependent variables (*Elapsed Time* and *Memory Consumed*) were both significant on the independent variable *Algorithm*, as depicted in Figure 37, it was appropriate to further examine the the Scheffe post-hoc analyses, depicted in Figure 38. I discuss results of the post hoc analyses, in more detail, in the *Interpretation of Inferential Results* part of Section 3.

## Multiple Comparisons

Scheffe		Multiple Comparisons					
Dependent Variable	(I) ALGORITHM	(J) ALGORITHM	Mean Difference (I-J)	Std. Error	Sig.	97.5% Confidence Interval	
						Lower Bound	Upper Bound
ELAPSED_TIME_2	A-star	Bellman-Ford	.00611*	.000342	.000	.00518	.00704
		Dijkstra	.00087	.000342	.039	-.00006	.00180
	Bellman-Ford	A-star	-.00611*	.000342	.000	-.00704	-.00518
		Dijkstra	-.00524*	.000342	.000	-.00617	-.00431
	Dijkstra	A-star	-.00087	.000342	.039	-.00180	.00006
		Bellman-Ford	.00524*	.000342	.000	.00431	.00617
MEMORY_CONSUMED_2	A-star	Bellman-Ford	-1.52492*	.003184	.000	-1.53358	-1.51626
		Dijkstra	-.08874*	.003184	.000	-.09739	-.08008
	Bellman-Ford	A-star	1.52492*	.003184	.000	1.51626	1.53358
		Dijkstra	1.43618*	.003184	.000	1.42752	1.44484
	Dijkstra	A-star	.08874*	.003184	.000	.08008	.09739
		Bellman-Ford	-1.43618*	.003184	.000	-1.44484	-1.42752

Based on observed means.

The error term is Mean Square(Error) = .003.

\*. The mean difference is significant at the

*Figure 38.* Post hoc results (Scheffe test) for elapsed time and memory consumed, per pathfinding algorithm.

Table 23

*Homogeneous Subsets (Scheffe): Elapsed Time*

	ALGORITHM	N	Subset		
			1	2	3
Scheffe <sup>a</sup>	Bellman-Ford	600	.10277		
	Dijkstra	600		.10801	
	A-Star	600			.10888
	Sig.		1.000	1.000	1.000

*Note.* Means for groups in homogeneous subsets are displayed.

Based on observed means.

The error term is Mean Square (Error) = 3.51E-005.

a. Uses Harmonic Mean Sample Size = 600.

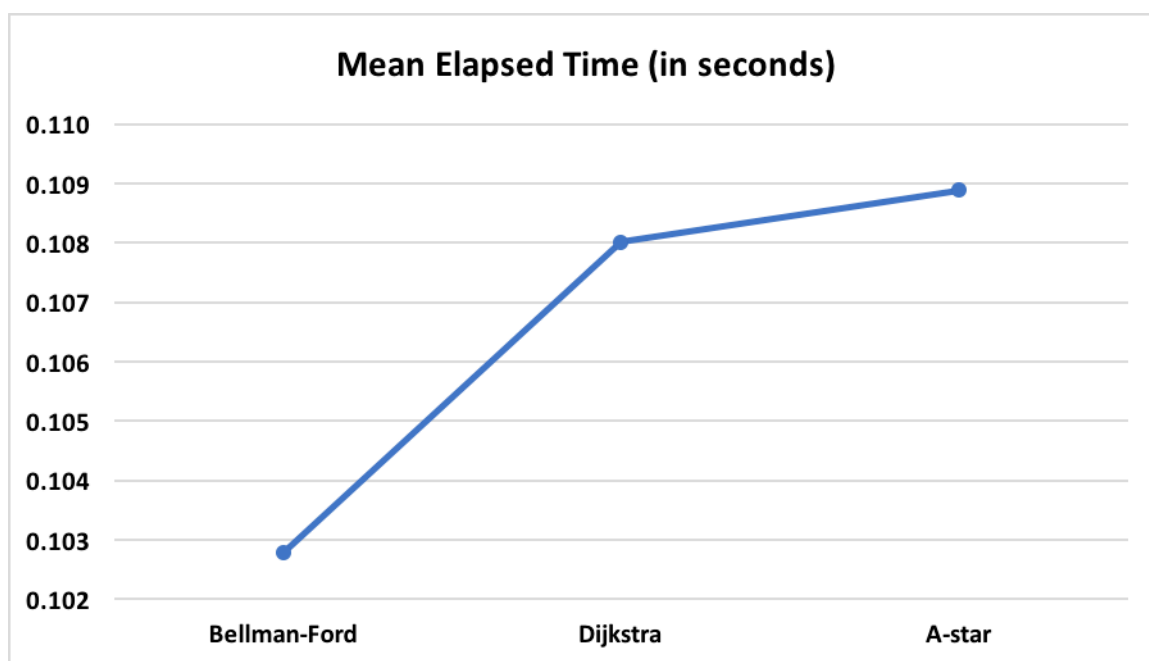


Figure 39. Homogeneous subsets: mean elapsed time per pathfinding algorithm.

Table 24

*Homogeneous Subsets (Scheffe): Memory Consumed*

	ALGORITHM	N	Subset		
			1	2	3
Scheffe <sup>a</sup>	A-star	600	1.08061		
	Dijkstra	600		1.16935	
	Bellman-Ford	600			2.60553
	Sig.		1.000	1.000	1.000

*Note.* Means for groups in homogeneous subsets are displayed.

Based on observed means. The error term is Mean Square (Error) = .003.

a. Uses Harmonic Mean Sample Size = 600.

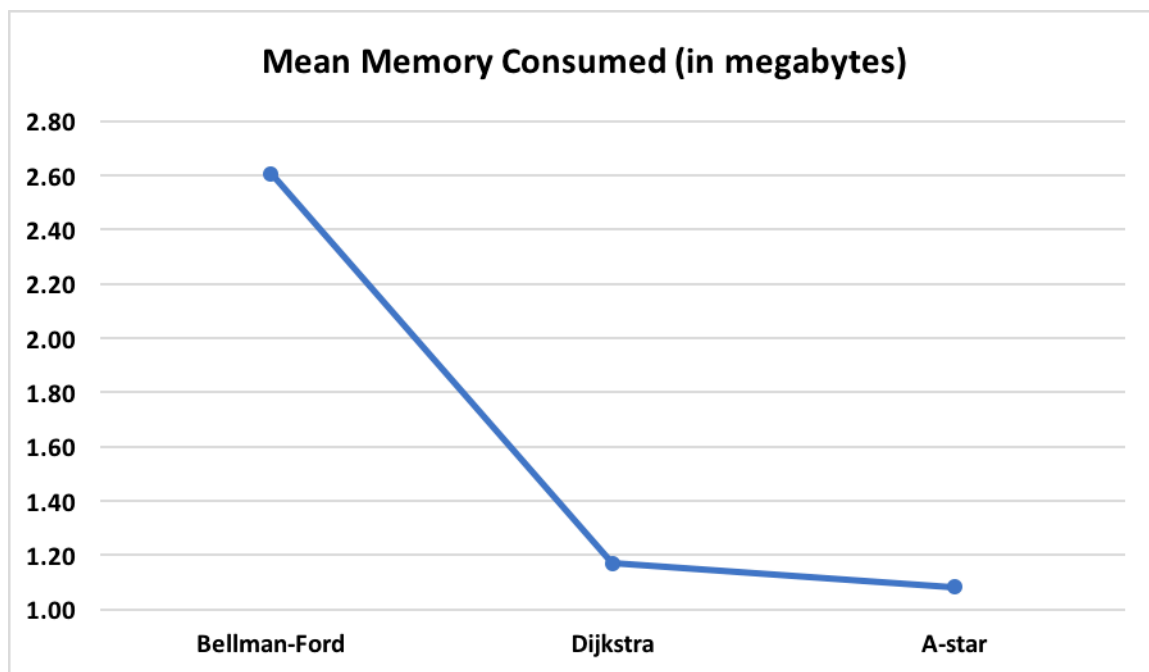


Figure 40. Homogeneous subsets: mean memory consumption per pathfinding algorithm.

## Interpretation of Inferential Results

The purpose of this study was to determine the nature of the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. The three independent variables were (a) pathfinding algorithm; (b) graph analysis framework; and (c) map complexity. The two dependent variables were (a) elapsed time; and (b) computer memory consumption. My research question was "What is the relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption?"

The null hypothesis ( $H_0$ ) stated there would be no relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. The alternative hypothesis ( $H_a$ ) stated there would be a relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time; and computer memory consumption.

Conducting a MANOVA using Pillai's Trace statistic, this study applied an alpha level of .05 to examine the  $p$ -value linked with the resulting multivariate  $F$  statistic. If the resulting  $p$ -value was less than the alpha level (.05), then the multivariate  $F$  statistic was significant, and the null hypothesis was rejected. Conversely, a  $p$ -value greater than the alpha level signified the relationship between the variables was not significant, and therefore the null hypothesis would be accepted.

Table 25

*Summary of the Multivariate (Pillai's Trace) Tests<sup>a</sup>*

Effect	Value	<i>F</i>	Hyp. df	Error df	<i>p</i> (sig.)	Partial Eta Squared
1. Algorithm	.995	885.68	4.0	3576.00	.000	.498
2. Framework	.999	720360.01 <sup>b</sup>	2.0	1787.00	.000	.999
3. Map Complexity	.992	112736.40 <sup>b</sup>	2.0	1787.00	.000	.992
4. Algorithm × Framework	.994	883.04	4.0	3576.00	.000	.497
5. Algorithm × Map Complexity	.283	147.12	4.0	3576.00	.000	.141
6. Framework × Map Complexity	.988	70669.98 <sup>b</sup>	2.0	1787.00	.000	.988
7. Algorithm × Framework × Map Complexity	.190	94.09	4.0	3576.00	.000	.095

*Note.* Statistical values are from Figure 36.

*a.* Design: Intercept + Algorithm + Framework + Map\_Complexity + Algorithm×Framework + Algorithm×Map\_Complexity + Framework×Map\_Complexity + Algorithm×Framework×Map\_Complexity.

*b.* Exact Statistic.

Analysis of the results of the Pillai's Trace statistic indicated that there was a significant relationship between the variables: pathfinding algorithm, graph analysis framework, map complexity, elapsed time, and computer memory consumption. A summary of the results of the Pillai's Trace statistic are depicted in Table 25.

For the three independent variables (*Pathfinding Algorithm*, *Graph Analysis Framework*, and *Map Complexity*) their main effects were all significant (although some more than others), as discussed next.

1. Pathfinding Algorithm: the Pillai's Trace value of .995 is significant,  $F(4, 3576) = 885.86, p < .001$ . The multivariate effect size,  $\eta^2 = .498$ , was moderate.

2. Graph Analysis Framework: the Pillai's Trace value of .999 is significant,  $F(2, 1787) = 720360.01, p < .001$ . The multivariate effect size,  $\eta^2 = .999$ , was very strong.

3. Map Complexity: the Pillai's Trace value of .992 is significant,  $F(2, 1787) = 112736.40, p < .001$ . The multivariate effect size,  $\eta^2 = .992$ , was very strong.

Next, the interaction effects for all combinations of the three independent variables, were all significant (some more than others), as discussed next.

4. Pathfinding Algorithm  $\times$  Graph Analysis Framework: the Pillai's Trace value of .994 is significant,  $F(4, 3576) = 883.04, p < .001$ . The multivariate effect size,  $\eta^2 = .497$ , was moderate.

5. Pathfinding Algorithm  $\times$  Map Complexity: the Pillai's Trace value of .283 is significant,  $F(4, 3576) = 147.12, p < .001$ . The multivariate effect size,  $\eta^2 = .141$ , was weak.

6. Graph Analysis Framework  $\times$  Map Complexity: the Pillai's Trace value of .988 is significant,  $F(2, 1787) = 70669.98, p < .001$ . The multivariate effect size,  $\eta^2 = .998$ , was very strong.

7. Pathfinding Algorithm  $\times$  Graph Analysis Framework  $\times$  Map-Complexity: the Pillai's Trace value of .190 is significant,  $F(4, 3576) = 94.09, p < .001$ . The multivariate effect size,  $\eta^2 = .095$ , was very weak.

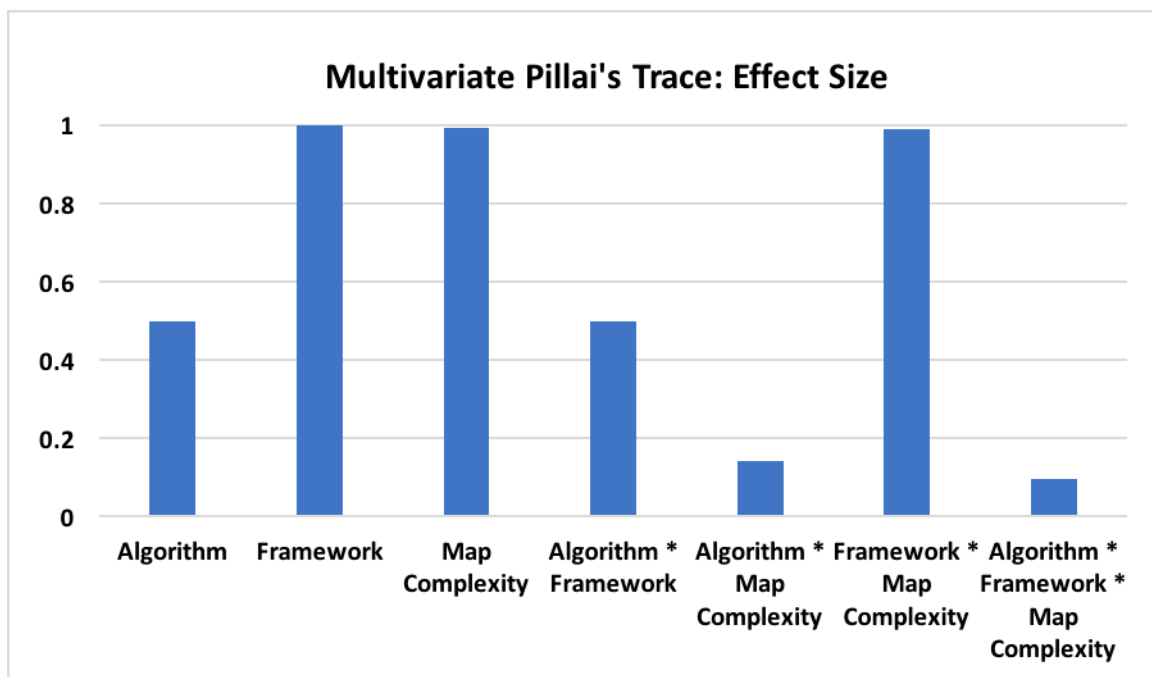


Figure 41. Multivariate effect sizes (Pillai's trace).

In summary, multivariate Pillai's Trace results were all significant for the independent variables, whether alone or in combination, but only the main effects for Framework, Map Complexity, and the interaction effect for "Framework  $\times$  Map Complexity," exhibited strong effect sizes (i.e., partial eta,  $\eta^2 > .90$ ). The remaining independent variables, while significant, had moderate to weak effect sizes ( $\eta^2 < .50$ ). A summary of the multivariate effect sizes is depicted in a column chart in Figure 41.

Because results of the MANOVA were statistically significant, multiple ANOVAs were conducted on the dependent variables (Elapsed Time, Memory Consumption) as follow up tests to the MANOVA, to evaluate the between-subject effects, as recommended by Howitt and Cramer (2014, p. 289). Prior to examination of the univariate ANOVA results, the alpha level was adjusted to  $\alpha = .025$  because two

dependent variables were analyzed, as recommended by Mertler and Reinhart (2017, p. 140). Univariate ANOVA results (see Figure 37) indicated several significant findings.

1. Pathfinding Algorithm: The univariate ANOVA indicated Algorithm significantly affected Elapsed Time,  $F(2, 1788) = 186.68, p < .001, \eta^2 = .173$ ; and Algorithm significantly affected Memory Consumed,  $F(2, 1788) = 144,492.53, p < .001, \eta^2 = .994$ .

2. Graph Analysis Framework: The univariate ANOVA indicated Framework significantly affected Elapsed Time,  $F(1, 1788) = 144,945.93, p < .001, \eta^2 = .988$ ; and Framework significantly affected Memory Consumed,  $F(1, 1788) = 118,798.37, p < .001, \eta^2 = .998$ .

3. Map Complexity: The univariate ANOVA indicated Map Complexity significantly affected Elapsed Time,  $F(1, 1788) = 224,654.23, p < .001, \eta^2 = .992$ ; and Map Complexity significantly affected Memory Consumed,  $F(1, 1788) = 606.07, p < .001, \eta^2 = .253$ .

4. Pathfinding Algorithm  $\times$  Graph Analysis Framework: The univariate ANOVA indicated Algorithm  $\times$  Framework significantly affected Elapsed Time,  $F(2, 1788) = 6.48, p = .002, \eta^2 = .007$ ; and Algorithm  $\times$  Framework significantly affected Memory Consumed,  $F(2, 1788) = 136,086.80, p < .001, \eta^2 = .993$ .

5. Pathfinding Algorithm  $\times$  Map Complexity: The univariate ANOVA indicated Algorithm  $\times$  Map Complexity significantly affected Elapsed Time,  $F(2, 1788) = 196.94,$



$p < .001$ ,  $\eta^2 = .181$ ; and Algorithm  $\times$  Map Complexity significantly affected Memory Consumed,  $F(2, 1788) = 109.95$ ,  $p < .001$ ,  $\eta^2 = .110$ .

6. Graph Analysis Framework  $\times$  Map Complexity: The univariate ANOVA indicated Framework  $\times$  Map Complexity significantly affected Elapsed Time,  $F(1, 1788) = 141,272.90$ ,  $p < .01$ ,  $\eta^2 = .988$ ; and Framework  $\times$  Map Complexity significantly affected Memory Consumed,  $F(1, 1788) = 1006.43$ ,  $p < .001$ ,  $\eta^2 = .360$ .

7. Pathfinding Algorithm  $\times$  Graph Analysis Framework  $\times$  Map Complexity: The univariate ANOVA indicated Algorithm  $\times$  Framework  $\times$  Map Complexity significantly affected Elapsed Time,  $F(2, 1788) = 5.97$ ,  $p = .003$ ,  $\eta^2 = .007$ ; and Algorithm  $\times$  Framework  $\times$  Map Complexity significantly affected Memory Consumed,  $F(2, 1788) = 195.921$ ,  $p < .001$ ,  $\eta^2 = .180$ .

In addition to the univariate ANOVAs, Scheffe post hoc tests were also conducted as follow-up tests. Analysis of the Scheffe post hoc results (see Figure 38) yielded significant findings for both elapsed time and memory consumption.

