2016

Algorithmic Foundations of Heuristic Search using Higher-Order Polygon Inequalities

Newton Henry Campbell Jr.
Nova Southeastern University, nc607@nova.edu

NSUWorks Citation
Newton Henry Campbell Jr.. 2016. *Algorithmic Foundations of Heuristic Search using Higher-Order Polygon Inequalities.* Doctoral dissertation. Nova Southeastern University. Retrieved from NSUWorks, College of Engineering and Computing. (374) https://nsuworks.nova.edu/gscis_etd/374.

Algorithmic Foundations of Heuristic Search using Higher-Order Polygon Inequalities

by

Newton H.Campbell Jr.

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in
Computer Science

College of Engineering and Computing
Nova Southeastern University

2016

Approval Signature Page

We hereby certify that this dissertation, submitted by Newton Campbell, Jr., conforms to acceptable standards and is fully adequate in scope and quality to fulfill the dissertation requirements for the degree of Doctor of Philosophy.

Michael J. Laszlo, Ph.D. Date
Chairperson of Dissertation Committee

Sumitra Mukherjee, Ph.D. Date
Dissertation Committee Member

Francisco J. Mitropoulos, Ph.D. Date
Dissertation Committee Member

Approved:

Amon B. Seagull, Ph.D. Date
Interim Dean, College of Engineering and Computing

College of Engineering and Computing
Nova Southeastern University

2016

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Algorithmic Foundations of Heuristic Search using Higher-Order Polygon Inequalities

by
Newton H. Campbell Jr.
November 2015

The shortest path problem in graphs is both a classic combinatorial optimization problem and a practical problem that admits many applications. Techniques for preprocessing a graph are useful for reducing shortest path query times. This dissertation studies the foundations of a class of algorithms that use preprocessed landmark information and the triangle inequality to guide A* search in graphs. A new heuristic is presented for solving shortest path queries that enables the use of higher order polygon inequalities. We demonstrate this capability by leveraging distance information from two landmarks when visiting a vertex as opposed to the common single landmark paradigm. The new heuristic's novel feature is that it computes and stores a reduced amount of preprocessed information (in comparison to previous landmark-based algorithms) while enabling more informed search decisions. We demonstrate that domination of this heuristic over its predecessor depends on landmark selection and that, in general, the denser the landmark set, the better heuristic performs. Due to the reduced memory requirement, this new heuristic admits much denser landmark sets.

We conduct experiments to characterize the impact that landmark configurations have on this new heuristic, demonstrating that centrality-based landmark selection has the best tradeoff between preprocessing and runtime. Using a developed graph library and static information from benchmark road map datasets, the algorithm is compared experimentally with previous landmark-based shortest path techniques in a fixed-memory environment to demonstrate a reduction in overall computational time and memory requirements. Experimental results are evaluated to detail the significance of landmark selection and density, the tradeoffs of performing preprocessing, and the practical use cases of the algorithm.

Acknowledgements

While I am finishing this final Computer Science degree at Nova Southeastern University in DC and Florida, my path started in Cleveland, OH. I thank my family and friends in Cleveland, who supported and nurtured my ambitions to get into this field. Knowing that I have my sisters, brother, parents, cousins, and everyone else in my huge family there when I need them has been a great comfort throughout the dissertation process. To my little sister Nicole, I'm ready to watch you be the next one to be called "doctor". Just keep pushing yourself. You'll get there. And to my grand-aunt, Phyllis Robinson, I wish you could be around to watch me walk across the stage one more time. Thank you for everything you've done. I'll carry the legacy that started with you into the next phase of my life and onward.

When I moved to SUNY Buffalo, in my mind, I was starting my own crusade. I started alone. I had a plan. But life had a different one. I ended up where I am at the time of this writing because of individuals who went out of their way to support me at every step along this journey. As my first research professor, Shambhu Upadhyaya, I thank you. Not only for introducing me to the world of research, but also for providing me with guidance and support both while at Buffalo and afterward. You were the first to ever tell me I should get a PhD. While I was too naïve to listen to you then, I eventually got around to it. I would also like to acknowledge other professors who were there for me after Buffalo, including Florian Buchholz, Brett Tjaden and Xunhua Wang. The time spent with you and lessons learned gave me the confidence and experience that I needed to pursue a PhD. Finally, thanks to my advisor, Michael Laszlo, my committee members and all the other professors and classmates I've interacted with at Nova Southeastern.

I've been a dreamer my entire life, imagining so many ways of helping the world but knowing, at some level, that many things were impossible. But with the support and nurture from the all the staff at BBN, I learned that I could build the impossible. This would have never happened without the support and valuable lessons from many of the seniors at BBN, including Jack Marin, Joshua Edmison, Richard Burne, Carl Powell, as well as all of the staff that I've worked with over the years. You are all just as responsible for shaping me into a contributor to science as all my experiences in University. Dr. John Everett (currently at DARPA) deserves a great deal of thanks for initially supporting the idea of being on my committee, nominating me and serving as a true mentor for the 2015 DARPA Rising session, and being the program manager for the first of many research efforts that I hope to have my name on. I now stand on the shoulders of these giants.

And to John-Francis Mergen, my mentor and one of my closest friends, I thank you and Lynne for always being there as family and for believing in me. While knowing you, I've seen you make the case for change in the world as only you can make it. In that time, you've taught me that I can do the same, and you can bet that I will. My lifelong friends deserve just as much acknowledgement for their support as everyone else mentioned here. So to Vera Neroni, Elizabeth Brown, Michelle Chu, Cyrus Chu, Douglas Campese, Jessica Lombardo, Scott Goldweber, and everyone else in the surrogate family that has joined me on my crusade, thank you for getting me to this point. Everyone on this page has contributed to the person I am today with a trust that I will strive to create a better tomorrow. I will not fail that trust. And I will not fail to try.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

From topic areas such as urban planning to space exploration, graph theory encompasses some of the oldest and most interesting areas of algorithmics. A graph, or network, is one of the most important types of models used in discrete applied mathematics (Strang, 2007). This model is used to analyze a wide variety of real-life applications. And as computable aspects of the real world are being analyzed more each day, the study of these large-scale interaction networks is a growing trend. Protein networks (Voevodski, Teng, & Xia, 2009a, 2009b), communications networks (Fortz & Thorup, 2000; Luo, Zhu, Wu, Chen, & Ieee, 2011), aircraft networks (Bard, Yu, & Arguello, 2001; Royset, Carlyle, & Wood, 2009), and road networks (Delling & Wagner, 2007; Geisberger, Sanders, Schultes, & Delling, 2008a) are studied frequently by abstracting them onto a graph. In practice, these networks are mined for structural and relational information to solve problems with respect to their domains.

One of the fundamental, most commonly studied problems in this space is the shortest path problem. The shortest path problem is a query for the lowest cost to get from one node of a graph to another by way of its edges. Computing this query quickly and in a resource efficient manner is beneficial for many applications. The brute force solution for the problem involves testing every path from source to destination in the graph. Methods for efficiently solving the shortest path problem apply a combination of dynamic programming and greedy algorithms to speed up the search. Though these methods are theoretically efficient solutions, their computational time and space requirement is insufficient for graphs at the practical scale of many modern, real-world networks. In this dissertation, a new class of algorithms for solving this problem for large-scale graphs is defined and evaluated through experimentation. In particular, this new method presents a feasible capability for storing basic information about the graph and using this information to guide future searches. To demonstrate its utility, this class of algorithms is applied to a set of benchmark datasets for navigational planning on road networks in a fixed-memory environment.

Background

The problem of pathfinding in a graph was mathematically established in early works by Euler through analysis of the map of Königsberg, a large city in pre-World War II Germany, shown in Figure 1 (Euler, 1736). In 1736, his Königsberg Bridge Problem, modernly known as the Eulerian circuit problem, represented the beginning of not only mathematical pathfinding, but of modern graph theory itself. Heavy research into the point to point shortest path (PPSP) problem started relatively late compared to most other

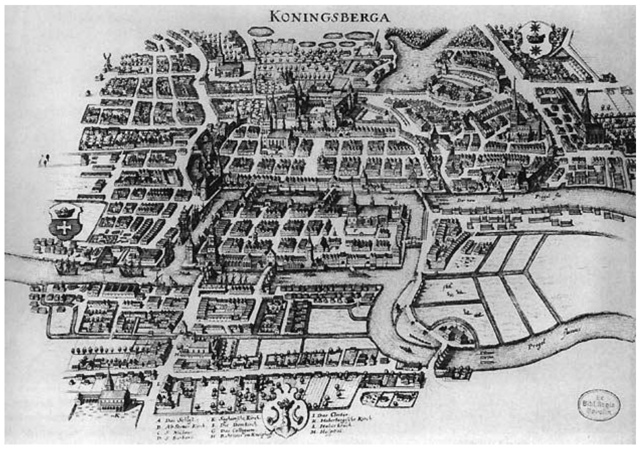


Figure 1 Map of the Seven Bridges of Königsberg, Euler's Inspiration for Studying the Königsberg Bridge Problem

combinatorial optimization problems in graph theory (Aardal, Nemhauser, & Weismantel, 2005). In all likelihood, this may have been because the size of data used for the problem was typically smaller, making the problem seem trivial while anything larger was deemed intractable. At the time of this writing, progress in practically solving the problem has only occurred in the last six decades. Much of the true scientific investigation started with Alfonso Shimbel, in his introduction of the all-pairs shortest path (APSP) problem (Shimbel, 1953). All possible path queries are automatically answered and stored for the APSP problem, while querying is done upon request for the PPSP problem. The solution to the PPSP problem requires an efficient computation of the shortest path between an arbitrary pair of nodes be established.

Shortly after Shimbel, Edsger W. Dijkstra was credited with discovering the algorithm that, at the time of this writing, is the best, most well-known, commonly used, and simplest method of solving the shortest path algorithm in a graph (Dijkstra, 1959). This algorithm is widely known as *Dijkstra's algorithm*. A decade after its creation, the *A* search algorithm* showed, by adding a heuristic that estimates distance, that it could run a shortest path query in significantly faster time than Dijkstra's algorithm (Hart, Nilsson, & Raphael, 1968). Fundamentally, the A* algorithm is Dijkstra's algorithm that takes into account a distance estimation heuristic derived from characteristics of the graph. While other algorithms have been developed in an attempt to contest them, these two greedy optimization algorithms serve as the basis for most modern day shortest path solutions.

As researchers find more use for graph theory in the storage, retrieval, and analysis of big data, extremely fast solutions to problems such as the shortest path problem are in great demand. However, not even Dijkstra's or the A* algorithm can solve the problem for massive datasets without a significant increase in their requirements for computational time and space. For this reason, modern research focuses on performing computations on the graph prior to allowing it to be queried for shortest path. The results of these computations are used to guide, narrow, or inform the search such that arbitrary queries can be performed significantly faster on graphs that represent huge data corpuses. Modern approaches typically exploit mathematical approximation techniques (Delling, Sanders, Schultes, & Wagner, 2009; Delling & Wagner, 2007; Goldberg & Harrelson, 2005; Jens Maue, Sanders, & Matijevic, 2010), large-scale storage (Duan, Pettie, & Siam/Acm, 2009; Goldman, Shivakumar, Venkatasubramanian, & Garcia-Molina, 1998; J. Sankaranarayanan & Samet, 2010; Thorup & Zwick, 2001), artificial intelligence algorithms (Awasthi, Lechevallier, Parent, & Proth, 2005; Yussof, Razali, Ong Hang, Ghapar, & Din, 2009; Zakzouk, Zaher, & El-Deen, 2010; Zongyan, Haihua, & Ye, 2012), and combinations of preprocessing algorithms (Sanders & Schultes, 2007). Of these approaches, the focus of this dissertation is an evaluation of strategies for aiding shortest path approximation known as landmark selection strategies. A series of landmark selection strategies is applied to a new class of algorithms to address one of the original applications of the problem, road navigation planning.

Problem Statement

Large-scale navigation planning requires the ability to regularly compute the shortest path for massive road networks. In such cases, preprocessing algorithms are used to increase the performance of queries. Many shortest path preprocessing algorithms require very heavy upfront computation and storage. In some cases, they require structural information about the graph that may not be able to be obtained in real-world applications. Moreover, many require a significant amount of information to be stored in order to yield reasonable speedups. Few algorithms concern themselves with the space complexity required by such preprocessing techniques. The problem that this dissertation addresses is that modern PPSP preprocessing algorithms have space and preprocessing time requirements for large-scale graphs that are impractical in terms of utility in real-world applications. While cloud computing is often used to perform navigation planning for devices that report location, network connectivity issues can prevent reasonable responses to navigation planning queries. For such mission-oriented devices that then must perform navigation planning locally, particularly with limited memory resources, these computational requirements must be reduced.

Dissertation Goal

The primary contribution of this dissertation is the description, software implementation, and experimental evaluation of a new class of algorithms for generating a heuristic function for the A* algorithm (Hart et al., 1968). Its novel feature is that it uses more information about the graph to generate the heuristic while requiring significantly less computational space, making it a favorable algorithm to use in a fixed memory environment. This new heuristic is based on a class of algorithms known as ALT (Goldberg & Harrelson, 2005). ALT describes a preprocessing technique for shortest path queries that chooses a relatively small number of landmark nodes in a graph, computes the distances between all vertices and these landmarks, and establishes lower bounds using this distance information and the triangle inequality during search queries. However, by using information about multiple landmarks, new lower bounds can be computed from other polygon inequalities. These inequalities can be derived from either generalized polygon inequalities or ones specific to a shape embedded within the graph. The use of these new lower bounds as a heuristic has resulted in a new class of algorithms called ALP, an acronym for A*, Landmarks, and Polygon Inequalities.

The ALT algorithm requires a spanning shortest path tree, rooted at each landmark to be generated and stored, in a process known as *landmark embedding*. However, through a process called *distributed landmark embedding*, hereafter referred to as distributed embedding, ALP generates shortest path trees only encompassing the local areas surrounding each landmark, resulting in a significant reduction in required memory. By using smaller shortest path trees with multiple landmarks to guide the search, ALP also reduces the amount of required apriori computation for shortest path search. In many practical cases, it also increases the efficiency of computing the A* heuristic. This heuristic's domination over ALT's depends on the landmark set that each is assigned. Therefore, if an optimal landmark set can be determined more efficiently under the ALP paradigm than under ALT, then ALP is the more efficient heuristic to use for A* search. The goal of this dissertation is to identify and characterize landmark selection techniques for a concrete ALP heuristic function that lends a significant memory and preprocessing time reduction while maintaining the experimental speedups that ALT provides.

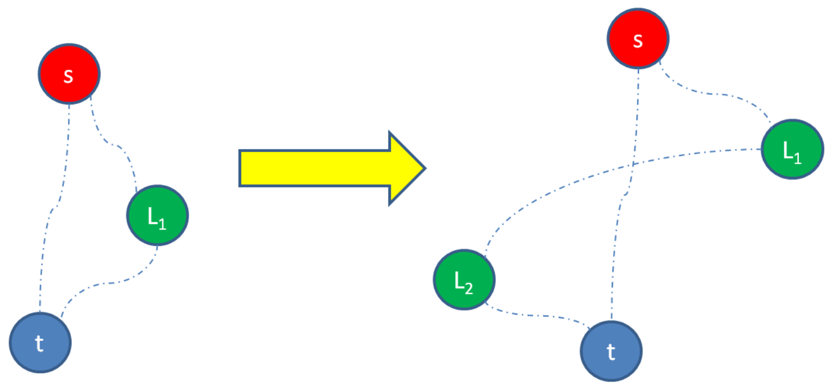


Figure 2 Notional diagram of changing the approach for guiding shortest path search from a Single Landmark(ALT) to a Dual Landmark approach (ALP)

The base case function for ALP, using one landmark to compute the A* heuristic function, is already characterized as the ALT algorithm. To begin to characterize the behavior of this class of algorithms with increasing information, this research theorizes and experiments with the behavior of A* using two landmarks as shown in Figure 2. The use of two landmarks, in this way, acts as an inductive step for using multiple landmarks to guide A* search. In the first three chapters of this report, the characterization of this dual landmark approach for ALP is formed. The ALP algorithm was implemented and tested using benchmark road graph datasets on which the ALT algorithm and several other major algorithms were tested (Demetrescu, Goldberg, & Johnson, 2006). The algorithm's performance bounds are compared with ALT's in common environments. ALP is tested using the most common modern landmark selection techniques to characterize its behavior for each of them. Data is collected to identify how large the shortest path tree actually has to grow for each landmark in the dataset to maintain an overall performance benefit. The scenarios in which each of the different shortest path preprocessing techniques and landmark selection techniques are optimal are characterized and experimentally tested. A suite of software tools for future use in situational shortest path solving is generated. In the end, an applicable algorithm for shortest path speedup under limited memory resources is demonstrated and verified.

Research Questions

The following questions pertain to the contribution of this effort and are answered through a combination of theory and experimentation:

- *What landmark selection techniques theoretically fit best with ALP?*

  The ALP class of algorithms differs in behavior from the ALT class of algorithms because of ALP's memory-reducing properties (i.e., distributed landmark embedding). These properties change the average expected computational performance of PPSP queries for each landmark selection technique. Some landmark selection techniques perform better under ALP while others will perform worse. However, because ALP with distributed embedding has to perform significantly less preprocessing, landmark selection techniques that result in heuristics that are on par with ALT's allow ALP to be leveraged as a more efficient approach than ALT.
- *Using ALP with distributed landmark embedding, what are the ideal characteristics for landmark shortest path trees? In other words, how much preprocessing and memory is required for ALP to maintain its key benefits?*

  Due to distributed landmark embedding, ALP requires preprocessing at a level significantly less than ALT. Each landmark grows significantly smaller shortest path trees in comparison. While guaranteed to be less than that of ALT, the exact amount of preprocessing is not theoretically defined as it is relative to the inputted graph information. If the graph has a very small number of partitions, the preprocessing may not see a significant reduction in compute time.

- *How does the algorithm behave as the number of landmarks used to guide the search increases?*

  A single landmark approach (ALT) and a dual landmark approach (ALP with distributed embedding) for guiding shortest path search using polygon inequalities is studied in this dissertation. These studies identify the benefits and drawbacks of each approach. Experimental results corresponding to each type of shape being identified in the graph are detailed in this effort. Future research will involve identifying other shapes with a larger number of sides (pentagons, hexagons, heptagons, etc.) to discover the benefits and detriments of continuously increasing the number of landmarks used for guiding the search.

The following open question pertains to how ALP's contributions can be further characterized.

- *In what ways can this be applied to path planning? What real-world applications exist for ALP that were previously impractical to solve with ALT?*

  In the real world, memory-limited capabilities for quickly computing shortest paths can enable smaller, memory-limited devices without constant internet or local network connection to navigate paths in large graph datasets. The reduced requirement of a persistent connection for path planning reduces the amount of energy required to power such devices. Such localized navigation planning also allows for more intelligent planning to occur in denied areas such as space or military domains.

Relevance and Significance

The shortest path problem is a classic problem in computer science (Dijkstra, 1959). Many developed preprocessing methods for Dijkstra’s algorithm efficiently solve the problem, but incur tradeoffs for large graphs that are impractical in some use cases. The need to analyze large real-world networks is steadily growing as more information is being accumulated about the real world and the use of digital services, networks, and devices grows. This scaling-up of networks creates a need for algorithms to be able to compute over large datasets without incurring a significant operational cost.

In areas such as navigation planning, smaller and smaller devices are required to do computing while using minimal bandwidth for communication. While newer devices are becoming more powerful, many still lack the ability to perform shortest path queries efficiently on large datasets using naïve algorithms. The required preprocessing for most real-world applications is slower for large-scale graphs, as the time to generate shortest path trees grows as a function of the number of graph elements. These problems need to be solved quickly, using minimal resources and, in some cases, limited preprocessing time. The problem has been cited in many other works and is commonly solved by pushing the problem off to an external memory source (A. Goldberg & R. Werneck, 2005; Hutchinson, Maheshwari, & Zeh, 2003). However, the problem must also be solved for devices that have very little to no external memory sources in various scenarios (Dong, ZuKuan, Jae-Hong, & ShuGuang, 2010; Santhosh, Sasiprabha, & Jeberson, 2010). For these types of devices, memory and processor usage play a large role in the energy consumption of the system and overall cost. Many modern approaches for pathfinding on these types of devices lack the dual benefit of low memory usage and efficient computation. In general, one is sacrificed for the other.

In particular, modern GPS-enabled devices are commonly tasked with computing the shortest path on the fly for downloaded map data (Bo & Dong, 2010; Holdsworth & Lui, 2009). Also, many such devices have very little external memory to store the massive amount of preprocessing information required by other methods. For small multipurpose devices without persistent network connections, this computation needs to be performed repeatedly on the same dataset as it is held in primary memory (Cerf et al., 2007; Jain, Fall, & Patra, 2004). A reduction in computation when solving this problem can reduce the amount of energy required for these devices, as well, while allowing them to efficiently perform other tasks at the same time. For these reasons, precomputing a reasonable amount of data to help guide the search such that a query can be practically executed on a device is a common need for individual consumers, businesses, and governments.

Aside from shortest path queries, landmark selection techniques are employed in a host of other applications. The notion of using landmarks to estimate distance information in a graph structure was actually conceptualized before their use in PPSP queries. Common routing protocols typically rely on landmarks such as key routing devices to decide whether or not other devices are too far away (Cowen, 1999). Internet distance information (in hops) and a concept of Internet coordinates is often measured using landmarks as a guide (Costa, Castro, Rowstron, & Key, 2004). Landmarks are naturally used by honey bees to estimate the flight path to their hives (Chittka, Geiger, & Kunze, 1995). And finally, landmarks have been used to create filters for string comparisons when detecting duplicates among large datasets (Weis & Naumann, 2004). In general, discoveries about the benefits and detriments of using multiple landmarks to perform estimation can benefit many landmark-based research efforts.

Barriers and Issues

The dual landmark heuristic demonstrated in this dissertation for ALP only outperforms ALT in certain scenarios. Over the same set of landmarks, the estimates computed by the dual landmark ALP heuristic has equal or worse performance than ALT. However, given ALP's ability to choose a denser landmark set, we see a performance increase over ALT. In this dissertation, we demonstrate how much more dense this set has to be for ALP. Regardless, even when ALP demonstrates little or no average time complexity reduction, its space complexity reduction is guaranteed.

One main goal of this dissertation is to explore the efficacy of landmark selection strategies that can optimize ALP algorithms. As of the time of this writing, this is still an open problem for ALT. Many authors have experimentally concluded that random selection of landmarks is good enough in many cases, with no theoretical backing (Goldberg & Harrelson, 2005; Potamias, Bonchi, Castillo, & Gionis, 2009). We characterize what "good enough" would mean for ALP in this dissertation, leaving the use of landmark selection techniques up to implementers. We are able to characterize this because of the ability to perform more experiments, a direct benefit of the smaller preprocessing time and space requirement of dual landmark ALP with distributed embedding. Therefore, a significant number of trials were performed for each experiment with a wide array of landmarks to obtain a better experimental characterization than seen in previous efforts.

Assumptions, Limitations, and Delimitations

This dissertation relies on a theoretically proven heuristic. Timing and memory usage, measured on a developer-class system, are recorded through program instrumentation for a host of metrics (e.g. number of nodes/edges explored, number of arithmetic operations, memory usage, % CPU usage, computed runtime in seconds, etc.). Conclusions about ALP's behavior in navigation planning environments shall be drawn from the measurements reported by this instrumentation. Such conclusions, however, fall prey to a small set of limitations. The first limitation stems from randomness (or more appropriately, pseudo-randomness). The random landmark selection technique is currently seen as a good technique for ALT (Goldberg, Kaplan, & Werneck, 2009). In ALT, testing a significant number of queries for various trials of random landmark selection becomes difficult because of the time that it takes to generate a shortest path tree from each node. For extremely large graph datasets, such computations for a significant number of trials ($\sim 10^6$) should sufficiently justify the behavior of the random landmark selection technique along with all other landmark selection techniques for ALP. The second primary limitation stems from the data used for experimentation. In this dissertation, the benchmark data used for experimentation is collected from the same sources used in each of the original research efforts that the algorithm will be compared against. Some of this data was not be able to be obtained due to insufficient citation of the source or simply a lack of access. For experimentation, all datasets used in these studies were either downloaded or replicated to sufficiently duplicate the results found in each study.

Intentionally excluded from this research is any experimentation using more than two landmarks for ALP. The main polygon inequalities that are used in this study are quadrilateral inequalities, as the use of two landmarks forms the shape of a quadrilateral in the graph. This allows the research to serve as a base demonstration of how a heuristic function behaves when more than one landmark is used to form a polygon in a graph. The focus, however, will not be on further increasing the number of landmarks that are used by the heuristic function. Rather, it will be on characterizing the behavior of the landmark selection techniques for ALP's dual landmark heuristic function. This full characterization provides an experimental template for future heuristics that use even more landmarks in their functions.

Definition of Terms

Throughout this dissertation, for clarity, a common set of graph theoretical definitions, concepts, and notations will be used. Let $G = (V,E)$ be an *undirected graph*, where V is the set of *vertices* in G and $E \subseteq V \times V$ is the set of *edges* in G, with $n = |V|$ and $m = |E|$. For any edge $e \in E$, let $w(e)$ be the positive real weight of the e. In an *unweighted* graph, for every edge $e \in E$, $w(e) = 1$. In a *weighted* graph, $w(e)$ is subject to the graph's application. A *finite* graph is one in which $|V| \neq \infty$ and $|E| \neq \infty$.If an edge $e \in E$ connects two vertices $v_i,v_j \in V$, v_i is called the *neighbor* of v_j and v_j the *neighbor* of v_i. The vertices v_i and v_j are also said to be *adjacent* to each other and *incident* to their shared edge e. A graph $H = (V(H), E(H))$ is a *subgraph* of G if $V(H) \subseteq V$ and $E(H) \subseteq E$, with edges of $E(H)$ incident to only the vertices in $V(H)$. A *spanning subgraph* H of G is a subgraph in which $V(H) = V$.

An *induced subgraph* H of G is a subgraph of G such that $V(H) \subseteq V(G)$ and two vertices of H are adjacent if and only if they are adjacent in G. In other words, H is an induced (Blondel, Guillaume, Lambiotte, & Lefebvre, 2008)subgraph of G if and only if it has exactly the edges that exist for G over the same vertex set. A *graph cluster, partition*, or *community* is a collection of vertices in a graph such that the vertices assigned to a particular community are similar or connected by some predefined criteria.

A sequence $(v_0,\ldots,v_{k-1})$, $k \geq 1$, of vertices of $G = (V,E)$ is known as a *path* from v_0 to v_{k-1} if there is an edge $(v_i, v_{i+1}) \in E$ for every $0 \leq i < k$. A *path* is denoted as $P(v_0,v_{k-1}) = \langle v_0,\ldots,v_{k-1}\rangle$. A path P is a subgraph of G. The *length* of P is the number of edges (i.e., $k - 1$) on the path $P(v_0,v_{k-1})$, denoted as $d(v_0,v_{k-1})$ or $d(P)$, and the *weight* of P is the sum of the weights of the path edges, denoted as $w(P)$ or $w(v_0,v_{k-1})$. If, for every pair of vertices $v_i,v_j \in V$, there exists a path from v_i to v_j, the graph is called *connected*. An acyclic, connected, spanning subgraph of G is called a *spanning tree* of G. In this dissertation, the experiments are performed on *finite*, *connected* graphs, both *directed* and *undirected*. Directed graphs will be *strongly connected*, meaning that each vertex can be reached from every other vertex in the graph.