1. Scheffe post hoc results for Elapsed Time and Pathfinding Algorithm: The elapsed time results for the A\* algorithm differed significantly from the Bellman-Ford algorithm ( $sig. \leq .001$ ). And the A\* algorithm differed significantly (but less so) from Dijkstra ( $sig. = .039$ ). Bellman-Ford significantly differed from Dijkstra ( $sig. \leq .001$ ).

2. Scheffe post hoc results for Memory-Consumption and Path Finding Algorithm: The memory consumption results for the A\* algorithm differed significantly from the Bellman-Ford algorithm ( $sig. \leq .001$ ). And the A\* algorithm differed

significantly from Dijkstra ( $sig. \leq .001$ ). Bellman-Ford significantly differed from Dijkstra ( $sig. \leq .001$ ).

Lastly, I compared the SPSS generated Homogenous Subsets results (see Table 23, Table 24, Figure 39, and Figure 40). The results of the Elapsed Time Homogeneous Subset tests, depicted in Table 23, indicated that the means (in elapsed seconds) for all three pathfinding algorithms were significantly different from each other. The Bellman-Ford algorithm was the fastest, and A\* was the slowest, while Dijkstra's elapsed time performance was in-between Bellman-Ford and A\*. A summary of the Elapsed Time means are depicted in a line chart in Figure 39.

The results of the Memory Consumption Homogeneous Subset tests, depicted in Table 24, indicated that the means (in memory consumed) for all three pathfinding algorithms were significantly different from each other. The A\* algorithm was the most memory efficient (i.e., consumed less memory), and Bellman-Ford consumed the most memory, while Dijkstra's memory consumption was in-between Bellman-Ford and A\*. The memory consumed mean values are depicted in a line chart in Figure 40.

A graphical relationship between Elapsed Time to Algorithm and Framework is depicted in Figure 42. It was evident that Network-X was slower than Graph-Tool, for all pathfinding algorithms tested in this study. A graphical relationship between Memory Consumed to Algorithm and Framework is depicted in Figure 43. It is evident that Network-X was more memory efficient (consumed less memory) than Graph-Tool, for all pathfinding algorithms tested in this study.

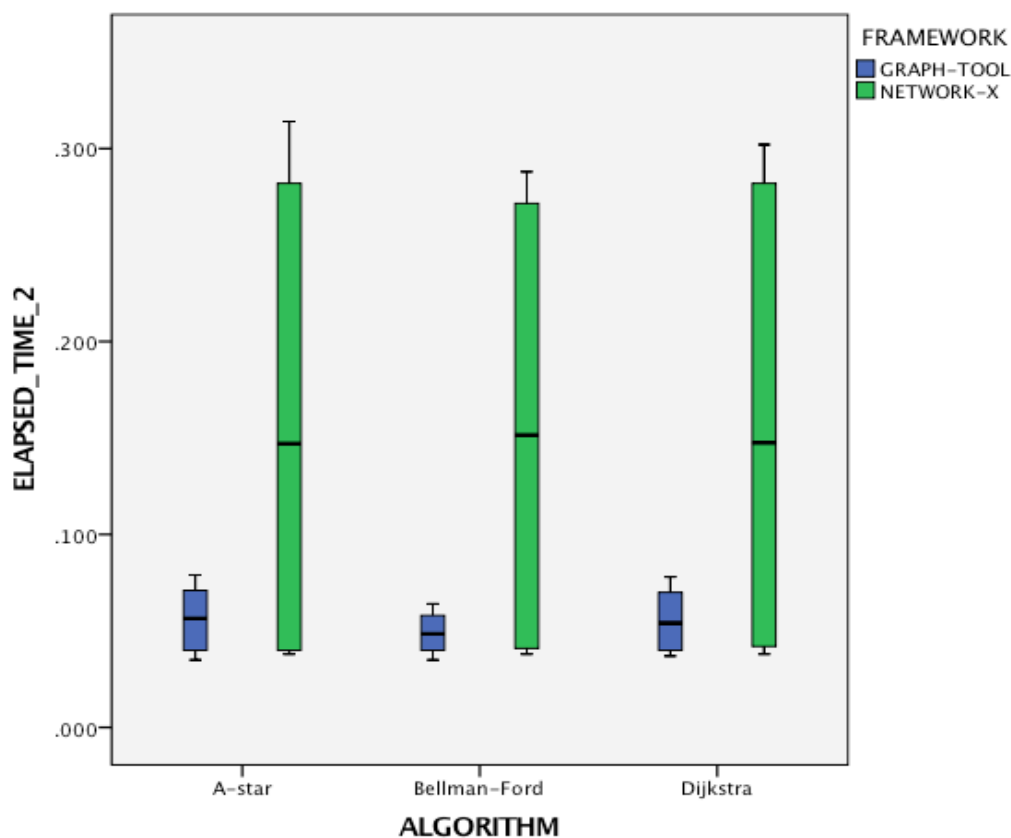


Figure 42. Elapsed time (seconds) per framework and algorithm.

A graphical relationship between Elapsed Time to Map Complexity and Algorithm is depicted in Figure 44. It is evident that map complexity had a noticeable impact on elapsed time. The high complexity map samples required more time to process than the low complexity map samples. Next, a graphical relationship between Memory Consumed to Map Complexity and Algorithm is depicted in Figure 45.

The relationship between Elapsed Time to Algorithm and Map Complexity is depicted in Figure 46. The relationship between Memory Consumed to Algorithm and Map Complexity is depicted in Figure 47. It is evident from Figure 47 that map

complexity impacted the maximum memory consumed by the Bellman-Ford algorithm, more than the maximum memory consumed by either the Dijkstra or A\* algorithms.

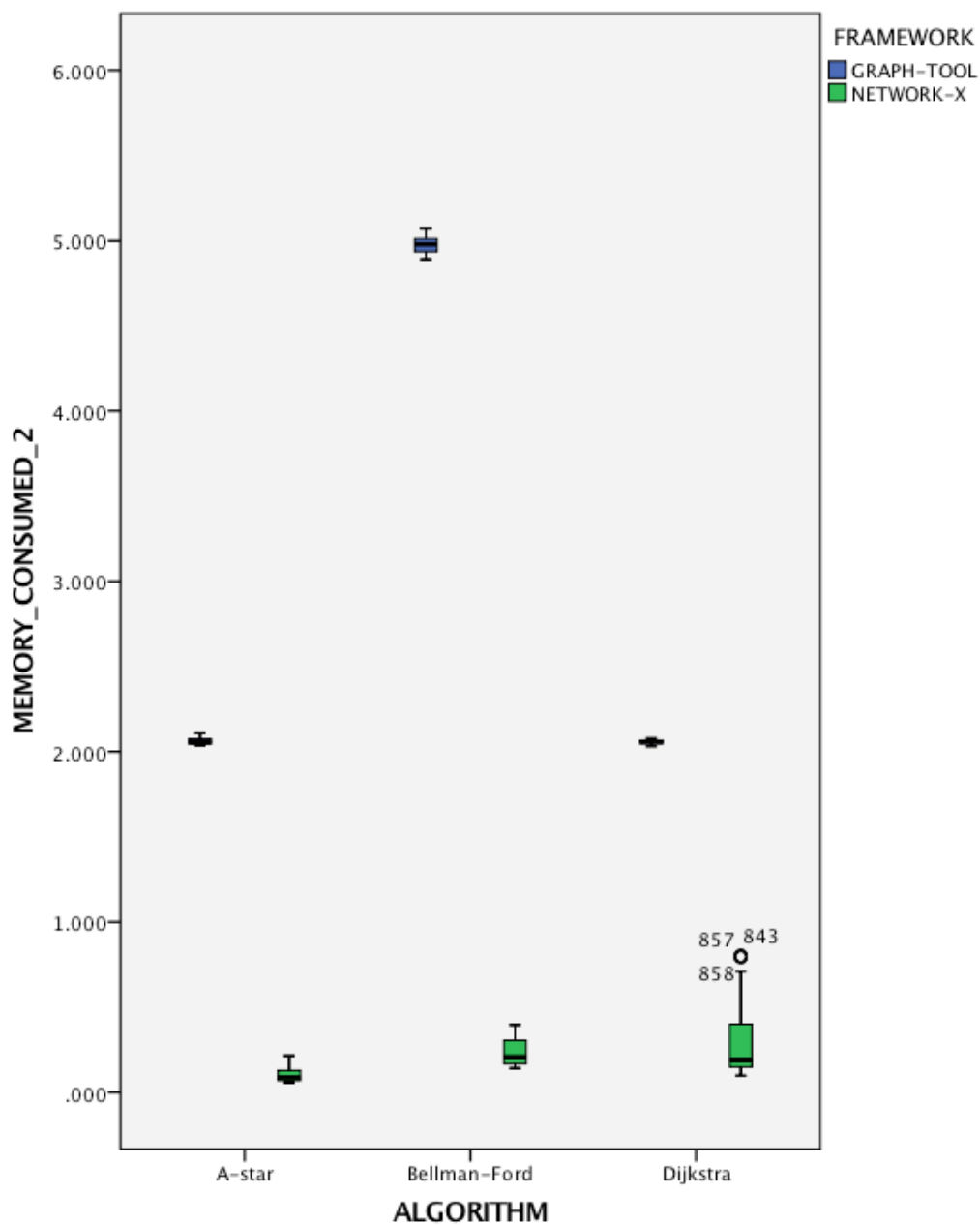


Figure 43. Memory consumed (megabytes) per framework and algorithm.

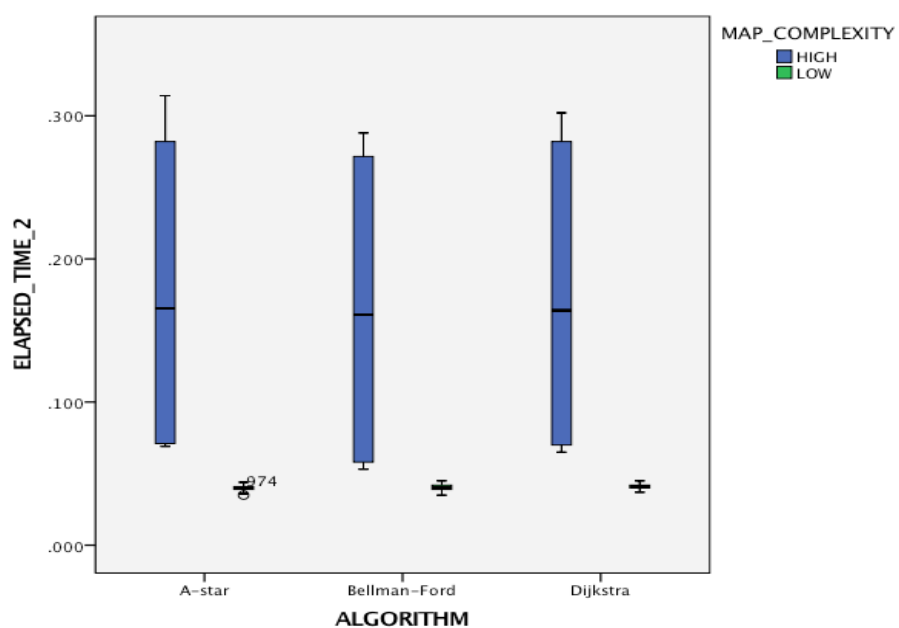


Figure 44. Elapsed time (seconds) per map complexity and algorithm.

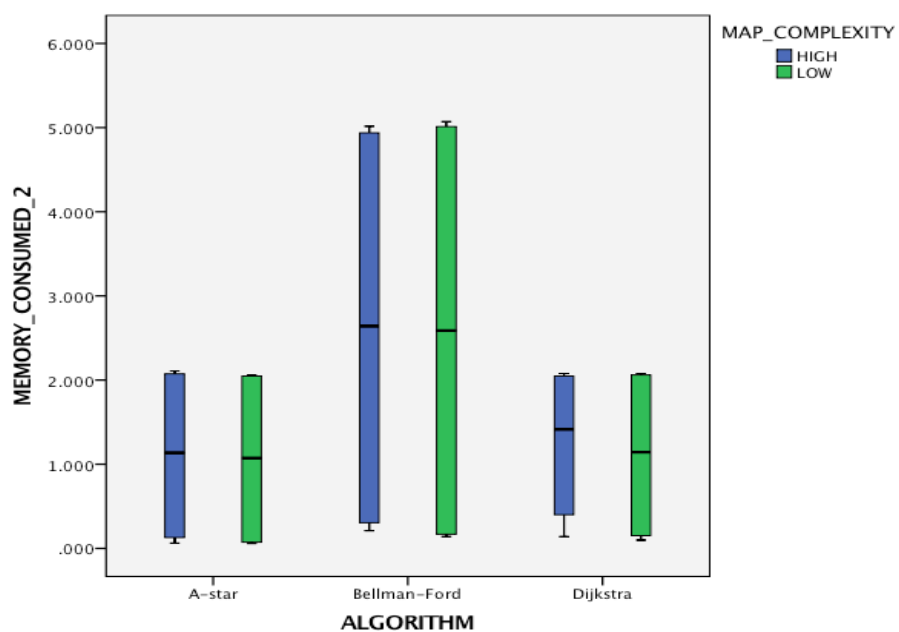


Figure 45. Memory consumed (megabytes) per map complexity and algorithm.

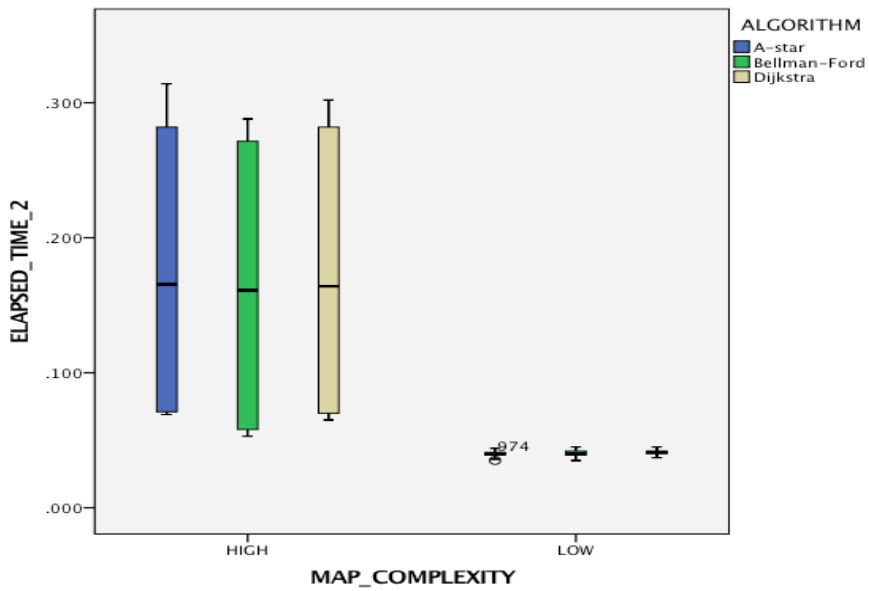


Figure 46. Elapsed time (seconds) per algorithm and map complexity.

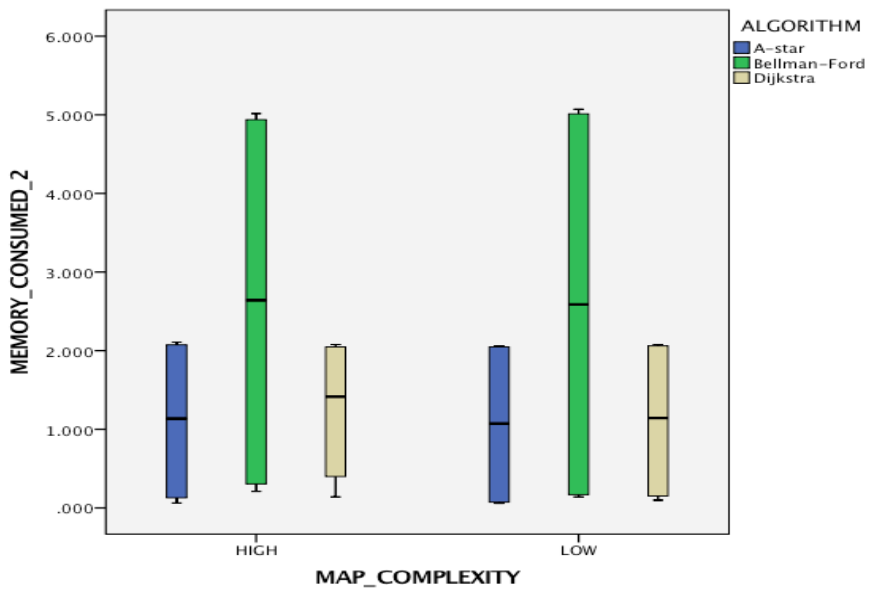


Figure 47. Memory consumed (megabytes) per algorithm and map complexity.

Table 26

*Means and Standard Deviations for the Dependent Variables for All Treatment Groups*

Dependent Variable	Independent Variable			<i>M</i>	<i>SD</i>	<i>N</i>	
	Pathfinding Algorithm	Graph Framework	Map Complexity				
Elapsed-Time	A*	Graph-Tool	High	.072	.003	150	
			Low	.040	.002	150	
		Network-X	High	.283	.013	150	
			Low	.040	.001	150	
		Bellman-Ford	Graph-Tool	High	.058	.003	150
				Low	.040	.002	150
	Network-X	High	.272	.006	150		
		Low	.041	.002	150		
	Dijkstra	Graph-Tool	High	.071	.003	150	
			Low	.040	.001	150	
		Network-X	High	.280	.013	150	
			Low	.042	.001	150	
Memory-Consumed		A*	Graph-Tool	High	2.074	.014	150
				Low	2.046	.006	150
	Network-X		High	.127	.043	150	
			Low	.075	.013	150	
	Bellman-Ford		Graph-Tool	High	4.942	.027	150
				Low	5.010	.025	150
	Network-X	High	.303	.046	150		
		Low	.167	.016	150		
	Dijkstra	Graph-Tool	High	2.050	.012	150	
			Low	2.065	.007	150	
		Network-X	High	.407	.172	150	
			Low	.155	.031	150	

The means for Elapsed Time (seconds) and Memory Consumption (megabytes) by Pathfinding Algorithm, Graph Analysis Framework, and Map Complexity, are depicted in Table 26.

## Summary and Theoretical Framework Implications

This study's findings indicated a clear statistical relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. Therefore, because of this relationship I rejected the null hypothesis ( $H_0$ ), and accepted the alternative hypothesis ( $H_a$ ). Thus, pathfinding algorithms, graph analysis frameworks, and map complexity can significantly affect elapsed time and computer memory consumption.

Furthermore, based on Pillai's Trace statistic (see Table 25, and Figure 41), the independent variables Graph Analysis Framework and Map Complexity (both as individual main effects, and as a combined interaction effect) had greater impacts on the dependent variables, in terms of effect size ( $ES$ ), than did Pathfinding Algorithm alone. Additionally, while Graph-Tool overall was faster than Network-X (see Figure 42), in terms of overall memory consumption when applied to the targeted 2D map samples, the Network-X framework consumed less memory than the Graph-Tool framework.

As related to my chosen theoretical framework, social network theory, the relevancy of this study's results is clear and can be summarized in two key points.

1. Regarding elapsed time, map complexity and algorithm choice both matter. Refer to Figure 44 and Figure 46. The 2D map samples with less complex map topology required less time to process by the pathfinding algorithms than the more complex map samples. Additionally, the Pillai's Trace results (see Table 25, and Figure 41) indicated that the more complex the network topology, the more time is required to do algorithmic pathfinding, which in social networks such as those described by Barnes (1954), De Sola



Pool and Kocheck (1979), and Korte and Milgram (1970), may imply more time is required to find connections and objects of interest in complex networks. This will have implications in terrorist network analysis, and criminal network analysis (see Eiselt & Bhadury, 2015; and see Medina, 2014). Based on the results of this quantitative study, computationally analyzing large social networks (e.g., complex terrorist networks), mapped as 2D graphs, will require more time to calculate shortest paths between nodes of interest, than doing the similar calculations in smaller networks.

2. Regarding memory consumption, the graph analysis framework and pathfinding algorithm choice both matter. Refer to Figure 41, Figure 45 and Figure 47. In terms of effect size (i.e., partial Eta squared,  $\eta^2$ ), the Pillai's Trace results indicate that on average while Bellman-Ford was the fastest algorithm, it also consumed the most memory, in fact over twice as much memory than either A\* or Dijkstra. This will have implications in terms of autonomous pathfinding for robots, as described by Dean (2013), and Kaur and Gangal (2015). Algorithm choice will impact the amount of memory needed in such autonomous agents. And it will likewise impact software agents working in network dead zones or GPS-denied environments, as described by Kang and Choo (2016), and by Wang, Zlatanova, Moreno, Van Oosterom, and Toro (2014).

In this quantitative study the Bellman-Ford algorithm consumed the most memory, but also demonstrated the fastest performance, as noted in Figures 43 and 44. There was some contradictory material on this topic. First, according to the seminal work of Cormen, Leiserson, Rivest, and Stein (2009), the run-time performance of Dijkstra's algorithm, with a good priority queue implementation, should be better than Bellman-

Ford (p. 658). However, in my quantitative study, the reverse was noted: the Bellman-Ford algorithm performed faster than Dijkstra's algorithm (see Figures 43, and 44).

Cormen et al. (2009) further suggest that the run-time performance of Dijkstra's algorithm (and by extension the A\* algorithm, which is a direct descendent of Dijkstra's algorithm) clearly depends on how the priority queue was implemented (Cormen et al, 2009, p. 661). However, in a later solo publication, Cormen (2013) claimed that while the  $O(n^3)$  runtime performance of Bellman-Ford may be considered slow, it is "not too bad" in applied practice, because the constant factors in the running times of the Bellman-Ford loops are low (Cormen, 2013, p. 106). By contrast, if Dijkstra is implemented with a Fibonacci heap for the priority queue, the constant hidden factors in the asymptotic notation (due to the Fibonacci implementation) are not as good as those for standard binary heaps (Cormen, 2013, p. 101). It remains to be seen how the priority queues for Dijkstra and A\* were implemented in the graph analysis frameworks tested in this study. Line by line source code analysis and further testing might confirm these conclusions, but such actions were beyond the scope of this study. Poor implementation of the priority queues could explain why Dijkstra and A\* performed worse in this study in terms of run-time speed (but in not memory consumption) than the Bellman-Ford algorithm. Additionally, according to Brodnik and Grgurovič (2017) Dijkstra's algorithm run-time performance will degrade with dense graphs, yielding slower performance than the Floyd-Warshall algorithm which exhibits asymptotic  $O(n^3)$  runtime performance (pp. 8-9), which is the same asymptotic runtime as Bellman-Ford's  $O(n^3)$ . The Dijkstra speed

assertion made by Brodnik and Grgurovič (2017, pp. 8-9) corresponds to the runtime performance of Dijkstra's algorithm seen in this quantitative study.

There is further supporting material describing situations where the Bellman-Ford algorithm could perform better (i.e., have faster runtime) than Dijkstra's algorithm. In the seminal work by Sedgewick and Wayne (2011), they indicated that while the worst-case runtime for Bellman-Ford is slower than Dijkstra's algorithm, in many of what they call "typical applications" Bellman-Ford will exhibit linear runtime performance (Sedgewick & Wayne, 2013, p. 682). In one of their tests involving a network of 250 vertices, Sedgewick and Wayne noticed that the Bellman-Ford algorithm completed pathfinding with fewer required path-length comparisons than Dijkstra's algorithm for the same network problem (2013, p. 675).

The Cormen (2013), and Sedgewick and Wayne (2011) publications clearly indicated scenarios where Bellman-Ford exhibits faster run-time performance, which was seen in this quantitative study. But in neither work (Cormen, 2013; Sedgewick & Wayne, 2011) was the memory consumption of Bellman-Ford predicted to be less than that of Dijkstra or A\*. This effect of Dijkstra and A\* consuming less memory than Bellman-Ford was confirmed in this quantitative study (see Figures 43 and 45). So, although in this quantitative study the Bellman-Ford algorithm was found to perform statistically faster than either Dijkstra's algorithm and the A\* algorithm (which was a surprising finding), Bellman-Ford also consumed more memory than the other two algorithms as was theorized and discussed by Sedgewick and Wayne (2011), and Cormen (2013) (which was not a surprising finding).

To discover why Bellman-Ford performed faster, the actual implementations of the priority queue(s) used in the Dijkstra algorithm implementations by the Network-X and Graph-Tool frameworks could be examined, since the source code to both frameworks is available online. However, such a microscopic line-by-line code analysis and comparison was beyond the scope of this quantitative study. My goal in this study was to measure and compare pathfinding algorithm performance between the selected graph analysis frameworks at a macro-level, not to perform a microscopic, line-by-line, source code analyses of the chosen frameworks (although doing such a line-by-line performance comparison of the pathfinding algorithm implementations of both frameworks could be a topic for further research).

There are other situations that may explain the surprisingly fast runtime results of the Bellman-Ford algorithm that were noted in this quantitative study. One is the use of parallel graph algorithms. Lenharth, Nguyen, and Pingali (2016) described situations for large complex networks (such as those of Facebook, Amazon and Netflix) where use of parallel graph algorithms may provide a way to efficiently analyze huge networks with over a billion nodes and edges (p. 78). For example, if one of the frameworks tested in this quantitative study used parallel-enhanced pathfinding algorithms, but the other did not do so, then perhaps that could explain the surprising faster runtime performance of Bellman-Ford against the other algorithms compared. However, validating if the pathfinding algorithms for the chosen graph analysis frameworks used parallel graph algorithm techniques was beyond the scope of this study, as that would require line-by-line source code analysis; yet doing so could be an avenue for future research.

Another situation that could cause unexpected run-time performance for pathfinding algorithms is the presence (or absence) and quantity of obstacles on the grid map, and the map topology itself. Obstacle placement (quantity and layout) can have a detrimental effect on grid-based path planning by increasing the complexity and difficulty of optimal path discovery, as discussed in Kang and Lee (2017, pp. 3, 5-6). Additionally, the type, quantity and placement of obstacles could negatively impact the heuristics used by heuristic search algorithms, such as (but not limited to) the A\* algorithm, as was discussed by Cavazza, Aranyi, and Charles (2017, pp. 2, 7). Also, Ammar, Bennaceur, Châari, Koubâa, and Alajlan (2015) suggested that high terrain blockage ratios impeded some algorithms (e.g., Dijkstra, A\*) more than other algorithms when calculating the shortest path in some maps with many obstacles. Related to my quantitative study's results, detecting the impact of obstacle placement on the runtime output of algorithm performance by the chosen graph analysis frameworks could be explored by doing a code review of the algorithm implementations, but as already noted, that was beyond the scope of this study (yet may be worthy of further research for those interested in discovering the root cause of the algorithmic performance results).

Another situation that may impact performance results (and could answer the reason why the Bellman-Ford algorithm was the fastest in this quantitative study) is the implementation of the pathfinding algorithms. Specifically, as discussed in Algfoor, Sunar, and Abdullah (2017, pp. 319-322, 324, 331); and in Kuipers, Feigenbaum, Hart, and Nilsson (2017, pp. 99-100), there are many different implementations of the A\* algorithm, and each may have different run-time performance characteristics. Likewise,

according to Lenharth, Nguyen, and Pingali (2016, p. 82), there are variants to the Bellman-Ford algorithm that may have different run-time performance characteristics. Additionally, per Kang and Lee (2017, p. 4), there are different implementations of Dijkstra's algorithm that yield different paths, depending on the need for *smooth* or *not smooth* paths, and therefore may exhibit different run-time performance. This implies that comparing the same-named algorithms from different graph analysis frameworks does not guarantee that one has compared the same algorithm implementation. Different implementations of same-named algorithms between frameworks may exhibit different runtime behavior. Detecting differences in the implementations would require deeper analysis of the source code, which was beyond the scope of this study.

One last reason to be discussed, which may account for the unexpected Bellman-Ford runtime speed (but not memory consumption) results may have to do with this study's random generated maps. There were only two types of adjacency matrices used in this study:  $200 \times 200$  graphs, or  $1000 \times 1000$  graphs. Additionally, so that all three algorithms could be fairly compared as discussed in Section 2 of this study, none of the random generated maps had negative weighted edges since only Bellman-Ford could support negative edge weights (Cormen, 2013). Next, I random-generated only small-world network graphs of the type described by Barnes (1954), De Sola Pool and Kocheck (1979), Korte and Milgram (1970), and Watts and Strogatz (1998). I did not random generate scale-free graphs of the type described by Albert, Jeong, and Barabási (1999), or Barabási (2016), nor random networks discussed by Erdős and Rényi (1961). Finally, I used uniform edge weight values of 1.0. These factors, together, could be surprisingly

favorable to Bellman-Ford. Perhaps use of scale-free graphs, or different edge weights, would have been more advantageous to Dijkstra's algorithm, or to the A\* algorithm. This could be implemented by having more categories for the *map complexity* independent variable (e.g., scale-free maps, variable edge weighted maps, random maps, regular maps, etc.), and therefore would have impacted this study by requiring more treatment groups. For example, 3 pathfinding algorithms  $\times$  2 graph analysis frameworks  $\times$  4 map complexities =  $3 \times 2 \times 4 = 24$  treatment groups, instead of the  $3 \times 2 \times 2 = 12$  treatment groups actually used in this study. Other options include using larger maps (e.g.,  $2000 \times 2000$ , or even larger). Clearly, using 24 (or more) factorial treatment groups may have generated more pathfinding algorithm results, but doing so was beyond the scope of this quantitative study, yet could be the topic of future research.

In conclusion, studying the problem of shortest path discovery through the lens of social network theory (while utilizing small-world network maps) is useful, as the results can be directly applied to the analysis of social networks, such as terrorist networks (Lenharth, Nguyen, & Pingali, 2016, p. 78), which is relevant today, given the often-reported instances in terrorist attacks over the last few years. But as noted above, small-world maps are only one type of graph. It would be useful to compare pathfinding algorithm performance with other types of graphs (e.g., regular grid maps, scale-free graphs, or even pure random networks). This means using small-world graphs alone only provides one set of answers. To obtain a more complete analysis of pathfinding algorithm performance in complex networks, more and different types of networks must be analyzed. Doing this could provide analyses of maps and networks more representative of

the real world, and therefore would provide results to a potentially broader audience, beyond those interested in only small-world networks and social network theory.

### **Applications to Professional Practice**

My results indicated that there was a significant impact on elapsed time and memory consumption by pathfinding algorithm, graph analysis framework, and map complexity. The implication is that the choice of graph analysis framework and pathfinding algorithm matter, but so does the structure of the underlying complex network. In short, this study's findings suggest that software engineers should try to know their problem domain (i.e., complex network) before choosing a graph analysis framework and pathfinding algorithm, because as was shown in Figure 46 and Figure 47, the complexity of the network map directly impacted elapsed time performance. This means that simply selecting a graph analysis framework and a pathfinding algorithm are insufficient if the software engineer is concerned about elapsed time performance. For example, if beforehand I have a general idea what kind of complex network I face, I could pick algorithms that are more compatible (e.g., more memory efficient) for that problem. For example, if the network I am working with has many nodes, then memory efficient frameworks and algorithms that can handle many nodes may be more useful than fast frameworks and algorithms that are less memory efficient.

The application of these findings to the professional practice indicated that not all open source frameworks exhibit the same runtime behavior. This may seem obvious in retrospect, but software engineers writing Python code to perform algorithmic pathfinding now have a starting point (the results of this study) to make a truly



quantitative assessment whether the Graph-Tool framework, or the Network-X framework is the right choice for their pathfinding problem. While Network-X used less memory than Graph-Tool (see Figure 43), it also exhibited slower run-time speed than Graph-Tool (see Figure 42). This represents an opportunity for the software engineer, who must decide which is more important: run-time performance, or memory consumption. This is a real-world tradeoff that software engineers often must balance, but if they lack comparative quantitative data on the frameworks in question then they may erroneously select the wrong software framework or algorithm for their problem. The results of this study may help prevent that error as I quantitatively evaluated two popular Python graph analysis frameworks, and provided data-driven statistics for software engineers in desperate need of real-world, comparative performance-oriented, applied algorithm and graph analysis framework advice.

Regarding social network analysis, the implications of this study are clear. In social network analysis finding connections between nodes is important (Korte & Milgram, 1970; Watts & Strogatz, 1998). If runtime speed is important, perhaps during a search for key players in criminal or terrorist networks (see Eiselt & Bhadury, 2015; and Medina, 2014), say because there is an insider tip regarding an impending terror attack, then the speed at which one can link nodes of interest together to find the key players in a complex terrorist network suggests a framework that performs quickly (e.g., Graph-Tool) may be more relevant. On the other hand, if one can process billions of nodes and edges with automated scripts, perhaps on a nightly basis when real-time speed is not necessary

but memory usage is critical due to huge node and edge volumes, then this suggests that a memory efficient framework may be more relevant (e.g., Network-X).

Regarding algorithmic pathfinding for resource-constrained smart agents (e.g., drones, self-driving cars, robots) located in network dead zones or GPS-denied environments, knowing the size and general complexity of the terrain maps are important, because, as was demonstrated in this study, map complexity and algorithm choice both impacted runtime performance in terms of elapsed time and memory consumption (see Figure 44, and Figure 45). Large and complex terrain maps may exhaust computer memory during pathfinding operations, therefore, using memory efficient algorithms, like the A\* algorithm (see Figure 47), may be the best choice.

### **Implications for Social Change**

There are two ways in which this study may immediately contribute to social change. First, from the perspective of terrorist and criminal network analyses, applying appropriate pathfinding algorithms and graph analysis frameworks may better enable law enforcement and intelligence agencies to find key players in criminal and terrorist networks of interest (see Eiselt & Bhadury, 2015; Medina, 2014), before they attack. Terrorist attacks happen, unfortunately, but by analyzing terrorist social networks it may be possible to identify and apprehend terror suspects and perpetrators (Lenharth, Nguyen, & Pingali, 2016, p. 78; see also McBride & Hewitt, 2013). This is particularly important given the recent terrorist attacks that occurred in (a) Manchester, UK, concert arena bombing on May 22, 2017; (b) St. Petersburg, Russia, metro train station suicide bombing on April 4, 2017; (c) Istanbul, Turkey, nightclub shooting on January 1, 2017;

(d) Orlando, FL, nightclub shooting on June 12, 2016; (e) Brussels, Belgium, airport and rail station bombings on March 22, 2016; (f) San Bernardino, CA, shooting on December 2, 2015; (g) Paris, France, shootings and Bataclan theatre bombing on November 13, 2015; and (h) the Charlie Hebdo shooting in Paris, France on January 7, 2015; to name just a few recent examples whose perpetrators were suspected to be involved in terrorist social networks. By combining pathfinding algorithms with complex network analysis and information technology, as demonstrated in this study, links between terror suspects might be detected before deadly attacks occur, giving law enforcement the chance to apprehend the terrorists, prevent loss of life, and thereby contribute to positive social change.

Second, this study may contribute to social change by providing a concrete, working example of the importance of gathering and analyzing quantitative data for the purpose of making informed technology decisions. From my 20+ years of experience as a software engineer, we are often asked to solve specific programming challenges, and are often given the flexibility to implement our own solutions. Yet if hard pressed for time, engineers sometimes choose the easiest solutions (e.g., use the same languages, tools, and methodologies already most familiar to us) because that is the short-term path of least resistance when facing tight time constraints. We do not always have sufficient time to quantitatively compare technologies, to make the best data-driven choice up front. The end result of rushed implementations and deployments is that sometimes we must re-engineer a previous so-called solution because it no longer scales well. And we may rely on word-of-mouth experiences (i.e., rumors and advice) from others, regarding which

technologies worked best for them (even though the specifics of their problem domain and circumstances may be different than ours). With this study, I presented not only comparisons of two popular Python graph analysis frameworks relevant to software engineers today and which Python programmers may immediately apply, but I also presented a method and working example for software engineers to follow which can be utilized to quantitatively compare many frameworks, for different problem domains. Software engineers can take this experience and apply it to their work, to discover which tools and algorithms work best for them, because such decisions would be supported by quantitative, data-driven facts, not by rumors or qualitative feelings.

### **Recommendations for Action**

This study is a call to action for all software engineers looking to move from a qualitative view of tools and technology, to a quantitative one. This study provides a working example how to quantitatively compare two or more algorithms, software tools and code frameworks (i.e., libraries). Engineers may start by picking the tools and services they wish to quantitatively compare. Then, consider creating a population pool of random generated objects relevant to their problem domain. Next, stratify the pool for subsequent stratified random sample selection. Finally, compare the results, quantitatively and statistically, as I demonstrated in this study. One tool may appear to dominate in many aspects, but not likely in all performance aspects. This is normal and is part of the engineering tradeoff that we often must make. By following the quantitative techniques demonstrated in this study, software engineers may be better informed and

educated how to make data-driven technology choices, not make guesses based solely on qualitative opinions.

The immediate application of this study's results should be considered by organizations implementing pathfinding software, whether it be for autonomous agents, or social network analysis, because the quantitative results may be relevant to their problem domains. Other organizations that perhaps do not use Python, or the Graph-Tool or Network-X frameworks may also benefit, because while the frameworks and languages may be interchangeable, the experience imparted by this study, in terms of how to generate local computer random-generated samples, stratify them, and then test them, can be applied to other experimental problem domains.

The results of this study may eventually be spread in peer-reviewed publications. I intend to publish aspects of this work in several peer-reviewed journals, such as, but not limited to, *Algorithms* (ISSN: 1999-4893), *The Journal of Discrete Algorithms* (ISSN: 1570-8667), *The Journal of Computational and Applied Mathematics* (ISSN: 0377-0427), and *The Python Papers* (ISSN: 1834-3147).