Many algorithms exist for identifying communities in graphs, a process known as *community detection*. A common community detection algorithm used throughout this dissertation is an algorithm dubbed the *Louvain method* (Blondel et al., 2008). The algorithm is a greedy optimization method that attempts to optimize a score known as *modularity*, a measurement of the fraction of edges that fall within a community minus the expected fraction if edges were distributed at random. The Louvain method occurs in two phases: In the first phase, the method identifies small communities by optimizing modularity locally. This is done by assigning each vertex in a network its own community, computing the modularity increase of moving the vertex into each of its neighbors' communities, and keeping the vertex in the community that resulted in the highest modularity increase for the graph (or in its own community, if no modularity increase occurs). This process is repeated for all nodes until no more modularity increases are possible. In the second phase, the nodes determined to be those of the same community are grouped together and a new graph is built where vertices are the communities from the first phase and weighted edges represent the edges between multiple border nodes from the first phase and self-loops for edges within the community. These two phases are repeated iteratively until a maximum modularity is attained and a hierarchy of communities, often modeled as a *dendrogram*, is formed for each phase. A dendrogram is a tree-like representation of the hierarchical clustering where each level of the tree represents the partitioning for the graph at that level, with the first level indicating maximum modularity for the Louvain method.

Also, this paper references several fundamental graph theoretic problems and algorithms. Given a graph $G = (V,E)$, the *point-to-point shortest path problem* (PPSP) is one of finding the path that comprises the shortest path in the graph from a specified vertex s, known as the *source*, to a specified vertex t, known as the *destination*. For two vertices $s,t \in V$, a path $P(s,t) \in G$ is called a *shortest path* from s to t if there exists no path $P'(s,t) \in G$ such that $d(P') < d(P)$ and $P' \neq P$. The *distance* between two vertices $s,t \in V$ is the sum of the weights on the shortest path and is denoted by $d(s,t)$. For weighted graphs, the *weight* of an individual edge is a numeric value that identifies the cost of traversing the edge in a path calculation. The weight of the edge that connects two vertices $v, v' \in V$ is denoted as $w(v, v')$. In Chapter 2, many of the reviewed algorithms apply to both weighted and unweighted graphs.

A *single-source shortest path tree* (SPT), is a spanning tree of a connected graph G, rooted at s, connecting all the vertices such that the length of the path to each vertex t in the tree is $d(s,t)$. The problem of computing this tree is known as the *single-source shortest path problem* (SSSP). The *all pairs shortest path problem* (APSP) attempts to find a shortest path from u to v for every pair of vertices $u,v \in V$.

With respect to algorithmic complexity, the *preprocessing time* of a shortest path algorithm refers to the worst-case time required to construct the data structure used to speed up shortest path queries. The *space complexity* is the worst-case size of such a data structure. And finally, the *query time* refers to the worst-case time required to compute either $d(s,t)$, $P(s,t)$, or both for $s,t \in V$.

Another important class of problems for large graphs involves the idea of probabilistic movement from one vertex of a graph to another vertex by way of incident edges. This is another way that graphs can characterize real-world interactions. For instance, a web surfer browsing from site to site or a disease spreading between humans by means of direct contact are two applications that can be modeled by probabilistic movement from vertex to vertex in a graph. In these problems, a *surfer* is an entity that is able to walk from vertex to vertex in the graph by way of its edges. A *random walk* on a graph is a finite, time-reversible Markov chain (Freedman, 1971). Given a graph $G = (V,E)$ and a starting vertex for the surfer, at each time step t, a neighbor is selected at random and the surfer moves to it. When the graph is unweighted, the surfer moves to a neighbor with uniform probability. When it is weighted, it moves to a neighbor with probability proportional to the weight of the incident edge.

The most common algorithm used to solve the shortest path problem in both directed and undirected graphs is known as Dijkstra's shortest path algorithm, or simply *Dijkstra's algorithm* (Dijkstra, 1959). Dijkstra's algorithm naturally creates an SPT in a graph, rooted at the source vertex, by finding the shortest path from the source vertex to one additional vertex at each iteration of the algorithm's primary loop. Each vertex $v \in V$ is in one of three states: *visited*, *unvisited*, or *settled*. The shortest path from the source vertex s to a vertex $u \in V$ is found once the state of u is settled. This settling occurs in the process specified by the pseudocode for the algorithm in Figure 3. Steps 11-15 are

```
Dijkstra($G = (V,E)$, $w: E$, $s$, $t \in V$)
1.    for each vertex $u \in V$
          Set the parent of $u$ to null
          Set the state of $u$ to unvisited
          Initialize $d(s,u)$ to $\infty$
2.    Set the state of $s$ as visited
3.    Set $d(s,s)$ to 0
4.    Insert all nodes  into Priority Queue $Q$                //Open Set
5.    while $Q$ is not empty and $t$ has not been visited
6.        Extract minimum $u \in V$ from Q
7.        Mark the state of $u$ as settled
8.        if u = t: stop
9.        For each vertex $v \in Q$ adjacent to $u$ that has not been settled
              //Relax the edge
10.           if $d(s,u) + w(\mathrm{u,v}) < d(s,v)$:
11.               Set the parent of $v$ to $u$
12.               Set $d(s,v) = d(s,u) + w(u,v)$
13.               if v is not visited:
14.                   Insert v into Q with priority d(s, v)
                      Set the state of $v$ to visited
                  Else:
15.                   Decrease the priority of $v$ in Q to $d(s,v)$
16.   return $d(s,t)$
```

Figure 3 Dijkstra's Algorithm for SSSP Queries

referred to as *relaxing an edge*.

This algorithm is an efficient greedy algorithm that effectively solves the single-source shortest path problem for graphs with non-negative edge weights. However, this restriction on edge weights can be removed using Johnson's algorithm to convert negative edge weights to non-negative in O($|V||E|$) (Johnson, 1977). Overall, the naïve version of Dijkstra's answers single-source shortest path queries in O($|V|^2$) time. The best version of the algorithm, using Fibonacci heaps (O(log $|V|$) deletions and insertions), manages to answer PPSP queries with a query time of O($|E|$ + $|V|$log $|V|$) (Fredman & Tarjan, 1987). For APSP, computing Dijkstra's from every vertex simply requires multiplying this query time by the total number of vertices, leaving the worst case bounds at O($|V||E|$ + $|V|^2$ log $|V|$). To date, there is no general sub-cubic algorithm that calculates an APSP solution for any type of simple graph, though faster solutions have been provided for graphs with certain constraints (Chan, 2007; Seidel, 1995). For general APSP, the Floyd-Warshall algorithm is the industry-standard algorithm with a time complexity of $O(|V|^3)$ (Floyd, 1962). If a target vertex *t* is provided, the bidirectional version of Dijkstra's algorithm can start a second search from the target vertex, alternating the search direction at each iteration and finishing when the frontiers of both searches meet.

The A* algorithm behaves similarly to Dijkstra's but with a *heuristic function*, π_t, guiding the search (Hart et al., 1968). Throughout this paper, $\pi_t(s)$ will denote the estimated cost of the shortest path from a vertex $s \in V$ to target vertex $t \in V$. This is also known as the *heuristic cost*. The A* search strategy uses this function to add additional knowledge about graph structure to the shortest path problem, pruning from the search space vertices that do not need to be considered. The pseudocode that demonstrates this addition is displayed in Figure 4. The figure also demonstrates that Dijkstra's algorithm is simply the A* algorithm without a search heuristic (or $\pi_t = 0$).

In terms of identifying shortest path, Dijkstra's algorithm is both *complete* and *optimal*, meaning that the algorithm both always finds the shortest path if one exists and it is guaranteed that there is no shorter path than the one that it finds, respectively (Russell & Norvig, 2009). However, A* possesses these properties only if the heuristic function π_t adheres to certain constraints. First, it must satisfy the constraints of Dijkstra's algorithm, meaning that the graph is finite and that it has non-negative edge weights. To

```
A*($G = (V,E), w: E, s,t \in V, \pi_t$)
1.    for each vertex $u \in V$
          Set the parent of $u$ to null
          Set the state of $u$ to unvisited
          Initialize $d(s,u)$ to $\infty$
2.    Set the state of $s$ as visited
3.    Set $d(s,s)$ to 0
4.    Insert all nodes  into Priority Queue $Q$        //Open Set
5.    Create empty set $R$          //Closed Set
5.    while $Q$ is not empty and $t$ has not been visited
6.        Remove minimum $u \in V$ from $Q$
7.        Mark the state of $u$ as settled
8.        if $u = t$: stop
9.        Add $u$ to $R$
10.       For each vertex $v \in V$ adjacent to $u \in V$
11.           $g' = d(s,u) + w(u,v)$
12.           $f' = g' + \pi_t(v)$     //$\pi_t$ is the A* heuristic function
13.           if $v \in R$ and $f' \geq d(s,v)$: continue
14.           if $v \notin Q$ or $f' < d(s,v)$:
15.               Set the parent of $v$ to $u$
16.               $g[v] = g'$
17.               $f[v] = f'$
18.               if $v \notin Q$: add $v$ to $Q$
```

Figure 4 A* Algorithm for PPSP Queries

achieve optimality, the first constraint is that the heuristic function, π_t, must be *admissible*, never overestimating the distance to the target vertex. This means that, in the case of graphs, for a heuristic function to be admissible, for any vertex $v \in V$,

$$\pi_t(v) \leq d(v,t) \qquad (1)$$

An intuitive example of an admissible heuristic is in the case of routing applications, in which the straight line distance to a target point is used as the admissible heuristic. Because the shortest distance between two points on a map is a straight line, it can never overestimate the distance of the path to the target at any point in the search.

The second constraint for optimality states that π_t must be *consistent*, meaning that the algorithm never traces its steps backward when attempting to settle the path (Russell & Norvig, 2009). More formally, when settling vertices on a path, if for every vertex n and every successor vertex n', the heuristic cost $\pi_t(n)$ should be no greater than the cost of getting to n' plus $\pi_t(n')$. So

$$\pi_t(n) \leq w(n,n') + \pi_t(n') \qquad (2)$$

Every consistent heuristic is also admissible, as it can never overestimate the cost of reaching the target vertex (Russell & Norvig, 2009). The consistency constraint requires a heuristic to obey the *triangle inequality*, which requires that one side of a triangle can be no longer than the sum of its other two sides. In the case of Equation 2, the triangle's endpoints are represented by n, n', and t.

For an A* query, let n be the vertex currently being visited on the search and let m be the previously visited node. An admissible heuristic π can be made into a consistent heuristic π' can by making the following adjustment:

$$\pi'_t(n) = \max\left(\pi_t(n), \pi_t(m) - w(m,n)\right) \qquad (3)$$

The equation for this heuristic is known as the *pathmax* equation and can be used to force consistency for any admissible heuristic. It is extremely useful when a proof of consistency has not been found for an admissible heuristic.

Finally, let $\pi_{\langle t,1\rangle}(v)$ and $\pi_{\langle t,2\rangle}(v)$ each be an admissible heuristic function for any vertex $v \in V$ of the graph, let

$$\pi_{\langle t,1\rangle}(v) \;\geq\; \pi_{\langle t,2\rangle}(v) \tag{4}$$

If Equation 3 holds, then $\pi_{\langle t,1\rangle}$ *dominates* $\pi_{\langle t,2\rangle}$, verifying that $\pi_{\langle t,1\rangle}$ is a more efficient heuristic. An A* search using $\pi_{\langle t,1\rangle}$ as a heuristic visits no more nodes than $\pi_{\langle t,2\rangle}$ on its way from source to target, allowing it to reach the target while visiting fewer nodes in the graph. A* can never suffer a performance degradation by switching from one heuristic to another consistent heuristic that dominates it (Pearl, 1984). Therefore, the best possible heuristic is the most dominant, consistent heuristic. Just as with Dijkstra's algorithm, A* also has a bidirectional variant. In the bidirectional variant, two heuristic functions are used with the same criteria of being consistent (and inherently, admissible).

A *metric space* is a set with a global distance function *d* known as a *metric* that, for any points *x*, *y* in the set, gives a nonnegative real number as the distance between them. A metric satisfies the following properties for all points *x*, *y*, *z* in the set:

- $d(x,y) \geq 0$ (nonnegative)
- $d(x,y) = 0$ if and only if $x = y$ (identity)
- $d(x,y) = d(y,x)$ (symmetry)
- $d(x,y) \leq d(x,z) + d(z,y)$ (the triangle inequality)

Using the shortest path between two vertices as the distance function, a finite, connected, undirected graph with positive edge weights fits each of these requirements and is, therefore, a metric space. A directed graph with non-negative edge weights is a *quasi-metric space*, meaning it has all the properties of a metric space except the symmetry property. The *triangle inequality*, originally proposed by Euclid in Elements around 300 BC, specifies that for three points in a metric space, the distance between any two of those points is no greater than the sum of the other two distances that form the triangle (Millman & Parker, 1991). For points *x*, *y*, *z* in a metric space, the triangle inequality states:

$$d(x,z) \leq d(x,y) + d(y,z) \tag{5}$$

This establishes an upper bound for the distance between points *x* and *z*. A lower bound can also be derived from the triangle inequality.

$$d(x,y) \geq \ |d(x,z) - d(y,z)| \tag{6}$$

This is known as the *reverse triangle inequality* and is derived from the triangle inequality as follows. First, subtract $d(y,z)$ from both sides from Equation 4:

$$d(x,y) \geq d(x,z) - d(y,z) \tag{7}$$

For $d(x,z) - d(y,z) \geq 0$, Equation 5 holds. Then, for $d(x,z) < d(y,z)$, we examine the following triangle inequality for points *y* and *z*.

$$d(y,z) \leq d(y,x) + d(x,z) \tag{8}$$

Subtracting $d(x,z)$ on both sides, we get

$$d(x,y) \geq d(y,z) - d(x,z) \tag{9}$$

By combining Equations 6 and 9, the new lower bound for x and y becomes

$$d(x,y) \geq \ |d(x,z) - d(y,z)| \tag{10}$$

Because obeying the triangle inequality is a required property of a metric space, the reverse triangle inequality is a required property, as well.

The triangle inequality can be generalized for all polygons through induction (Millman & Parker, 1991). Given a set of points P_1, P_2, …, P_n in a metric space,

$$d(P_1, P_n) \leq d(P_1, P_2) + d(P_2, P_3) + \cdots + d(P_{n-1}, P_n) \tag{11}$$

This is known as the *generalized polygon inequality* and follows from induction from the triangle inequality.

Finally, another geometry-based inequality for metric spaces is known as *Ptolemy's Inequality*. For four points w, x, y, z in a metric space, Ptolemy's Inequality states that

$$d(w, x) \cdot d(y, z) + d(x, y) \cdot d(w, z) \geq d(w, y) \cdot d(x, z) \tag{12}$$

This inequality is derived from measuring the sides of quadrilaterals (Kay, 2011).

PageRank is an edge analysis algorithm that is used to compute the probability that a vertex in a network will be visited on a random walk of the network (Brin & Page, 1998). Its initial intention was to act as a ranking system for distinct vertices (web pages), indicating their individual popularity in a random walk of the graph. However, the algorithm has demonstrated utility in a wide variety of graph applications in which analyzing the priority of particular vertices is a concern (Andersen, Chung, & Lang, 2006; J. Chen, Bardes, Aronow, & Jegga, 2009; P. Chen, Xie, Maslov, & Redner, 2007; Liu, Bollen, Nelson, & Van de Sompel, 2005).

PageRank is an eigenvector centrality measure that is computed as follows. Given a graph G with $n = |V|$ vertices and vertices numbered 1 through n, an *adjacency matrix* A is an $n \times n$ matrix formed such that

$$A_{ij} = \begin{cases} 1, & An\ edge\ exists\ between\ vertices\ i\ and\ j \\ 0, & Otherwise \end{cases} \tag{13}$$

for $i,j \epsilon [1,n]$. This is the simplest type of adjacency matrix. In other applications, the weight of the edge or number of edges between two nodes is used for edges between two vertices.

After forming the adjacency matrix, an $n \times n$ *transition probability matrix P'* is computed, where each element P'_{ij} contains the probability that a surfer would move from vertex i to vertex j. For each vertex $i \epsilon V$ represented by a row A_i in the adjacency matrix, let $L(i)$ represent the set of vertices adjacent to i. P'_{ij} is then computed as follows:

$$P'_{ij} = \begin{cases} \frac{1}{|L(i)|}, & if\ j \in L(i) \\ \frac{1}{|V|}, & if L(i) = \emptyset \\ 0, & otherwise \end{cases} \tag{14}$$

(Page, Brin, Motwani, & Winograd, 1999)

The goal of PageRank is to identify the principal eigenvector of the transformation of this matrix that takes into account *surfer teleportation*, the likelihood of a surfer to move to another vertex without following any specific path in the graph. To compare this to web browsing behavior, this is the likelihood of a surfer “getting bored” and finding a new web page to start surfing. Let $\alpha \epsilon [0,1]$ represent this probability. Then P, the transition probability matrix taking into account surfer teleportation, is computed as follows:

$$P_{ij} = (1 - \alpha)P'_{ij} + \frac{\alpha}{|V|} \tag{15}$$

The principal eigenvector of P can be computed by a variety of different methods for speed or application (Das Sarma, Gollapudi, & Panigrahy, 2011; Kamvar, Haveliwala, & Golub, 2004; Sun, Deng, & Deng, 2008). The basic algorithm that is used to quickly approximate the principal eigenvector is known as the *power method* (Mises &

Pollaczek-Geiringer, 1929). A delta vector δ and initial guess vector x_0 for x of size n with arbitrary inputs is created and is continuously updated by

$$x_k = x_{k-1}\mathrm{P} \tag{16}$$

until

$$|x_k - x_{k-1}| < \delta \tag{17}$$

The final derived vector x_k is known as the PageRank vector, with the value in $x_k[i]$, $1 \leq i \leq n$, representing the PageRank value of the vertex corresponding to i. Using this method, PageRank maintains a time complexity of O(|*E*|) (Bao, Feng, Liu, Ma, & Wang, 2006).

Summary

Modern day techniques for preprocessing large graphs to aid shortest path queries are insufficient in many real-world applications for devices with limited resources. Some algorithms rely on large amounts of memory, removing the ability for the device to perform other operations while performing navigation planning. Others rely on heavy compute resources, which can be expensive at smaller scales and consume a large amount of energy. To address this problem, this dissertation characterizes and compares the theoretical and practical performance of ALP, a new class of algorithms against ALT, the preprocessing technique from which it was derived. When combined with distributed embedding, ALP’s novel feature is that it can rely on more precomputed distance information than ALT to derive a heuristic for A* while realizing a significant reduction in both space complexity and preprocessing time. Its ability to quickly perform preprocessing lends itself to better landmark selection, as more trials to vet landmarks can occur. It is also able to compute and store more landmark information with a fixed amount of required memory. Because of its improved preprocessing, heuristics can be generated that are on par or even better than those generated by ALT. The algorithms' characterization will occur through the identification of optimal landmark selection strategies in an effort to advise future users of the algorithm of the initial computations that need to be performed in a network. Such experiments will occur with both synthetic and real world benchmark data to truly test the algorithms in a variety of scenarios. In the end, a set of portable graph libraries, a theoretical and experimental characterization of ALP against ALT, and a characterization of landmark selection techniques for the ALP approach will be generated.

This dissertation is organized as follows. Chapter 2 introduces the problem of preprocessing the shortest path algorithm and reviews existing methodologies for path planning and landmark selection. Chapter 3 introduces the motivations for using the polygon inequality to guide A* shortest path searching, laying the foundations of the ALP class of algorithms and establishes several theoretical techniques for identifying landmarks. Chapter 4 describes data analysis, findings, and results of experimentation with respect to the bounds and landmark selection algorithms for ALP contrasted with that of ALT. Chapter 5 summarizes the conclusions of the study based on the analysis described in Chapter 4 in relation to the theoretical characterization described in Chapter 3.

Chapter 2

Review of the Literature

To understand the principles of preprocessing a graph to perform shortest path queries, identify new methods of approximate distance estimation, address techniques for identifying landmark elements of the graph from which to base distance estimation, and develop algorithms that maintain realistic space complexity, this chapter provides a review of key papers from the academic literature.

Metric-Independent Shortest Path Preprocessing

Significant work has been done in preemptively analyzing graphs to store information that can assist in solving the point-to-point shortest path (PPSP) problem (Awasthi et al., 2005; Duan et al., 2009; Lin, Kwok, & Lau, 2003; Sanders & Schultes, 2007). Performance for algorithms that attempt to maintain exact distance information degrades for large-scale graphs. In this literature review, algorithms that focus on distance estimation are described. In particular, because ALP and ALT algorithms rely on the same fundamental principles, the preprocessing algorithms in this review have been vetted through their comparison to ALT algorithms.

In practice, the applications of a graph are taken into account to create metrics that advise shortest path search queries (Delling, Goldberg, Pajor, & Werneck, 2011). The development of such preprocessing algorithms is an acknowledgement, on behalf of the academic community, that more efficient algorithms than normal Dijkstra’s or A* are needed to handle the challenges of real-world pathfinding applications. While this dissertation is concerned with practical applications of shortest path search, the goal is to make practical a general class of algorithms for shortest path preprocessing. Therefore, the preprocessing performed by the ALP algorithm will be compared and contrasted with other forms of *metric-independent preprocessing*, which are preprocessing algorithms that only take the graph topology as input (Delling et al., 2011). Such algorithms have the shortcoming of producing a large amount of auxiliary data for use during query time. As shown in Figure 5 below, metric-independent preprocessing commonly involves performing some computations and storage of a subset of possible distance information for key points in a graph prior to running PPSP queries. One of the main contributions of this dissertation is to demonstrate a class of algorithms that significantly reduce the amount of auxiliary data while maintaining a practical speedup to the A* algorithm.

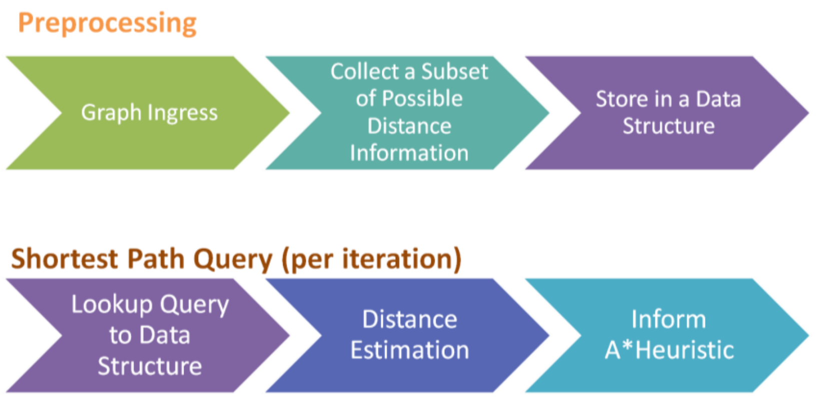

Preprocessing
Graph Ingress
Collect a Subset of Possible Distance Information
Store in a Data Structure
Shortest Path Query (per iteration)
Lookup Query to Data Structure
Distance Estimation
Inform A*Heuristic

Figure 5 Common Paradigm for Metric-Independent Preprocessing

A, Landmarks, and Triangle Inequality (A. V. Goldberg & Harrelson, 2005)*

While many other metric-independent preprocessing algorithms exist, ALT, developed by Goldberg and Harrelson, was the original algorithm to propose using landmark methods to speed up A*. ALT describes a class of algorithms that compute a heuristic for A* by using precomputed shortest path trees (SPTs). These SPTs are rooted at strategically chosen landmark vertices in the graph. Using the triangle inequality, the distance information stored by these SPTs is exploited to estimate the distance between a visited vertex and a search target (Goldberg & Harrelson, 2005). The ALT algorithm is one of the central focuses of this dissertation. Both the ALT and ALP algorithms depend on the same fundamental principles to estimate distances in a graph. Specifically, we will investigate landmark selection methods that optimize heuristics for the new ALP class of algorithms and how they compare to the landmark selection methods created for ALT.

Goldberg and Harrelson's original work provided three contributions. First, their main contribution was a preprocessing technique for computing distance bounds that depends on identifying a carefully chosen, relatively small (in comparison to |*V*|) number of vertices, called *landmarks*, in a graph. Second, they provided the first exact shortest path preprocessing algorithm for arbitrary graphs (no restricted graph classes). And finally, they tested this algorithm in an experimental study comparing new and previously known algorithms both on synthetic graphs and on real-world road graphs.

In ALT, a PPSP query uses computed distance estimate, derived from the triangle

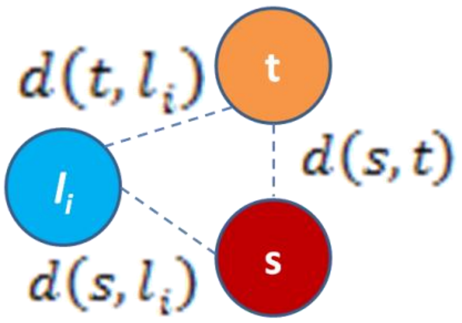


Figure 6 Illustration of distance information for three vertices not necessarily incident to each other in a graph

inequality, to guide the search. Using the distances illustrated in Figure 6 for a graph $G = \{V, E\}$ this inequality yields two important equations for any three vertices $s, t, l_i \in V$:

$$d(s,t) \leq d(s,l_i) + d(l_i,t) \tag{18}$$

$$d(s,t) \geq |d(s,l_i) - d(l_i,t)| \tag{19}$$

Let $\boldsymbol{L} \subseteq V$ be the set of landmarks with distance $d\ (v, l_i)$ stored for all vertices $v \in V$ and any landmark $l_i \in \boldsymbol{L}$, $1 \leq i \leq |\boldsymbol{L}|$. Due to the triangle inequality, the following equation holds for vertices $s,t \in V$:

$$d(s,t) \geq |d(s,l_i) - d(t,l_i)| \tag{20}$$

Based on the above arguments, the ALT algorithm works as follows: In a preprocessing step, the Dijkstra's SPT algorithm is used to compute and store the distances to each landmark in $\boldsymbol{L}$ from all other vertices in V. Then, during PPSP queries, the triangle inequality is used as follows: let $\pi_t^{\boldsymbol{L}}\ (v)$ be the heuristic function based on landmarks that will be used for the A* algorithm seen in Figure 4. Then the following equation represents the heuristic function when visiting vertex $v \in V$ on the way to a target vertex t:

$$\pi_t^{\boldsymbol{L}}\ (v) = max_{l_i \epsilon \boldsymbol{L}}\ \{|d(v,l_i) - d(t,l_i)|\} \tag{21}$$

Recall that a dominating heuristic function for A* yields a larger estimate than other heuristics without overestimating distance. For this reason, in ALT, to compute the best estimate, the maximum triangle inequality estimate is taken over all landmarks. Using this heuristic for A* tailors the bounds to the graph being analyzed, greatly reducing the search space, along with memory requirements and processing time. The proof that $\pi_t^{\boldsymbol{L}}$ is an admissible heuristic for the shortest path between two vertices *s,t* ϵ *V* follows:

Proof. Let *P*(*s*,*t*) be a shortest *s*-*t* path. For any *vᵢ* ϵ *V*, *i*≤1<*k*, $d(v_{i+1}) - \pi_t^{\boldsymbol{L}}\,(v_i) + \pi_t^{\boldsymbol{L}}\,(v_{i+1}) \geq 0$. Therefore, $\sum_{i=0}^{k-2} d(v_i, v_{i+1}) \geq \sum_{i=0}^{k-1} [\pi_t^{\boldsymbol{L}}\,(v_i) - \pi_t^{\boldsymbol{L}}\,(v_{i+1})]$. Because of this, $d(s,t) \geq \pi_t^{\boldsymbol{L}}(s) - \pi_t^{\boldsymbol{L}}(t) = \pi_t^{\boldsymbol{L}}(s)$ (Bauer, Columbus, Katz, Krug, & Wagner, 2010).