Additionally, at work I am a technology leader and plan to speak about how software engineers can make the transition from qualitative, *feelings-based* decision making, to quantitative, *data-driven* decision making. After publication I will also consider spreading this knowledge at my work place and beyond. The fora most appropriate for distribution of this knowledge include industry conferences and symposia. It could also include creation, or contribution to, one or more open source projects that are related to aspects of this doctoral study.

Finally, after I become a doctor, I will become an educator. In that role, I plan to spread knowledge of applied quantitative techniques to my students. This is because one way to have lasting impact on your profession is by positively influencing the next generation.

### **Recommendations for Further Study**

There are many ways one may approach comparative algorithm analysis. The concrete example provided by this doctoral study represented just one way to do so. Further research could utilize and compare more frameworks (including proprietary options, not just the open source frameworks used in this study); or use other computer languages (not just Python); and employ more complex map types instead of the two options used in this study. Another research approach could study the impact of dynamic maps on algorithmic pathfinding, as discussed by Zhang, Chan, Yang, and Deng (2017). Other research options include (a) use of 3D maps and 3D-oriented pathfinding algorithms instead of the 2D options used here; (b) utilize other hardware and operating systems (instead of the Apple laptop, and Mac OS used in this study); or (c) use virtual machines.

More exotic research options include testing multi-threaded implementations of pathfinding algorithms (e.g., see how many threads is the *optimal* number). Another option would be to research parallel processing oriented algorithms, systems and architectures as discussed in Chakaravarthy, Checconi, Murali, Petrini, and Sabharwal (2016); and in Ediger, Jiang, Riedy, and Bader (2013). Further research in this area might

indicate if parallel processing enhances pathfinding algorithm performance, and if so then at what price in terms of memory usage, increased code complexity, or other factors.

There are many options available when selecting a research design and methodology. Other avenues for further research may involve different experimental approaches or different research designs and methods, for example, instead of the *between-groups post-test only* approach used in this study, one could try a (a) longitudinal study; or (b) repeated-measures study; or (c) combine a repeated-measures, pre-test and post-test approach; or (d) combine several algorithms on very complex maps or problems, to see if combined solutions perform better than single algorithm solutions. One might explore trying qualitative approaches to algorithm comparison or evaluation, instead of the strictly quantitative approach I followed in this study. Also, changing the lens by which the study is interpreted, away from *social network theory* (used in this study) to another theoretical framework, such as *chaos theory* as discussed in Hung and Tu (2014) for example, might lead to new and interesting revelations.

Using a different statistic is an option worth considering. Perhaps finding causation within *random generated* data, as was the focus of this study, may not be as important as finding a more general correlation between someone else's *existing data*. In this case, not using MANOVA, and instead using another statistical method, such as logistic regression as discussed by Arcuri and Briand (2014), or other statistical methods, such as multiple regression, factor analysis, or discriminant analysis, could lead to interesting results and applications thereof. Changing the statistic utilized may require changing the number, and type, of dependent and independent variables used, as

discussed in Mertler and Reinhart (2017), but such changes could present many new opportunities from which interesting, useful algorithm research studies may emerge.

Regarding the limitations discussed in Section 1 of this doctoral study, several are still relevant. First, use of only two graph analysis frameworks (Network-X and Graph-Tool) may be less relevant to potential readers, particularly if their pathfinding framework choice is neither of the options I tested during this doctoral study. There are other graph frameworks that can also be tested in future research to broaden the appeal of this study, such as the iGraph framework discussed in Nocke, et al., (2015), and the Pajek framework, as discussed in Ma, Fukuda, and Schmöcker (2013), among others. Another limitation noted in Section 1 of this study was the deliberate avoidance of discovering "why" a particular framework and/or algorithm performed as it did, since that may have required line-by-line code analysis and/or code profiling, which was clearly beyond the scope of this study. Some readers may be interested in knowing exactly *why* the Bellman-Ford algorithm exhibited faster runtime performance over Dijkstra in this study, when according to the algorithm theory discussed in Cormen, Leiserson, Rivest, and Stein (2009), the Bellman-Ford algorithm should have exhibited generally slower runtime performance than Dijkstra. The quality of the pathfinding algorithm implementation is important as discussed in Kapanowski and Gałuszka (2016), and may have impacted the runtime performance results of this study, as was my deliberate use of Python, not Java, C, or some other compiled computer language, although Python is a good language for data structures development, as noted by Kapanowski and Gałuszka (2016, p. 1).



Next, regarding the delimiters I discussed in Section 1, several are still relevant and deserve discussion. I delimited this study to only use the Python computer language. As discussed in Farooq, Khan, Ahmad, Islam, and Abid (2014), each computer language has its own characteristics. By choosing Python, I unfortunately limited this study mainly to personal computers (PCs), laptops and servers, but not to mobile devices. Had I selected the Objective-C or Swift programming languages for implementation, I could have run the algorithm comparison tests on mobile Apple devices such as the iPhone, iPad, or iWatch smart watch. This would have allowed me to collect algorithm performance data on Apple mobile devices, which may have broadened the appeal of this study. Alternately, choosing Java as the implementation programming language would have made it easier to benchmark the pathfinding algorithms on Android mobile devices, and servers, laptops and PCs, since Java runs on many of these devices (but not on Apple mobile phones it should be noted). However, had I used Objective-C, Swift, I would have had to find different graph analysis frameworks because it seemed very difficult to use Graph-Tool and Network-X on Apple devices, so by switching to the Objective-C or Swift languages, I would likely have had to choose other graph analysis frameworks too. Java seemed to face a similar situation, regarding the chosen graph analysis frameworks. In conclusion, while using Objective-C, Swift or Java may have broadened the appeal of this study, doing so would have significantly complicated my experiment, and would have caused delays.

As discussed earlier, another self-imposed boundary (i.e., delimiter), was the deliberate use of only two different map complexities (i.e., *low complexity* and *high*

*complexity*). Not all real-world maps nicely fit into a  $200 \times 200$  adjacency matrix, nor a  $1000 \times 1000$  adjacency matrix. Real world complex networks may be quite complicated and certainly not represented by the two map complexity options used in this study.

Adding more map options may broaden the appeal of this study, and certainly represent a future research opportunity. Some complex network types, such as scale-free networks (Barabási, 2016), random networks (Erdős & Rényi, 1961), and even dynamically changing maps as discussed in Franke and Ivanova (2014) represent future research opportunities.

Finally, use of parallel programming techniques and algorithms, as discussed in Bazregar, Piltan, Nabaee, and Ebrahimi (2013) is something that was not explicitly tested in this experiment, but could be useful to readers working with billions of nodes and in need of high performance graph analysis frameworks that support parallel programming. This study does not answer how parallel programming enhanced pathfinding algorithms would perform, and at what price in terms of CPU, memory, and increased programming complexity, as those topics were deemed beyond the scope of this study, yet they could represent future research opportunities for the next intrepid researcher.

### **Reflections**

In conducting research on pathfinding algorithms, using graph analysis frameworks, I had some pre-conceived notions (garnered from my days as a hardcore C and C++ programmer at Microsoft, back in the 1990s) that any framework written in C or C++ (e.g., Graph-Tool) would be superior in all aspects of run-time performance, to a framework written in an interpreted language such as Python (e.g., Network-X).

However, while the Python framework (Network-X) was not the fastest (Graph-Tool was faster), Network-X was not too much slower, but it was much easier to use and program the Network-X framework, than the Graph-Tool framework. This represented a classic dilemma we software engineers sometimes face: "option A" is easy to program and debug, but has slow run-time performance; while "option B" is harder to program and debug, but has faster run-time performance. My pre-conceived notion that Python would exhibit very poor performance overall, was unjust and has since been corrected, due to what I learned during this study about the Python computer language, and Python runtime performance. Python is a useful language, it has many tools and supporting frameworks, it is easy to learn (I was new to Python when I started this doctoral study), and now I intend to use Python much more.

Regarding the DIT program, I honestly did not know much about the value of the scholar-practitioner role before embarking on the DIT journey. I just wanted a doctoral degree, but I did not have the time to earn one at a traditional brick-and-mortar college. Being an early student in the DIT program has its challenges, including the sad reality that there were no DIT doctoral studies from which I could draw experience and knowledge. I had to rely on DBA studies, which while useful up to a point, are not the same as a true DIT doctoral study.

I have learned to become a much better researcher, and now I have a much better grasp of statistics, and statistics-friendly tools such as SPSS, Excel, and even the R programming language. These facts and skills will help me greatly at work, and will help me be a productive researcher. Another pre-conceived notion I had regarding PhDs vs.

applied doctoral degree holders, was my assumption that in traditional PhD programs one creates a new theory, and that holders of applied doctoral degrees were second class citizens because they did not create new theories. That viewpoint has since been corrected. Applied doctorates, such as the DIT, are valuable, because someone must take that new theory, test it, and then apply it to a real-world problem to help prove the utility of that theory in the real world. As an applied scholar practitioner and applied researcher, I am that bridge between pure theory and the real world. I did not know I would become the bridge when I first started on the DIT path, but I know it now, and I will be forever empowered by it.

### **Summary and Study Conclusions**

Algorithm choice matters. Framework choice matters. Knowledge of one's problem domain matters. Software engineers are responsible for implementing the software which runs many aspects of the modern world (e.g., self-driving cars, planetary surface rovers, implantable biomedical devices, drones, automated financial trading systems). While it is possible to get by as a software engineer using only qualitative methods to assess software frameworks and algorithm choices, we and our customers can benefit from our use of data-driven, quantitative approaches to software engineering. Software engineers can benefit, long term, by spending some time, up front, in quantitative evaluation of the performance of a product, framework, or service, prior to implementation and eventual production deployment. This may help prevent unscalable software solutions from being implemented, to the benefit of our customers, and ultimately, to us software engineers (who ultimately must support such software).

Based on the MANOVA analysis of the frameworks, algorithms, and population pool tested in this quantitative experimental study, the null hypothesis was rejected. It was determined that there was a statistically significant causal relationship between pathfinding algorithms, graph analysis frameworks, map complexity, elapsed time, and computer memory consumption. This information will aid decision makers (e.g., software engineers, software architects, systems designers, technical managers) determine which graph analysis frameworks and algorithms to use. But more broadly, the information provided by this study should empower decision makers with a working example of quantitative data-driven analysis and decision making, and the knowledge how to quantitatively compare most any software framework or algorithm. If the product or service being compared produces output that can be measured, it can likely be quantitatively compared, and therefore its performance statistically analyzed. If this can be done early, before implementing and deploying the proposed product or service into production, it may save software engineers time over the long run, prevent customer frustration due to poor performance, and help avoid costly future rework.

## References

- Algfoor, Z. A., Sunar, M. S., & Abdullah, A. (2017). A new weighted pathfinding algorithms to reduce the search time on grid maps. *Expert Systems with Applications*, *71*, 319-331. doi:10.1016/j.eswa.2016.12.003
- Algfoor, Z. A., Sunar, M. S., & Kolivand, H. (2015). A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. *International Journal of Computer Games Technology*, 2015. doi:10.1155/2015/736138
- Abdulkadir, S. I., Fadzli, S. A., Jamal, A. A., Makhtar, M., Awang, M. K., Mohamad, M., & Susilawati, F. (2015). Indoor Global Path Planning Based on Critical Cells Using Dijkstra Algorithm. *Journal of Theoretical and Applied Information Technology*, *79*(1). Retrieved from <http://www.jatit.org>
- Adewumi, A., Kagamba, J., & Alochukwu, A. (2016). Application of Chaos Theory in the Prediction of Motorized Traffic Flows on Urban Networks. *Mathematical Problems in Engineering*, 2016. doi:10.1155/2016/5656734
- Afuah, A. (2013). Are network effects really all about size? The role of structure and conduct. *Strategic Management Journal*, *34*(3), 257-273. doi:10.1002/smj.2013
- Akeret, J., Gamper, L., Amara, A., & Refregier, A. (2015). HOPE: A Python just-in-time compiler for astrophysical computations. *Astronomy and Computing*, *10*, 1-8. doi:10.1016/j.ascom.2014.12.001
- Albert, R., & Barabási, A. L. (2002). Statistical Mechanics of Complex Networks. *Reviews of Modern Physics*, *74*(1), 47. doi:10.1103/RevModPhys.74.47
- Albert, R., Jeong, H., & Barabási, A. L. (1999). Internet: Diameter of the World-Wide

Web. *Nature*, 401(6749), 130-131. doi:10.1038/43601

- Almaghairbe, R., & Roper, M. (2016). Separating passing and failing test executions by clustering anomalies. *Software Quality Journal*, 1-38. doi:10.1007/s11219-016-9339-1
- Alotaibi, E. T. S., & Al-Rawi, H. (2016). MRPPSim: A Multi-Robot Path Planning Simulation. *International Journal of Advanced Computer Science and Applications*, 7(8). Retrieved from <http://thesai.org/Publications>
- Amin, H. U., Malik, A. S., Kamel, N., Chooi, W. T., & Hussain, M. (2015). P300 correlates with learning & memory abilities and fluid intelligence. *Journal of Neuroengineering and Rehabilitation*, 12(1), 87. doi:10.1186/s12984-015-0077-6
- Ammar, A., Bennaceur, H., Châari, I., Koubâa, A., & Alajlan, M. (2015). Relaxed Dijkstra and A\* with Linear Complexity for Robot Path Planning Problems in Large-Scale Grid Environments. *Soft Computing*, 1-23. doi:10.1007/s00500-015-1750-1
- Arcuri, A., & Briand, L. (2014). A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, 24(3), 219-250. doi:10.1002/stvr.1486
- Ariel, B., Sutherland, A., Henstock, D., Young, J., Drover, P., Sykes, J., ... & Henderson, R. (2016). "Contagious Accountability" A Global Multisite Randomized Controlled Trial on the Effect of Police Body-Worn Cameras on Citizens' Complaints Against the Police. *Criminal Justice and Behavior*, 0093854816668218. doi:10.1177/0093854816668218

- Baingana, B., & Giannakis, G. B. (2017). Tracking Switched Dynamic Network Topologies From Information Cascades. *IEEE Transactions on Signal Processing*, 65(4), 985-997. doi:10.1109/TSP.2016.2628354
- Balaguru, S., Nallathamby, R., & Robin, C. R. (2015). A Novel Approach for Analyzing the Social Network. *Procedia Computer Science*, 48, 687-692. doi:10.1016/j.procs.2015.04.202
- Balakrishnan, R., & Penno, M. (2014). Causality in the Context of Analytical Models and Numerical Experiments. *Accounting, Organizations and Society*, 39(7), 531-534. doi:10.1016/j.aos.2013.09.004
- Barabási, A. L. (2016). *Network science*. Cambridge, United Kingdom: Cambridge University Press.
- Barabási, A. L., & Albert, R. (1999). Emergence of Scaling in Random Networks. *Science*, 286(5439), 509-512. doi:10.1126/science.286.5439.509
- Barnes, J. A. (1954). Class and Committees in a Norwegian Island Parish. *Human Relations*, 7(1), 39-58. doi:10.1177/001872675400700102
- Barnes-Mauthe, M., Gray, S. A., Arita, S., Lynham, J., & Leung, P. (2015). What determines social capital in a social–ecological system? Insights from a network perspective. *Environmental Management*, 55(2), 392-410. doi:10.1007/s00267-014-0395-7
- Barnham, C. (2015). Quantitative and qualitative research: perceptual foundations. *International Journal of Market Research*, 57(6), 837-854. doi:10.2501/IJMR-2015-070



- Bazregar, M., Piltan, F., Nabae, A., & Ebrahimi, M. M. (2013). Parallel Soft Computing Control Optimization Algorithm for Uncertainty Dynamic Systems. *International Journal of Advanced Science and Technology*, 51, 93-106. Retrieved from <http://www.sersc.org/journals/IJAST>
- Becton, M., & Wang, X. (2015). Grain-size dependence of mechanical properties in polycrystalline boron-nitride: a computational study. *Physical Chemistry Chemical Physics*, 17(34), 21894-21901. doi:10.1039/c5cp03460d
- Bettany-Saltikov, J., & Whittaker, V. J. (2014). Selecting the most appropriate inferential statistical test for your quantitative research study. *Journal of Clinical Nursing*, 23(11-12), 1520-1531. doi:10.1111/jocn.12343
- Bezerra, L. C., Goldberg, E. F., Goldberg, M. C., & Buriol, L. S. (2013). Analyzing the impact of MOACO components: An algorithmic study on the multi-objective shortest path problem. *Expert Systems with Applications*, 40(1), 345-355. doi:10.1016/j.eswa.2012.07.052
- Bird, K. D., & Hadzi-Pavlovic, D. (2014). Controlling the maximum familywise Type I error rate in analyses of multivariate experiments. *Psychological Methods*, 19(2), 265. doi:10.1037/a0033806
- Blasco-Arcas, L., Hernandez-Ortega, B., & Jimenez-Martinez, J. (2013). Adopting television as a new channel for e-commerce. The influence of interactive technologies on consumer behavior. *Electronic Commerce Research*, 13(4), 457-475. doi:10.1007/s10660-013-9132-1
- Boguchwal, L. (2015). Shortest Path Algorithms for Functional Environments. *Discrete*

*Optimization*, 18, 217-251. doi:10.1016/j.disopt.2015.09.006

Bohács, G., Gyimesi, A., & Rózsa, Z. (2015). Development of an Intelligent Path Planning Method for Materials Handling Machinery at Construction Sites. *Periodica Polytechnica Transportation Engineering*, 44(1), 13-22. doi:10.3311/PPtr.8035

Brodnik, A., & Grgurovič, M. (2017). Solving all-pairs shortest path by single-source computations: Theory and practice. *Discrete Applied Mathematics*. doi:10.1016/j.dam.2017.03.008

Brooks, B., Hogan, B., Ellison, N., Lampe, C., & Vitak, J. (2014). Assessing Structural Correlates to Social Capital in Facebook Ego Networks. *Social Networks*, 38, 1-15. doi:10.1016/j.socnet.2014.01.002

Buliung, R. N., Larsen, K., Faulkner, G. E., & Stone, M. R. (2013). The “path” not taken: Exploring structural differences in mapped-versus shortest-network-path school travel routes. *American journal of public health*, 103(9), 1589-1596. doi:10.2105/AJPH.2012.301172

Capaldo, A., & Giannoccaro, I. (2015). Interdependence and Network-Level Trust in Supply Chain Networks: A Computational Study. *Industrial Marketing Management*, 44, 180-195. doi:10.1016/j.indmarman.2014.10.001

Cavazza, M., Aranyi, G., & Charles, F. (2017). BCI Control of Heuristic Search Algorithms. *Frontiers in Neuroinformatics*, 11. doi:10.3389/fninf.2017.00006

Chakaravarthy, V., Checconi, F., Murali, P., Petrini, F., & Sabharwal, Y. (2016). Scalable single source shortest path algorithms for massively parallel systems. *IEEE*