The runtime of ALT's preprocessing, not including the actual selection of l landmarks, is $O(l \cdot (|E| + |V| \log|V|))$, as a breadth-first search is performed from each landmark to form each SPT. Because an SPT is computed from every chosen landmark, ALT's data structure requires $O(l \cdot |V|)$ space. Since $l \leq |V|$, the theoretical space requirement for ALT is $O(|V|^2)$. This quadratic space requirement means that the preprocessing algorithm does not scale well in terms of memory. As a dataset (or more specifically, its number of vertices) grows, the number of chosen landmarks must be increased in order to maintain an appropriate distribution of distances.

The ALT algorithm's preprocessing technique is faster than other preprocessing techniques for shortest path search, due to the fact that it only performs one shortest path search from each landmark to create each SPT. In experimentation on large European roadmap datasets ($\sim 6.7 \times 10^6$ nodes), it was shown that preprocessing only 16 landmarks can lead to a speedup factor of nearly 50 using the bidirectional implementation of A*(Jens Maue, 2006). However, identifying the set of landmarks that optimizes overall performance during preprocessing and querying on any graph is an NP-hard problem known as MINALT(Bauer et al., 2010).

For sparse graphs, a larger number of landmarks are also required by ALT to be effective. Storing distance information for each landmark is quite space intensive, as an individual measurement of distance must be kept for each node-landmark pair. Therefore, the ALT algorithms lack the ability to maintain reasonable space complexity while achieving efficient speedup for sparse graphs.

Increasing the number of landmarks or the size of the graph can present another drawback to the ALT approach. Note that, for ALT, as each vertex is visited for A*, $\pi_t^L\ (v)$ must be computed, such that for *l* landmarks, *l* subtraction operations need to occur along with a *max* operation (of time complexity *O*(*l*)). For a large enough *l* or for long enough paths, performing this many operations for every visit to a vertex in the graph can drastically slow down a query's actual runtime. In some cases, this will result in Dijkstra's algorithm (A* with a 0 heuristic) outperforming A* with the ALT heuristic. This dissertation advocates that the number of visited vertices cannot be the only reliable measure of the effectiveness when defining a new heuristic function for A*. Future research must measure the actual number of operations that occur during queries and not simply the size of the search space to clarify an algorithm's behavior.

Precomputed Cluster Distances (J Maue, Sanders, Matijevic, Alvarez, & Serna, 2006)

The precomputed cluster distances (PCD) algorithm was designed with the intention of reducing the space requirements of metric-independent preprocessing algorithms such as ALT. PCD uses the distances between graph clusters to inform the heuristic for A* (Jens Maue et al., 2010). The preprocessing step of the PCD algorithm assumes that the graph has been partitioned into *k* clusters that will be used in the query process to maintain an upper bound, where *k* is predetermined. This preprocessing method is metric-independent, as clustering is seen as a part of topology input. Also, the algorithm operates in the same manner regardless of the type of clustering and, in practical cases, this clustering is done ad-hoc by quickly splitting the graph into cells. These ad-hoc methods are much faster than more accurate methods as the Louvain algorithm (Blondel et al., 2008).

To begin PCD preprocessing, the minimum distance between each pair of clusters is computed by connecting, with zero weight, a single vertex to all *border vertices* of a cluster and computing the shortest path from that "single source". A border vertex is a vertex with an adjacent vertex that is in another cluster *C*. Border vertices realize the shortest distance to other clusters in the graph. These cluster distances are then used to advise A* during query time. Only k^2 shortest paths are calculated with this approach and only k^2 distances are then stored. The impact of this preprocessing step is dependent on the structure, size, and number of clusters that the graph is partitioned on. But with adequate parameters, the algorithm is flexible enough to allow many different types of clustering.

PCD's preprocessing method is significant as it experimentally provides greater speedup than the ALT algorithm and achieves drastically reduced space complexity. The PCD algorithm only computes and stores distance information for border nodes of partitions of the graph. Therefore, the algorithm benefits from a significant reduction in preprocessing time and required memory.

The querying step for PCD is a modification of a bidirectional version of Dijkstra's algorithm. This means that the lower and upper bounds that need to be updated are computed differently based on the iteration of the search. From the start vertex and end vertex, lower bounds for the length of any path from source to target containing a settled vertex in an intermediate cluster are repeatedly estimated. Let **C** be the set of clusters in a graph G. The shortest path between two clusters P,Q is

$$d(P,Q) = min\{p \in P, q \in Q \mid d(p,q)\} \quad (22)$$

For an intermediate vertex, $u \in V$, being settled, the lower bounds of the shortest path between vertices s,$t \in V$ can be estimated to be

$$d(s,t) \geq d(s,u) + d(U,T) + d(t,t^{'}) \quad (23)$$

$$d(s,t) \geq d(s,s^{'}) + d(S,U) + d(t,u) \quad (24)$$

where S, T, U are clusters that respectively contain s,t,$u \in V$, and cluster border vertices s',$t' \in V$.

The upper bound is also updated at every iteration of the search. The settled vertex gets pruned if the path from the source to destination using it is greater than the maintained upper bound. For clusters $\sigma, \tau, \varphi \in G$ and source-target pair $s \in \sigma$, and $t \in \tau$, let $s_{\varphi\tau} \in \varphi$, $t_{\varphi\tau} \in \tau$ represent the source-target pair for the shortest path from cluster φ to τ. This target pair is denoted $\langle s_{\varphi\tau}, t_{\varphi\tau} \rangle$. Also, let $s_{\sigma\varphi} \in \varphi, t_{\sigma\varphi} \in \varphi$ represent the source-target pair for the shortest path from cluster φ to σ, denoted $\langle s_{\sigma\varphi}, t_{\sigma\varphi} \rangle$. The upper bound is initialized as the sum of the diameters of the source and target clusters and the precomputed distance between their clusters using one of the following equations:

$$d(s,t) \leq d(s, s_{\varphi\tau}) + d(\varphi,\tau) + d(t_{\varphi\tau}, t) \quad (25)$$

$$d(s,t) \leq d(s, s_{\sigma\varphi}) + d(\sigma,\varphi) + d(t_{\sigma\varphi}, t) \quad (26)$$

$$d(s,t) \leq d(s, s_{\varphi\tau}) + d(\varphi,\tau) + 2r(\tau) \quad (27)$$

$$d(s,t) \leq 2r(\sigma) + d(\sigma,\varphi) + d(t_{\sigma\varphi}, t) \quad (28)$$

where, for a cluster $A \in G$, $r(A)$ denotes the *radius* of the cluster. Each of these equations hold for the upper bound of $P(s,t)$. The upper bound is then maintained with one of these equations based on the upper bound and whether or not $s_{\varphi\tau}$, $t_{\varphi\tau}$, $s_{\sigma\varphi}$, or $t_{\sigma\varphi}$is settled.

Attempting to set bounds on a search space to prune the space has been a common technique for speeding up shortest path queries. Often, however, many algorithms require a significant amount of storage, inherently rendering them not scalable for larger datasets (Lauther, 2004; Jagan Sankaranarayanan, Samet, & Alborzi, 2009; Wagner, Willhalm, & Zaroliagis, 2005). The previously discussed ALT algorithm maintains a space complexity of $O(|V|^2)$ for l landmarks. The ALT algorithm was also cited by PCD's authors as a key reason for developing their own space-efficient algorithm.

PCD's chief benefit is that while, in practice, it requires more preprocessing than landmarks, it achieves PPSP speedups through far more space-efficient means. In Maue's work, when comparing the amount of space required by PCD to ALT, he notes that the space complexity for PCD is $\theta(k^2 + B)$ compared to ALT's $\theta(l \cdot |V|)$, where B is equal to the number of border nodes for clusters. However, since the actual clustering information is stored, as well, the space complexity is actually $\theta(k^2 + B + |V|)$, as information about which cluster every vertex belongs to needs to be referenced. In Maue's experiment, the landmark method also had an experimental average speedup to normal PPSP less than that of PCD (Jens Maue et al., 2010) and a higher preprocessing time complexity. However, as shown later in the methodology for ALP, the space requirement for landmarks can be significantly reduced while benefiting from a sufficient performance increase. PCD will be a key algorithm to compare ALP against when using speed as a metric.

Note also that the clustering takes place before preprocessing, meaning that the algorithm itself ignores the type of clusters when computing distances. Clustering information is presumed to be input parameters, limiting the application of this algorithm. The downside to this algorithm is that the complexity benefits are only gained if the clusters inherently come with the topology information or are quickly computed. This is computationally intensive and is optimal only for graphs that have the proper structure for clustering, such as small-world or scale-free graphs. The fastest known algorithms for graph clustering rely on modularity optimization, another NP-hard problem, and run experimentally in $O(|V| \log |V|)$ (Blondel et al., 2008).

The key issue here is that data that can be overlaid onto a graph does not necessarily cluster or partition well. This can have a significant impact on the PCD algorithm. Optimal clustering (with maximum modularity) can sometimes result in clusters that are extremely small, which could potentially require PCD's preprocessing algorithm to store nearly as much information as ALT preprocessing. In such cases, while the space benefit is still clearly better, the performance benefit of PCD over ALT for a high number of clusters has not been tested.

Reach-Based Routing (Goldberg et al., 2009; Gutman, 2004)

Reach-based pruning is another method for speeding up shortest-path queries such as Dijkstra's algorithm. *Reach* is a centrality measure that identifies how central a vertex is on a shortest path (Gutman, 2004). The reach of a vertex $v \in V$ is larger when v is closer to the middle of a shortest path and smaller otherwise. Based on this measure, the algorithm was created to deal with large-scale graphs, which inherently contain shortest paths that are larger in size.

Let the reach of a node $v \in V$ be denoted as $R_P(v)$ for shortest path P. For a reach metric m and a path P, let $m(P)$ represent the sum of $m(e)$ over all edges e of P (or zero for $|P| = 1$). Then for two nodes $u,v \in V$, $m(u,v,P)$ represents $m(Q)$ where Q is the subpath in P from u to v. Formally, for path $P(s,t)$ and graph G,

$$R_P(v) = \min\{m(s,v,P), m(v,t,P)\} \tag{29}$$

$$R_G(v) = \max_{\{P \in SP \mid v \in P\}} R_P(v) \tag{30}$$

where $R_G(v)$ is the reach of v in G, SP the set of all shortest paths in G, and $P \in SP \mid v \in P$ represents any shortest path in G containing v.

For the purposes of creating a feasible algorithm, computing exact reaches for all elements in a graph is not scalable. Therefore, an upper bounds for $R_G(v)$, denoted as $\overline{R_G}(v)$, is computed instead. Let $\underline{d}(P)$ be the lower bound of $d(P)$. If, for a source-target pair $s,t \in V$, $\overline{R_G}(v) < \underline{d}(P(s,v))$ and $\overline{R_G}(v) < \underline{d}(P(v,t))$, then v is not on a shortest path from s to t. Therefore, reach-based pruning for shortest path search occurs as follows. During a run of Dijkstra's algorithm (seen in Figure 3), before inserting a vertex $v \in V$ into the priority queue, a test is run on the reach values for v. Vertex v is inserted into the priority queue if

$$R_G(v) \geq R_P(v) \tag{31}$$

Otherwise, the vertex is not considered to be on the shortest path. These reach upper bounds are computed during the preprocessing phase. Lower bounds are iteratively computed. The bidirectional variant is achieved by setting implicit bounds in both directions. Note that in the bidirectional variant, searching between $s,t \in V$ by way of vertex $v \in V$ the goal is to identify $d(s,v)$, $d(v,t)$, $P(s,v)$, and $P(v,t)$. With this in mind, $R_P(v)$ is likely to be high, making v a high-reach vertex. This bidirectional variant is often used to optimize the speedup.

In practice, the reach measure along with reach-based pruning is combined with other approaches such as contraction hierarchies (Geisberger, Sanders, Schultes, & Delling, 2008b) or ALT (Goldberg et al., 2009). In this research, the combination of reach-based pruning and ALT, known as REAL, is studied. REAL is a partial landmark algorithm which stores landmark distances for all vertices with high reach, set by establishing a reach threshold *R*. A query begins by running normal bidirectional Dijkstra's (or A* with no heuristic) with normal reach-based pruning. Bidirectional Dijkstra's continues until either the algorithm terminates or the search frontiers, both forward and backward, have crossed into the region of vertices with reach *R* or higher.

Once the search radii of the front and backward searches have crossed the threshold, the algorithm then uses ALT to accomplish the remainder of its task. The way that the remainder of the path is found in forward search is symmetrical to the way it is found in backward search in the following description. For identifying $P(s,t)$, suppose that s has low reach. Denote s' as the *proxy*, or highest reach vertex closest to s. The vertex s' is computed either during preprocessing or by a multiple-source version of Dijkstra's algorithm. Then store the length of the shortest path between s' and s, $d(s',s)$. The lower bound for the vertex where both search frontiers meet is computed using the precomputed landmark distances. For a landmark *L*, the lower bound on $d(s,v)$ using distances to *L* is specified by

$$d(s,v) \geq |d(s',L) - d(v,L) - d(s',s)| \tag{32}$$

The lower bounds from target *t* are computed in the same way. This algorithm's performance is strongly dependent on the quality of the lower bound. This bound is determined by both the number of landmarks and the reach threshold. For too high of a threshold, the lower bounds will be inaccurate. The number of landmarks and landmark selection vary the performance of the algorithm in the same manner that they do in regular ALT.

Other Preprocessing Algorithms

Maue's PCD algorithm demonstrated practical performance benefits over both the Arc Flags (M et al., 2007) and Geometric Containers (Wagner et al., 2005) preprocessing algorithms. The Geometric Containers algorithm relies on the concept of *edge labeling*, where preprocessing attaches a label to each edge in a graph that represents all nodes to which a shortest path starts with the individual edge. Specifically, a geometric object, known as a *container*, is created that contains at least the edges within a given graph region. PPSP queries are then answered by Dijkstra's algorithm as restricted to the edges that lie inside a container. While geometric containers algorithms maintain only a linear space requirement, the preprocessing step requires a single source shortest path search from every node, making it impractical for large-scale graphs.

For Arc Flags algorithms, an input graph is partitioned such that a flag is computed for each edge within a partition, or region, which indicates whether the edge is on a shortest path to any node in that partition. It is similar to the Geometric Containers algorithm in that it considers only the edges whose flag correspond to a specific region.

This algorithm still realizes a high preprocessing time, as one shortest path search from every border node of a region is required.

Finally, it has been noted, from experimentation, that landmark methods such as ALT begin to drastically underestimate the shortest path when approximating short distances (relative to the size of the graph) (Maruhashi, Shigezumi, Yugami, & Faloutsos, 2012). For this reason, *EigenSP* uses eigenvalues and eigenvectors to directly compute distance. The eigenvalues and eigenvectors of a graph adjacency matrix can indicate path capacity between any two vertices in an undirected, connected graph (Harary & Schwenk, 1979)**.** The adjacency matrix *A* for an undirected, connected graph *G* is a symmetric matrix with real eigenvalues. This means that *A* is a *Hermitian matrix*. Because of this, the eigenvalues and eigenvectors for *A* can be used to count the number of paths between an arbitrary pair $s, t \in V$. Note, from applied mathematics, $A = X\Lambda X^T$, where Λ is the diagonal matrix for the eigenvalues of *A* and *X* is an orthonormal matrix containing its eigenvectors as columns. Then, from the orthonormality of *X*, for $k \in \mathbb{Z}^+$:

$$A^k = X\Lambda^k X^T \tag{33}$$

From spectral graph theory, the elements of A^k represent the number of paths of length *k*. Specifically, an element *e* in the *i*th row and *j*th column of matrix A^k represents the number of paths from vertex *i* to *j* in *G*. If there is no path of length *k* from vertex *i* to *j* in A^k, *e* = 0. Therefore, for source and target vertices *s* and *t*, the eigenvectors and eigenvalues of a graph's adjacency matrix are related to their shortest path length by the following equation:

$$d(s,t) = min\left\{k| \sum_{r=1}^{n} x_{rs}\lambda_r^k x_{rt} > 0, k \in \mathbb{Z}^+\right\} \tag{34}$$

where x_{rs} is the sth entry of the rth eigenvector, λ_r^k is the rth eigenvalue of the adjacency matrix and n is the number of orthogonal eigenvectors.

At query time, EigenSP tests a series of values for k to respond to a query. To speed up PPSP queries, a set of eigenvectors and corresponding eigenvalues are precomputed. While this leads to extremely fast PPSP queries, this method of precomputation does not scale well. Even when using some of the most efficient algorithms for computing eigensystems (Cullum & Willoughby, 2002), it is simply infeasible to rely on the number of computations to calculate $d(s,t)$ directly for large-scale practical implementations. However, as in the Geometric Containers or Arc Flags algorithms, if a smaller region *R* of the graph can be extracted such that the shortest path from any vertex in *R* to any other vertex in *R* only traverses edges within *R*, then EigenSP can be simply run on the subgraph for *R*. This is a potential area of future research.

Landmark Selection Algorithms

Landmark selection is crucial to the performance of ALT and ALP algorithms. In this section, the most common landmark techniques for ALT are reviewed. Identifying the particular set of vertices to select as landmarks such that the expected number of settled vertices for shortest path queries is minimal, or what is known as the MINALT problem, is NP-Hard (Bauer et al., 2010). Comparing, contrasting, and understanding the fundamental reasons behind modern landmark selection techniques is critical in identifying new ones for the ALP class of algorithms. The algorithms that work well under the ALT paradigm do not necessarily work well under the ALP paradigm when multiple landmarks are used. Studying the development process of these algorithms also suggests methods of creating new ones for ALP. The study of the behavior of these landmark selection algorithms in ALP, modification of their parameters, and the development of any new ones are the main focus of this dissertation.

Search Space

In terms of pathfinding, the *search space* is the feasible region of solutions for a given query. For a set of landmarks $\boldsymbol{L}$, the search space, $V_L(s,t)$ (Bauer et al., 2010), of an ALT query can be explicitly defined as follows:

$$V_L(s,t) = \{v \in V | d(s,v) + \pi_t^{\boldsymbol{L}}(v) \leq d(s,t) \wedge v < t\} \tag{35}$$

In this space, $v < t$, denotes that the search space expands until the target *t* is reached. For ALT, this definition implies that there are no vertices outside of this search space for $v < t$ that satisfy $d(s,v) + \pi_t^{\boldsymbol{L}}(v) \leq d(s,t)$. Overall, this definition shows that, for any given set of landmarks, the search space for ALT only takes into account paths that are less than or equal to the distance between *s* and *t*. If landmarks are chosen strategically, the number of vertices in this search space can decrease, inherently reducing the search time. Using this definition, the MINALT problem is explicitly defined as follows:

Problem: $MinALT(G,k) = \ arg\ min_L \ \sum_{s,t \in V} |V_L(s,t)|$

In other words, the MINALT problem is the problem of identifying the set of landmarks that minimizes the summation of all search spaces for any two vertices $s, t \in V$. In general, increasing the number of landmarks *k* improves the speedup performance of ALT search. The *optimal* solution to this problem, however, minimizes the preprocessing time, preprocessing space complexity, and average query time. Identifying the solution to this problem is NP-hard. This has been shown by a polynomial time reduction to the MAXCOVER problem (Fuchs, 2010). Typically, an optimization method is used to get close to a good solution for MINALT. These landmark selection techniques, also known as *embedding methods*, typically fall into three categories: *global*, *local*, and *distance-based* (Sommer, 2012). Global techniques rely on the classic paradigm of using the entire graph for landmarks, having each landmark relate to all vertices in the graph. Local techniques require a vertex $v \in V$ to compute path information only to certain landmarks, usually only recording the shortest path between v and a subset of the landmarks. In these cases, the nearest landmarks to v are typically the ones that have information stored. Finally, distance-based methods vary in the distance information that is stored, many times storing information about different subsets of the graph.

Basic Methods

The first proposed landmark selection algorithm and perhaps the most intuitive is *random* landmark selection (Goldberg & Harrelson, 2005). Based on the number of vertices in the graph, *k* vertices are chosen at random to serve as landmarks. A series of sample queries are run with each landmark to determine the best set. This is a brute force method of performing landmark selection for ALT. However, in terms of lower bounds, random landmarks demonstrate better performance than any of the following methods of landmark selection (Potamias et al., 2009).

Goldberg & Harrelson immediately recognized this as a flawed, brute-force method of choosing landmarks and proposed *farthest landmark selection* (Goldberg & Harrelson, 2005). The algorithm works as follows: Identify a start vertex $v \in V$ and find the vertex $v' \in V$ farthest, in terms of path weight, away from it. Add v' to the set of landmarks. Then, proceed in iteration by finding the next vertex v''farthest away from the current set of landmarks and adding v'' to the set. The next vertex that is farthest away maximizes the distance to the closest vertex in the set. Continue until k landmarks have been identified.

Also initially proposed was *planar landmark selection* (Goldberg & Harrelson, 2005). This landmark selection algorithm uses graph layout information to divide a graph into sectors. The vertices of the graph are all given polar coordinates. Based on these coordinates, a point is placed in the middle of the graph and the sectors are created. For each sector, the farthest point is selected to be a landmark. If two points for different sectors happen to be on the border of their respective sector and adjacent to each other, one of them is removed.

A later version of farthest landmark selection was introduced that computed farthest based on path distance instead of path weight, meaning that the cost of moving from vertex to vertex is 1 (A. V. Goldberg & R. F. Werneck, 2005). This will be denoted here as *farthest-d selection*. This biases farthest selection to choose separate, dense regions of the graph to place landmarks in. While the selection algorithm takes a smaller amount of time than most, there are still better methods of identifying more optimal landmarks.

Avoid landmark selection, a commonly used and modified landmark selection algorithm, begins by computing the SPT T_r, rooted at some arbitrary vertex $r \in V$ (A. V. Goldberg & R. F. Werneck, 2005). Often, r is chosen at random. For Avoid, the term *weight* is defined differently and will be denoted here as *A-weight*. For a set of landmarks

L, the A-weight of a vertex $v \in V$ is the difference between its distance $d(r,v)$ and the lower bound of $d(r,v)$ as computed in the ALT algorithm. Let T_v be a subtree of T_r. For every $v \in V$, the *size* $s(v)$, or the sum of the weights of all vertices in T_v, is computed. If *w* is the vertex with the maximum size, T_w is traversed, following the child with the largest size until a leaf is reached. The first leaf that is reached is a new landmark. This approach "avoids" existing landmarks to improve coverage of landmarks over the graph.

Advanced Landmark Selection Algorithms

In the previous section, we detailed some very basic embedding methods for estimating the shortest path using the ALT algorithms. The following algorithms perform more in-depth graph analysis to strategically select landmarks.

Betweenness Centrality Embedding (Potamias et al., 2009)

One of the first advanced landmark selection algorithms that has shown promise is based on the betweenness centrality of landmarks. Such mining of the graph before selecting landmarks has proven to be several orders of magnitude faster than current methods.

The basic principle behind using betweenness centrality as a guide for landmark selection stems from the following observations:

Observation 1: Let a landmark node *l* exist on the shortest path between two nodes *s* and *t*. Then $d(s,t) = d(s,l) + d(l,t)$.

Observation 2: Let a node *s* exist on the shortest path between two nodes *l* and *t* or let *t* exist on the shortest path between nodes *s* and *l*. Then $d(s,t) = |d(s,l) - d(l,t)|$

Based on these observations, this work attempts to solve a problem that is similar to the MINALT problem. It proposes the *LANDMARKS$_d$* problem, which attempts to cover all (or most) shortest path pairs in the graph by ensuring there are landmarks between them.

Problem *LANDMARKS$_d$(G, k)*: Is there a set of landmarks $L \subseteq V$ of size at most *k* such that the number of pairs of vertices $(u, v) \in V \times V$ covered by L is maximized?

A landmark *covers* a pair of vertices (u, v) if there exists at least one landmark in L that lies on the shortest path from u to v. If a chosen landmark lies on the path between two nodes u and v, then the shortest path distance is simply the upper bounds of the triangle inequality for that landmark. In other words, for a given landmark-source-target set $(l, s, t) \in V,\ d(s, t) = \ d(s, l) + \ d(l, t).$ This allows the upper bound of the triangle inequality to be the answer to the shortest path problem. It follows, then that the optimal landmarks for the *LANDMARKS$_d$* problem are the ones with maximum betweenness centrality in the graph. The *LANDMARKS$_d$* problem is demonstrated to be NP-hard by proving that *LANDMARKS-COVER* is NP-hard. *LANDMARKS-COVER* is proven to be NP-hard because there exists a polynomial-time transformation to it from the NP-hard VERTEX-COVER problem.

Problem *LANDMARKS-COVER(G,k)*: Is there a number of landmarks $L \subseteq V$ of size at most k such that all pairs of vertices $(u, v) \in V \times V$ are covered?

Problem *VERTEX-COVER(G,k)*: Is there a *vertex cover*, or set of vertices such that each edge of the graph is incident to at least one vertex of the set, of size at most *k* in *G*?

For a vertex $v \in V$, let $\sigma_{st}(v)$ denote the number of paths from s to t containing v. Also, let $|P(s, t)|$ simply denote the total number of paths from s to t. Then betweenness centrality of *v* is formally defined as

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{|P(s,t)|} \tag{36}$$

For landmark selections, the optimal landmarks are those with highest betweenness centrality (Potamias et al., 2009). However, series of nodes with high betweenness centrality will be clumped together in the graph, reducing their utility. Therefore, two other metrics that are taken into account are degree and closeness centrality. To select nodes based on degree, the nodes of the graph are simply sorted from lowest to highest degree and the highest degree nodes are chosen. Also, choosing a node with the lowest *closeness centrality* has demonstrated utility. For a source-target pair $s, t \in V$, closeness centrality C_C of a vertex $u \in V$ is defined as

$$\mathrm{C_C}(u) = \frac{1}{|V|} \sum_{v \in V} \mathrm{d}(s,t) \tag{37}$$

Choosing the k vertices with lowest closeness centrality is the common convention. However, both the closeness centrality and the betweenness centrality are very difficult to compute in large scale graphs. Therefore, partitioning the graph into sections and identifying nodes with the highest betweenness centrality, lowest closeness centrality, or degree produce the most optimal results. In a series of experiments, the centrality measures proved to be far more robust than the degree measures, primarily because centrality measures produce results more indicative of the path structure than simple degree measures.

Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme (Miao, 2014)

Recent work has considered the differences between globally selected, query-independent landmark selection and local, query dependent methods. The global methods discussed inherently incur a larger relative error (underestimates), particularly for close nodes, than local ones. By establishing tighter bounds, the search space is inherently narrowed. By identifying a query-dependent local landmark, the search no longer falls prey to a global setting that could be less than optimal for local queries. This dissertation effort will propose, implement, and test a hybrid, query-independent approach to landmark selection for the ALP class of algorithms. For breadth, this work in query-dependent, local embedding is reviewed.

A notional example can be made from the graph in Figure 7. Based on the given *global landmark* l_1 to the right of the graph, if we were to estimate the distance between *a* and *b* using ALT, the following would result:

$$d(a,b) \leq d(a,l_1) + \ d(l_1,b) \tag{38}$$

However, a more accurate estimate could be made from node *c*, which is much closer to *a* and b. This would result in the following estimation:

$$d(a,b) \leq d(a,c) + \ d(c,b) \tag{39}$$

This estimation is clearly tighter, therefore narrowing the search space. Node c is then referred to as a *local landmark*.

Identifying such local landmarks demonstrates a benefit by narrowing the search space. However, the method for actually identifying these landmarks is not intuitive. Recall that once landmark nodes have been selected, for a given landmark l_i, ALT identifies the shortest path between l_i and every other node in the graph by performing a breadth-first search that spawns an SPT. By preserving this SPT structure, one can identify, at query-time, the *least common ancestor*, or *LCA*, between a source and target node pair as a local landmark. The LCA of two nodes $s,t \in V$ in an SPT is the node

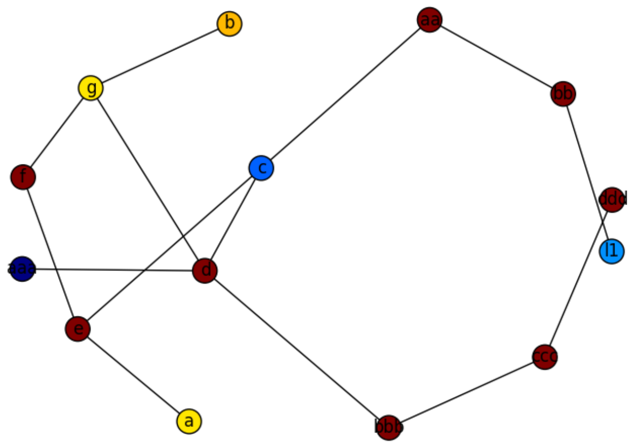


Figure 7 Local Landmarks Example

furthest from the root that is an ancestor of both *s* and *t*. In the example in Figure 7, node *c* was the LCA. Unless the global landmark is the only common ancestor, the LCA will always be closer to the two query nodes than the global landmark, therefore reducing the search space.

Storing information in this SPT-based local landmark scheme can incur serious space complexity costs. Three key pieces of information are stored for this algorithm:

1. Embedded Distances: Basic ALT requires $O(l \cdot n)$ space to record the distance between landmarks and all other nodes of the graph.
2. Shortest path trees: Each shortest path tree requires $O(n)$ space. Also, arrays that are used to quickly calculate the LCA for larger SPTs require $O(n)$ space. The theoretical space complexity for SPTs and these arrays is also $O(l \cdot n)$.
3. Range Minimum Query Index Tables: Tables used to efficiently identify least common ancestors. Also requires $O(l \cdot n)$ space.

Further optimizations are made for this algorithm to enhance performance using lossless graph compression to limit the amount of space required by landmarks and local search algorithms to further narrow the search space. The theoretical space requirements led to massive practical requirements when tested on real data. While the actual search did not use all the data in memory, each of the separate structures necessary for the algorithm to be executed required being loaded into memory. Therefore, while certainly increasing the overall time complexity of the ALT algorithm with a new and innovative method of identifying landmarks at query-time, this algorithm sacrifices large amounts of memory to be carried out on large datasets.

Chapter 3

Methodology

Overview

The fundamental problem that this dissertation addresses is the optimization of landmark selection for the A*, landmarks, and polygon inequalities (ALP) class of algorithms. In Chapter 2, the ALT methodology for estimating shortest path distances for A* was described, along with the most modern landmark selection techniques that attempt to optimize the algorithm's speedup ratio and comparable shortest path preprocessing algorithms. Further, other metric-independent shortest path preprocessing algorithms were highlighted. In this chapter, we demonstrate that using multiple landmark vertices to guide A* search grants the ability to perform less computations at both preprocessing and query time. Using a process dubbed *distributed embedding*, we demonstrate that ALP has a significantly smaller space requirement in comparison to ALT and can provide better landmark selection. It is also noted, in this chapter, that the base heuristic for ALP, using a single landmark, has already been verified and validated as the ALT algorithm. To begin to characterize ALP's behavior when using multiple landmarks, the approach in this effort sought to use two landmarks to guide the search query.

In this chapter, the methodology for the dissertation is presented in its entirety. The Methodology chapter provides the framework that guided the design and implementation of a shortest path software library that includes the ALP dual landmark capability. The design for the dissertation's experiments, along with their corresponding metrics are described to further demonstrate that domination of one heuristic over the other depends on the landmark set each is assigned and, in general, the denser the landmark set, the better the heuristic. The methodology details five specific concepts: (1) mathematical detail of the lower bounds that are created by the use of two landmark vertices in the graph as reference points; (2) further theoretical specification of the use of two landmarks in distributed embedding; (3) theoretical specification of ALT landmark selection techniques in the ALP environment; (4) new landmark selection techniques that apply to the characteristics of the ALP environment; and (5) description of the experimentation and measurements required to fully characterize the ALP algorithm.

A key goal in developing this methodology was to establish the design of the software experimentation framework that allowed for rapid updating of landmark selection technique and heuristic function implementations, trivial collection of metrics, and extraction of details about the data operating environment (i.e., graph structure and characterization of shortest path queries). The Research Methods section details the algorithms that were used to characterize ALP and its landmark selection techniques. The Validation and Verification section contains a high-level explanation of the ALP software library and dissertation experiments. Finally, the Summary recapitulates the scope of the complete effort and maps the methodology to the overall contributions of the effort.

Research Methods

Quadrilateral Properties in Graphs

Previous implementations of embedding methods compute shortest path trees (SPTs) that cover the entire graph from a selected set of landmarks and use the triangle inequality at query time to establish a lower bound for A* (Goldberg & Harrelson, 2005). The use of this geometric inequality can be expanded to allow for more lower bounds to be derived. Such bounds are derived by forming other types of polygons, of higher order than triangles, in the graph. Using quadrilaterals, we explain how these heuristics can be derived by identifying any polygon in a graph and setting the heuristic values for A* equal to the maximum derived lower bound of one side of the polygon. The development of the ALT algorithm provides a base case for such a hypothesis. The use of two landmarks, as seen in this dissertation, provides an inductive step for the proof of the hypothesis. We begin with a description of how to form a triangle in a graph to establish the triangle inequality as a lower bound. This proof was derived from the reverse triangle inequality proof for a metric space, detailed in Chapter 1.

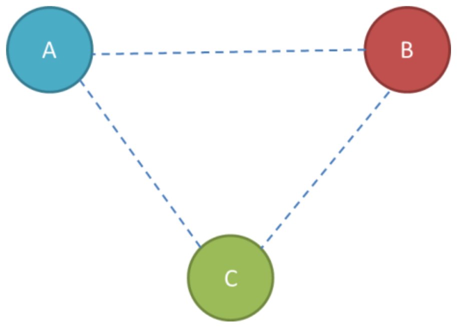


Figure 8 Three vertices within a sample connected graph. The dotted lines represent shortest paths between each of the vertices

Shown in Figure 8, for a connected graph *G*[1], containing vertices $A, B, C \in V$, the shortest path distances between each vertex form a metric space. If *G* is undirected, for the distances between vertices A, B, C, the following triangle inequalities hold:

$$d(A,B) \leq d(C,A) + d(B,C) \qquad (40)$$

$$d(B,C) \leq d(A,B) + d(C,A) \qquad (41)$$

[1] Recall from Chapter 1 that we are addressing graphs that are either directed or undirected. If directed, they are strongly connected.

Both of these inequalities apply to the three vertices in *G*. The reverse triangle inequality, which is used as a lower bound for A* in ALT, is derived from these inequalities as shown in Table 1.

#	Statements		Reasons
1.	$d(A,B) \leq d(C,A) + d(B,C)$	$d(B,C) \leq d(A,B) + d(C,A)$	Triangle Inequality
2.	$d(A,B) - d(B,C) \leq d(C,A)$	$d(B,C) - d(A,B) \leq d(C,A)$	Subtraction on both sides (#1)
3.	$\boldsymbol{\lvert d(A,B) - d(B,C)\rvert \leq d(C,A)}$		**Absolute Value Definition (#2)**

Table 1 Derivation of the Reverse Triangle Inequality in Simple, Connected Graphs

ALT uses this reverse triangle inequality to create a heuristic that estimates the distance between vertices *C* and *A* by setting vertex *B* equal to a landmark *l* such that

$$|d(A,l) - d(l,C)| \leq d(C,A) \tag{42}$$

By computing and storing the values $d(A,l)$ and $d(l,C)$ before performing any PPSP queries, this lower bound is then used as a heuristic to the A* algorithm. Because it is the lower bound, it will never overestimate the distance between vertices *A* and *C*.

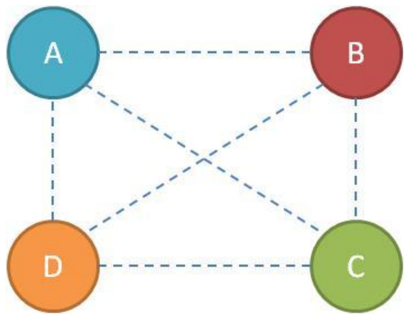


Figure 9 Four vertices within a sample connected graph. The dotted lines represent shortest paths between each of the vertices

For a quadrilateral, the lower bound of one of its sides can also be calculated using the other three sides. This reverse quadrilateral inequality can also be used to establish the lower bounds for the shortest path of a graph. Illustrated in Figure 9, for a graph *G* with vertices $A, B, C, D \in V$, the lower bound can be derived from the following system of inequalities for quadrilaterals:

$$d(B,A) \leq d(C,B) + d(D,C) + d(A,D) \tag{43}$$

$$d(C,B) \leq d(B,A) + d(D,C) + d(A,D) \tag{44}$$

$$d(D,C) \leq d(B,A) + d(C,B) + d(A,D) \tag{45}$$

Similar to the triangle inequality for Figure 8, a set of inequalities describe the lower bounds for distances between vertices of the graph represented in Figure 9. Shown in Table 2, the reverse quadrilateral inequality is derived in a manner similar to that of the reverse triangle inequality.

#	Statements			Reasons
	A	**B**	**C**	
1.	$d(B,A) \leq d(C,B) + d(D,C) + d(A,D)$	$d(C,B) \leq d(B,A) + d(D,C) + d(A,D)$	$d(D,C) \leq d(B,A) + d(C,B) + d(A,D)$	Quadrilateral Inequality (Given)
2.	$d(B,A) - d(C,B) - d(D,C) \leq d(A,D)$	$d(C,B) - d(B,A) - d(D,C) \leq d(A,D)$	$d(D,C) - d(C,B) - d(B,A) \leq d(A,D)$	Subtraction on both sides #1
3.	$d(B,A) - d(D,C) - d(C,B) \leq d(A,D)$	$d(C,B) - d(D,C) - d(B,A) \leq d(A,D)$	$d(D,C) - d(B,A) - d(C,B) \leq d(A,D)$	Subtraction on both sides #1
4.	$\boldsymbol{\lvert d(B,A) - d(C,B)\rvert - d(D,C) \leq d(A,D)}$			Absolute Value Definition (#2A/2B)
5.	$\boldsymbol{\lvert d(C,B) - d(D,C)\rvert - d(B,A) \leq d(A,D)}$			Absolute Value Definition (#2C/3B)
6.	$\boldsymbol{\lvert d(B,A) - d(D,C)\rvert - d(C,B) \leq d(A,D)}$			Absolute Value Definition (#3A/3C)

Table 2 Derivation of the Reverse Quadrilateral Inequality in Simple, Connected Graphs

The resulting inequalities that bound the distance between two vertices, *A* and *D*, are

$$|d(B,A) - d(C,B)| - d(D,C) \leq d(A,D) \tag{46}$$

$$|d(C,B) - d(D,C)| - d(B,A) \leq d(A,D) \tag{47}$$

$$|d(B,A) - d(D,C)| - d(C,B) \leq d(A,D) \tag{48}$$

A potential problem with these inequalities is that they have the ability to generate negative lower bound estimates, which is useless for a nonnegative distance metric. For utility, when attempting to estimate the lower bounds of a quadrilateral, other geometric inequalities should be considered such that the highest possible lower bound can be used. In this dissertation, we use two such estimations to inform the heuristic. The first, Ptolemy's inequality (Kay, 2011) for quadrilaterals is used as follows for the dual landmark heuristic to yield a lower bound for the distance between *A* and *D*. First, we begin with the original inequality:

$$d(A,C) \cdot d(B,D) \leq d(B,A) \cdot d(D,C) + d(C,B) \cdot d(A,D) \tag{49}$$

Note that when considering these alternative inequalities, we maintain the same notation for each distance term, as to not disturb the inequality when a directed graph is used. Then to estimate the distance between *A* and *D*, using simple algebra,

$$\frac{d(A,C) \cdot d(B,D) - d(B,A) \cdot d(D,C)}{d(C,B)} \leq d(A,D) \tag{50}$$

In practical cases, information regarding the values of $d(A,C)$ and $d(B,D)$ (the diagonals) may be unknown. Therefore, the distance between can be estimated as follows. First, suppose all the values on the right side of the above equation are known and the values on the left side are unknown (except, of course, the distance between vertices *A* and *D*). Using the reverse triangle inequality[2], we understand that

$$0 \leq |d(B,A) - d(C,B)| \leq \ d(A,C) \tag{51}$$

$$0 \leq |d(C,B) - d(D,C)| \leq \ d(B,D) \tag{52}$$

Because they are non-negative, we also know that

[2] Taking directionality into account.

$$0 \leq |d(A,B) - d(B,C)| \cdot |d(B,C) - d(C,D)| \leq d(A,C) \cdot d(B,D) \tag{53}$$

Using these lower bounds, we can rewrite Ptolemy's inequality with respect to the lower bound for the distance between vertices *A* and *D* as

$$\frac{|d(B,A) - d(C,B)| \cdot |d(C,B) - d(D,C)| - d(B,A) \cdot d(D,C)}{d(C,B)} \leq d(A,D) \tag{54}$$

Because we use Ptolemy's inequality here, this can become a perfect estimate when a cyclic quadrilateral is formed from the four endpoint vertices, $A, B, C, D \in V$. Understanding how to form a cyclic quadrilateral in a graph or quickly verify that a quadrilateral formed in a graph is cyclic, however, is outside of the scope of this dissertation effort.

The use of Ptolemy's inequality, here, serves as one of three examples of using multiple data points to vary heuristics for A* search in a graph. Because multiple data points are used, more inequalities can be generated to estimate distances. The maximum over the set of lower bounds derived by these inequalities can be used to tighten the lower bound. With that said, the second example gives two more lower bounds for the distance between *A* and *D*, derived from the triangle inequality, are noted here:

$$|d(B,A) - d(D,B)| \leq d(A,D) \tag{55}$$

$$|d(C,A) - d(D,C)| \leq d(A,D) \tag{56}$$

As stated earlier in regards to Ptolemy's inequality, $d(A,C)$ and $d(B,D)$ are commonly unknown[3]. Though, in this case, we cannot derive a similar inequality by using the two values' lower bounds. However, in ALP's case, we will see later that these equations will come in handy when *B* = *C*. Therefore, we add it to the set of lower bounds.

[3] These would be the diagonals of the quadrilateral

The third example is taken from the *four-point condition* on metric spaces that is valid for trees with weighted edges, such as in the case of a shortest path tree. The four-point condition states that for the nodes in Figure 9, the shortest path tree holds the following property:

$$d(A,C) + d(D,B) \leq \ max\{d(B,A) + d(D,C), d(A,D) + d(C,B)\} \tag{57}$$

Just like with Ptolemy's, $d(A,C)$ and $d(D,B)$ are commonly unknown. Therefore, we replace these terms with their lower bounds in the equation:

$$|d(B,A) - d(C,B)| + |d(D,C) - d(C,B)| \tag{58}$$

$$\leq \ max\{d(B,A) + d(D,C), d(A,D) + d(C,B)\}$$

Therefore, we have

$$|d(B,A) - d(C,B)| + |d(D,C) - d(C,B)| - d(C,B) \leq \ d(A,D) \tag{59}$$

if and only if the following condition holds:

$$d(A,D) + d(C,B) \geq d(B,A) + d(D,C) \tag{60}$$

In conclusion, when estimating the distance between two points in a graph such as the one in Figure 9, the maximum of the following seven equations can result in the tightest lower bound for the distance between vertices A and D.

$$|\boldsymbol{d(B,A)-d(C,B)}|-\boldsymbol{d(D,C)} \leq \boldsymbol{d(A,D)}$$

$$|\boldsymbol{d(C,B)-d(D,C)}|-\boldsymbol{d(B,A)} \leq \boldsymbol{d(A,D)}$$

$$|\boldsymbol{d(B,A)-d(D,C)}|-\boldsymbol{d(C,B)} \leq \boldsymbol{d(A,D)}$$

$$|\boldsymbol{d(B,A)-d(D,B)}| \leq \boldsymbol{d(A,D)}$$

$$|\boldsymbol{d(C,A)-d(D,C)}| \leq \boldsymbol{d(A,D)}$$

$$\frac{|\boldsymbol{d(B,A)-d(C,B)}|\cdot|\boldsymbol{d(C,B)-d(D,C)}|-\boldsymbol{d(B,A)}\cdot\boldsymbol{d(D,C)}}{\boldsymbol{d(C,B)}} \leq \boldsymbol{d(A,D)}$$

$$|\boldsymbol{d(B,A)-d(C,B)}|+|\boldsymbol{d(D,C)-d(C,B)}|-\boldsymbol{d(C,B)} \leq \boldsymbol{d(A,D)}$$

Figure 10 Quadrilateral Inequalities for Graphs

A, Landmarks, and Polygon Inequalities*

Just as with the reverse triangle inequality, the lower bound produced by the reverse quadrilateral inequality can be used as a heuristic for the A* algorithm. The establishment of this new heuristic is known as ALP, for its use of the A* algorithm, Landmarks, and Polygon Inequalities. By choosing two landmark vertices to act as endpoints *B* and *C* from the last section, a new dual landmark heuristic is achieved as follows: For source and target nodes $s,t \in V$ and two valid landmark vertices $l_1, l_2 \in V$ in a graph *G*, the following lower bounds hold for the shortest path:

$d(s,t) \geq \lvert d(l_1,s)-d(l_2,l_1)\rvert - d(t,l_2)$	**Reverse Quadrilateral Inequalities**
$d(s,t) \geq \lvert d(l_1,s)-d(t,l_2)\rvert - d(l_2,l_1)$	
$d(s,t) \geq \lvert d(l_2,l_1)-d(t,l_2)\rvert - d(l_1,s)$	
$d(s,t) \geq \lvert d(l_1,s)-d(t,l_2)\rvert$	***l_1=l_2***
$d(s,t) \geq \lvert d(l_1,s)-d(t,l_2)\rvert$	***l_1=l_2***
$d(s,t) \geq \dfrac{\lvert d(l_1,s)-d(l_2,l_1)\rvert\cdot\lvert d(l_2,l_1)-d(t,l_2)\rvert - d(l_1,s)\cdot d(t,l_2)}{d(l_2,l_1)}$	**Ptolemy's Inequality**
$d(s,t) \geq \lvert d(l_1,s)-d(l_2,l_1)\rvert + \lvert d(t,l_2)-d(l_2,l_1)\rvert - d(l_2,l_1)$	**Four-Point Condition**

Table 3 Inequalities for a source, target, and two landmark vertices in a directed graph

These seven lower bounds can all become heuristics for the ALP algorithm. Because it is based on dual landmarks (DL), let $\pi^{\boldsymbol{DL}}_{\langle t,i\rangle}(v),\ i \in [1,7]$ denote each new heuristic at a visited vertex $v \in V$. For two given landmarks, l_1, l_2, the following seven heuristics can be used for the A* algorithm:

$$\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v) = |d(l_1, v) - d(l_2, l_1)| - d(t, l_2) \tag{61}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,2\rangle}(v) = |d(l_1, v) - d(t, l_2)| - d(l_2, l_1) \tag{62}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,3\rangle}(v) = |d(l_2, l_1) - d(t, l_2)| - d(l_1, v) \tag{63}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,4\rangle}(v) = |d(l_1, v) - d(t, l_1)| \tag{64}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,5\rangle}(v) = |d(l_2, v) - d(t, l_2)| \tag{65}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,6\rangle}(v) = \frac{|d(l_1, v) - d(l_2, l_1)| \cdot |d(l_2, l_1) - d(t, l_2)| - d(l_1, v) \cdot d(t, l_2)}{d(l_2, l_1)} \tag{66}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,7\rangle}(v) = |d(l_1, v) - d(l_2, l_1)| + |d(t, l_2) - d(l_2, l_1)| - d(l_2, l_1) \tag{67}$$

Each of these are new, admissible heuristics for A* based on polygon inequalities, specifically for quadrilaterals. The following is the optimal dual landmark heuristic now for ALP.

$$\pi^{\boldsymbol{DL}}_{t}\ (v) = \ max_i\ \{\pi^{\boldsymbol{DL}}_{\langle t,i\rangle}(v)\} \tag{68}$$

As a word of caution, one has to be careful when in the case of directed graphs. In the undirected case, there is no difference between estimating the distance from v to t $(d(v, t))$and from t to v $(d(t, v))$. However, as shown in Figure 11, to generalize ALP for the directed and undirected case, directionality of the distance terms must be taken into account. For a directed graph, the shortest path metric space is formed with these as the distances between four points. For any four-vertex configuration of the graph,

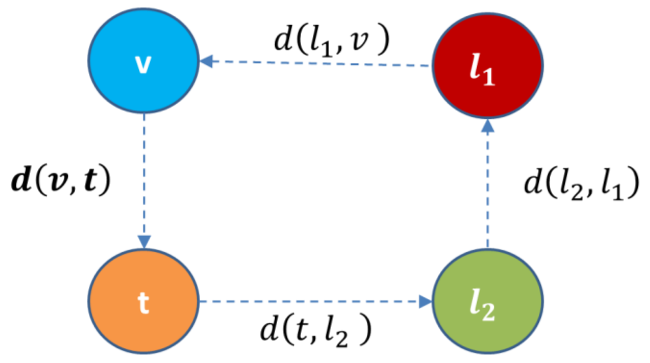


Figure 11 Four vertices within a sample directed connected graph. The dotted lines represent shortest paths between each of the vertices

preprocessing must yield instant access to the three distance values in the figure not in bold in order to derive this new heuristic.

For ALP, the A* algorithm, described in Chapter 1, is used with this new heuristic function as input, just as in ALT, with one change. This change involves a process known as *distributed landmark embedding*, or simply *distributed embedding*. The distributed embedding process is further detailed in a later section. In summary, for dual landmark ALP, the process works as follows. After landmark selection, each vertex in the graph is assigned to a single landmark within its respective partition. Distance information is then computed from each partition's landmark to (and from, in the directed case) the other vertices subgraph, as well as between all landmarks in the landmark set of the graph. These vertices contain distance information for only the landmark to which they are assigned. As a vertex *v* is visited, if *v* does not have distance information at its current landmark node, l_1, the landmark that does have distance information for *v* is used to bound the search. For unidirectional A*, the l_2 landmark remains the same for the target node, as it is the only one containing distance information for that node. This fact, of course, would change for the bidirectional variant of A*. Note that, when using distributed embedding, $\pi^{\boldsymbol{DL}}_{\langle t,4\rangle}(v)$ and $\pi^{\boldsymbol{DL}}_{\langle t,5\rangle}(v)$ can only be used when both the visited node *v* and target node *t* share the same landmark. Otherwise, the information needed for this heuristic cannot be computed. If the source and target vertex share the same landmark (i.e., $l_1 = l_2$), then the ALP heuristic is reduced to the ALT heuristic (i.e., $l_1 = l_2 = l$) as follows:

$$\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v) = |d(v,l) - d(l,l)| - d(l,t) = |d(v,l)| - d(l,t) = d(v,l) - d(l,t) \quad (69)$$

$$\pi^{\boldsymbol{DL}}_{\langle t,2\rangle}(v) = |d(v,l) - d(l,t)| - d(l,l) = |d(v,l) - d(l,t)| = |d(v,l) - d(l,t)| \quad (70)$$

$$\pi^{\boldsymbol{DL}}_{\langle t,3\rangle}(v) = |d(l,l) - d(l,t)| - d(v,l) = |-d(l,t)| - d(v,l) = d(l,t) - d(v,l) \quad (71)$$

Because we are taking the maximum, $\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}$ and $\pi^{\boldsymbol{DL}}_{\langle t,3\rangle}$ simplify to the reverse triangle inequality. $\pi^{\boldsymbol{DL}}_{\langle t,4\rangle}$ and $\pi^{\boldsymbol{DL}}_{\langle t,5\rangle}$ are, by their very definition, equal to the reverse triangle inequality, as well. $\pi^{\boldsymbol{DL}}_{\langle t,6\rangle}$ cannot be used over the same set of landmarks because its equation would result in a division by zero. Finally, $\pi^{\boldsymbol{DL}}_{\langle t,7\rangle}$ cannot hold because its constraint would violate the triangle inequality. Therefore, the dual landmark ALP heuristic function always reduces to the ALT heuristic ($\pi^{\boldsymbol{DL}}_{\langle t,4\rangle}(v)$ and $\pi^{\boldsymbol{DL}}_{\langle t,5\rangle}(v)$) when the currently visited and target nodes share landmarks.

It should be noted that there are other polygon-based inequalities for special cases and shapes that could also be used to define A* heuristics, as they, too, can yield estimates that never overestimate the shortest path. Future research can include the use and selection of varying heuristics for special quadrilaterals along with that of other types of polygons induced on the graph. Such research would address the difficult problem of extracting information such as angle and inscribed shapes before the heuristic could be computed. In this dissertation, however, we will conduct experimentation using only the heuristics defined in this section. The dual landmark ALP heuristic for the inequalities derived in this section will be characterized in the following section and will be used for experimentation.

Characterizing ALP Heuristics

For a source and target vertex pair, the following theorems for the ALP heuristic function, $\pi_t^{\boldsymbol{DL}}$, apply:

Theorem 1: $\pi_t^{\boldsymbol{DL}}$ is an admissible heuristic.

Proof. The proofs for the inequalities used for the heuristic are all derived in the previous section. Because the heuristic function has an upper bound set at the actual shortest path to the target, the heuristic will never overestimate the distance to the target, rendering it admissible.

Theorem 2: Using distributed embedding, $\pi_t^{\boldsymbol{DL}}$ is not consistent.

Proof. This is proven by contradiction. Let *c* be the cost of transitioning with A* from vertex v to v', for $v, v' \in V$. Recall that *c* is nonnegative for the A* algorithm. Let $\pi_{\langle t,1\rangle}^{\boldsymbol{DL}}(v')$ be the maximum chosen for $\pi_t^{\boldsymbol{DL}}$ for both of these iterations. Then, for $\pi_t^{\boldsymbol{DL}}$ to be consistent,

$$|d(v,l_1) - d(l_1,l_2)| - d(t,l_2) \ \leq c + \ |d(v',l_1) - d(l_1,l_2)| - d(t,l_2) \qquad (72)$$

Because c is non-negative and the heuristic takes into account whether or not it moves towards or away from its landmark, $d(v',l_1) = d(v,l_1) - c$ or $d(v',l_1) = d(v,l_1) + c$, respectively. Therefore, this equation holds and demonstrates monotonicity over the same set of landmarks for successive iterations. However, allow the selection of landmarks for a query to change during the query, due to distributed embedding. For the heuristic to be consistent, with vertex v belonging to landmark l_i and v' belonging to landmark l_j, once again let $\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v')$ be the maximum chosen for $\pi^{\boldsymbol{DL}}_t$ for both of these iterations. The following equation must then hold for $\pi^{\boldsymbol{DL}}_t$ to be consistent.

$$|d(v,l_i) - d(l_i,l_2)| - d(t,l_2) \le c + \left|d(v',l_j) - d(l_j,l_2)\right| - d(t,l_2) \tag{73}$$

Let l_j be a chosen landmark such that $d(l_i,l_2) > c + d(l_j,l_2)$ and $d(v,l_i) < d(v',l_j)$. This scenario yields a contradiction for the equation such that $\pi^{\boldsymbol{DL}}_t$ is not consistent.

Theorem 3: $\pi^{\boldsymbol{DL}}_t$ does not dominate $\pi^{\boldsymbol{L}}_t$ over the same set of landmarks.

Proof. In the previous section, we demonstrated that the dual landmark heuristic reduces to the triangle inequality heuristic over the same set of landmarks. This means that when a visited vertex and target share the same landmark, the heuristic estimates for $\pi^{\boldsymbol{DL}}_t$ and $\pi^{\boldsymbol{L}}_t$ will always be equal. For one heuristic to dominate another, all of its values must be greater than or equal to the corresponding values of the other heuristic. Therefore, for $\pi^{\boldsymbol{DL}}_t$ to dominate $\pi^{\boldsymbol{L}}_t$ over the same set of landmarks, $\pi^{\boldsymbol{DL}}_t$ would have to dominate $\pi^{\boldsymbol{L}}_t$ when a visited vertex and target do not share landmarks. We take two landmarks $l_1, l_2 \in |V|$ (for $l_1 \neq l_2$),that reference the visited vertex *v* and target *t*, respectively. For $\pi^{\boldsymbol{DL}}_t$ to dominate $\pi^{\boldsymbol{L}}_t$, any one of the following inequalities must hold:

1. $|d(v,l_1) - d(l_1,l_2)| - d(l_2,t) \ge max_{l_i \epsilon \boldsymbol{L}}\,\{|d(v,l_i) - d(t,l_i)|\}$
2. $|d(v,l_1) - d(l_2,t)| - d(l_1,l_2) \ge max_{l_i \epsilon \boldsymbol{L}}\,\{|d(v,l_i) - d(t,l_i)|\}$
3. $|d(l_1,l_2) - d(l_2,t)| - d(v,l_1) \ge max_{l_i \epsilon \boldsymbol{L}}\,\{|d(v,l_i) - d(t,l_i)|\}$
4. $\frac{|d(l_1,v) - d(l_2,l_1)| \cdot |d(l_2,l_1) - d(t,l_2)| - d(l_1,v) \cdot d(t,l_2)}{d(l_2,l_1)} \ge max_{l_i \epsilon \boldsymbol{L}}\,\{|d(v,l_i) - d(t,l_i)|\}$
5. $|d(l_1,v) - d(l_2,l_1)| + |d(t,l_2) - d(l_2,l_1)| - d(l_2,l_1) \ge max_{l_i \epsilon \boldsymbol{L}}\,\{|d(v,l_i) - d(t,l_i)|\}$

Because l_1 and l_2 are in the set $\boldsymbol{L}$, we can eliminate the first three equations from validity as there is no way to guarantee (outside of very specific landmark selection) that

$$|d(v, l_1) - d(l_2, t)| \geq max_{l_i \epsilon \boldsymbol{L}} \{|d(v, l_i) - d(t, l_i)|\}$$

For the final two inequalities, we can easily identify the same contradiction for both. Let all distance values used on the left hand side of the equations equal to one. This results in a negative left-hand side for the inequality. The right-hand side of the inequality has the benefit that it can never be negative. Therefore, we no equations left where $\pi_t^{\boldsymbol{DL}}$ provides a greater estimate than $\pi_t^{\boldsymbol{L}}$.

Theorem 4: $\pi_t^{\boldsymbol{L}}$ does not dominate $\pi_t^{\boldsymbol{DL}}$ over different landmark sets.

Proof. This can be proven by contradiction. Let $\pi_{\langle t,1\rangle}^{\boldsymbol{DL}}(v')$be the maximum chosen value for $\pi_t^{\boldsymbol{DL}}$. For the triangle inequality heuristic to dominate the dual landmark heuristic:

$$|d(v, l) - d(t, l)| \geq \left|d(v, l_i) - d\left(l_i, l_j\right)\right| - d\left(t, l_j\right) \tag{74}$$

where l is the landmark that maximizes $\pi_t^{\boldsymbol{L}}$ and l_i and l_j are the landmarks for v and t, respectively. Let $d\left(l_i, l_j\right) \gg d(v, l_i) + d\left(t, l_j\right) > |d(v, l) - d(t, l)|$, meaning the distance between the two landmarks are much greater than the sum of the landmark distances for the visited and target vertex. Then it follows that $\left|d(v, l_i) - d\left(l_i, l_j\right)\right|$ is significantly larger than all other terms in the equation. If we let the distance between both l and the visited vertex and target nodes be equal for every landmark, the term $|d(v, l) - d(t, l)|$ will be significantly small. Then the above equation does not hold for landmarks that are significantly far apart and we have a contradiction.

To summarize, according to Theorem 1, ALP's dual landmark heuristic is an admissible heuristic, making it a viable candidate for the A* algorithm, even though it is not consistent when using distributed embedding, as shown in the proof of Theorem 2.

We address Theorem 2 in experimentation for both ALT and ALP by implementing pathmax for A*, forcing consistency for both heuristics. From Theorem 3, this heuristic for ALP does not dominate the heuristic for ALT over the same set of landmarks. From Theorem 4, it is demonstrated that there are scenarios in which the ALP heuristic gives a higher estimation than the ALT algorithm. In the proof for Theorem 4, a possible scenario for ALT (with the visited vertex *v* being very far from the target *t*) is used to theoretically demonstrate that it can have a lower heuristic estimate than ALP. The proof inherently shows the reverse, as well: that ALP can have a lower heuristic estimate than ALT. Theorem 4 highlights landmark selection as the key to one heuristic theoretically outperforming the other. We delve into further detail for this finding in the next section. These four theorems and their respective proofs are the justification for the investigation of landmark selection techniques for ALP. If landmark selection techniques for ALP allow for a more informed A* search capability, then it is the overall optimal heuristic as its landmark selection is inherently faster than that of ALT's.

A major contribution of this dissertation an experimental characterization of the real, practical scenarios for better distance estimates with respect to landmark selection for the ALP and ALT heuristics. Specifically, given that distributed embedding allows the practical preprocessing time and space complexity to be significantly less, it is worth exploring the cases that ALP heuristic does outperform the ALT heuristic and vice-versa. Recall, from Chapter 1, that one heuristic outperforms the other, in terms of the number of vertices that are searched, by creating a higher estimation of the shortest path lower bound. Let $l_\alpha \in \boldsymbol{L}$ be the landmark chosen for ALT that maximizes its heuristic and $l_1, l_2 \in \boldsymbol{L}$ be the landmarks for the current vertex and the target, respectively. For each possible landmark setup, the following are the scenarios in which the ALP dual landmark heuristic outperforms the ALT triangle inequality heuristic in the context of number of explored vertices. As the dual landmark heuristic uses seven separate equations to derive its heuristic, the equations that actually cause the ALP heuristic to dominate ALT are specified here. Note that the ALP heuristics that are recommended in each of these scenarios can, but are not guaranteed to, dominate ALT and are not inclusive of all dual landmark ALP estimates that can dominate ALT. These scenarios specify situations in which the dual landmark ALP heuristic has a high likelihood of dominating the ALT heuristic, and will be experimentally verified throughout the dissertation.

Scenario 1: $l_1 = l_\alpha \neq l_2$

Outperforms ALT when $|d(l_1, l_2) - d(t, l_2)| - d(v, l_1) \geq |d(v, l_1) - d(t, l_1)|$

This scenario, in particular, outperforms ALT at the beginning of a search in a large graph, for $\pi^{\boldsymbol{DL}}_{\langle t,2\rangle}(v)$, **when the distances between the two landmarks is significantly large**. Particularly, if $|d(l_1, l_2) - d(t, l_2)| \gg d(t, l_1) > d(v, l_1)$, the heuristic dominates. As such, $\pi^{\boldsymbol{DL}}_{\langle t,3\rangle}(v)$ and $\pi^{\boldsymbol{DL}}_{\langle t,6\rangle}(v)$ are the estimates that have a higher likelihood of yielding stronger results than the triangle inequality here.

Scenario 2: $l_1 \neq l_\alpha = l_2$

Outperforms ALT when $|d(l_1, l_2) - d(v, l_1)| - d(t, l_2) \geq |d(v, l_2) - d(t, l_2)|$

Particularly, if $|d(l_1, l_2) - d(v, l_1)| \gg d(v, l_2) > d(t, l_2)$, the heuristic dominates. Since we cannot rely on $d(v, l_1)$ to always be significantly larger than $d(v, l_2)$, the heuristic relies on the distance between the respective landmarks being significantly large to dominate. Therefore, in this scenario, the ALP heuristic dominates ALT **when the distance between the two landmarks is significantly large.** As such, $\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v)$ and

$\pi^{\boldsymbol{DL}}_{\langle t,6\rangle}(v)$ are the estimates that have a higher likelihood of yielding stronger results than the triangle inequality here.

Scenario 3: $l_1 = l_\alpha = l_2$

Always has the same performance as ALT.

$d(l_\alpha, l_\alpha)$=0, by definition. Therefore, all of the possible equations for the ALP heuristic are reduced to the triangle inequality. And **the ALP heuristic becomes the ALT heuristic**.

Scenario 4: $l_1 = l_2 \neq l_\alpha$

Outperforms ALT when $|d(v, l_1) - d(l_1, t)| - d(l_1, l_1) \geq |d(v, l_\alpha) - d(t, l_\alpha)|$

$d(l_1, l_1)$=0, by definition. Therefore, $\pi^{\boldsymbol{DL}}_t(v, 6)$ is eliminated as an option for the dual landmark heuristic. Because this occurs and because the ALT heuristic chooses the landmark that maximizes the triangle inequality, the best we can hope for is that the ALP heuristic is reduced to the heuristic for ALT. Therefore, **when the ALP algorithm's search is within the same partition, the ALP algorithm never dominates the ALT algorithm**.

Scenario 5: $l_1 \neq l_\alpha \neq l_2$

Outperforms ALT when

$\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v) \geq |d(v, l_\alpha) - d(t, l_\alpha)|$ or

$\pi^{\boldsymbol{DL}}_{\langle t,2\rangle}(v) \geq |d(v, l_\alpha) - d(t, l_\alpha)|$ or

$\pi^{\boldsymbol{DL}}_{\langle t,3\rangle}(v) \geq |d(v, l_\alpha) - d(t, l_\alpha)|$ or

$\pi^{\boldsymbol{DL}}_{\langle t,6\rangle}(v) \geq |d(v, l_\alpha) - d(t, l_\alpha)|$ or

$\pi^{\boldsymbol{DL}}_{\langle t,7\rangle}(v) \geq |d(v, l_\alpha) - d(t, l_\alpha)|$

$\pi^{\boldsymbol{DL}}_{\langle t,4\rangle}(v)$ or $\pi^{\boldsymbol{DL}}_{\langle t,5\rangle}(v)$ can only reach the equivalence of the ALT heuristic's estimate over the same set of landmarks or for landmarks with similar distances to the one's used in ALT.

Scenario 5 is the most common situational scenario and will promise interesting experimental results. This is also the scenario that most significantly demonstrates that when the landmarks that would be used for both ALT and ALP differ, the heuristic value for ALP is not always greater than the heuristic value for ALT, producing the results of Theorems 3 and 4. The key insight here is that if more efficient algorithms for selecting a better landmark set for ALP exist, ALP will often outperform ALT in practical scenarios. All of these observations about ALP's performance are summarized in Table 4.

	Scenario	**Outperforms ALT when…**
1.	$l_1 = l_\alpha \neq l_2$	$\lvert d(l_1,l_2) - d(t,l_2)\rvert \gg d(t,l_1) > d(v,l_1)$
2.	$l_1 \neq l_\alpha = l_2$	$\lvert d(l_1,l_2) - d(v,l_1)\rvert \gg d(v,l_2) > d(t,l_2)$
3.	$l_1 = l_\alpha = l_2$	$Never. Always\ equal\ performance.$
4.	$l_1 = l_2 \neq l_\alpha$	$Never. At\ best, equal\ performance.$
5.	$l_1 \neq l_\alpha \neq l_2$	$\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v) \geq \lvert d(v,l_\alpha) - d(t,l_\alpha)\rvert$ or $\pi^{\boldsymbol{DL}}_{\langle t,2\rangle}(v) \geq \lvert d(v,l_\alpha) - d(t,l_\alpha)\rvert$ or $\pi^{\boldsymbol{DL}}_{\langle t,3\rangle}(v) \geq \lvert d(v,l_\alpha) - d(t,l_\alpha)\rvert$ or $\pi^{\boldsymbol{DL}}_{\langle t,6\rangle}(v) \geq \lvert d(v,l_\alpha) - d(t,l_\alpha)\rvert$ or $\pi^{\boldsymbol{DL}}_{\langle t,7\rangle}(v) \geq \lvert d(v,l_\alpha) - d(t,l_\alpha)\rvert$

Table 4 When ALP Beats ALT

Distributed Embedding

For a set of landmarks $\boldsymbol{L}$, the ALT algorithm has a space complexity of $\theta(|\boldsymbol{L}| \times |V|)$ from computing and storing distance information for all shortest paths between each landmark and *V*. (J Maue et al., 2006) However, when using ALP, this space complexity

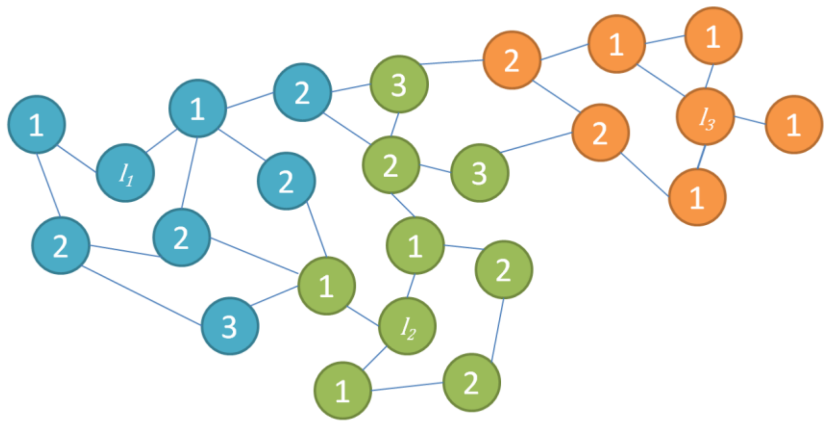


Figure 12 An Example of Distributed Embedding for a Simple Graph with Three Partitions

can be reduced to $\theta(|\boldsymbol{L}|^2 + |V|)$ using the following technique, called *distributed landmark embedding*. In the dual landmark preprocessing for ALP, each landmark only computes the shortest path tree to a specified set of vertices, called a *graph partition*, around it[4]. The only other operation is a shortest path calculation among the landmark set, as the distance between each landmark is needed to compute the ALP heuristic. For best results, the subgraph induced by each partition should be connected to increase the likelihood that the shortest path from the landmark to any vertex in the partition lies within the subgraph induced by the graph partition, though this is not a requirement.

As shown in Figure 12, during preprocessing, each vertex in the graph needs to be labeled with an identifier, signifying its landmark partition and the distance to (and from, in the case of directed graphs) its corresponding landmark. When all landmarks have been chosen, an SPT for each landmark in $\boldsymbol{L}$ is then computed for its respective partition. To preserve space, this partitioning information is not explicitly stored. Rather, each vertex maintains distance information about the landmark to which it belongs along with

[4] In this work, we identify the graph partitions first and select landmarks inside of these partitions (rather, we see the partitions as input to the algorithm, just as with PCD(Jens Maue, 2006)). Future work can explore the initially identifying landmarks in the graph first and then use these landmarks to form partitions.

a reference to that landmark. The only information that a landmark maintains is distance information between it and all other landmarks. For landmark selection algorithms, if an algorithm requires understanding of all vertices that belong to a particular partition, then the partition can be discovered by finding all vertices with a common landmark reference. During query time, ALP carries out the normal A* algorithm with the ALP heuristic function, $\pi_t^{\boldsymbol{DL}}$, that relies on polygon inequalities for quadrilaterals.

Recall from Chapter 2 that the time complexity of ALT's preprocessing, not including the selection of l landmarks, is $O(|\boldsymbol{L}| \cdot (|E| + |V| \log|V|))$, as an SPT is generated with Dijkstra's algorithm, rooted at each landmark. Each of these SPTs covers the entire graph. For ALP, multiple SPTs are grown with the landmarks as roots such that the union of their vertices covers all vertices of the graph. Distance information is only maintained by vertices for one other vertex (i.e., the landmark vertex at the root of its SPT). For this to occur, it simply grows the Dijkstra SPT from a given landmark until all vertices in the landmark's partition are a part of the tree. For overlapping graph partitions, ALP grows the shortest path tree from each landmark to cover the vertices in its partition, as usual. During query time, the algorithm uses the set of landmarks with known distances that produce the highest lower bounds.

The memory and practical runtime saved by doing this is the novelty of distributed embedding. Note that the theoretical time complexity for preprocessing of ALP remains the same as that of ALT. The actual shortest path between two vertices within a graph partition could include vertices from outside the partition. This means that, in the worst case, the generated SPT includes the entire vertex set of the graph. This, of course, would rarely happen in practice. In practice, the SPT is significantly small in comparison to the size of the graph and its generation runs in a fraction of the time. Therefore, for a graph in which the vertices of each partition match the vertices in a partition's shortest path tree, let E' be the average number of edges in each partition and V' the average number of vertices in each partition. Then the average runtime of ALP preprocessing, not including landmark selection, is

$$O(|\boldsymbol{L}| \cdot (|E'| + |V'| \log|V'|)) \tag{75}$$

Because the shortest path tree is computed from every chosen landmark and distance along with an all-pairs shortest path calculation for the landmarks, ALT's data structure requires $\theta(|\boldsymbol{L}| \cdot |V|)$ space[5]. Since $|\boldsymbol{L}| \leq |V|$, the theoretical space requirement for ALT can be said to be $O(|V|^2)$. Note that this upper limit is only theoretical, as a relatively small number of landmarks are chosen for any particular graph. Therefore, the $\theta(|\boldsymbol{L}| \cdot |V|)$ space requirement is a more practical specification. For ALP, shortest path data is stored for the landmark-vertex pairs of each graph partition and the pairwise distances between landmarks. Therefore, ALP's data structure requires $\theta(|V| + |\boldsymbol{L}|^2)$ space. Once again, because $|\boldsymbol{L}| \leq |V|$, the space requirement for ALP can also be described as $O(|V| + |V|^2) = O(|V|^2)$, which is theoretically larger than the worst-case ALT requirement. Therefore, the ALP space requirement is an improvement on the ALT space requirement as long as

$$|\boldsymbol{L}| < \sqrt{|V|^2 - |V|} \tag{76}$$

[5] It should be noted that for directed graphs, we compute the shortest path tree to and from every landmark, requiring twice the space from ALT and twice the number of subgraph vertices to be stored for ALP $(2 \cdot |V| + |\boldsymbol{L}|^2)$.

Finally, recall that, during an arbitrary shortest path query, ALT attempts to maximize its heuristic by using the triangle inequality for each landmark at each visited vertex of the search:

$$\pi_t^{\boldsymbol{L}}(v) = max_{l_i \epsilon \boldsymbol{L}} \{\pi_t^{\boldsymbol{L}}\} \tag{77}$$

For a growing number of landmarks, computing this many estimates at each step becomes computationally expensive. However, the dual landmark heuristic, $\pi_t^{\boldsymbol{DL}}$, only requires that, at most, four estimates be computed and compared at each iteration. This should drastically reduce ALP's compute time in comparison to ALT.

Algorithm Degradation

Thus far, when describing ALP's performance in comparison to ALT, performance has been measured by the value calculated by a heuristic function. For A*, this value determines the size of the search space for any given query. For an admissible heuristic, the higher the estimates, the smaller the search space and the assumption is always that this leads to better overall performance. However, one thing that is not taken into account in this and many shortest path performance surveys is the amount of processing needed to compute the actual heuristic as each vertex is being visited. As stated in Chapter 2, for each PPSP query, at each vertex, a number of subtractions equal to the number of landmarks is performed as well as a *max* operation. This means a $\theta(|\boldsymbol{L}|)$ runtime for each visited node. For large-scale graphs, which require more landmarks to be preprocessed, this can significantly add to the overall compute time of queries. In comparison, with the dual landmark ALP heuristic, if the visited vertex and target vertex are owned by different landmarks, exactly twelve subtraction operations, two multiplication operations, two additions, and a division operation occurs with a $\theta(4)$ *max* operation. If they are owned by the same landmarks, only one subtraction operation occurs (to compute the reverse triangle inequality). This means that, in terms of practical, processor-based performance measurements, over the same set of landmarks, it is possible for dual landmark ALP to outperform ALT. In particular, for graphs with longer average path lengths, the search performance for an ALT heuristic with higher estimates can suffer degradation at a rate significantly less than ALP's heuristic.

The implementation of operations such as multiplication and division can vary from system to system and therefore would have an impact on the search strongly dependent on the processor. As computer architectures and optimization methods for arithmetic operations and *max* functions vary greatly, there is no formal computation model upon which we can compare and contrast this level of detail in performance for the heuristics. Future research could involve the ALP algorithm being experimentally tested against ALT over a series of different processor architectures to concretize their performance on modern day systems. Also, clever ways to reduce the number of operations for each heuristic calculation while maintaining asymptotic complexity should be explored.

In this dissertation, experiments not only measure the number of visited nodes when comparing performance of shortest path algorithms. During experimentation, the number of each type of arithmetic operation and the computational runtime performed during each query are stored as measurements. This type of measurement is performed to better characterize the behavior of ALT and ALP as graph sizes scale.

ALT Landmark Selection in ALP

In ALT, solutions to the problem of choosing the best landmarks seek to reduce the average search space for arbitrary shortest path queries. Recall the search space, as defined in Chapter 2, is

$$V_L(s,t) = \{v \in V | d(s,v) + \pi_t^{\boldsymbol{L}}(v) \leq d(s,t) \ \land \ v < t\} \tag{78}$$

The MinALT problem seeks to choose the minimum set of landmarks L that reduces the overall search space for arbitrary shortest path queries and can be denoted as follows:

Problem: $MinALT(G,k) = \ arg\ min_L \ \sum_{s,t \in V} |V_L(s,t)|$

Landmark selection techniques in ALP seek to solve the exact same problem. The search space for ALP using the dual landmark heuristic to guide the search is simply defined as

$$V_{DL}(s,t) = \{v \in V | d(s,v) + \pi_t^{\boldsymbol{DL}}(v) \leq d(s,t) \ \land \ v < t\} \tag{79}$$

We denote the problem of choosing the minimum set of landmarks L, which reduces this overall search space for arbitrary shortest path queries as

Problem: $MinALP(G,k) = \ arg\ min_L \ \sum_{s,t \in V} |V_{DL}(s,t)|$

While the goals of the proposed solutions to MinALT and MinALP are the same, algorithms that have been generated to solve them must differ because of the graph partitioning requirement of ALP. Further, the goals of these algorithms must differ because of the arithmetic that maximizes each heuristic. To state the differences explicitly, high heuristic estimates for the ALT algorithm rely on a landmark being extremely far from the vertex being visited during the search and extremely close to the target vertex, or vice-versa. In other words, for $\pi_t^{\boldsymbol{L}}$, either $d(l,v)$ should approach the graph diameter while $d(t, l)$ approaches 0 or $d(t, l)$ should approach the graph diameter while $d(l, v)$ approaches 0 to maximize estimates, thereby maximizing performance. For the dual landmark ALP heuristic, distributed embedding will typically force smaller values for $d(l_1, v)$ and $d(t, l_2)$. Therefore, the best strategies for dual landmark ALP will seek to maximize $d(l_1, l_2)$ for any point in the search while minimizing $d(l_1, v)$ and $d(t, l_2)$.

The following subsections detail how the embedding methods typically used in ALT can be applied to ALP and the theoretical details of their impacts when using the dual landmark heuristic. Each of these algorithms rely on a partitioning of the graph that attempts to minimize the relative number of edges between partitions in comparison to the number of edges within partitions. These landmark selection algorithms are designed with partitioning configurations generated by algorithms such as the Louvain algorithm (Blondel et al., 2008) that maximize modularity amongst graph partitions in mind. Such an algorithm can produce partitions that are dense in their number of edges, inherently reducing preprocessing time and presenting an optimal scenario for higher heuristic calculations.