*Transactions on Parallel and Distributed Systems.*

doi:10.1109/TPDS.2016.2634535

- Chewning, L. V., & Doerfel, M. L. (2013). Integrating crisis into the organizational lifecycle through transitional networks. *International Journal of Humanities and Social Science*, 3, 39-52. Retrieved from <http://www.ijhssnet.com/journals>
- Clair, T. S., Cook, T. D., & Hallberg, K. (2014). Examining the internal validity and statistical precision of the comparative interrupted time series design by comparison with a randomized experiment. *American Journal of Evaluation*, 35(3), 311-327. doi:10.1177/1098214014527337
- Cohen, J. (1992). A Power Primer. *Psychological Bulletin*, 112(1), 155. Retrieved from <http://www2.psych.ubc.ca>
- Cokley, K. O., & Awad, G. H. (2013). In defense of quantitative methods: Using the “master’s tools” to promote social justice. *Journal for Social Action in Counseling and Psychology*, 5(2), 26-41. Retrieved from <http://jsacp.tumblr.com>
- Collins, C. S., & Cooper, J. E. (2014). Emotional intelligence and the qualitative researcher. *International Journal of Qualitative Methods*, 13(1), 88-103. Retrieved from <https://ejournals.library.ualberta.ca>
- Cormen, T. (2013). *Algorithms Unlocked*. Cambridge, MA: Massachusetts Institute of Technology
- Cormen, T., Leiserson, C., Rivest, R., & Stein, C. (2009). *Introduction to Algorithms (3rd ed.)*. Cambridge, MA: Massachusetts Institute of Technology
- Csardi, G., & Nepusz, T. (2006). The iGraph software package for complex network

- research. *InterJournal, Complex Systems*, 1695(5), 1-9. Retrieved from <http://www.interjournal.org>
- D'Angelo, G., D'Emidio, M., & Frigioni, D. (2014). A loop-free shortest-path routing algorithm for dynamic networks. *Theoretical Computer Science*, 516, 1-19.  
doi:10.1016/j.tcs.2013.11.001
- Daigneault, P. M., & Jacob, S. (2014). Unexpected but Most Welcome Mixed Methods for the Validation and Revision of the Participatory Evaluation Measurement Instrument. *Journal of Mixed Methods Research*, 8(1), 6-24.  
doi:10.1177/1558689813486190
- Dantas-Torres, F. (2015). Climate change, biodiversity, ticks and tick-borne diseases: the butterfly effect. *International Journal for Parasitology: Parasites and Wildlife*, 4(3), 452-461. doi:10.1016/j.ijppaw.2015.07.001
- Dawson, J. Q., Munzner, T., & McGrenere, J. (2015). A search-set model of path tracing in graphs. *Information Visualization*, 14(4), 308-338.  
doi:10.1177/1473871614550536
- Day, C. (2014). Python Power. *Computing in Science and Engineering*, 16(1), 88.  
doi:10.1109/MCSE.2014.26
- De Sola Pool, I., & Kochen, M. (1979). Contacts and Influence. *Social Networks*, 1(1), 5-51. doi:10.1016/0378-8733(78)90011-4
- Dean, D. J. (2013). Finding Optimal Travel Routes with Uncertain Cost Data. *Transactions in GIS*, 17(2), 159-181. doi:10.1111/j.1467-9671.2012.01360.x
- Delen, D., Kuzey, C., & Uyar, A. (2013). Measuring firm performance using financial

- ratios: A decision tree approach. *Expert Systems with Applications*, 40(10), 3970-3983. doi:10.1016/j.eswa.2013.01.012
- Dierbach, C. (2014). Python as a First Programming Language. *Journal of Computing Sciences in Colleges*, 29(6), 153-154. Retrieved from <http://dl.acm.org>
- do Carmo Machado, I., McGregor, J. D., Cavalcanti, Y. C., & De Almeida, E. S. (2014). On strategies for testing software product lines: A systematic literature review. *Information and Software Technology*, 56(10), 1183-1199. doi:10.1016/j.infsof.2014.04.002
- Donaldson, L., Qiu, J., & Luo, B. N. (2013). For rigour in organizational management theory research. *Journal of Management Studies*, 50(1), 153-172. doi:10.1111/j.1467-6486.2012.01069.x
- Donofrio, B., Class, Q., Lahey, B., & Larsson, H. (2014). Testing the Developmental Origins of Health and Disease Hypothesis for Psychopathology Using Family-Based, Quasi-Experimental Designs. *Child Development Perspectives*, 8(3), 151-157. doi:10.1111/cdep.12078
- Drislane, L. E., Patrick, C. J., Sourander, A., Sillanmäki, L., Aggen, S. H., Elonheimo, H., ... & Kendler, K. S. (2014). Distinct variants of extreme psychopathic individuals in society at large: Evidence from a population-based sample. *Personality Disorders: Theory, Research, and Treatment*, 5(2), 154. doi:10.1037/per0000060
- Drost, R. (2013). Memory and decision making: determining action when the sirens sound. *Weather, Climate, and Society*, 5(1), 43-54. doi:10.1175/WCAS-D-11-

00042.1

- du Plessis, H., & Van Niekerk, A. (2014). A new GISc framework and competency set for curricula development at South African universities. *South African Journal of Geomatics*, 3(1), 1-12. Retrieved from <http://www.ajol.info>
- Dunn, J. G., & Weissman, J. S. (2016). Plastid: nucleotide-resolution analysis of next-generation sequencing and genomics data. *BMC Genomics*, 17(1), 958. doi:10.1186/s12864-016-3278-x
- Dunn, S. L., Arslanian-Engoren, C., DeKoekkoek, T., Jadack, R., & Scott, L. D. (2015). Secondary data analysis as an efficient and effective approach to nursing research. *Western Journal of Nursing Research*, 37(10), 1295-1307. doi:10.1177/0193945915570042
- Dybå, T., Kampenes, V. B., & Sjøberg, D. I. (2006). A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8), 745-755. doi:10.1016/j.infsof.2005.08.009
- Ediger, D., Jiang, K., Riedy, E. J., & Bader, D. A. (2013). GraphCT: Multithreaded algorithms for massive graph analysis. *IEEE Transactions on Parallel and Distributed Systems*, 24(11), 2220-2229. doi:10.1109/TPDS.2012.323
- Eiselt, H. A., & Bhadury, J. (2015). The Use of Structures in Communication Networks to Track Membership in Terrorist Groups. *Journal of Terrorism Research*, 6(1). doi:10.15664/jtr.1073
- Erdős, P., & Rényi, A. (1961). On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1-2), 261-267. doi:10.1007/BF02066689

- Erikson, E. (2013). Formalist and Relationalist Theory in Social Network Analysis. *Sociological Theory*, 31(3), 219-242. doi:10.1177/0735275113501998
- Ersoy, M., & Akbulut, Y. (2014). Cognitive and affective implications of persuasive technology use on mathematics instruction. *Computers & Education*, 75, 253-262. doi:10.1016/j.compedu.2014.03.009
- Farooq, M. S., Khan, S. A., Ahmad, F., Islam, S., & Abid, A. (2014). An Evaluation Framework and Comparative Analysis of the Widely Used First Programming Languages. *PLoS One*, 9(2), 02. doi:10.1371/journal.pone.0088941
- Faul, F., Erdfelder, E., Buchner, A., & Lang, A. G. (2009). Statistical power analyses using G\* Power 3.1: Tests for correlation and regression analyses. *Behavior Research Methods*, 41(4), 1149-1160. doi:10.3758/brm.41.4.1149
- Fernández-Berrocal, P., Cabello, R., Castillo, R., & Extremera, N. (2012). Gender differences in emotional intelligence: The mediating effect of age. *Psicologia Conductual*, 20(1), 77. Retrieved from <https://www.researchgate.net>
- Field, A. (2013). *Discovering Statistics using IBM SPSS Statistics (4th ed.)*. London: United Kingdom. SAGE Publications Ltd.
- Fraigniaud, P., & Giakkoupis, G. (2014). Greedy Routing in Small-World Networks with Power-Law Degrees. *Distributed Computing*, 27(4), 231-253. doi:10.1007/s00446-014-0210-y
- Franke, R., & Ivanova, G. (2014). FALCON or How to Compute Measures Time Efficiently on Dynamically Evolving Dense Complex Networks? *Journal of Biomedical Informatics*, 47, 62-70. doi:10.1016/j.jbi.2013.09.005

- Freeman, J. (2015). Open source tools for large-scale neuroscience. *Current Opinion in Neurobiology*, 32, 156-163. doi:10.1016/j.conb.2015.04.002
- Fritz, M. S., Cox, M. G., & MacKinnon, D. P. (2015). Increasing statistical power in mediation models without increasing sample size. *Evaluation & the Health Professions*, 38(3) 343-366. doi:10.1177/0163278713514250
- Gass, O., Meth, H., & Maedche, A. (2014). PaaS Characteristics for Productive Software Development: An Evaluation Framework. *IEEE Internet Computing*, 18(1), 56-64. doi:10.1109/MIC.2014.12
- Gassen, J. (2014). Causal Inference in Empirical Archival Financial Accounting Research. *Accounting, Organizations and Society*, 39(7), 535-544. doi:10.1016/j.aos.2013.10.004
- Gaston, A., Wilson, P. M., Mack, D. E., Elliot, S., & Prapavessis, H. (2013). Understanding physical activity behavior and cognitions in pregnant women: An application of self-determination theory. *Psychology of Sport and Exercise*, 14(3), 405-412. doi:10.1016/j.psychsport.2012.12.009
- Gerber, N., Bell, B., Gavaghan, K., Weisstanner, C., Caversaccio, M., & Weber, S. (2014). Surgical planning tool for robotically assisted hearing aid implantation. *International Journal of Computer Assisted Radiology and Surgery*, 9(1), 11-20. doi:10.1007/s11548-013-0908-5
- Gibson, H., & Vickers, P. (2016). Using adjacency matrices to lay out larger small-world networks. *Applied Soft Computing*, 42, 80-92. doi:10.1016/j.asoc.2016.01.036
- González-Bailón, S. (2013). Social Science in the Era of Big Data. *Policy & Internet*,



5(2), 147-160. doi:10.1002/1944-2866.POI328

Gorelick, M., & Ozsvald, I. (2014). *High Performance Python: Practical Performant Programming for Humans*. Sebastopol, CA: O'Reilly Media, Inc.

Granato, D., de Araújo Calado, V. M., & Jarvis, B. (2014). Observations on the use of statistical methods in food science and technology. *Food Research International*, 55, 137-149. doi:10.1016/j.foodres.2013.10.024

Groeneveld, S., Tummers, L., Bronkhorst, B., Ashikali, T., & Van Thiel, S. (2015). Quantitative methods in public administration: Their use and development through time. *International Public Management Journal*, 18(1), 61-86. doi:10.1080/10967494.2014.972484

Hair, J. F., Anderson, R. E., Babin, B. J., & Black, W. C. (2010). *Multivariate Data Analysis (7th ed.)*. Upper Saddle River, NJ: Pearson.

Hancox, J. E., Quested, E., Thøgersen-Ntoumani, C., & Ntoumanis, N. (2015). An intervention to train group exercise class instructors to adopt a motivationally adaptive communication style: a quasi-experimental study protocol. *Health Psychology and Behavioral Medicine*, 3(1), 190-203. doi:10.1080/21642850.2015.1074075

Harenberg, S., Bello, G., Gjeltrema, L., Ranshous, S., Harlalka, J., Seay, R., ... & Samatova, N. (2014). Community detection in large-scale networks: a survey and empirical evaluation. *Wiley Interdisciplinary Reviews: Computational Statistics*, 6(6), 426-439. doi:10.1002/wics.1319

Hayes, A. F., & Preacher, K. J. (2014). Statistical mediation analysis with a

- multicategorical independent variable. *British Journal of Mathematical and Statistical Psychology*, 67(3), 451-470. doi:10.1111/bmsp.12028
- Hearnshaw, E. J., & Wilson, M. M. (2013). A Complex Network Approach to Supply Chain Network Theory. *International Journal of Operations & Production Management*, 33(4), 442-469. doi:10.1108/01443571311307343
- Henderson, V. C., Kimmelman, J., Fergusson, D., Grimshaw, J. M., & Hackam, D. G. (2013). Threats to validity in the design and conduct of preclinical efficacy studies: a systematic review of guidelines for in vivo animal experiments. *PLoS Med*, 10(7), e1001489. doi:10.1371/journal.pmed.1001489
- Hoare, Z., & Hoe, J. (2013). Understanding Quantitative Research: Part 2. *Nursing Standard*, 27(18), 48-55. doi:10.7748/ns2013.01.27.18.48.c9488
- Horga, G., Kaur, T., & Peterson, B. S. (2014). Annual Research Review: Current limitations and future directions in MRI studies of child-and adult-onset developmental psychopathologies. *Journal of Child Psychology and Psychiatry*, 55(6), 659-680. doi:10.1111/jcpp.12185
- Howitt, D., & Cramer, D. (2014). *Introduction to SPSS Statistics in Psychology* (6th ed.). Edinburgh Gate, UK: Pearson Education Ltd.
- Hric, D., Peixoto, T. P., & Fortunato, S. (2016). Network structure, metadata, and the prediction of missing nodes and annotations. *Physical Review X*, 6(3), 031038. doi:10.1103/PhysRevX.6.031038
- Huang, L., Zhang, B., Yuan, X., Zhang, C., & Ma, A. (2016). A novel Bi-Ant colony optimization algorithm for solving multi-objective service selection problem.

- Journal of Intelligent & Fuzzy Systems*, 31(2), 873-884. doi:10.3233/JIFS-169018
- Hung, S. C., & Tu, M. F. (2014). Is small actually big? The chaos of technological change. *Research Policy*, 43(7), 1227-1238. doi:10.1016/j.respol.2014.03.003
- Hunt, J. M. (2015). Python in CS1 - Not. *Journal of Computing Sciences in Colleges*, 31(2), 172-179. Retrieved from <http://dl.acm.org>
- Hwang, K., Hallquist, M. N., & Luna, B. (2013). The development of hub architecture in the human functional brain network. *Cerebral Cortex*, 23(10), 2380-2393. doi:10.1093/cercor/bhs227
- Ionel-Alin, L., & Irimie Emil, P. (2013). Conceptual delimitations on sustainable development. *Annals of the University of Oradea Economic Science Series*, 22(1), 252-261. Retrieved from <http://anale.steconomieuoradea.ro/en/journal-archive>
- Ittner, C. D. (2014). Strengthening Causal Inferences in Positivist Field Studies. *Accounting, Organizations and Society*, 39(7), 545-549. doi:10.1016/j.aos.2013.10.003
- Jackson, M. R. (2015). Resistance to qual/quant parity: Why the “paradigm” discussion can’t be avoided. *Qualitative Psychology*, 2(2), 181. doi:10.1037/qup0000031
- Johnston, K., Tanner, M., Lalla, N., & Kawalski, D. (2013). Social Capital: The Benefit of Facebook ‘Friends’. *Behaviour & Information Technology*, 32(1), 24-36. doi:10.1080/0144929X.2010.550063
- Jukna, S., & Schnitger, G. (2016). On the optimality of Bellman–Ford–Moore shortest path algorithm. *Theoretical Computer Science*, 628, 101-109. doi:10.1016/j.tcs.2016.03.014

- Kang, B., & Choo, H. (2016). Network-based Algorithms for Evacuation: A Survey. *International Journal of Disaster Risk Reduction*. doi:10.1016/j.ijdr.2016.10.004
- Kang, J. Y., & Lee, B. S. (2017). Optimisation of pipeline route in the presence of obstacles based on a least cost path algorithm and laplacian smoothing. *International Journal of Naval Architecture and Ocean Engineering*. doi:10.1016/j.ijnaoe.2017.02.001
- Kapanowski, A., & Gałuszka, Ł. (2016). Weighted Graph Algorithms with Python. *The Python Papers*, 11(3). Retrieved from <http://ojs.pythonpapers.org/index.php/tpp/issue/view/37>
- Katz, J. (2015). A theory of qualitative methodology: The social system of analytic fieldwork. *Méthod (e) s: African Review of Social Sciences Methodology*, 1(1-2), 131-146. doi:10.1080/23754745.2015.1017282
- Kaur, M., & Gangal, A. (2015). Comparative Analysis of Various Routing Protocol in MANET. *International Journal of Computer Applications*, 118(8). doi:10.5120/20766-3207
- Kaye, J., Whitley, E. A., Lund, D., Morrison, M., Teare, H., & Melham, K. (2015). Dynamic consent: a patient interface for twenty-first century research networks. *European Journal of Human Genetics*, 23(2), 141-146. doi:10.1038/ejhg.2014.71
- Kepner, J., Bade, D., Buluç, A., Gilbert, J., Mattson, T., & Meyerhenke, H. (2015). Graphs, Matrices, and the GraphBLAS: Seven Good Reasons. *Procedia Computer Science*, 51, 2453–2462. doi:10.1016/j.procs.2015.05.353
- Kirkwood, A., & Price, L. (2013). Examining some assumptions and limitations of

research on the effects of emerging technologies for teaching and learning in higher education. *British Journal of Educational Technology*, 44(4), 536-543. doi:10.1111/bjet.12049

Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K., & Rosenberg, J. (2002). Preliminary guidelines for empirical research in software engineering. *IEEE Transactions on Software Engineering*, 28(8), 721-734. doi:10.1109/TSE.2002.1027796

Kleinberg, J. (2000). The Small-World Phenomenon: An Algorithmic Perspective. In *ACM Symposium on the Theory of Computing, Thirty-Second Annual Proceedings* (pp. 163-170). ACM. doi:10.1145/335305.335325

Klimaszewski, J. (2014). The efficiency of the A\* algorithm's implementations in selected programming languages. *Journal of Theoretical and Applied Computer Science*, 8(2), 63-71. Retrieved from <http://www.jtacs.org>

Koç, Ç., Bektaş, T., Jabali, O., & Laporte, G. (2016). Thirty years of heterogeneous vehicle routing. *European Journal of Operational Research*, 249(1), 1-21. doi:10.1016/j.ejor.2015.07.020

Korkmaz, S., Goksuluk, D., & Zararsiz, G. (2014). MVN: an R package for assessing multivariate normality. *The R Journal*, 6(2), 151-162. Retrieved from <https://journal.r-project.org/archive/2014-2/>

Korte, C., & Milgram, S. (1970). Acquaintance Networks Between Racial Groups: Application of the Small World Method. *Journal of Personality and Social Psychology*, 15(2), 101-108. Retrieved from <http://www.apa.org>

- Koujaku, S., Takigawa, I., Kudo, M., & Imai, H. (2016). Dense Core Model for Cohesive Subgraph Discovery. *Social Networks*, *44*, 143-152.  
doi:10.1016/j.socnet.2015.06.003
- Krause, J., Croft, D. P., & James, R. (2007). Social network theory in the behavioral sciences: potential applications. *Behavioral Ecology and Sociobiology*, *62*(1), 15-27. doi:10.1007/s00265-007-0445-8
- Krauss, M., Burghaus, R., Lippert, J., Niemi, M., Neuvonen, P., Schuppert, A., ... & Görlitz, L. (2013). Using Bayesian-PBPK modeling for assessment of inter-individual variability and subgroup stratification. *In Silico Pharmacology*, *1*(1).  
doi:10.1186/2193-9616-1-6
- Krishnan, S. S., & Sitaraman, R. K. (2013). Video stream quality impacts viewer behavior: inferring causality using quasi-experimental designs. *IEEE/ACM Transactions on Networking*, *21*(6), 2001-2014.  
doi:10.1109/TNET.2013.2281542
- Kuipers, B., Feigenbaum, E. A., Hart, P. E., & Nilsson, N. J. (2017). Shakey: From Conception to History. *AI Magazine*, *38*(1), 88-103. Retrieved from <http://ai.stanford.edu/%7Enilsson/publications.html#essays>
- Kumar, S., Nilsen, W. J., Abernethy, A., Atienza, A., Patrick, K., Pavel, M., ... & Hedeker, D. (2013). Mobile health technology evaluation: the mHealth evidence workshop. *American Journal of Preventive Medicine*, *45*(2), 228-236.  
doi:10.1016/j.amepre.2013.03.017
- Lakshmi, S., & Mohideen, M. A. (2013). Issues in Reliability and Validity of Research.