Random Landmark Selection

The baseline strategy for ALP, just as with ALT, is random landmark selection. Two landmark selection methods for ALP are attempted in this work. Both algorithms take in a graph topology (including partitioning information) as their parameter and randomly, with uniform distribution, designates a single vertex within each partition as a landmark vertex. This is where the first algorithm, *random-p*, stops. The landmarks used by ALP are the landmarks that were selected. The second algorithm much, like the ALT variant, continues with an initial set of test queries to ensure good landmarks have been chosen. For a number of trials *k*, we compare the average search space size of these each trial. The landmark configuration with the lowest search space size is the final landmark configuration that will be used by ALP. Note that the partitioning is considered a part of the graph topology and will not be changed during this selection process. This second landmark selection algorithm is denoted *random-opt*. The pseudocode for both of these algorithms follow:

```
Random-p($G = (V,E)$)
1.    landmark_set <- list
2.    for each partition $\boldsymbol{H} \subset \boldsymbol{G}$
3.        Choose a random vertex $v \in \boldsymbol{V}(\boldsymbol{H})$
```

Figure 13 Random Landmark Selection

```
Random-opt($G = (V,E)$, num_trials)
1.    landmark_set <- list
2.    for each partition $\boldsymbol{H} \subset \boldsymbol{G}$
3.        $v$ = ALT_Random(H, num_trials)  //Perform ALT random landmark selection
```

Figure 14 Optimized Random Landmark Selection

Farthest-d

Farthest-d landmark selection takes in, as parameters, a graph topology (including partitioning information). As with normal *farthest-d* selection, this landmark selection algorithm works as follows for ALP. Let $\{C_1, \ldots, C_n\} \in \boldsymbol{C}$ be the set of partitions in the input graph. Identify a start vertex $v \in V$ in partition C_i and find the vertex $v' \in V$ farthest, in terms of distance, in a partition C_j, away from it. Add v' to the set of landmarks. Then, proceed in iteration by finding the next vertex v'' in partition C_m farthest away from the current set of landmarks and adding v'' to the set. If, on a particular iteration, the next farthest vertex is in a partition that has a landmark designated to it, find the next farthest landmark in a neighboring partition that does not have a vertex in the set of landmarks. Continue until all partitions have an established landmark. Just as with ALT, this algorithm is denoted *farthest-d*.

Planar

The planar landmark selection algorithm is suited for ALP's use of partitioning. This landmark selection algorithm uses graph layout information to divide a graph into sectors[6]. Each of these sectors is the respective graph partition for ALP. For dual landmark ALP, we leverage the partitioning algorithm described in the next section to implement planar landmark selection. By referencing the partition as sectors, the landmark for each partition will be selected by identifying the set of vertices within that partition with maximum eccentricity. If multiple vertices within the partition have the same eccentricity, one of them is chosen at random to be added to the set. In other words, we will identify the set of vertices from which the distance to all other vertices within its partition is maximal. For each sector, this typically is the farthest vector from any center node. This algorithm is known as *planar*.

[6] Planar landmark selection for ALP does not assume graph itself is planar.

```
planar($G = (V,E)$)
1.    landmark_set <- list
2.    for each partition $\boldsymbol{H} \subset \boldsymbol{G}$
3.        Compute the eccentricity of H
```

Figure 15* ALP *Planar Landmark Selection

Betweenness Centrality-Based

Betweenness centrality is a preferred method for choosing landmarks in ALT. For ALP, this landmark selection algorithm iterates through each partition in the graph. For each partition, we induce a subgraph G' from the vertices in the partition. The vertex with the largest betweenness centrality in G' is designated as the landmark for that partition. If G' is not connected, the largest connected subgraph of G' is used to compute betweenness centrality and for landmark identification. This algorithm is known as *betweenness*.

New Landmark Selection for ALP

Here, we discuss landmark selection techniques not based on those from ALT research.

Centrality-Based Landmark Selection

Here, we detail a new landmark selection method, based on PageRank (Brin & Page, 1998). We will identify this selection technique as *PageRank-P*. Landmarks need to be created such that the likelihood of passing through a landmark on a path in the graph is maximized while ensuring that landmarks are not too close to each other. Therefore, the probability of encountering a vertex during a random walk of each subgraph H_i generated by a partition $C_i \in \boldsymbol{C}$ can be used to decide which vertex in the subgraph will be a landmark. The PageRank algorithm, an eigenvector centrality computation, requires $O(n+m)$ time to compute a PageRank vector for a graph. (Han, Lee, Pham, & Yu, 2010) Each subgraph induced by each partition has a basic PageRank calculation run on it. For k partitions, k PageRank vectors will be computed. The vertex with highest PageRank in its partition (and its respective vector) is chosen as the landmark for that partition. As with *betweenness,* if the partition is disconnected, the PageRank calculation will be run on the largest connected subgraph of the partition and a landmark will be chosen from that.

Formally, let k be the number of partitions in G and $L \subset V$ is the set of landmarks. The goal is to compute the set L of size k. For each partition C_i, $1 \leq i \leq k$, and its induced subgraph H_i, a landmark $l_i \epsilon L$ is chosen by the following equation[7]:

$$l_i = \max_{V_j \in H_i} \frac{1-\mathrm{d}}{\mathrm{N}} + \mathrm{d} \sum_{V_k \in M(V_j)} \frac{\mathrm{PR}(\mathrm{V}_j)}{\mathrm{L}(\mathrm{V}_k)} \tag{80}$$

where V_j represents a vertex in H_i, N the number of vertices in C_i, d a dampening factor, $M(V_j)$ the set of vertices that link to a page V_j, $L(V_k)$ the number of outbound links from V_k, and PR(Vj) the PageRank of V_j. This selection technique probabilistically chooses appropriate landmarks with comparable computational speed in comparison to the others.

During experimentation, for PageRank, we establish two more landmark selection techniques, where we choose landmarks with the minimum and mode scores, as well. These techniques are denoted *PageRank-Min* and *PageRank-Mode*. Further, the same paradigm is used for the following centrality measures: Closeness centrality, Load centrality, and Katz centrality (Freeman, 1979; Goh, Kahng, & Kim, 2001; Katz, 1953; Newman, 2001). We denote these as *closeness*, *load*, and *katz*, respectively.

[7] Just as in the other landmark selection methods, we determine partitions here using the Louvain method.

The *closeness centrality* of a particular landmark is simply the reciprocal of its *farness*, which is the sum of all distances from all other nodes. Therefore, using the notation above, closeness landmark selection chooses a subgraph's landmark using the following equation:

$$l_i = \max_{V_j \in H_i} \frac{1}{\sum_{x \in V_j} d(l_i, x)} \tag{81}$$

Load centrality is a variant of betweenness centrality in that it is defined through a hypothetical flow process. The score for an individual node is the fraction of all shortest paths that pass through that node. Using the notation for betweenness centrality from Chapter 2, for a vertex $v \in V$, let $\sigma_{st}(v)$ denote the number of shortest paths from s to t containing v. Also, let $|P(s,t)|$ simply denote the total number of paths from s to t. Then betweenness centrality of v is formally defined as

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{|P(s,t)|} \tag{82}$$

Katz centrality is similar to eigenvalue centrality and PageRank measures. It computes centrality scores by measuring the number of first degree vertices and all other vertices that connect to the vertex under consideration through these immediate neighbors.

Centrality measures are an intuitive way of keeping the distances among the landmark set for ALP large relative to the distances between landmarks and the vertices they own.

Farthest-ECC

The Farthest-d algorithm for ALT is feasible for the small number of landmarks supported by the algorithm. However, with ALP, many more landmarks are able to be selected. Attempting to run this many shortest path computations becomes intensive and reduces ALP's preprocessing benefits. Ideally, identifying nodes with maximum eccentricity within each partition would be the optimal approach. But this does not address the computational intensity problem. Therefore, another method was identified for attempting to find landmarks in the distributed embedding environment that were farthest away from the other landmarks. This version of farthest seeks to identify landmarks within each graph partition that are farthest away from a sample set of nodes, chosen through a uniform random distribution, in the graph. To do this, we first reverse the graph, so that we are computing distances to each landmark. A set of nodes within each subgraph, also chosen through uniform random distribution, grow their shortest path trees out to the full graph's sample set. The node within each subgraph that has the maximum distance from the full graph's sample set of nodes is chosen as the landmark. The goal of this version of farthest, dubbed *farthest-ecc*, was too maximize $d(l_2, l_1)$ such that it would unbalance the heuristic estimates, providing the largest possible guesses, especially over long distances.

Validating and Verification

We end this Chapter with an overview of two experiments used to validate and verify the claims made in the methodology. In order to characterize the practical performance of ALP, experiments with both real world and synthetic data must occur. The main goals of experimentation were to verify ALP's relatively smaller preprocessing (for both time and space), validate its behavior in the context of ALT, and gain insight into the benefits and detriments of using one algorithm over another. They also establish the validity and utility of the ALP algorithm in comparison the ALT algorithm.

Experiment 1: Performance and Bounds

To understand how to perform optimal landmark selection in ALP, the algorithm's basic behavior must be defined. The only way to do this is in the context of another landmark-based class of algorithms, ALT. Therefore, Experiment 1 was an initial investigation of the ALP dual landmark heuristic's behavior and its performance bounds based on the scenarios defined earlier in the chapter for ALT. For the base implementations, comparison between ALT and ALP using the experimental benchmark road data from Maue's PCD research and Goldberg's ALT research occurred. Random selection was used for a series of controlled trials comparing the two algorithms on these datasets. To initially test ALP's heuristics, the algorithm will first be tested without distributed embedding. An implementation with distributed embedding will be created after initial testing. The Louvain algorithm (Blondel et al., 2008) will be used for the partitioning of the graph.

After initial testing, the ALP heuristic was exercised such that its computational bounds can be verified. This experiment sought the parameters that maximize and minimize ALP's computational performance and memory requirements. Using scenarios defined in this chapter, we were able to identify the optimal conditions for the heuristic, when it breaks even with the ALT heuristic, and its worst performance conditions. By the end of Experiment 1, a full characterization of the performance bounds of ALP algorithms against ALT algorithms was derived.

In this chapter, we have demonstrated that the advantage of using the ALP heuristic is that it practically admits more landmarks than ALT and performs faster landmark selection over the same number of landmarks. However, during query time, over the same set of landmarks, ALT dominates ALP (though ALT requires more space to store landmark distance information). The results of trials generated during this experiment also generate further characterizations of the algorithms to guide later application, as well as informing how the algorithm compares to other metric-independent preprocessing algorithms.

Experiment 2: ALP vs. ALT

Experiment 2 fulfilled the key contribution for this dissertation by identifying optimal landmark selection techniques for dual landmark ALP with distributed embedding. This experiment sought to arbitrate between each of the aforementioned algorithms for landmark selection in the ALP environment. Each technique was vetted using a common partitioning algorithm for multiple graph datasets, both real and synthetic. Like PCD, the way that the graph partitions are shaped and the actual partitioning is not determined by the algorithm (J Maue et al., 2006). For this approach, we continued to leverage an extremely fast algorithm for partitioning graphs known as the Louvain algorithm (Blondel et al., 2008). This algorithm relies on maximizing modularity within a graph, ensuring that there is a significantly higher proportion of edge connections within partitions than between partitions. It has become a standard algorithm for community detection in graphs and, as such, will lend a significant demonstration and characterization for ALP's behavior to this common type of input.

Summary of Experiments

The table below summarizes each of the experiments in this dissertation. Experiments are described in much further detail in the next chapter.

<u>Experiment 1</u> **ALP Performance Bounds**	Goal	Investigate and understand the computational bounds of ALP dual landmark heuristics in comparison with ALT
	Research Questions	• Using ALP with distributed landmark embedding, what are the ideal characteristics for landmark shortest path trees? In other words, how much preprocessing and memory is required for ALP to maintain its key benefits? • How does the algorithm behave as the number of landmarks used to guide the search increases? • What landmark selection techniques theoretically fit best with ALP?
<u>Experiment 2</u> **ALT vs ALP**	Goal	Compare and contrast the ALP and ALT algorithms to characterize utility
	Research Questions	• What are the key benefits of using the (dual landmark) ALP heuristic over the ALT heuristic when performing shortest path queries? • In what ways can this be applied to path planning? • What real-world applications exist for ALP that did not exist for ALT?

Table 5 Dissertation Experiments

Once sufficient data was collected from the first experiment, Experiment 2 trials were carried out with guidance from the results of Experiment 1. Each experiment underwent more than 10^6 trials to sufficiently compare and characterize the two algorithms under experimentation. Each experiment relied on available data used to characterize the other metric-independent preprocessing algorithms mentioned in the literature review, as well as benchmark models common to modern graph libraries. This ensured that the experiments that are performed here can be replicated and validated upon publication.

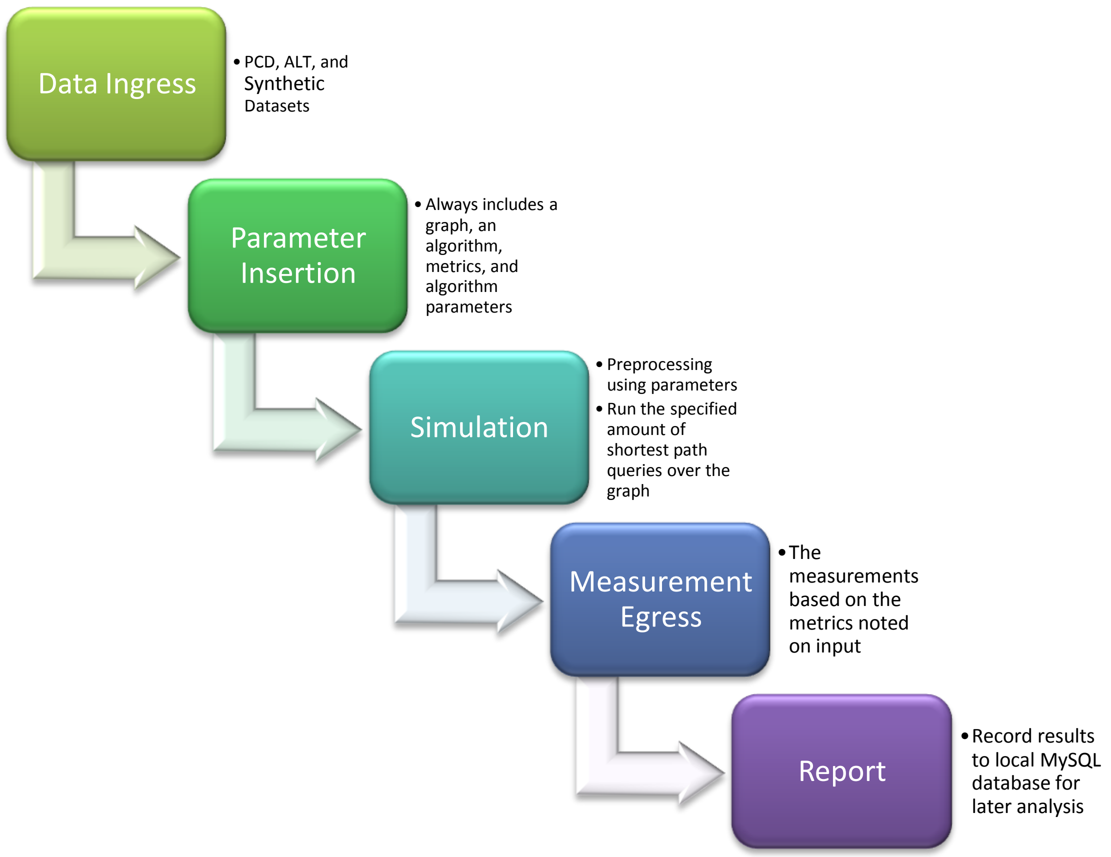

Data Ingress
• PCD, ALT, and Synthetic Datasets
Parameter Insertion
• Always includes a graph, an algorithm, metrics, and algorithm parameters
Simulation
• Preprocessing using parameters
• Run the specified amount of shortest path queries over the graph
Measurement Egress
• The measurements based on the metrics noted on input
Report
• Record results to local MySQL database for later analysis

Figure 16 The flow of each trial during Experimentation

The trials run for each experiment followed the flow shown in Figure 16. Data for the particular experiment is loaded into memory. All information regarding the structure and characterization of this data were previously recorded. The specific parameters for a given trial will then be established. During simulation, these parameters are used for searching over a user-specified number of shortest path queries on the particular dataset. A measurement harness monitors the simulation to extract information related to the specified metrics for preprocessing and shortest path queries. Finally, the measurements gathered by the harness will be sent to a relational database that will be used for analysis and to draw conclusions.

Summary

This chapter describes the foundations of a class of algorithms that reduce the amount of preprocessed information necessary to perform preprocessed shortest path queries. A new class of algorithms is presented for solving shortest path queries using the A* algorithm, landmarks, and polygon inequalities (ALP). Its novel feature is that it computes and stores a reduced amount of preprocessed information while making more informed search decisions. This new heuristic is applied by using distance information about two landmarks in a single query to guide the A* algorithm from a source node to a destination node. A new paradigm for landmark selection, known as distributed embedding, is proposed for this heuristic. Using this process for shortest path search reduces the amount of preprocessed information that needs to be stored while also reducing the level of computation required at each step of the search. In a fixed space environment, ALP has the potential to have more informed searches than ALT, as it is able to leverage more landmarks. Domination of one heuristic over the other depends on the landmark set each is assigned and, in general, the denser the landmark set, the better the heuristic. While ALP theoretically does not dominate the ALT heuristic, the ALT heuristic, in turn, does not dominate it. In Chapter 4, we will establish, through experimentation, that in cases in which the ALT heuristic has greater average estimates than the ALP dual landmark heuristic, ALP can still computationally outperform ALT. Therefore, a key contribution of this effort will be the analysis of scenarios in which this heuristic and its competitors should be used. This will give guidance to future users of shortest path algorithms.

Chapter 4

Results

This chapter provides an objective description and analysis of the findings, results, and outcomes of the research. The experiments for the dissertation are described in detail. The trials conducted in each of these experiments were strongly motivated by previous studies for ALT (Fuchs, 2010; A. Goldberg & R. Werneck, 2005; Goldberg & Harrelson, 2005; Potamias et al., 2009; Takes & Kosters, 2014). In this chapter, the use of charts, tables, and figures are limited to those that are needed to support the final conclusions. All other illustrations and summary data can be found in the appendices. The Data Analysis section describes the methods of collecting the data and summaries of what has been collected, pointing out ambiguities, inconsistencies, patterns and themes in the data. In the Findings section, the results described in the Data Analysis section are synthesized in light of the dissertation's research questions, literature review, and methodologies. In the Summary section, the research questions posed in Chapter 1 are explicitly answered by summarizing the Data Analysis and Findings sections, enumerating the theoretical and practical implications of the information relayed by those sections.

In this Chapter, experimentation with ALP, using two landmarks for distance estimation, compares the class of algorithm's performance and benefits against ALT, the class of algorithms from which it was derived. This experimentation also fully characterizes the heuristics, identifies the optimal, average, and worst-case input parameters, and thoroughly compares the dual landmark ALP algorithm to its predecessor, the ALT algorithm. Experiments are initially performed on synthetic graph datasets to characterize the algorithm's performance based on structure. Then, benchmark datasets that have been called out in academic literature, based on city and state maps, are used for applied characterization. Experiment 1 resulted in a characterization of the performance of ALP as a heuristic for A* with regard to graph structure and landmark selection. Experiment 2 highlights differences in performance of ALP and ALT as heuristics for A*, with final trials for the experiment simulating the comparative behavior of both algorithms in a fixed-memory environment. The combined theoretical and experimental characterization of this algorithm offers the Computer Science community insight into the applications of the algorithm in other spaces. In the end, a shortest path analysis software library, the theoretical and experimental characterizations of ALP, and data sufficient to evidence the innovative claims of this dissertation are contributed.

Data Analysis

This section describes the implementation of the ALP experimentation environment, the datasets used for experimentation, and the metrics used for characterization. 9,653 trials, each corresponding to at least 1,000 shortest path queries were run to vet the performance and bounds of the ALP algorithm, landmark selection in its environment, and how local/global optima of its performance compares to that of ALT. In total, over 1.84×10^7 shortest path queries were answered by the experimental testbed. The data that is analyzed in this section is derived from these queries. Table 6 summarizes the experiment sessions, trials, and queries performed for the experiments in this dissertation.

	Road Graph Queries	Synthetic Graphs Queries	Total Queries	Landmark Selection Techniques Attempted
Dijkstra's	4,144,759	2,826,206	6,970,965	N/A
ALT	3,258,983	1,321,295	4,580,278	5
ALP	4,068,893	2,826,097	6,894,990	13

Table 6 Summary of Experimental Runs

Datasets

Experiments were run on multiple classes of synthetic graphs and graphs of real road networks. Shown in Table 7, the synthetic graphs used for experimentation have structures that model data across many fields of study. The use of these graphs allowed us to experimentally glean how ALP can behave in different environments, and not simply during road navigation. The number of nodes and edges is not included in Table 7 as a parameter for these graphs, as they vary throughout experimentation.

Descriptions and further details about the structure of each graph are found in Appendix A. In-depth detail about the number of nodes and edges that provided specific

Table 7 Synthetic Graph Problem Families

Name	Graph Type	Graph Parameters	DB Name
M1	Barabási–Albert (BA) model	Preferential Attachment = 2 Edges/Node	NETWORKX.BARABASI_ALBERT_2
M2	Barabási–Albert (BA) model	Preferential Attachment = 3 Edges/Node	NETWORKX.BARABASI_ALBERT_3
M3	Barabási–Albert (BA) model	Preferential Attachment = 5 Edges/Node	NETWORKX.BARABASI_ALBERT_5
M4	Barabási–Albert (BA) model	Preferential Attachment = 7 Edges/Node	NETWORKX.BARABASI_ALBERT_7
M5	Barabási–Albert (BA) model	Preferential Attachment = 9 Edges/Node	NETWORKX.BARABASI_ALBERT_9
M6	Barabási–Albert (BA) model	Preferential Attachment = 11 Edges/Node	NETWORKX.BARABASI_ALBERT_11
M7	Barabási–Albert (BA) model	Preferential Attachment = 13 Edges/Node	NETWORKX.BARABASI_ALBERT_13
M8	Barbell Graph	Equivalent Number of Nodes on each side	NETWORKX.BARBELL_GRAPH_EVEN
M9	Barbell Graph	2/3 Nodes on Left Barbell, 1/3 Nodes on Right Barbell	NETWORKX.BARBELL_GRAPH_ODD
M10	Circular Ladder Graph		NETWORKX.CIRCULAR_LADDER_GRA
M11	Complete Graph		NETWORKX.COMPLETE_GRAPH
M12	Cycle Graph		NETWORKX.CYCLE_GRAPH
M13	Erdős–Rényi model	Edge Creation = 15%	NETWORKX.ERDOS_RENYI_15
M14	Erdős–Rényi model	Edge Creation = 30%	NETWORKX.ERDOS_RENYI_30
M15	Ladder Graph		NETWORKX.LADDER_GRAPH
M16	Path Graph		NETWORKX.PATH_GRAPH
M17	Random Lobster	Pbackbone=45%, PBeyondBackbone=45%	NETWORKX.RANDOM_LOBSTER_45
M18	Random Lobster	Pbackbone=90%, PBeyondBackbone=90%	NETWORKX.RANDOM_LOBSTER_90
M19	Watts–Strogatz model	10% nearest neighbor connections, 10% Prewiring	NETWORKX.WATTS_STROGATZ_10
M20	Watts–Strogatz model	20% nearest neighbor connections, 20% Prewiring	NETWORKX.WATTS_STROGATZ_20
M21	Waxman Graph	alpha=0.4,beta=0.1,domain=(0,0,1,1)	NETWORKX.WAXMAN_GRAPH

results of analysis on these graphs can be found in Appendix C.

Graph Type	Average Transitivity	Average Clustering Coefficient
NETWORKX.BARABÁSI_ALBERT_2	2.90E-02	0.06
NETWORKX.BARABÁSI_ALBERT_3	6.49E-02	0.10
NETWORKX.BARABÁSI_ALBERT_5	7.97E-02	0.10
NETWORKX.BARABÁSI_ALBERT_7	5.82E-02	0.07
NETWORKX.BARABÁSI_ALBERT_9	1.33E-01	0.15
NETWORKX.BARABÁSI_ALBERT_11	1.48E-01	0.16
NETWORKX.BARABÁSI_ALBERT_13	1.62E-01	0.17
NETWORKX.BARBELL_GRAPH_EVEN	9.96E-01	0.67
NETWORKX.BARBELL_GRAPH_ODD	9.98E-01	0.80
NETWORKX.CIRCULAR_LADDER_GRAPH	0.00E+00	0.00
NETWORKX.COMPLETE_GRAPH	0.00E+00	0.00
NETWORKX.CYCLE_GRAPH	0.00E+00	0.00
NETWORKX.ERDOS_RENYI_15	1.51E-01	0.16
NETWORKX.ERDOS_RENYI_30	3.04E-01	0.30
NETWORKX.LADDER_GRAPH	0.00E+00	0.00
NETWORKX.PATH_GRAPH	0.00E+00	0.00
NETWORKX.RANDOM_LOBSTER_45	0.00E+00	0.00
NETWORKX.RANDOM_LOBSTER_90	0.00E+00	0.00
NETWORKX.WATTS_STROGATZ_10	9.36E-02	0.10
NETWORKX.WATTS_STROGATZ_20	4.12E-01	0.42
NETWORKX.WAXMAN_GRAPH	7.97E-02	0.08

Table 8 Average Synthetic Graph Transitivity and Local Clustering Coefficient

Each of these structures varies in terms of several main properties. In experimentation, we specifically focus on their *average clustering coefficient* and *transitivity*, as shown in Table 8. The *clustering coefficient* of each vertex in a graph is the fraction of triangles connected to the vertex divided by its number of *triples*, or sets of two edges connected to the vertex. Therefore, the average clustering coefficient for a graph is the mean clustering coefficient over all vertices. *Transitivity* is a relative measure of the number of triangles in a graph divided by the total number of connected triples of nodes. Transitivity is also known as the global clustering coefficient of a graph. Average clustering coefficient and transitivity measures give strong indications of the clustering of vertices in the graph. They are significant to the findings in this effort as distributed embedding relies on a partitioning of the graph and the partitions used in these experiments (primarily provided by the Louvain method) are strongly dependent on these properties (Soundarajan & Hopcroft, 2015).

Summary information for the real road graphs that were used in experimentation is shown in Table 9. These graphs were taken from datasets used in the 9th DIMACS Implementation Challenge (Demetrescu et al., 2006). This is a benchmark dataset for much of the shortest path research that occurs in academia at the time of this writing. These datasets allowed for testing of ALP's behavior on directed graphs. In some cases, for testing purposes, we executed trials using real road graphs as undirected graphs. The differences are noted when reporting summary data.

In general, a vertex in these graphs represents a single intersection of two roads and an edge represents a road segment. While many previous research efforts with ALT

Table 9 Road Graph Problem Families

Description	**# Vertices**	**# Edges**
Pennsylvania	1,087,562	1,541,514
Rome	3,353	4,831
Belgium	746,333	767,786
Luxembourg	84,136	85,579
NYC	264,346	365,050
Washington DC	9,599	14,909
Rhode Island	53,288	68,496
United States (Western)	6,262,104	15,248,146
United States (Central)	14,081,816	34,292,496
United States (Eastern)	3,598,623	8,778,114
United States (Bay Area)	321,270	800,172
Hawaii	64,892	76,809
Great Lakes	2,758,119	6,885,658
New Mexico	467,259	567,084

(and other shortest path preprocessing methods) required the dataset to be processed using only subgraphs of the roadmap, the datasets used in this effort could be used in their entirety when experimenting with ALP. Subgraphs are only used in ALP during experimentation to increase the number of trials, not because of computational hardware limits. Cases in which subgraphs are used are noted in the experiment data. For all datasets, we analyze the graph's largest *strongly connected component*, or the induced subgraph in which all vertices can reach all other vertices.

In the context of the original work, for each query, source-target pairs among all vertices are chosen at random using a uniform distribution (Goldberg & Harrelson, 2005). Testing queries with path lengths uniformly distributed from zero to the diameter of the graph was necessary in order to adequately characterize the behavior of each algorithm in each graph. Because the source-target pairs in our runs are chosen with uniform random distribution, path lengths span the possible distances of the graph.