*International Journal of Management Research and Reviews*, 3(4), 2752.

Retrieved from <http://ijmrr.com>

Lampis, F., Díaz-Empanaza, I., & Banerjee, A. (2015). How to use SETAR models in  
gretl. *Computational Economics*, 46(2), 231-241. doi:10.1007/s10614-014-9445-8

Lamprecht, D., Strohmaier, M., Helic, D., Nyulas, C., Tudorache, T., Noy, N. F., &  
Musen, M. A. (2015). Using ontologies to model human navigation behavior in  
information networks: A study based on wikipedia. *Semantic Web*, 6(4), 403-422.  
doi:10.3233/SW-140143

Landrum, B., & Garza, G. (2015). Mending fences: Defining the domains and approaches  
of quantitative and qualitative research. *Qualitative Psychology*, 2(2), 199.  
doi:10.1037/qup0000030

Largerone, C., Mougel, P. N., Rabbany, R., & Zaïane, O. R. (2015). Generating Attributed  
Networks with Communities. *PLoS One*, 10(4): e0122777.  
doi:10.1371/journal.pone.0122777

Lenharth, A., Nguyen, D., & Pingali, K. (2016). Parallel graph analytics.  
*Communications of the ACM*, 59(5), 78-87. doi:10.1145/2901919

Lewis, T. G. (2013). Cognitive stigmergy: A study of emergence in small-group social  
networks. *Cognitive Systems Research*, 21, 7-21.  
doi:10.1016/j.cogsys.2012.06.002

Li, D., Chen, J., Guo, C., Liu, Y., Zhang, J., Zhang, Z., & Zhang, Y. (2013). IP-  
geolocation mapping for moderately connected Internet regions. *IEEE  
Transactions on Parallel and Distributed Systems*, 24(2), 381-391.

doi:10.1109/TPDS.2012.136

Li, X., Zhou, W., & Liu, D. (2012). Application Source Codes Profiling for ASIP Memory Subsystem Design. *Procedia Engineering*, 29, 3160-3164.

doi:10.1016/j.proeng.2012.01.458

Liapis, A., Yannakakis, G. N., & Togelius, J. (2015). Constrained novelty search: A study on game content generation. *Evolutionary Computation*, 23(1), 101-129.

doi:10.1162/EVCO\_a\_00123

Lim, K. L., Seng, K. P., Yeong, L. S., Ang, L. M., & Ch'ng, S. I. (2015). Uninformed pathfinding: A new approach. *Expert Systems with Applications*, 42(5), 2722-2730. doi:10.1016/j.eswa.2014.10.046

doi:10.1016/j.eswa.2014.10.046

Liu, M., Egan, G. K., & Santoso, F. (2015). Modeling, autopilot design, and field tuning of a UAV with minimum control surfaces. *IEEE Transactions on Control Systems Technology*, 23(6), 2353-2360. doi:10.1109/TCST.2015.2398316

doi:10.1109/TCST.2015.2398316

Liu, T. M., & Lyons, D. M. (2015). Leveraging Area Bounds Information for Autonomous Decentralized Multi-Robot Exploration. *Robotics and Autonomous Systems*, 74, 66-78. doi:10.1016/j.robot.2015.07.002

doi:10.1016/j.robot.2015.07.002

Liyanagunawardena, T. R., Adams, A. A., & Williams, S. A. (2013). MOOCs: A systematic study of the published literature 2008-2012. *The International Review of Research in Open and Distributed Learning*, 14(3), 202-227. Retrieved from <http://www.irrodl.org>

<http://www.irrodl.org>

Lordan, O., Sallan, J. M., & Simo, P. (2014). Study of the topology and robustness of airline route networks from the complex network approach: a survey and research



- agenda. *Journal of Transport Geography*, 37, 112-120.  
doi:10.1016/j.jtrangeo.2014.04.015
- Luft, J., & Shields, M. D. (2014). Subjectivity in Developing and Validating Causal Explanations in Positivist Accounting Research. *Accounting, Organizations and Society*, 39(7), 550-558. doi:10.1016/j.aos.2013.09.001
- Lunde, Å., Heggen, K., & Strand, R. (2013). Knowledge and Power Exploring Unproductive Interplay Between Quantitative and Qualitative Researchers. *Journal of Mixed Methods Research*, 7(2), 197-210.  
doi:10.1177/1558689812471087
- Ma, J., Fukuda, D., & Schmöcker, J. D. (2013). Faster hyperpath generating algorithms for vehicle navigation. *Transportmetrica A: Transport Science*, 9(10), 925-948.  
doi:10.1080/18128602.2012.719165
- Maciejewski, W., & Puleo, G. J. (2014). Environmental evolutionary graph theory. *Journal of Theoretical Biology*, 360, 117-128. doi:10.1016/j.jtbi.2014.06.040
- Madill, A. (2015). Qualitative research is not a paradigm: Commentary on Jackson (2015) and Landrum and Garza (2015). *Qualitative Psychology*, 2, 214-220.  
doi:10.1037/qup0000032
- Maertens, A., & Barrett, C. B. (2013). Measuring social networks' effects on agricultural technology adoption. *American Journal of Agricultural Economics*, 95(2), 353-359. doi:10.1093/ajae/aas049
- Majeed, F., & Rahman, S. (2015). Graph Visualization Tools: A Comparative Analysis. *Journal of Independent Studies and Research: Computing*, 13(1), 20-26.

Retrieved from <http://jisr.szabist.edu.pk/JISR-C>

- Malliaros, F. D., & Vazirgiannis, M. (2013). Clustering and Community Detection in Directed Networks: A Survey. *Physics Reports*, 533(4), 95-142.  
doi:10.1016/j.physrep.2013.08.002
- Marozzi, M. (2016). Inter-industry financial ratio comparison with application to Japanese and Chinese firms. *Electronic Journal of Applied Statistical Analysis*, 9(1), 40-57. doi:10.1285/i20705948v9n1p40
- Marsh-Hunkin, K. E., Gochfeld, D. J., & Slattery, M. (2013). Antipredator responses to invasive lionfish, *Pterois volitans*: interspecific differences in cue utilization by two coral reef gobies. *Marine Biology*, 160(4), 1029-1040. doi:10.1007/s00227-012-2156-6
- Mayorga, A., & Gleicher, M. (2013). Splatterplots: Overcoming overdraw in scatter plots. *IEEE Transactions on Visualization and Computer Graphics*, 19(9), 1526-1538. doi:10.1109/TVCG.2013.65
- McBride, M., & Hewitt, D. (2013). The Enemy you can't see: An Investigation of the Disruption of Dark Networks. *Journal of Economic Behavior & Organization*, 93, 32-50. doi:10.1016/j.jebo.2013.07.004
- McClymont, K., Keedwell, E., & Savic, D. (2015). An analysis of the interface between evolutionary algorithm operators and problem features for water resources problems. A case study in water distribution network design. *Environmental Modelling & Software*, 69, 414-424. doi:10.1016/j.envsoft.2014.12.023
- Mears, D., & Pollard, H. B. (2016). Network science and the human brain: Using graph

theory to understand the brain and one of its hubs, the amygdala, in health and disease. *Journal of Neuroscience Research*, 94(6), 590-605.

doi:10.1002/jnr.23705

Medina, R. M. (2014). Social Network Analysis: A Case Study of the Islamist Terrorist Network. *Security Journal*, 27(1), 97-121. doi:10.1057/sj.2012.21

Merchant, G. (2012). Unravelling the social network: theory and research. *Learning, Media and Technology*, 37(1), 4-19. doi:10.1080/17439884.2011.567992

Mertler, C. A., & Reinhart, R. A. (2017). *Advanced and Multivariate Statistical Methods: Practical Application and Interpretation (6th ed.)*. New York, NY: Routledge.

Mills, B. J., Clark, J. J., Peeples, M. A., Haas, W. R., Roberts, J. M., Hill, J. B., . . . & Shackley, M. S. (2013). Transformation of Social Networks in the Late Pre-Hispanic US Southwest. *Proceedings of the National Academy of Sciences*, 110(15), 5785-5790. doi:10.1073/pnas.1219966110

Milner, G. (2016). What is GPS?. *Journal of Technology in Human Services*, 34(1), 9-12. doi:10.1080/15228835.2016.1140110

Mitzenmacher, M. (2015). Theory Without Experiments: Have We Gone Too Far? *Communications of the ACM*, 58(9), 40-42. doi:10.1145/2699413

Mora, A. M., Merelo, J. J., Castillo, P. A., & Arenas, M. G. (2013). hCHAC: A family of MOACO algorithms for the resolution of the bi-criteria military unit pathfinding problem. *Computers & Operations Research*, 40(6), 1524-1551. doi:10.1016/j.cor.2011.11.015

Muller, E., Bednar, J. A., Diesmann, M., Gewaltig, M. O., Hines, M., & Davison, A. P.

- (2015). Python in Neuroscience. *Frontiers in Neuroinformatics*, 9.  
doi:10.3389/fninf.2015.00011
- Murphy, R. R., O'Connell, J., Cox, A. J., & Schulz-Trieglaff, O. (2015). NxRepair: error correction in de novo sequence assembly using Nextera mate pairs. *PeerJ*, 3, e996. doi:10.7717/peerj.996
- Nanongkai, D. (2014). Distributed approximation algorithms for weighted shortest paths. *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*, (2014), 565-573. doi:10.1145/2591796.2591850
- NASA (n.d.). *In-situ Exploration and Sample Return: Autonomous Planetary Mobility* [Online digital image]. Retrieved January 27, 2017 from [http://mars.nasa.gov/mer/technology/is\\_autonomous\\_mobility-02.html](http://mars.nasa.gov/mer/technology/is_autonomous_mobility-02.html)
- Nash, A., & Koenig, S. (2013). Any-Angle Path Planning. *AI Magazine*, 34(4), 85-107. doi:10.1609/aimag.v34i4.2512
- Neall, A. M., & Tuckey, M. R. (2014). A methodological review of research on the antecedents and consequences of workplace harassment. *Journal of Occupational and Organizational Psychology*, 87(2), 225-257. doi:10.1111/joop.12059
- Newman, M. E., Watts, D. J., & Strogatz, S. H. (2002). Random Graph Models of Social Networks. *Proceedings of the National Academy of Sciences*, 99(Suppl. 1), 2566-2572. doi:10.1073/pnas.012582999
- Nilsen, J. K. (2007). Montepython: Implementing quantum monte carlo using python. *Computer Physics Communications*, 177(10), 799-814. doi:10.1016/j.cpc.2007.06.013

- Nocke, T., Buschmann, S., Donges, J. F., Marwan, N., Schulz, H. J., & Tominski, C. (2015). Review: visual analytics of climate networks. *Nonlinear Processes in Geophysics*, 22(5), 545. doi:10.5194/npg-22-545-2015
- Nunn, C. L., Jordán, F., McCabe, C. M., Verdolin, J. L., & Fewell, J. H. (2015). Infectious disease and group size: more than just a numbers game. *Phil. Trans. R. Soc. B*, 370(1669), 20140111. doi:10.1098/rstb.2014.0111
- Opsahl, T., Vernet, A., Alnuaimi, T., & George, G. (2017). Revisiting the Small-World Phenomenon: Efficiency Variation and Classification of Small-World Networks. *Organizational Research Methods*, 20(1), 149-173. doi:10.1177/1094428116675032
- Orchard, D., & Rice, A. (2014). A Computational Science Agenda for Programming Language Research. *Procedia Computer Science*, 29, 713-727. doi:10.1016/j.procs.2014.05.064
- Öztürk, S., & Kuzucuoğlu, A. E. (2016). A Multi-Robot Coordination Approach for Autonomous Runway Foreign Object Debris (FOD) Clearance. *Robotics and Autonomous Systems*, 75, 244-259. doi:10.1016/j.robot.2015.09.022
- Paiva, C. E., Barroso, E. M., Carnesecca, E. C., de Pádua Souza, C., dos Santos, F. T., López, R. V. M., & Paiva, S. B. R. (2014). A critical analysis of test-retest reliability in instrument validation studies of cancer patients under palliative care: a systematic review. *BMC Medical Research Methodology*, 14(1), 1. doi:10.1186/1471-2288-14-8
- Papaneophytou, C. P., & Kontopidis, G. (2014). Statistical Approaches to Maximize

- Recombinant Protein Expression in Escherichia Coli: A General Review. *Protein Expression and Purification*, 94, 22-32. doi:10.1016/j.pep.2013.10.016
- Peters, D. H. (2014). The Application of Systems Thinking in Health: Why Use Systems Thinking. *Health Research Policy and Systems*, 12, 51. doi:10.1186/1478-4505-12-51
- Pettengill, J. B., Pightling, A. W., Baugher, J. D., Rand, H., & Strain, E. (2016). Real-Time Pathogen Detection in the Era of Whole-Genome Sequencing and Big Data: Comparison of k-mer and Site-Based Methods for Inferring the Genetic Distances among Tens of Thousands of Salmonella Samples. *PLoS One*, 11(11), e0166162. doi:10.1371/journal.pone.0166162
- Phillips, J. D., Schwanghart, W., & Heckmann, T. (2015). Graph Theory in the Geosciences. *Earth-Science Reviews*, 143, 147-160. doi:10.1016/j.earscirev.2015.02.002
- Pluye, P., & Hong, Q. N. (2014). Combining the power of stories and the power of numbers: mixed methods research and mixed studies reviews. *Annual Review of Public Health*, 35(1), 29. doi:10.1146/annurev-publhealth-032013-182440
- Poisot, T. (2013). An a posteriori measure of network modularity. *F1000Research*, 2. doi:10.12688/f1000research.2-130.v3
- Puckett, B. J., Eggleston, D. B., Kerr, P. C., & Luettich, R. A. (2014). Larval dispersal and population connectivity among a network of marine reserves. *Fisheries Oceanography*, 23(4), 342-361. doi:10.1111/fog.12067
- Qasem, A. A. A., & Viswanathappa, G. (2016). Teacher perceptions towards ICT

integration: Professional development through blended learning. *Journal of Information Technology Education: Research*, 15, 561-575. Retrieved from <http://www.informingscience.org>

Quick, J., & Hall, S. (2015). Part Three: The quantitative approach. *Journal of Perioperative Practice*, 25(10), 192-196. Retrieved from <http://www.afpp.org.uk/books-journals/Journal-of-Perioperative-Practice>

Quinn, S. C., Kass, N. E., & Thomas, S. B. (2013). Building trust for engagement of minorities in human subjects research: is the glass half full, half empty, or the wrong size?. *American Journal of Public Health*, 103(12), 2119-2121. doi:10.2105/AJPH.2013.301685

Raich, M., Müller, J., & Abfalter, D. (2014). Hybrid analysis of textual data: Grounding managerial decisions on intertwined qualitative and quantitative analysis. *Management Decision*, 52(4), 737-754. doi:10.1108/MD-03-2012-0247

Raz, S., Bar-Haim, Y., Sadeh, A., & Dan, O. (2014). Reliability and validity of the online continuous performance test among young adults. *Assessment*, 21(1), 108-118. doi:10.1177/1073191112443409

Redondo, J. M., & Ortin, F. (2015). A Comprehensive Evaluation of Common Python Implementations. *IEEE Software*, 32(4), 76-84. doi:10.1109/MS.2014.104

Rezvanian, A., & Meybodi, M. R. (2015). Sampling Social Networks Using Shortest Paths. *Physica A: Statistical Mechanics and its Applications*, 424, 254-268. doi:10.1016/j.physa.2015.01.030

Riazi, A. M., & Candlin, C. N. (2014). Mixed-methods research in language teaching and

- learning: Opportunities, issues and challenges. *Language Teaching*, 47(02), 135-173. doi:10.1017/S0261444813000505
- Rodeh, O., Bacik, J., & Mason, C. (2013). BTRFS: The Linux B-tree filesystem. *ACM Transactions on Storage (TOS)*, 9(3), 9. doi:10.1145/2501620.2501623
- Rohden, M., Witthaut, D., Timme, M., & Meyer-Ortmanns, H. (2017). Curing critical links in oscillator networks as power flow models. *New Journal of Physics*, 19(1), 013002. doi:10.1088/1367-2630/aa5597
- Rooney, A. A., Cooper, G. S., Jahnke, G. D., Lam, J., Morgan, R. L., Boyles, A. L., ... & Walker, T. D. (2016). How credible are the study results? Evaluating and applying internal validity tools to literature-based assessments of environmental health hazards. *Environment International*, 92, 617-629. doi:10.1016/j.envint.2016.01.005
- Rosa, I. C., Rocha, R. J., Lopes, A., Cruz, I. C., Calado, R., Bandarra, N., ... & Rosa, R. (2016). Impact of air exposure on the photobiology and biochemical profile of an aggressive intertidal competitor, the zoanthid *Palythoa caribaeorum*. *Marine Biology*, 163(11), 222. doi:10.1007/s00227-016-3002-z
- Rossant, C., & Harris, K. D. (2013). Hardware-accelerated interactive data visualization for neuroscience in Python. *Frontiers in Neuroinformatics*, 7. doi:10.3389/fninf.2013.00036
- Rutledge, B. L., Jones, E. S., Bailey, J. H., & Stewart, J. H. (2014). Evolution of Medical Students' Understanding of Systems-Based Practice: A Qualitative Account. *The Qualitative Report*, 19(27), 1. Retrieved from <http://nsuworks.nova.edu>



- Salmela, L., & Rivals, E. (2014). LoRDEC: accurate and efficient long read error correction. *Bioinformatics*, btu538. doi:10.1093/bioinformatics/btu538
- Sayama, H., Pestov, I., Schmidt, J., Bush, B. J., Wong, C., Yamanoi, J., & Gross, T. (2013). Modeling complex systems with adaptive networks. *Computers & Mathematics with Applications*, 65(10), 1645-1664. doi:10.1016/j.camwa.2012.12.005
- Schreier, F. (2017). Computational aspects of speed-dependent Voigt profiles. *Journal of Quantitative Spectroscopy and Radiative Transfer*, 187, 44-53. doi:10.1016/j.jqsrt.2016.08.009
- Sedgewick, R., & Wayne, K. (2011). *Algorithms (4th ed.)*. Upper Saddle River, NJ: Pearson Education, Inc.
- Severance, C. (2015). Guido van Rossum: The Modern Era of Python. *Computer*, 48(3), 8-10. doi:10.1109/MC.2015.73
- Sharon, G., Stern, R., Goldenberg, M., & Felner, A. (2013). The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence*, 195, 470-495. doi:10.1016/j.artint.2012.11.006
- Shi, B., & Weninger, T. (2016). Discriminative predicate path mining for fact checking in knowledge graphs. *Knowledge-Based Systems*, 104, 123-133. doi:10.1016/j.knosys.2016.04.015
- Singh, A., & Mishra, P. K. (2014). Performance Analysis of Floyd Warshall Algorithm vs Rectangular Algorithm. *International Journal of Computer Applications*, 107(16). doi:10.5120/18837-0372

- Singh, N., Browne, L. M., & Butler, R. (2013). Parallel astronomical data processing with Python: Recipes for multicore machines. *Astronomy and Computing*, 2, 1-10. doi:10.1016/j.ascom.2013.04.002
- Slade, S., & Prinsloo, P. (2013). Learning analytics ethical issues and dilemmas. *American Behavioral Scientist*, 57(10), 1510-1529. doi:10.1177/0002764213479366
- Steininger, T., Greiner, M., Beaujean, F., & Enßlin, T. (2016). d2o: a distributed data object for parallel high-performance computing in Python. *Journal of Big Data*, 3(1), 17. doi:10.1186/s40537-016-0052-5
- Stern, R., Felner, A., van den Berg, J., Puzis, R., Shah, R., & Goldberg, K. (2014). Potential-Based Bounded-Cost Search and Anytime Non-Parametric A\*. *Artificial Intelligence*, 214, 1-25. doi:10.1016/j.artint.2014.05.002
- Stevenson, A., & Cordy, J. R. (2014). A survey of grammatical inference in software engineering. *Science of Computer Programming*, 96, 444-459. doi:10.1016/j.scico.2014.05.008
- Subarno, T., Siregar, V. P., Agus, S. B., & Sunuddin, A. (2016). Modelling Complex Terrain of Reef Geomorphological Structures in Harapan-kelapa Island, Kepulauan Seribu. *Procedia Environmental Sciences*, 33, 478-486. doi:10.1016/j.proenv.2016.03.100
- Sun, H., Ha, W., Teh, P. L., & Huang, J. (2016). A Case Study on Implementing Modularity in Software Development. *Journal of Computer Information Systems*, 1-9. doi:10.1080/08874417.2016.1183430

- Sung, Y., Kwak, J., & Park, J. H. (2015). Graph-based motor primitive generation framework. *Human-centric Computing and Information Sciences*, 5(1), 35. doi:10.1186/s13673-015-0051-0
- Tabachnick, B. G., & Fidell, L. S. (2014). *Using Multivariate Statistics: Pearson New International Edition (6th ed.)*. Essex, United Kingdom. Pearson Publishing.
- Taylor, T. E., O'Dell, C. W., Partain, P. T., Cronk, H. Q., Nelson, R. R., Rosenthal, E. J., ... & Gunson, M. R. (2016). Orbiting Carbon Observatory-2 (OCO-2) cloud screening algorithms: validation against collocated MODIS and CALIOP data. *Atmospheric Measurement Techniques*, 9(3), 973. doi:10.5194/amt-9-973-2016
- Thakur, S., & Guttman, D. S. (2016). A De-Novo Genome Analysis Pipeline (DeNoGAP) for large-scale comparative prokaryotic genomics studies. *BMC Bioinformatics*, 17(1), 260. doi:10.1186/s12859-016-1142-2
- Thomas, D. B., Luk, W., Leong, P. H., & Villasenor, J. D. (2007). Gaussian random number generators. *ACM Computing Surveys (CSUR)*, 39(4), 11. doi:10.1145/1287620.1287622
- Tonidandel, S., & LeBreton, J. M. (2013). Beyond step-down analysis: A new test for decomposing the importance of dependent variables in MANOVA. *Journal of Applied Psychology*, 98(3), 469. doi:10.1037/a0032001
- Traag, V. A., Krings, G., & Van Dooren, P. (2013). Significant scales in community structure. *Scientific Reports*, 3. doi:10.1038/srep02930
- Tsang, E. W. (2014). Case Studies and Generalization in Information Systems Research: A Critical Realist Perspective. *The Journal of Strategic Information Systems*,

- 23(2), 174-186. doi:10.1016/j.jsis.2013.09.002
- Turner, T. L., Balmer, D. F., & Coverdale, J. H. (2013). Methodologies and Study Designs Relevant to Medical Education Research. *International Review of Psychiatry, 25*(3), 301-310. doi:10.3109/09540261.2013.790310
- Vegas, S., Apa, C., & Juristo, N. (2016). Crossover Designs in Software Engineering Experiments: Benefits and Perils. *IEEE Transactions on Software Engineering, 42*(2), 120-135. doi:10.1109/TSE.2015.2467378
- Veletsianos, G., & Kimmons, R. (2016). Scholars in an increasingly open and digital world: How do education professors and students use Twitter?. *The Internet and Higher Education, 30*, 1-10. doi:10.1016/j.iheduc.2016.02.002
- Venkatesh, V., Brown, S. A., & Bala, H. (2013). Bridging the qualitative-quantitative divide: Guidelines for conducting mixed methods research in information systems. *MIS Quarterly, 37*(1), 21-54. Retrieved from <http://misq.org>
- Vesović, M., Smiljanić, A., & Kostić, D. (2016). Performance of shortest path algorithm based on parallel vertex traversal. *Serbian Journal of Electrical Engineering, 13*(1), 31-43. doi:10.2298/SJEE1601031V
- Wang, X. G. (2015). A Network Evolution Model Based on Community Structure. *Neurocomputing, 168*, 1037–1043. doi:10.1016/j.neucom.2015.05.021
- Wang, Z., Zlatanova, S., Moreno, A., Van Oosterom, P., & Toro, C. (2014). A data model for route planning in the case of forest fires. *Computers & Geosciences, 68*, 1-10. doi:10.1016/j.cageo.2014.03.013
- Warne, R. T. (2014). A Primer on Multivariate Analysis of Variance (MANOVA) for

- Behavioral Scientists. *Practical Assessment, Research & Evaluation*, 19(17), 1-10. Retrieved from <http://pareonline.net>
- Wasserman, S., & Faust, K. (1994). *Social Network Analysis: Methods and Applications* (Vol. 8). New York, NY: Cambridge University Press.
- Watts, D. J., & Strogatz, S. H. (1998). Collective Dynamics of ‘Small-World’ Networks. *Nature*, 393(6684), 440-442. doi:10.1038/30918
- Wen, L., Çatay, B., & Eglese, R. (2014). Finding a minimum cost path between a pair of nodes in a time-varying road network with a congestion charge. *European Journal of Operational Research*, 236(3), 915-923. doi:10.1016/j.ejor.2013.10.044
- White, A. V., & Perrone-McGovern, K. (2017). Influence of Generational Status and Financial Stress on Academic and Career Self-Efficacy. *Journal of Employment Counseling*, 54(1), 38-46. doi:10.1002/joec.12049
- Wildiers, H., Mauer, M., Pallis, A., Hurria, A., Mohile, S. G., Luciani, A., ... & Cohen, H. J. (2013). End points and trial design in geriatric oncology research: a joint European organisation for research and treatment of cancer–Alliance for Clinical Trials in Oncology–International Society of Geriatric Oncology position article. *Journal of Clinical Oncology*, 31(29), 3711-3718. doi:10.1200/JCO.2013.49.6125
- Williamson, T., & Olsson, R. A. (2014). PySy: A Python Package for Enhanced Concurrent Programming. *Concurrency and Computation: Practice and Experience*, 26(2), 309-335. doi:10.1002/cpe.2981
- Wohlin, C., & Aurum, A. (2015). Towards a decision-making structure for selecting a

- research design in empirical software engineering. *Empirical Software Engineering*, 20(6), 1427-1455. doi:10.1007/s10664-014-9319-7
- Xu, J., & Chen, H. (2008). The Topology of Dark Networks. *Communications of the ACM*, 51(10), 58-65. doi:10.1145/1400181.1400198
- Xu, Y., Liu, P., Li, X., & Ren, W. (2014). Discovering the Influences of Complex Network Effects on Recovering Large Scale Multiagent Systems. *The Scientific World Journal*, 2014. doi:10.1155/2014/407639
- Yamamoto, M., Ono, M., Nakashima, K., & Hirai, A. (2016). Unified performance profiling of an entire virtualized environment. *International Journal of Networking and Computing*, 6(1), 124-147. Retrieved from <http://www.ijnc.org>
- Yang, L., Qi, J., Song, D., Xiao, J., Han, J., & Xia, Y. (2016). Survey of Robot 3D Path Planning Algorithms. *Journal of Control Science and Engineering*, 2016. doi:10.1155/2016/7426913
- Yang, Y., Poon, J. P., Liu, Y., & Bagchi-Sen, S. (2015). Small and Flat Worlds: A Complex Network Analysis of International Trade in Crude Oil. *Energy*, 93, 534-543. doi:10.1016/j.energy.2015.09.079
- Yang, Z., Algesheimer, R., & Tessone, C. J. (2016). A Comparative Analysis of Community Detection Algorithms on Artificial Networks. *Scientific Reports*, 6. doi:10.1038/srep30750
- Yoon, S., Yoon, S. E., Lee, U., & Shim, D. H. (2015). Recursive Path Planning Using Reduced States for Car-Like Vehicles on Grid Maps. *IEEE Transactions on Intelligent Transportation Systems*, 16(5), 2797-2813.

doi:10.1109/TITS.2015.2422991

- Zaglia, M. E. (2013). Brand communities embedded in social networks. *Journal of Business Research*, 66(2), 216-223. doi:10.1016/j.jbusres.2012.07.015
- Zhang, A., Li, C., & Bi, W. (2016). Rectangle expansion A\* pathfinding for grid maps. *Chinese Journal of Aeronautics*, 29(5), 1385-1396. doi:10.1016/j.cja.2016.04.023
- Zhang, C., Anzalone, N. C., Faria, R. P., & Pearce, J. M. (2013). Open-source 3D-printable optics equipment. *PloS One*, 8(3), e59840.  
doi:10.1371/journal.pone.0059840
- Zhang, M., Su, C., Liu, Y., Hu, M., & Zhu, Y. (2016). Unmanned Aerial Vehicle Route Planning in the Presence of a Threat Environment Based on a Virtual Globe Platform. *ISPRS International Journal of Geo-Information*, 5(10), 184.  
doi:10.3390/ijgi5100184
- Zhang, X., Chan, F. T., Yang, H., & Deng, Y. (2017). An adaptive amoeba algorithm for shortest path tree computation in dynamic graphs. *Information Sciences*, 405, 123-140. doi:10.1016/j.ins.2017.04.021
- Zhang, Z., & Wang, K. (2013). A trust model for multimedia social networks. *Social Network Analysis and Mining*, 3(4), 969-979. doi:10.1007/s13278-012-0078-4
- Zhu, L., & Chiu, Y. C. (2015). Transportation Routing Map Abstraction Approach: Algorithm and Numerical Analysis. *Transportation Research Record: Journal of the Transportation Research Board*, (2528), 78-85. doi:10.3141/2528-09.
- Zohrabi, M. (2013). Mixed method research: Instruments, validity, reliability and reporting findings. *Theory and Practice in Language Studies*, 3(2), 254. Retrieved

from <http://www.academypublication.com/issues>

Zou, Z., Wu, J., Gao, J., & Xu, X. (2014). Cascade defense in urban road network by inserting modular topologies. *Kybernetes*, 43(5), 750-763. doi:10.1108/K-11-2013-0250

Zou, Z., Xiao, Y., & Gao, J. (2013). Robustness analysis of urban transit network based on complex networks theory. *Kybernetes*, 42(3), 383-399.  
doi:10.1108/03684921311323644



## Appendix A: Graph-Tool A\* Algorithm Instrument

*Instrument Background:* Graph-Tool is an open source graph analysis framework, available as a Python module, and freely available from the following website:

<https://graph-tool.skewed.de>

Graph-Tool supports several pathfinding algorithms through its extensive application programmer interface (API). Official documentation of the complete Graph-Tool API can be found at the following website:

<https://graph-tool.skewed.de/static/doc/index.html>

Graph-Tool's A\* pathfinding algorithm is supported in a Python function named *astar\_search* and is fully described in the Graph-Tool online documentation here:

[https://graph-tool.skewed.de/static/doc/search\\_module.html?highlight=astar#graph\\_tool.search.astar\\_search](https://graph-tool.skewed.de/static/doc/search_module.html?highlight=astar#graph_tool.search.astar_search)

*Versioning:* The Graph-Tool version used in this study: 2.18

*Instructions:* The A\* (pronounced "A star") algorithm is supported by a Graph-Tool Python function named: *astar\_search*

Software engineers writing Python source code to utilize Graph-Tool's A\* pathfinding algorithm, may call the aforementioned function using Python. This is the primary method my quantitative study utilizes Graph-Tool's A\* algorithm.

In summary, these are the steps to use the Graph-Tool A\* API function:

1. Load the 2D terrain map file
2. Assign the start and destination nodes.

3. Call the A\* function: *astar\_search*
4. Two result parameters are returned, one of which is the list of predecessors from the destination node, back to the start node. This contains the shortest path.
5. Iterate through the list of predecessor nodes until the complete path is generated.
6. Count the number of nodes in that list to obtain the final path length.

An example how to use the API is depicted next in Figure A1.

```

import graph_tool.all as gt
g = gt.Graph()
g = gt.load_graph( input_path_file )
dist, pred = gt.astar_search(g, startNode, weight=weights) #Call A* function
i = 0
path = [] #build the list of node predecessors.
for p in pred:
    path.append( [i, int(p)] )
    i += 1

astarPath = [] #create an empty path.
astarPath.append( int(destNode) ) # append the dest node, then find predecessor
currNode = destNode #set current node to destination node
while currNode != startNode: #start with destination node...
    astarPath.append( path [int(currNode)][1] ) #append the predecessor node
    currNode = path [int(currNode)][1] #update the current node
astarPath.reverse() # now reverse the list, so it displays in correct order
print("A-Star Path = %s" % astarPath) # the final shortest path
print("A-Star Path Length = %d" % (len(astarPath) -1) ) # the final path length

```

Figure 48. Abbreviated Graph-Tool A\* (a-star) algorithm API demonstration.

*Results Interpretation:* As shown in the Python source code snippet above, the variable 'astarPath', used in the penultimate line, contains a string with the full path from

start node, to destination node. The length of this node list contains the path length, as shown above in the last line of Python code.

*Statistical Validity and Reliability:* The validity and reliability of the algorithm instrument were successfully verified in a pilot test, as described theoretically in Section 2, and statistically in Section 3, using the Wilcoxon Signed Ranks statistic in a repeated-measures design with no intervention. See Section 3 for the statistical details.

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned graph analysis framework and pathfinding API, is freely available for review at this URL: <https://github.com/professor-moran/graph-theory>

## Appendix B: Graph-Tool Bellman-Ford Algorithm Instrument

*Instrument Background:* Graph-Tool is an open source graph analysis framework, available as a Python module, and freely available from the following website:

<https://graph-tool.skewed.de>

Graph-Tool supports several pathfinding algorithms through its extensive application programmer interface (API). Official documentation of the complete Graph-Tool API can be found at the following website:

<https://graph-tool.skewed.de/static/doc/index.html>

Graph-Tool's Bellman-Ford pathfinding algorithm is supported in a Python function named *shortest\_path* and is fully described in the Graph-Tool online documentation here: [https://graph-tool.skewed.de/static/doc/topology.html?highlight=shortest\\_path#graph\\_tool.topology.shortest\\_path](https://graph-tool.skewed.de/static/doc/topology.html?highlight=shortest_path#graph_tool.topology.shortest_path)

*Versioning:* The Graph-Tool version used in this study: 2.18

*Instructions:* The Bellman-Ford algorithm is supported by a Graph-Tool Python function named: *shortest\_path* but requires that the `negative_weights` parameter is set to Boolean True, in order for Bellman-Ford to function to be activated, as discussed in the aforementioned documentation URL.

Software engineers writing Python source code to utilize Graph-Tool's Bellman-Ford pathfinding algorithm, may call the aforementioned function using Python. This is the primary method my quantitative study utilizes Graph-Tool's Bellman-Ford algorithm.

In summary, these are the steps to use the Graph-Tool Bellman-Ford API function:

1. Load the 2D terrain map file
2. Assign the start and destination nodes.
3. Call the Bellman-Ford function: *shortest\_path*, and with the *negative\_weights* parameter is set to Boolean True.
4. Two result parameters are returned, one of which is the list of vertices from the destination node, back to the start node. This contains the shortest path.
5. Iterate through the list of nodes until the complete path is generated.
6. Count the number of nodes in that list to obtain the final path length.

An example how to use the API is depicted next in Figure B1.

```
import graph_tool.all as gt
g = gt.Graph()
g = gt.load_graph( input_path_file )
vertlist, edgelist = gt.shortest_path(g, startNode, destNode, negative_weights=True)
bellmanFordPath = [] #create an empty path.
for v in vertlist:
    bellmanFordPath.append( int(v) )
print (bellmanFordPath)    #this is the complete Bellman-Ford shortest path.
print (len(bellmanFordPath)-1)    #this is the shortest path length
```

Figure 49. Abbreviated Graph-Tool Bellman-Ford algorithm API demonstration.

*Results Interpretation:* As shown in the Python source code snippet above, the variable 'bellmanFordPath', used in the penultimate line, contains a string with the full

path from start node, to destination node. The length of this node list contains the path length, as shown above in the last line of Python code.

*Statistical Validity and Reliability:* The validity and reliability of the algorithm instrument were successfully verified in a pilot test, as described theoretically in Section 2, and statistically in Section 3, using the Wilcoxon Signed Ranks statistic in a repeated-measures design with no intervention. See Section 3 for the statistical details.