For each experiment, a series of trials was run over these synthetic and road graphs at various scales to vet the overall performance of both ALT and ALP. A trial describes a specific configuration of parameters for a set of shortest path queries. Over 1000 variations of synthetic graphs, as well as over 100 different subgraphs of real road datasets were used. The two tables shown in Figure 17 categorize each class of graph used during experimentation by size. Each graph instance falls under categories that are deemed *vertex scales* and *edge scales*. These scales are defined by lower and upper bounds for the number of vertices and edges contained in a single graph, respectively. Performance of the shortest path preprocessing algorithms is vetted for each of these vertex and edge scales.

Implementation

The implementations used for each experiment were based on the pseudocode and descriptions in Chapters 1-3. Experimentation was carried out under a 64-bit CentOS 7 instance on a custom-built server, which has 8 GB of RAM and a 2.20GHz Intel(R) Core(TM) 2 Duo CPU E4500 processor. An additional 40GB of swap space was allocated on the server. Of note, this swap space was never tapped for ALP processing for large scale graphs and regularly tapped for ALT.

For the software implementations, all experimentation for ALT and ALP was implemented using Python. The synthetic graphs for these experiments are generated by the NetworkX library (Developers, 2010) using Python 2.7. NetworkX's scripts for pathfinding (A*, Dijkstra's algorithm) were instrumented such that metrics such as search space size could be recorded for each query. The library was also extended by adding a capability to only grow a Dijkstra SPT until it covers a desired set of vertices. This capability serves preprocessing in both the ALT and ALP environments. The NetworkX source code for the A* algorithm was duplicated and modified such that the pathmax equation was used by default to force consistency.

For smaller graphs (V1-V4), to map vertices to their corresponding landmarks and partitions, we use NetworkX's vertex labeling mechanisms to give each vertex an attribute called "ALP_<*landmark_id*>" with a value of its distance from its partition's

Category	# Vertices	# Experimented Graphs
V1	1-100	2098
V2	101-1000	315
V3	1001-5000	133
V4	5001-20000	85
V5	20001-100000	92
V6	100001-250000	1
V7	250000-1000000	4

Category	# Vertices	# Experimented Graphs
E1	1-100	1375
E2	101-1000	812
E3	1001-5000	170
E4	5001-20000	146
E5	20001-100000	131
E6	100001-250000	40
E7	250000-1000000	35

Figure 17 Vertex and Edge Graph Scales

landmark. For larger graphs (V4-V7), we use separate Python dictionaries as data structures for ALT and ALP, respectively to address memory issues[8]. For ALP, three separate dictionaries serve the following functions:

(1) Relating a vertex to its reference landmark
(2) Storing the distances from all landmarks and vertices of the subgraph owned by a landmark to that landmark
(3) Storing the distances to all landmarks and vertices of the subgraph owned by a landmark from that landmark

For ALT, only two dictionaries are needed[9] that serve the functions of storing vertex distances to and from landmarks, respectively.

Unless otherwise specified in this chapter, a NetworkX implementation of the Louvain method was used for graph partitioning (Aynaud, 2010; Blondel et al., 2008). Other partitioning methods that grant the flexibility of creating a desired number of partitions are used and described later in the Chapter for specific trials.

For experimentation with larger graph datasets, NetworkX objects under Python proved to be too large to run on the basic experimentation server. Because of this, Cython was used to convert modified NetworkX shortest path algorithms, all preprocessing algorithms, and all querying mechanisms to C code (Behnel et al., 2011; Summerfield, 2013; Surhone, Tennoe, & Henssonow, 2011). Using GCC 4.9.2, the running binary for this code was optimized to run each trial for the experiments (Griffith, 2002). The following GCC flags were used:

[8] When attempting to use NetworkX labeling, a dictionary is populated for every node, creating substantial overhead in the case of large graphs.
[9] These grow to become much significantly larger than ALP's dictionaries because they must store landmark distance information for each landmark to and from all other vertices in the graph.

```
gcc -flto -fuse-linker-plugin -Ofast -fivopts -fdata-sections -floop-
parallelize-all -ftree-parallelize-loops=4 -funroll-loops -mtune=native
-march=native -I/usr/include/python2.7
```

Figure 18 GCC Optimizations for Large Graph Runs

The optimizations are tailored toward the server processor and are focused as much as possible on speed, not the size of the resulting binary executable. Substantial efficiency increases stemmed from the combination of the conversion to C code and the optimizations for GCC.

Appendix B details the structure of our data storage for queries and trials.

Metrics

Throughout this chapter, the following metrics are used to characterize ALP as an A* heuristic and to compare and contrast it with ALT. *Efficiency* is the primary metric identified by the creators of ALT to measure query performance (Goldberg & Harrelson, 2005).[10] The average efficiency over a set of shortest path queries is used to characterize a heuristic. Recall that the *search space size* is the number of vertices visited to discover the shortest path. The efficiency of a single query is computed as follows:

$$Dijkstra\ efficiency = \frac{|P(s,t)|}{|V(s,t)|} \qquad ALT\ efficiency = \frac{|P(s,t)|}{|V_L(s,t)|}$$

$$ALP\ efficiency = \frac{|P(s,t)|}{|V_{DL}(s,t)|}$$

In other words, the efficiency is defined as the number of vertices on the shortest path divided by the number of vertices explored by the search for a single query. An optimal heuristic would have 100% efficiency. For example, for ALP, a perfect search would mean that $|P(s,t)| = |V_{DL}(s,t)|$. This is a machine and scale independent method

[10] We call this measure "efficiency" because of its use in the original ALT publications.

of understanding ALP performance. We use this metric throughout both experiments for evaluating shortest path algorithm performance.

To further identify utility of each algorithm, the *tradeoff* metric is used to identify the utility of using each algorithm over a user-defined number of queries. Tradeoff is calculated as follows:

$$t(n) = t_0 + nt_q$$

where $t(n)$ is the time to process *n* queries, t_0 is the preprocessing time, and t_q is the average time (in seconds) to process is each query. Note that this makes tradeoff an application-based metric which can vary based on the number of queries being executed. *Preprocessing time* is the physical time in seconds that it takes to actually run a landmark selection algorithm plus the time that it takes to actually grow the shortest path trees for each landmark. In general, a good heuristic brings tradeoff values as close to zero as possible. It is a machine and implementation-dependent metric that complements efficiency to provide better understanding of practical performance for ALP and other shortest path algorithms that require preprocessing.

For some analysis, we take a look at the number of landmarks used for a particular landmark configuration and the average efficiency of a run with that landmark configuration respectively as (x, y) coordinates. This allows us to measure the *performance gain* that stems from growing the number of landmarks by computing the slope of these coordinates. Here, we define performance gain as a simple measure of how the performance of ALP or ALT increases as the number of landmarks increases.

Approximation error is another common metric used in the literature for ALT to understand the efficacy of an embedding on the graph. For a given query, approximation error is defined as follows:

$$approximation\ error = \frac{\left|d(s,t) - \pi_t^{\boldsymbol{DL}}(s)\right|}{d(s,t)}$$

The approximation error for Dijkstra's algorithm is always 1, as Dijkstra's algorithm is equivalent to A* with a zero heuristic. Like efficiency, it is a measure of the quality of a heuristic. The two numbers are typically proportional to each other. However, both average efficiency and average approximation error are needed to measure the quality of a heuristic. For instance, if a heuristic were to only make good estimates at key waypoints in a larger graph, the average efficiency from such a heuristic would be large while the average approximation error would be large, as well. A good heuristic keeps average efficiency large and approximation error small. Approximation error is a good indicator of a heuristic being applicable across many datasets. In summary, efficiency is a good measure of a heuristic's quality for shortest path search (performance) while approximation error is a good measure of a heuristic's quality for estimating distance in a metric space (utility).

To recap, the metrics used to characterize performance during experimentation were:

- *Efficiency*
- *Tradeoff*
- *Performance Gain*
- *Approximation Error*

Experiment 1: ALP Performance and Bounds

This section describes the activities carried out in Experiment 1. Experiment 1 sought to characterize the performance and bounds of ALP with distributed embedding as a heuristic for A* in the experimentation environment described above. The implementations and schemas used in this first experiment established an operational experimentation environment for shortest path preprocessing. Note that highly-detailed, supplemental or extra interesting data from all experimentation can be found in Appendix C.

Description of Trials

Each trial tested a variety of graph configurations and parameters for ALP such that its computational bounds could be identified. Unless otherwise noted, we leveraged optimized random landmark selection to select landmarks for ALP. For every query, we also ran Dijkstra's algorithm as A* with a zero heuristic for a consistent sanity check and basis of comparison. The results of Dijkstra's algorithm queries are recorded, as well[11]. In this experiment, we looked at scenarios from a variety of vantage points, teasing out the performance and bounds of ALP. Table 10 briefly describes the types of trials, or sub-experiments that were run to vet ALT's performance and bounds. Results for Experiment 1 yield information about the performance and bounds for ALP in the context of each of these trials.

[11] For instance, we verify that path lengths are equal for both Dijkstra and ALP to ensure correctness of each algorithm. Also, if ALP has larger search space size than Dijkstra, it means overestimates have occurred.

Table 10 ALP Performance and Bounds Trials

Trial	**Description**
Varying Graph Structure	Characterize the efficiency and approximation error of ALP heuristic when run on 20 different synthetic graph structures as well as real road graphs.
Number of Landmarks	Identify the degree to which ALP performance increases as the number of landmarks used is increased.
Landmark Selection	Details performance of ALP for landmarks chosen through a set of landmark selection algorithms defined in Chapter 3.

Varying Graph Structure

In this set of trials, 1000 shortest path queries were run on each synthetic graph structure in the dataset using the ALP algorithm at all vertex and edge scales[12]. The number of landmarks that were used for each trial was always equal to the number of graph partitions for the input graph. The lowest number of partitions made available by the Louvain dendrogram was used for distributed embedding.

Table 11 describes the number of runs and average ALP efficiency for each graph class, shown in alphabetical order. For each type of synthetic graph, the efficiency at each vertex or edge scale was quite similar. We enumerate, in Appendix C, a set of tables that show every permutation of a graph structure against the average efficiency of queries on that graph. Here, we highlight noteworthy correlations between graph structures. Table 11 and Figure 19 are sufficient for examining ALP's behavior for different graph structures. These results imply that efficiency should grow in proportion to transitivity. Conversely, graphs such as path graphs, cycle graphs, ladder graphs, and random lobsters with zero transitivity (having no triangles), exhibit high efficiency rates, as well. Their high efficiency rates are due to the fact that the very structure of each graph significantly tightens the quadrilateral inequalities.

[12] We run these queries for each of the road graphs, as well. This data is found in the Appendix.

Graph Type	Average Efficiency	Average Approximation Error
NETWORKX.BARABÁSI_ALBERT_2	9.83%	83.36%
NETWORKX.BARABÁSI_ALBERT_3	16.22%	71.74%
NETWORKX.BARABÁSI_ALBERT_5	11.64%	73.63%
NETWORKX.BARABÁSI_ALBERT_7	7.35%	75.94%
NETWORKX.BARABÁSI_ALBERT_9	13.29%	71.99%
NETWORKX.BARABÁSI_ALBERT_11	13.66%	70.00%
NETWORKX.BARABÁSI_ALBERT_13	14.49%	70.10%
NETWORKX.BARBELL_GRAPH_EVEN	32.26%	54.53%
NETWORKX.BARBELL_GRAPH_ODD	24.09%	57.84%
NETWORKX.CIRCULAR_LADDER_GRAPH	41.20%	28.14%
NETWORKX.COMPLETE_GRAPH	9.18%	99.25%
NETWORKX.CYCLE_GRAPH	80.41%	22.51%
NETWORKX.ERDOS_RENYI_15	27.52%	65.73%
NETWORKX.ERDOS_RENYI_30	24.75%	62.63%
NETWORKX.LADDER_GRAPH	48.90%	16.30%
NETWORKX.PATH_GRAPH	93.34%	20.43%
NETWORKX.RANDOM_LOBSTER_45	61.91%	22.85%
NETWORKX.RANDOM_LOBSTER_90	42.50%	27.52%
NETWORKX.WATTS_STROGATZ_10	19.15%	69.36%
NETWORKX.WATTS_STROGATZ_20	14.87%	68.13%
NETWORKX.WAXMAN_GRAPH	4.76%	75.10%

Table 11 Efficiency and Approximation Error for Varying Synthetic Graph Structures

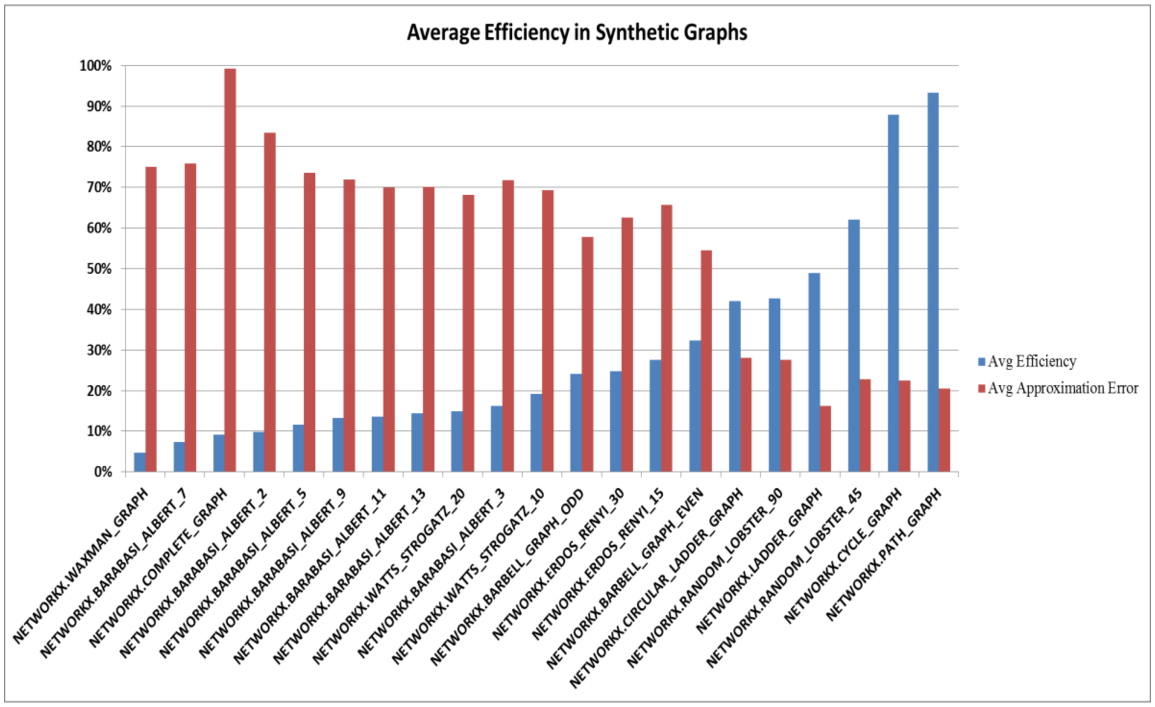


Figure 19 Average Efficiency and Error for Synthetic Graphs

Analysis of each of the experimental graph structures reveals that performance of ALP for these graph models does not seem to be a significant correlation between the transitivity or average clustering coefficient of the graph and the average efficiency of an ALP shortest path query (Figure 20). The only noticeable correlation is that when these structural properties tend to be zero, the efficiency gets closer to 100. Measures for both transitivity and average clustering coefficient are zero for ladder, circular ladder, random lobster, cycle, and path graphs. For each of those graphs, the prediction of where A* should move next is successful roughly 50% at each vertex visit.

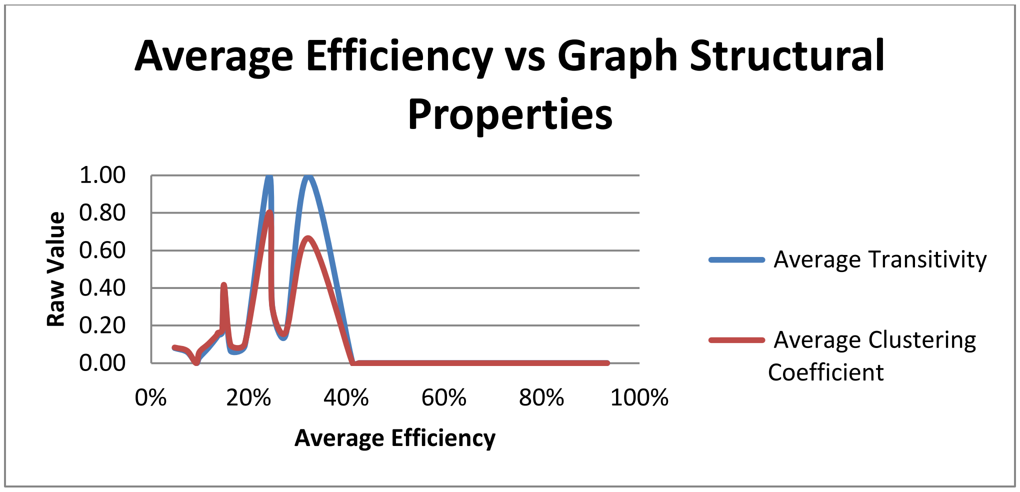


Figure 20 Average Efficiency of 1000 Queries vs Structural Properties of Graphs[13]

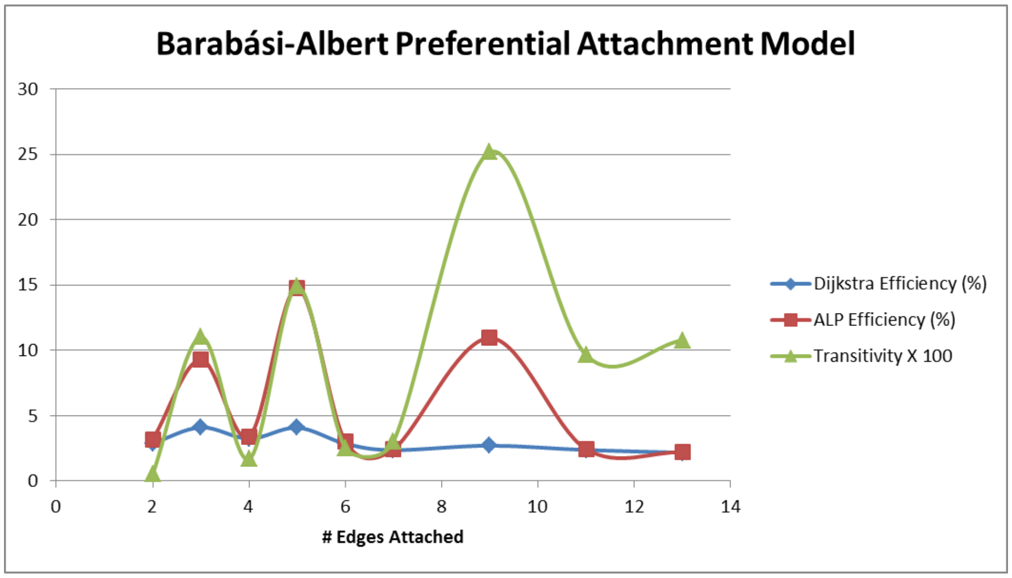


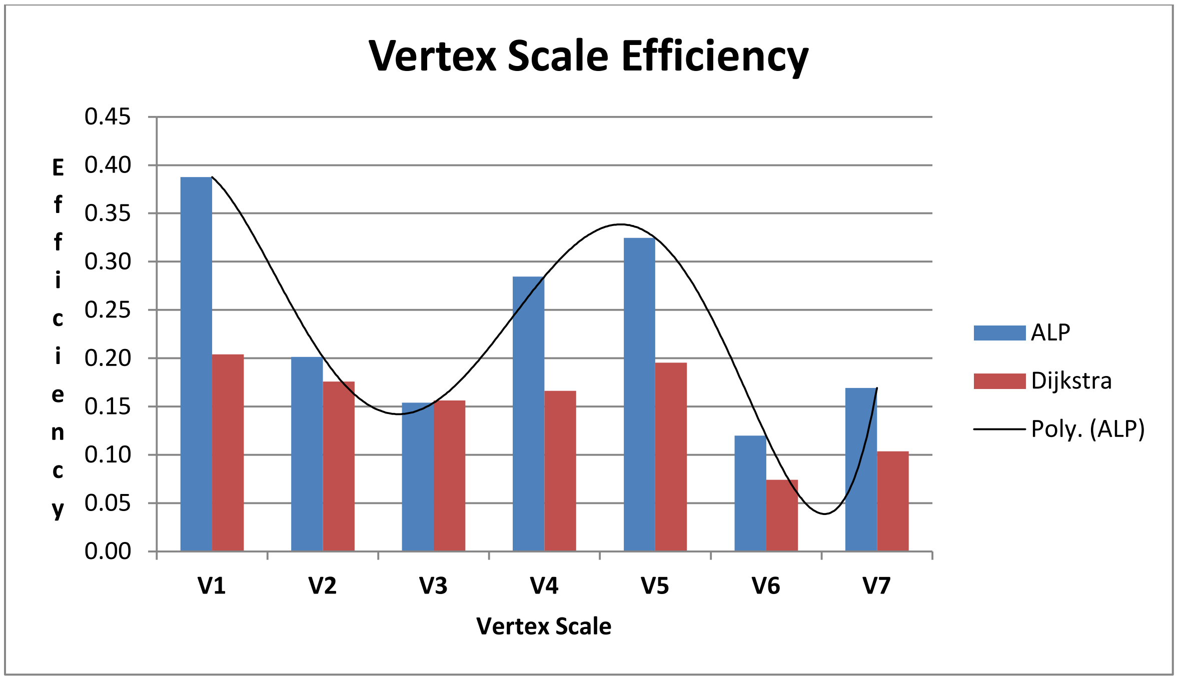
Figure 21 Graph of Efficiency measures for Dijkstra's Algorithm and ALP shortest path queries on Barabási-Albert preferential attachment graphs[14]

Figure 21 further highlights correlations by examining the relationship between transitivity, efficiency, and the parameters for generating the Barabási-Albert graph. In the figure, we multiply the transitivity by 100 to demonstrate its variability in relation to Dijkstra and ALP efficiency. We see that it varies in a way quite similar to ALP and

[13] Initial results show no immediate correlation between efficiency and the properties
[14] The green line on the plot shows the transitivity of each graph for the # of edges attached

Dijkstra’s efficiency for those graphs. ALP’s performance seems to depend on both transitivity and average clustering.

For each of the synthetic graph structures, Figure 22 illustrates the difference between ALP and Dijkstra over growing vertex and edge scales. These figures demonstrate that ALP’s efficiency decreases as the graph gets larger. This behavior is the same for ALT and Dijkstra’s algorithm, as well. This is why preprocessing as opposed to simply using Dijkstra’s algorithm becomes more valuable as graphs get larger. We simply note a decrease in efficiency as paths get larger, a fundamental property of the search shared by ALT. Results show that these measurements are not correlated in any meaningful way with respect to growing graph scale.



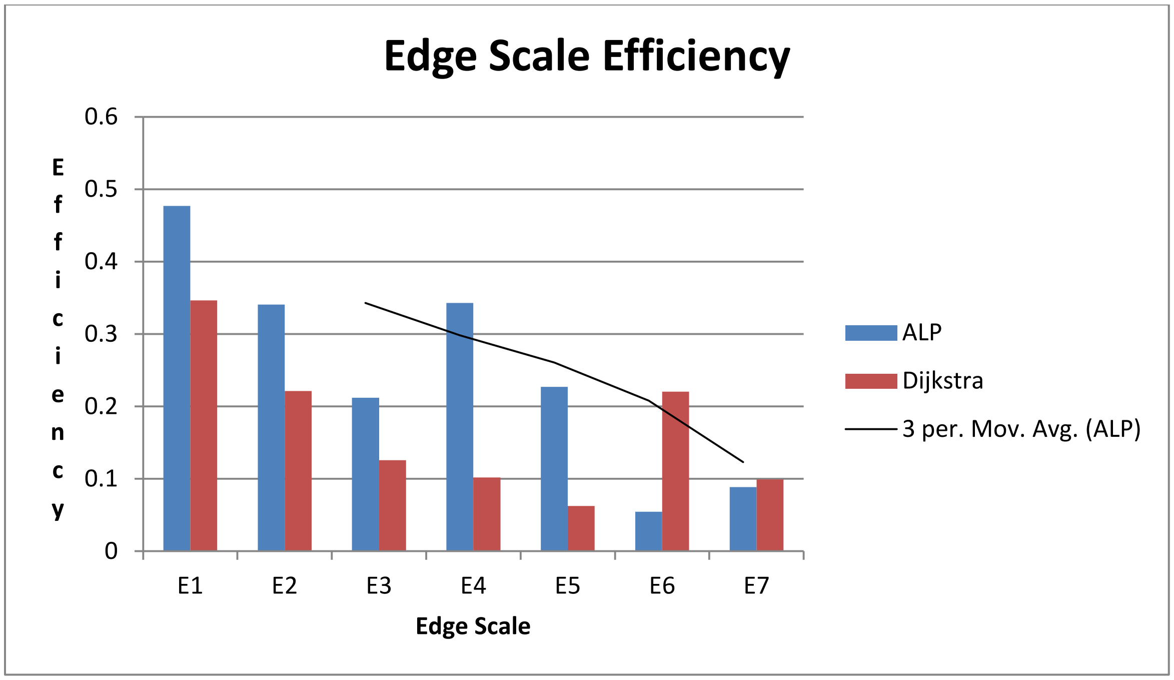


Figure 22 ALP Efficiency at each Graph Scale

Number of Landmarks

The structure of the landmark SPTs used by ALP are constrained by partitioning. One strategic method of increasing ALP's efficiency is to increase the number of landmarks that are used, which shortens the SPTs used for ALP. These series of trials provide evidence as to the degree to which ALP performs better in the context of larger or smaller SPTs from each landmark. These trials are performed on the following four road graphs:

Dataset	# Nodes	# Edges	Average Clustering	Transitivity
Rome	3353	4831	3.027E-02	3.7358E-02
Washington DC	9522	14832	3.919E-02	4.6936E-02
Vermont	95671	209764	1.603E-02	2.8579E-02
New York City	264328	730012	2.077E-02	2.5438E-02

Table 12 Road Graphs for Increasing Landmark Trials

Note that the average clustering and transitivity of these graphs are closest to the Barabási–Albert, Waxman, and Watts-Strogatz graphs in our synthetic graph dataset. In this series of trials, we leverage the hierarchies of Louvain algorithm community detection to increase the number landmarks. We partition each graph by the first level of the Louvain dendrogram (with the least partitions), then the second, the third, and up to the fourth. This results in a growing number of landmarks used for ALP (as well as shorter SPTs). For each real graph available in our dataset, we run 1000 shortest path queries on uniform random source-target pairs. Below, in Table 13, we detail the average efficiency, average error, and the proportion of the graph searched during 1000 ALP queries for each of these road graphs. The data for these vertex classes most clearly demonstrated the differences in efficiency as the number of landmarks grew.