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned graph analysis framework and pathfinding API, is freely available for review at this URL: <https://github.com/professor-moran/graph-theory>

## Appendix C: Graph-Tool Dijkstra Algorithm Instrument

*Instrument Background:* Graph-Tool is an open source graph analysis framework, available as a Python module, and freely available from the following website:

<https://graph-tool.skewed.de>

Graph-Tool supports several pathfinding algorithms through its extensive application programmer interface (API). Official documentation of the complete Graph-Tool API can be found at the following website:

<https://graph-tool.skewed.de/static/doc/index.html>

Graph-Tool's Dijkstra pathfinding algorithm is supported in a Python function named *dijkstra\_search* and is fully described in the Graph-Tool online documentation here: [https://graph-tool.skewed.de/static/doc/search\\_module.html?highlight=dijkstra\\_search#graph\\_tool.search.dijkstra\\_search](https://graph-tool.skewed.de/static/doc/search_module.html?highlight=dijkstra_search#graph_tool.search.dijkstra_search)

*Versioning:* The Graph-Tool version used in this study: 2.18

*Instructions:* The Dijkstra algorithm is supported by a Graph-Tool Python function named: *dijkstra\_search*

Software engineers writing Python source code to utilize Graph-Tool's Dijkstra pathfinding algorithm, may call the aforementioned function using Python. This is the primary method my quantitative study utilizes Graph-Tool's Dijkstra algorithm.

In summary, these are the steps to use the Graph-Tool Dijkstra API function:

1. Load the 2D terrain map file
2. Assign the start and destination nodes.
3. Call the Dijkstra function: *dijkstra\_search*

4. Two result parameters are returned, one of which is the list of predecessors from the destination node, back to the start node. This contains the shortest path.
5. Iterate through the list of predecessor nodes until the complete path is generated.
6. Count the number of nodes in that list to obtain the final path length.

An example how to use the API is depicted next in Figure C1.

```

import graph_tool.all as gt
g = gt.Graph()
g = gt.load_graph( input_path_file )
dist, pred = gt.dijkstra_search (g, startNode, weight=weights) #Call Dijkstra
i = 0
path = [] #build the list of node predecessors.
for p in pred:
    path.append( [i, int(p)] )
    i += 1
dijkPath = [] #create an empty path.
dijkPath.append( int(destNode) ) # append the dest node, then find predecessor
currNode = destNode #set current node to destination node
while currNode != startNode: #start with destination node...
    dijkPath.append( path [int(currNode)][1] ) #append the predecessor node
    currNode = path [int(currNode)][1] #update the current node
dijkPath.reverse() # now reverse the list, so it displays in correct order
print("Dijkstra Path = %s" % dijkPath) # the final shortest path
print("Dijkstra Path Length = %d" % (len(dijkPath) -1) ) # the final path length

```

Figure 50. Abbreviated Graph-Tool Dijkstra algorithm API demonstration.

*Results Interpretation:* As shown in the Python source code snippet above, the variable 'dijkPath', used in the penultimate line, contains a string with the full path from



start node, to destination node. The length of this node list contains the path length, as shown above in the last line of Python code.

*Statistical Validity and Reliability:* The validity and reliability of the algorithm instrument were successfully verified in a pilot test, as described theoretically in Section 2, and statistically in Section 3, using the Wilcoxon Signed Ranks statistic in a repeated-measures design with no intervention. See Section 3 for the statistical details.

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned graph analysis framework and pathfinding API, is freely available for review at this URL: <https://github.com/professor-moran/graph-theory>

## Appendix D: Network-X A\* Algorithm Instrument

*Instrument Background:* Network-X is an open source graph analysis framework, available as a Python module, and freely available from the following website:

<http://Network-X.readthedocs.io/en/stable/index.html>

Network-X supports several pathfinding algorithms through its extensive application programmer interface (API). Official documentation of the complete Network-X API can be found here: [http://Network-](http://Network-X.readthedocs.io/en/stable/reference/index.html)

[X.readthedocs.io/en/stable/reference/index.html](http://Network-X.readthedocs.io/en/stable/reference/index.html)

Network-X's A\* (pronounced "A star") pathfinding algorithm is supported in a Python function named *astar\_path* and is fully described in the Network-X online documentation here: [http://Network-X.readthedocs.io/en/stable/reference/generated/networkx.algorithms.shortest\\_paths.astar.astar\\_path.html?highlight=astar\\_path](http://Network-X.readthedocs.io/en/stable/reference/generated/networkx.algorithms.shortest_paths.astar.astar_path.html?highlight=astar_path)

*Versioning:* The Network-X version used in this study: 1.11

*Instructions:* The A\* algorithm is supported by a Network-X function named: *astar\_path*

Software engineers writing Python source code to utilize Network-X's A\* pathfinding algorithm, may call the aforementioned function using Python. This is the primary method my quantitative study utilizes Network-X's A\* algorithm.

In summary, these are the steps to use the Network-X A\* API function:

1. Load the 2D terrain map file
2. Assign the start and destination nodes.

3. Call the A\* function: *astar\_path*
4. A list of nodes, from the start to destination node, is returned. This contains the shortest path.
5. Print the node list to display the path from start to destination.
6. Count the number of nodes in that list to obtain the final path length.

An example how to use the API is depicted next in Figure D1.

```
import networkx as nx
G = nx.Graph( input_data.values )
nodeList = G.nodes()
aStarPath = nx.astar_path(G, startNode, destNode ) # call A*
aStarPathLength = len(aStarPath) # get A* shortest path length
print ("A* path = %s" % aStarPath) # print the shortest path
print ("A* path length = %d" % aStarPathLength) # print the path length
```

Figure 51. Abbreviated Network-X A\* (a-star) algorithm API demonstration.

*Results Interpretation:* As shown in the Python source code snippet above, the variable 'aStarPath', used in the penultimate line, contains a string with the full path from start node, to destination node. The length of this node list contains the path length, as shown above in the last line of Python code.

*Statistical Validity and Reliability:* The validity and reliability of the algorithm instrument were successfully verified in a pilot test, as described theoretically in Section 2, and statistically in Section 3, using the Wilcoxon Signed Ranks statistic in a repeated-measures design with no intervention. See Section 3 for the statistical details.

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned graph analysis framework and pathfinding API, is freely available for review at this URL: <https://github.com/professor-moran/graph-theory>

## Appendix E: Network-X Bellman-Ford Algorithm Instrument

*Instrument Background:* Network-X is an open source graph analysis framework, available as a Python module, and freely available from the following website:

<http://networkx.readthedocs.io/en/stable/index.html>

Network-X supports several pathfinding algorithms through its extensive application programmer interface (API). Official documentation of the complete Network-X API can be found here:

<http://networkx.readthedocs.io/en/stable/reference/index.html>

Network-X's Bellman-Ford pathfinding algorithm is supported in a Python function named *bellman\_ford* and is fully described in the Network-X online documentation here:

[http://networkx.readthedocs.io/en/stable/reference/generated/networkx.algorithms.shortest\\_paths.weighted.bellman\\_ford.html?highlight=bellman\\_ford](http://networkx.readthedocs.io/en/stable/reference/generated/networkx.algorithms.shortest_paths.weighted.bellman_ford.html?highlight=bellman_ford)

*Versioning:* The Network-X version used in this study: 1.11

*Instructions:* The Bellman-Ford algorithm is supported by a Network-X function named: *bellman\_ford*

Software engineers writing Python source code to utilize Network-X's Bellman-Ford pathfinding algorithm, may call the aforementioned function using Python. This is the primary method my quantitative study utilizes Network-X's Bellman-Ford algorithm.

In summary, these are the steps to use the Network-X Bellman-Ford API function:

1. Load the 2D terrain map file

2. Assign the start and destination nodes.
3. Call the Bellman-Ford function: *bellman\_ford*
4. Two result parameters are returned, one of which is the list of predecessors from the destination node, back to the start node. This contains the shortest path.
5. Iterate through the list of predecessor nodes until the complete path is generated.
6. Count the number of nodes in that list to obtain the final path length.

An example how to use the API is depicted next in Figure E1.

```
import networkx as nx
G = nx.Graph( input_data.values )
nodeList = G.nodes()
pred, dist = nx.bellman_ford(G, startNode ) # call Bellman-Ford
path = dict(pred) # convert Predecessor list to Key-Value dictionary
while currNode != startNode: # start with destination node
    bfPath.append( path[currNode] ) # append its predecessor node
    currNode = path[currNode] #update current node, keep looping back to source node
bfPath.reverse() # now reverse the list, so path displays in correct order
bfPathLengthsAll = dict(dist)
bfPathLength = bfPathLengthsAll[destNode] # get Bellman-Ford shortest pathlength
print ("Bellman-Ford path = %s" % bfPath) # print the shortest path
print ("Bellman-Ford path length = %d" % bfPathLength) # print the path length
```

Figure 52. Abbreviated Network-X Bellman-Ford algorithm API demonstration.

*Results Interpretation:* As shown in the Python source code snippet above, the variable 'bfPath', used in the penultimate line, contains a string with the full path from start node, to destination node. The length of this node list contains the path length, as shown above in the last line of Python code.

*Statistical Validity and Reliability:* The validity and reliability of the algorithm instrument were successfully verified in a pilot test, as described theoretically in Section 2, and statistically in Section 3, using the Wilcoxon Signed Ranks statistic in a repeated-measures design with no intervention. See Section 3 for the statistical details.

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned graph analysis framework and pathfinding API, is freely available for review at this URL: <https://github.com/professor-moran/graph-theory>

## Appendix F: Network-X Dijkstra Algorithm Instrument

*Instrument Background:* Network-X is an open source graph analysis framework, available as a Python module, and freely available from the following website:

<http://networkx.readthedocs.io/en/stable/index.html>

Network-X supports several pathfinding algorithms through its extensive application programmer interface (API). Official documentation of the complete Network-X API can be found here:

<http://networkx.readthedocs.io/en/stable/reference/index.html>

Network-X's Dijkstra pathfinding algorithm is supported in a Python function named *dijkstra\_path* and is fully described in the Network-X online documentation here:

[http://networkx.readthedocs.io/en/stable/reference/generated/networkx.algorithms.shortest\\_paths.weighted.dijkstra\\_path.html?highlight=dijkstra\\_path](http://networkx.readthedocs.io/en/stable/reference/generated/networkx.algorithms.shortest_paths.weighted.dijkstra_path.html?highlight=dijkstra_path)

*Versioning:* The Network-X version used in this study: 1.11

*Instructions:* The Dijkstra algorithm is supported by a Network-X function named: *dijkstra\_path*

Software engineers writing Python source code to utilize Network-X's Dijkstra pathfinding algorithm, may call the aforementioned function using Python. This is the primary method my quantitative study utilizes Network-X's Dijkstra algorithm.

In summary, these are the steps to use the Network-X Dijkstra API function:

1. Load the 2D terrain map file
2. Assign the start and destination nodes.
3. Call the Dijkstra function: *dijkstra\_path*

4. A list of nodes, from the start to destination node, is returned. This contains the shortest path.
5. Print the node list to display the path from start to destination.
6. Count the number of nodes in that list to obtain the final path length.

An example how to use the API is depicted next in Figure F1.

```
import networkx as nx
G = nx.Graph( input_data.values )
nodelist = G.nodes()
dijkstraPath = nx.dijkstra_path(G, startNode, destNode ) # Call Dijkstra
dijkstraPathLength = len(dijkstraPath) # get Dijkstra shortest path length
print ("Dijkstra path = %s" % dijkstraPath) # print the shortest path
print ("Dijkstra path length = %d" % dijkstraPathLength) # print the path length
```

Figure 53. Abbreviated Network-X Dijkstra algorithm API demonstration.

*Results Interpretation:* As shown in the Python source code snippet above, the variable 'dijkstraPath', used in the penultimate line, contains a string with the full path from start node, to destination node. The length of this node list contains the path length, as shown above in the last line of Python code.

*Statistical Validity and Reliability:* The validity and reliability of the algorithm instrument were successfully verified in a pilot test, as described theoretically in Section 2, and statistically in Section 3, using the Wilcoxon Signed Ranks statistic in a repeated-measures design with no intervention. See Section 3 for the statistical details.

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned graph analysis framework and pathfinding API, is freely available for review at this URL: <https://github.com/professor-moran/graph-theory>



## Appendix G: Python TimeIt Instrument

*Instrument Background:* Like other mature computer programming languages, Python has many built-in utility functions, and supports an extensive application programmer interface (API).

To measure execution time of small code snippets, Python has the *timeit* module. Official documentation of the *timeit* module can be found here:

<https://docs.python.org/2/library/timeit.html>

*Versioning:* The Python (and *timeit*) version used in this study: 2.7.11

*Instructions:* Software engineers writing Python source code to utilize Python's built-in *timeit* function can do so with either the command-line interface (CLI), or the callable interface. This study uses the *timeit* callable interface.

In summary, these are the steps to use the Python's *timeit* function:

1. Create a variable to hold the start time, using *timeit*.
2. Call the Python function whose elapsed time is to be measured.
3. Create a variable to hold the end time, using *timeit*.
4. Subtract the end time from the start time to calculate the elapsed time.
5. Repeat, if or as needed.

An example how to use the API is depicted next in Figure G1.

```

import timeit
start_time = timeit.default_timer() #get the start time
#Call the A* search function:
p = multiprocessing.Process(target=runAstar, args=(state,))
end_time = timeit.default_timer() #get the end time
elapsed_time = end_time - start_time
print ("Elapsed time = %f seconds" % elapsed_time)

```

Figure 54. Abbreviated python TimeIt API demonstration

*Results Interpretation:* As shown in the Python source code above, the variable 'elapsed\_time', used in the last line, contains the number of seconds elapsed between the start time and ending time.

*Statistical Validity and Reliability:* The validity and reliability of the *timeit* module were discussed earlier in Section 2, with corroborated scholarly support listed in Table 11. For ease of reference, the scholarly articles supporting the validity and reliability of the *timeit* Python module, are as follows: (a) Akeret, Gamper, Amara, and Refregier (2015); (b) Gorelick and Ozsvald (2014); (c) Pettengill et al. (2016); (d) Schreier (2017); and (e) Steininger, Greiner, Beaujean, and Enßlin (2016).

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned API, is freely available for review at this URL:

<https://github.com/professor-moran/graph-theory>

## Appendix H: Python Memory\_Profiler Instrument

*Instrument Background:* Like other mature computer programming languages, Python has many built-in utility functions, and supports an extensive application programmer interface (API). Additionally, the Python Software Foundation has many other libraries, modules and source code freely available for download at the Python Package Index (PyPI) site: <https://pypi.python.org/pypi>

To measure execution time of small code snippets, Python has the *memory\_profiler* module, provided and supported by PyPI. Official documentation of the *memory\_profiler* module can be found here:

[https://pypi.python.org/pypi/memory\\_profiler](https://pypi.python.org/pypi/memory_profiler)

*Versioning:* The *memory\_profiler* version used in this study: 0.41

*Instructions:* Software engineers writing Python source code may utilize Python's *memory\_profiler* functionality to monitor the amount of memory used in Python programs, on a line-by-line basis.

However, users of *memory\_profiler* should be informed in advance that, as discussed in Gorelick and Ozsvald (2014), *memory\_profiler* results may vary between experimental test runs, due to (a) the nondeterministic way memory is handled between the Python memory manager and the operating system memory manager; and (b) Python garbage collection is not instantaneous, so recently deleted memory objects may be unavailable to the programmer, yet still take up memory because they are not yet garbage collected, thereby affecting the *memory\_profiler* results (p. 43).

This suggests that, depending on the target operating system and environment, repeat runs of *memory\_profiler* may be recommended to establish baseline statistics which may help to identify data outliers (to be transformed, or removed, prior to subsequent statistical analyses), in the event recently deleted memory objects are still cached in memory between sequential uses of *memory\_profiler* (p. 289).

In summary, these are the steps to use the Python's *memory\_profiler* function:

1. Decorate the Python function to be profiled, with the "@profile" special script.
2. [OPTIONAL] In the decorator, indicated the level of precision desired. E.g., "@profile(precision = 4)" provides accuracy up to the ten-thousandths place.
3. Call the Python function whose memory use is to be monitored, from a Python program (script).
4. Output from *memory\_profiler* displays (i.e., prints) to the console window. Redirect this output to a text file for later parsing, so as to extract the memory consumption results for the function that was decorated with the "@profile" decorator (as described above in step 1).

An example how to use the API is depicted next in Figure H1.

```

from memory_profiler import profile
import networkx as nx

@profile(precision=4)
def runAstar(state):
    #this is the function being profiled...
    G = nx.Graph( input_data.values )
    nodeList = G.nodes()
    aStarPath = nx.astar_path(G, startNode, destNode )
    aStarPathLength = len(aStarPath)
    print ("Path length = %d" % (aStarPathLength - 1) )
    print ("Path = %s" % aStarPath)
    return aStarPath, aStarPathLength

def main():
    #Call the function to be memory profiled
    runAstar(state) #memory_profiler output will automatically write to console.
    #redirect console output to text file in order to parse it later,
    #and therefore extract the memory consumption results.

IN THE CONSOLE (or redirected text file output) RESULTS MAY APPEAR SIMILAR TO THIS:

Filename: networkx-astar-pathfinding.py

Line #   Mem usage   Increment   Line Contents
=====
275  20.2344 MiB   0.0000 MB   @profile(precision=4)
276                                     def runAstar(state):
277
283  22.7500 MiB   0.6406 MB   G = nx.Graph( input_data.values )
286  22.7578 MiB   0.0078 MB   nodeList = G.nodes()
299  22.8203 MiB   0.2586 MB   aStarPath = nx.astar_path(G) # call A* algorithm
301  22.8203 MiB   0.0000 MB   aStarPathLength = len(aStarPath)
306  22.8203 MiB   0.0000 MB   print ("Path length=%d" % (aStarPathLength-1))
307  22.8203 MiB   0.0000 MB   print ("Path = %s" % aStarPath)
308  23.3828 MiB   0.5625 MB   return aStarPath, aStarPathLength

```

Figure 55. Abbreviated python memory\_profiler API demonstration.

*Results Interpretation:* As shown in the Python source code above, and in the subsequent console output, the *increment* value next to the line that does the pathfinding (in this abbreviated example, "aStarPath = nx.astar\_path(G)") contains the memory consumed by that line of code, in this case, 0.2586 MB, as determined by *memory\_profiler*. This value is saved and analyzed later during my statistical analyses.

*Statistical Validity and Reliability:* The validity and reliability of the *memory\_profiler* module were discussed earlier in Section 2, with corroborated scholarly support listed in Table 11. For ease of reference, the scholarly articles supporting the validity and reliability of the *memory\_profiler* Python module, are as follows: (a) Dunn and Weissman (2016); (b) Gorelick and Ozsvald (2014); (c) Li, Zhou, and Liu (2012); (d) Rossant and Harris (2013); and (e) Murphy, O’Connell, Cox, and Schulz-Trieglaff (2015).

*Doctoral study source code location:* A full, working version of my source code, which uses the aforementioned API, is freely available for review at this URL:  
<https://github.com/professor-moran/graph-theory>.