The first and most apparent result is that ALP appears to have greater efficiency

Name	# Landmarks	Level	Efficiency	% Graph Searched	Average Error
Rome	48	1	7.00049%	30.98052%	60.34290%
Rome	58	2	7.68830%	28.70882%	55.71418%
Rome	187	3	11.03445%	21.38073%	40.01324%
Rome	818	4	25.13997%	10.43306%	17.37964%
Washington DC	73	1	5.64145%	21.49114%	40.39575%
Washington DC	136	2	6.31846%	18.45890%	33.44018%
Washington DC	624	3	10.96116%	11.21931%	19.90601%
Washington DC	2855	4	31.55521%	4.27612%	7.27658%
Vermont	658	1	0.76603%	58.06448%	87.96134%
Vermont	718	2	0.99790%	51.84114%	40.18168%
Vermont	1923	3	1.01485%	51.19348%	37.31285%
Vermont	7405	4	1.04701%	51.01309%	34.76163%
NYC	418	1	1.44018%	16.43274%	26.45864%
NYC	429	2	1.44114%	15.77193%	25.58507%
NYC	926	3	1.75182%	13.79238%	21.55280%
NYC	3908	4	2.82053%	9.06403%	13.66942%

Table 13 ALP Performance for Increasing Landmarks

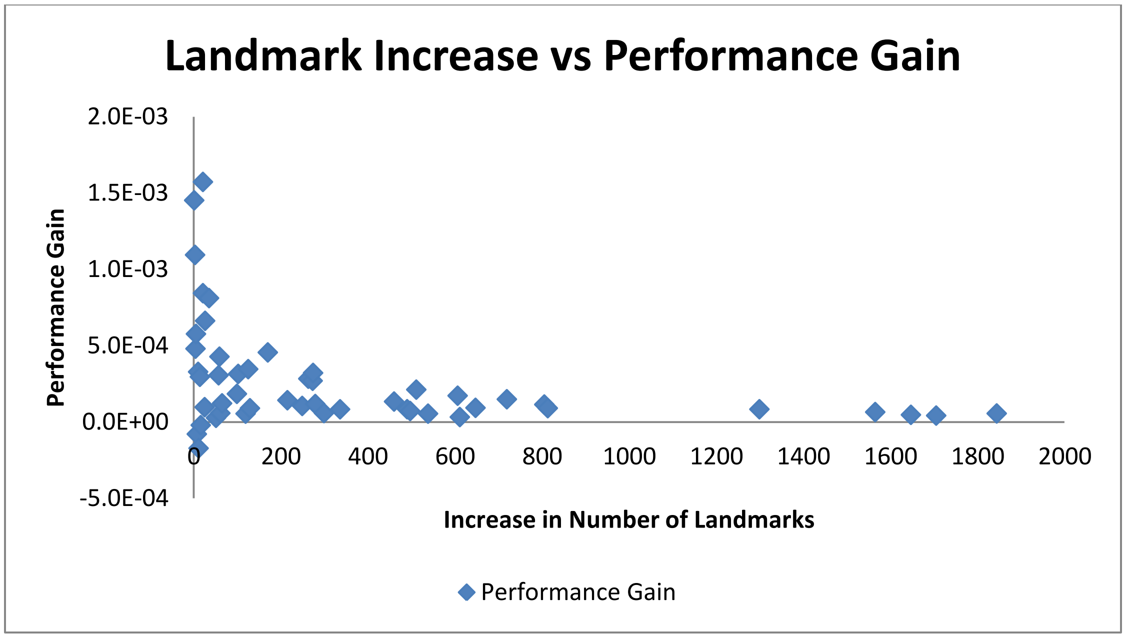


Figure 23 Landmark Increase vs Performance Gain

as the number of landmarks embedded in the graph grows. For Washington DC, we see as high as 25% efficiency between the use of level 1 and 4 for the dendrogram. Further analysis of this increase in efficiency is illustrated in Figure 23. This figure illustrates this performance gain[15] in relation to the increase in sheer number of landmarks for each run. As the ratio of the number of vertices to landmarks increases, the performance gain converges.[16] This means that growing the number of landmarks is beneficial up to a limit for ALP. However, the actual amount that it benefits decreases as the maximum possible partitioning is approached.

Shown in the tables and plots above, the efficiency of ALP always improves when the number of landmarks embedded in the graph grows. However, performance gain converges to zero as the ratio of nodes to landmarks continues to grow. Understanding this convergence is the key to understanding the optimal number of landmarks for ALP.

[15] Defined earlier in Metrics

[16] Experimental graphs for performance gains as the landmarks increase appear to be a Cauchy sequence. While the data does not precisely confirm this over all graphs, the limit of this function converges as it approaches 0.

Name	# Landmarks	Level	Preprocessing Time (s)
Rome	48	1	10.8281069
Rome	58	2	9.5090308
Rome	187	3	15.122344
Rome	818	4	34.9846501
Washington DC	73	1	39.6760621
Washington DC	136	2	46.523139
Washington DC	624	3	93.329982
Washington DC	2855	4	296.3415701
Vermont	658	1	918.328876
Vermont	718	2	995.6715961
Vermont	1923	3	2083.487783
Vermont	7405	4	6984.66541
NYC	418	1	2264.852974
NYC	429	2	1981.283189
NYC	926	3	4085.070291
NYC	3908	4	4610.982617

Table 14 # Landmarks vs Preprocessing Time

Further, understanding this convergence can inform partitioning algorithms such as the Louvain method as to the average size that clusters need to be for optimal behavior.

Results also show (Table 14) that preprocessing time typically coincides directly with the number of landmarks being used. Preprocessing time is measured as the combined time that it takes to both choose the set of landmarks and then grow the shortest path trees. The time that it takes to choose the set of landmarks varies based on the landmark selection technique used. The table below shows that for random landmark selection, the preprocessing time increases in linear proportion to the number of landmarks used.

Landmark Selection

The proposed landmark selection techniques from Chapter 3 were implemented in the Python implementation to identify the critical points of performance for each method. For reference, these techniques are summarized in Table 15.

Embedding Method	**Description**
Optimized Random	Within each subgraph, choose a set of candidate landmarks at random and run a series of ALT queries within the subgraph. Choose the landmark with the most efficient runs.
Farthest-d	Chooses a single landmark in each subgraph partition that is farthest in distance from all other already chosen landmarks
Farthest-ECC	Chooses a single landmark in each graph partition that is farthest from all vertices (highest eccentricity)
Planar	Choose a single landmark in each graph partition that is a border vertex and farthest from all other already chosen landmarks.
Betweenness Centrality	Compute the betweenness centrality of the largest connected subgraph of the partition. Select the vertex with the highest betweenness centrality
PageRank Maximum	Compute the PageRank of the largest connected subgraph of the partition. Select the vertex with the highest PageRank value
PageRank Minimum	Compute the PageRank of the largest connected subgraph of the partition. Select the vertex with the lowest PageRank value
PageRank Mode	Compute the PageRank of the largest connected subgraph of the partition. Choose a vertex with a PageRank value equal to the mode of vertex PageRank values
Closeness Centrality	Compute the closeness centrality of the largest connected subgraph of the partition. Select the vertex with the highest closeness centrality
Katz Centrality	Compute the Katz centrality of the largest connected subgraph of the partition. Select the vertex with the highest Katz centrality
Load Centrality	Compute the load centrality of the largest connected subgraph of the partition. Select the vertex with the highest load centrality

Table 15 Experimental Landmarks Selection Techniques for ALP

Each of these landmark selection techniques was applied to graphs in the road graph dataset. The goal of landmark selection is to optimize query performance and the tradeoff for the time required by preprocessing. 1000 queries were run on each graph, iterating through each landmark selection method, for the lower two levels of the dendrogram produced by the Louvain algorithm for partitioning. In the previous trials, we experienced intractably high preprocessing times for Farthest-d. We also saw that Katz centrality did not always converge in quite a few graphs. This is a fundamental property of Katz centrality, as it is primarily suited for directed acyclic graphs. Because these techniques were inconsistent in allowing meaningful results to be obtained, the Farthest-d and Katz centrality are not included in the summaries in this chapter. Their behavior and the edge cases where they optimize the ALP algorithm can be found in the results shown in the appendix. Figure 24 and Figure 25 describe the efficiency and tradeoff, respectively, of each of these runs for two road graphs as a bar chart. The numbers following the geographical locations for the chart labels describe the number of landmarks that were used for ALP. Two levels of the Louvain method dendrogram were used for each graph to appropriately characterize the selection algorithm's behavior.

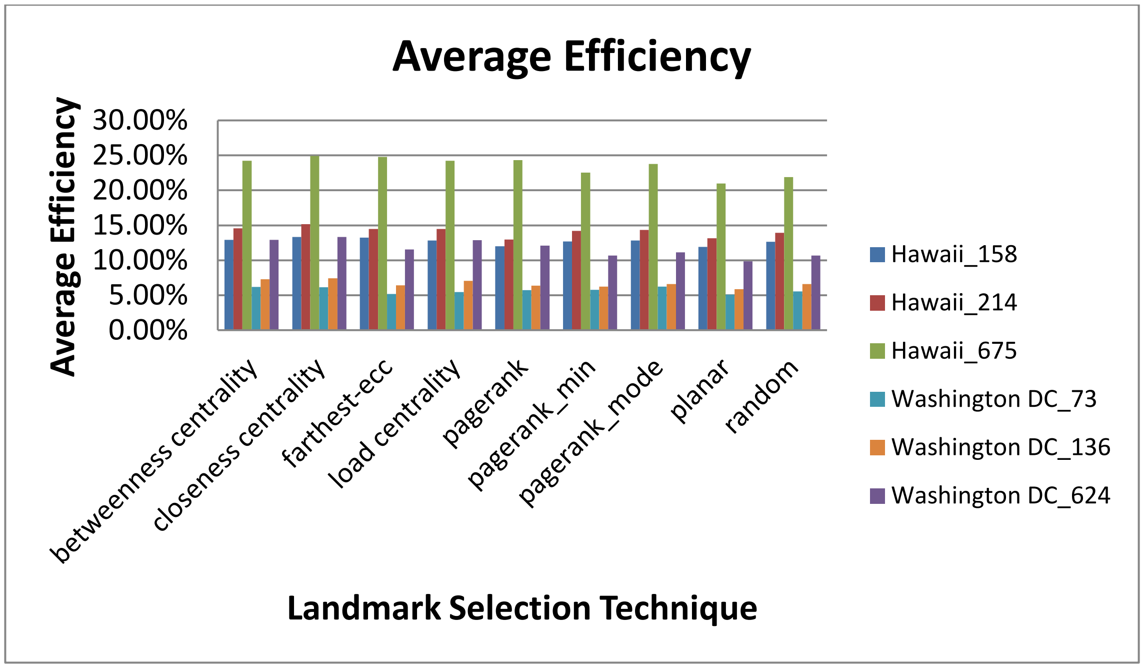


Figure 24 Landmark Selection Efficiency on Two Graphs for 1000 Query Trials

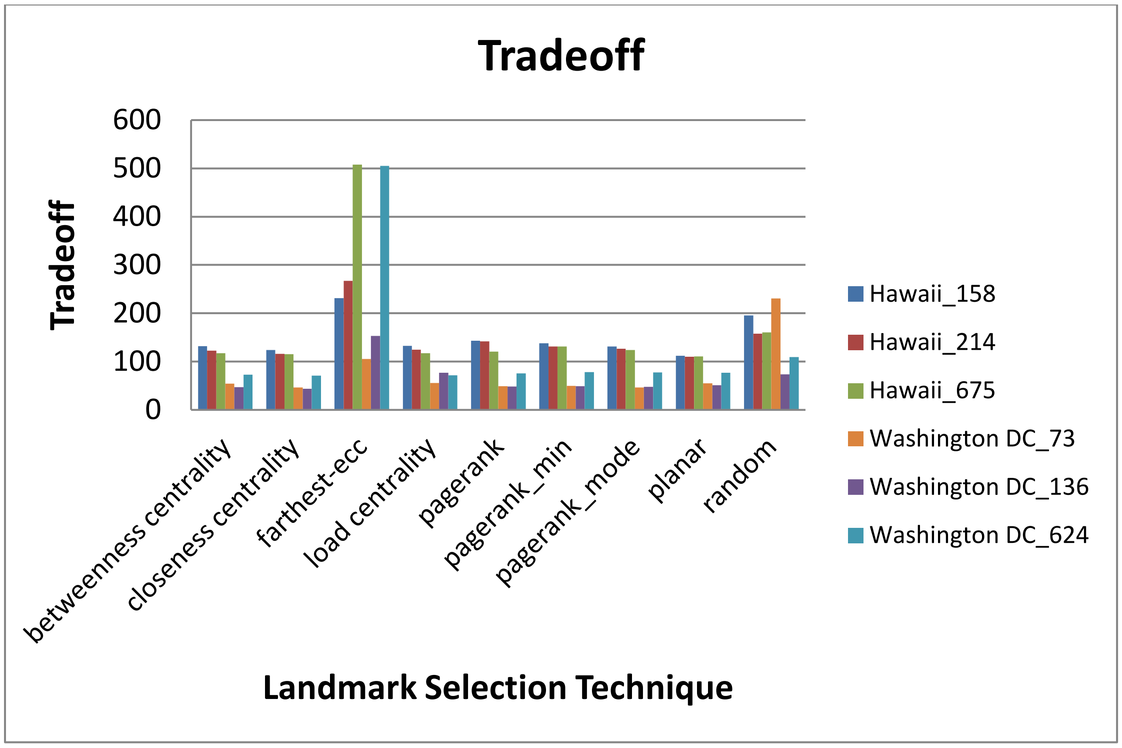


Figure 25 Landmark Selection Tradeoff on Two Graphs for 1000 Query Trials

As stated before, landmark selection is used to optimize the average efficiency of the ALP algorithm. This is apparent in Figure 24, as we see at most a 4% difference in the efficiency for any given graph, with *Farthest-ecc* showing highest efficiency for the largest graphs. In Figure 25, we see that the total clock time for both preprocessing and total query time can vary significantly based on landmark selection. *Farthest-ecc* demonstrates the largest tradeoff. Unfortunately, this is because its preprocessing time is the longest for each graph, as seen in Figure 26 for a 1000 query run on the graph of New Mexico[17]. Just as stated by Goldberg for some of ALT's original work, one cannot expect an improvement of an order of magnitude the average performance (Goldberg & Harrelson, 2005). These results indicate that this property applies to ALP, as well, which

[17] Remember, Farthest-ecc requires computing the graph eccentricity, a very expensive computation, particularly for large graphs.

is why we see random landmark selection still performing reasonably well in comparison to other algorithms.

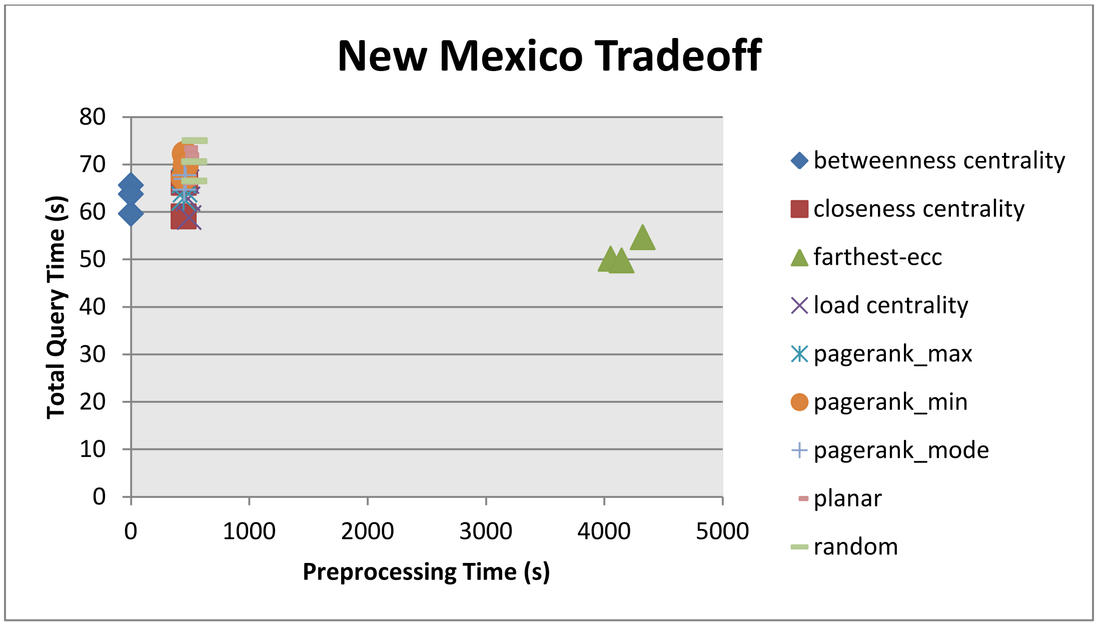


Figure 26 Preprocessing Time vs Total Query Time for Landmark Selection Techniques on the New Mexico Graph Dataset

Figure 27 illustrates the average approximation error for each of these runs as a bar chart. PageRank (max) and Planar landmark selection have the most error in these scenarios. Meanwhile, PageRank (min and mode), Farthest (eccentricity), and betweenness, closeness, and load centrality landmark selection techniques have average approximation errors below that of random. We also see that ALP makes better average approximations for graphs that are larger.[18]

[18] Graphs in Figure 27 are sorted from largest to smallest.

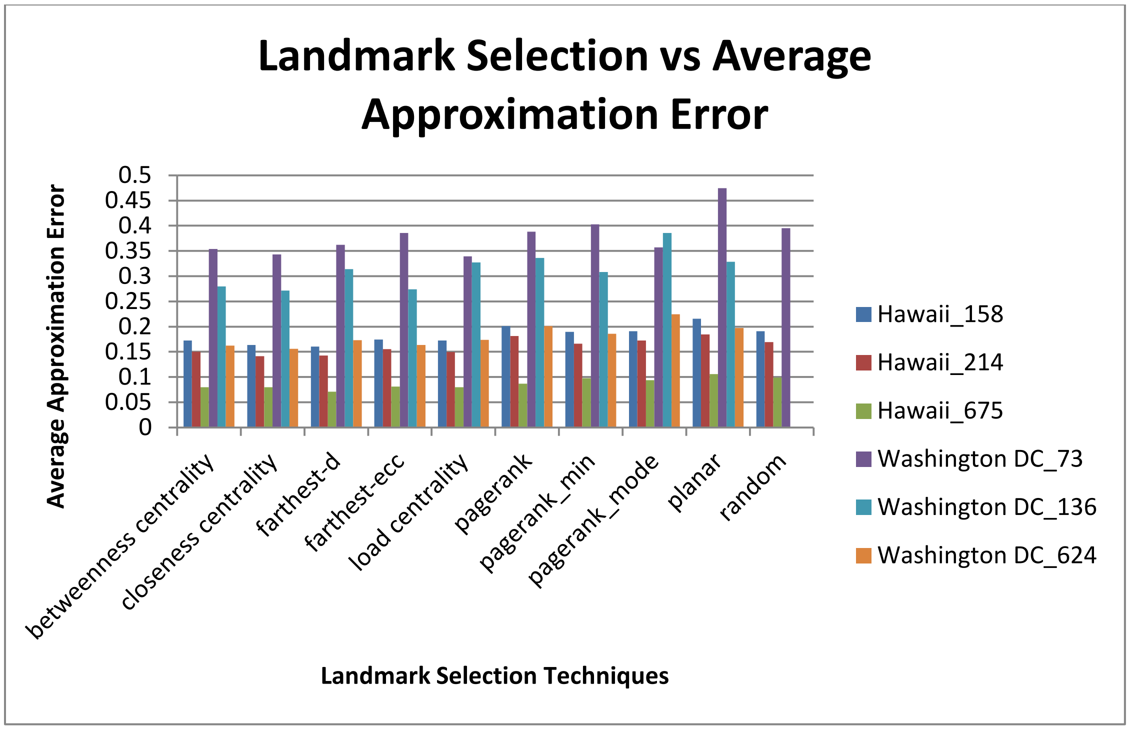


Figure 27 Landmark Selection Approximation Error on Three Graphs for 1000 Query Trials[19]

For each landmark selection technique, Figure 28 illustrates the approximation error of ALP queries using each landmark selection technique in the context of actual path lengths, indicating that ALP has a tighter approximation over larger distances. The landmark selection techniques do not impact the average approximation error as the path lengths become larger.

Each of the landmark selection methods exhibit similar average efficiency, tradeoff, and average error as distances become larger. *Farthest-ecc* has the best efficiency but the worst tradeoff, as the preprocessing time is significant for an insignificant benefit in query time. It also maintains the lowest error as path lengths grow. Random selection demonstrates the best overall tradeoff. ALP Planar is the least efficient,

[19] The labels of the graphs indicate the geographic location prior to the underscore and the number of chosen landmarks after the underscore.

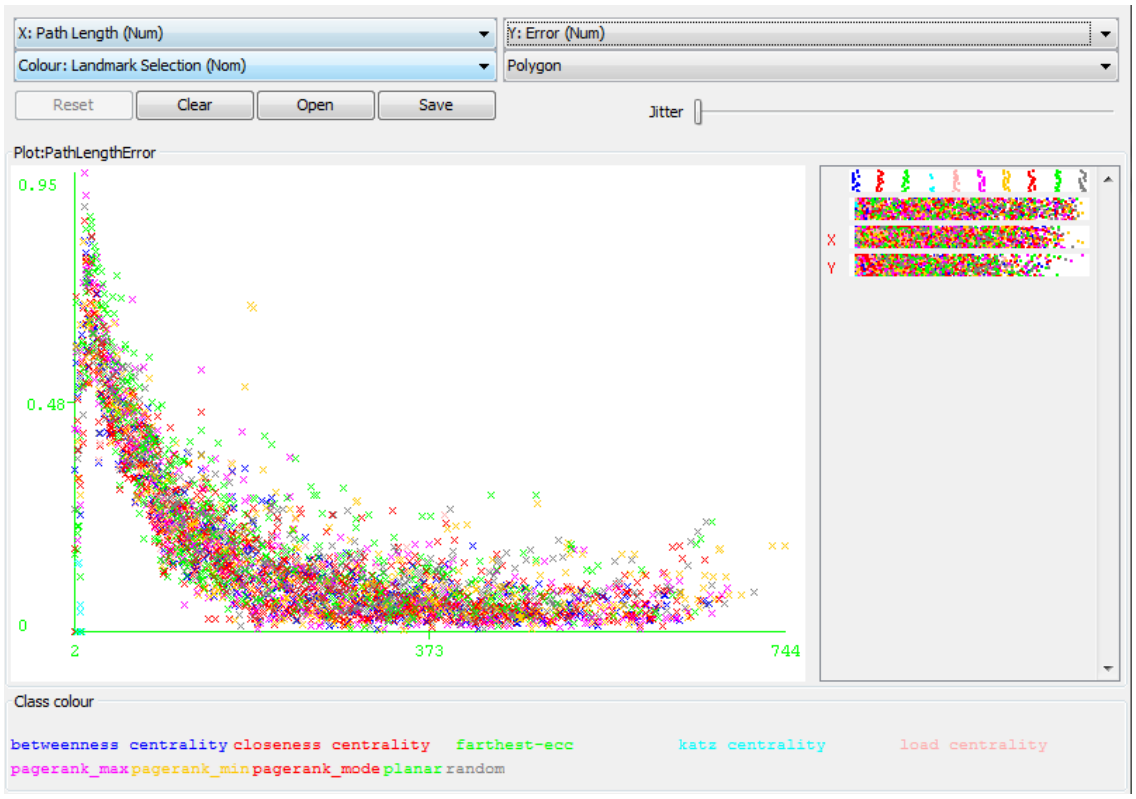


Figure 28 Path Length(X) vs Approximation Error (Y)

has the worst tradeoff, and exhibits the highest average error of all the featured landmark selection techniques.

The landmark selection techniques used for ALP can make a difference in its average efficiency. However, for the datasets used throughout experimentation, at their size, only a 4-6% difference in efficiency is ever observed. *Farthest-ECC* shows the best performance in the context of efficiency, but takes longer time than many other measures to compute. Therefore, for critical applications, when even the smallest speedup for query-time is needed, *Farthest-ECC* demonstrates the best performance, because of its ability to space landmarks out in the graph. However, its preprocessing time can, in some cases, be impractical. Overall, all centrality measure-based landmark selection techniques[20] gave reliable performance for centrality that can be computed for most datasets. They all demonstrated better performance than simple random landmark selection. However, when it comes to common applications, when high landmark selection times are detrimental to an application, closeness, and load centrality demonstrated the most consistent performance across all datasets and were quick to compute landmarks for ALP.

Experiment 2: ALT vs ALP

Experiment 2 leveraged all of the implementations, data gathering, and knowledge gleaned from Experiment 1. We used this information to identify the key benefits of using ALP over ALT in practical scenarios. Notably, we do not focus heavily on the fact that ALT outperforms ALP over the same set of landmarks in terms of our efficiency metric, as mathematics tells us that the lower bound of the triangle inequality will always be tighter under that scenario. Rather, the trials in this Experiment focus on the preferred graph and landmark configurations for their practical use. Therefore, we compared the tradeoffs of ALT and ALP to answer research questions regarding utility of each algorithm.

Description of Trials

We again leverage the Python 2.7/NetworkX 1.9 implementations to perform experimentation. We run each individual trial by inputting a graph dataset, setting up a number of shortest path source-target pairs, preprocessing both ALT and ALP, and then executing queries using the ALT, ALP, and uninformed (Dijkstra's) heuristic. We use the

[20] This is with exception to Katz centrality, which had trouble establishing an appropriate eigenvector for many datasets.

Trial Categories	**Description**
ALT vs. ALP: Runtime	Analyze the comparative runtimes for shortest path queries from Experiment 1 trials
ALT vs. ALP: Equal Landmarks	Compare and contrast the efficiency and average error between ALT and ALP when the same landmarks are chosen for both ALT and ALP
ALT vs. ALP: Fixed-Memory	ALT and ALP go head to head in a fixed memory environment for four road graph datasets.

Table 16 ALT vs ALP Trials

pathmax equation for A* such that the heuristics are consistent. First, we compare the runtimes of ALT and ALP in the previous graph trials. Next we highlight the behavior of ALT and ALP when they use the same set of landmarks and gain a comparative understanding of how the algorithms behave given the same parameters. And finally, the featured trial established a fixed amount of memory and ran each of the algorithms under varied parameters as gleaned from this study and the academic literature to understand their utility.

ALT vs. ALP: Runtime

For first comparisons of ALP and ALT, the performance of both algorithms was analyzed for the experimental benchmark road data from DIMACS and all available synthetic graphs (up to size 10^6 nodes) from Experiment 1. Random landmark selection was used for each trial run of the two algorithms on these datasets. The Louvain algorithm was used again for the partitioning of each graph prior to distributed embedding. As illustrated in Figure 29, queries for paths with distances between 1 and 501 were called 10^5 times. While ALT nearly always out-estimated the dual landmark ALP algorithm, the resulting data show significant improvement of the runtime of the dual landmark ALP heuristic over the ALT heuristic on a diverse set of graphs with larger path lengths, as well as an inherent reduction in required memory. This is a result

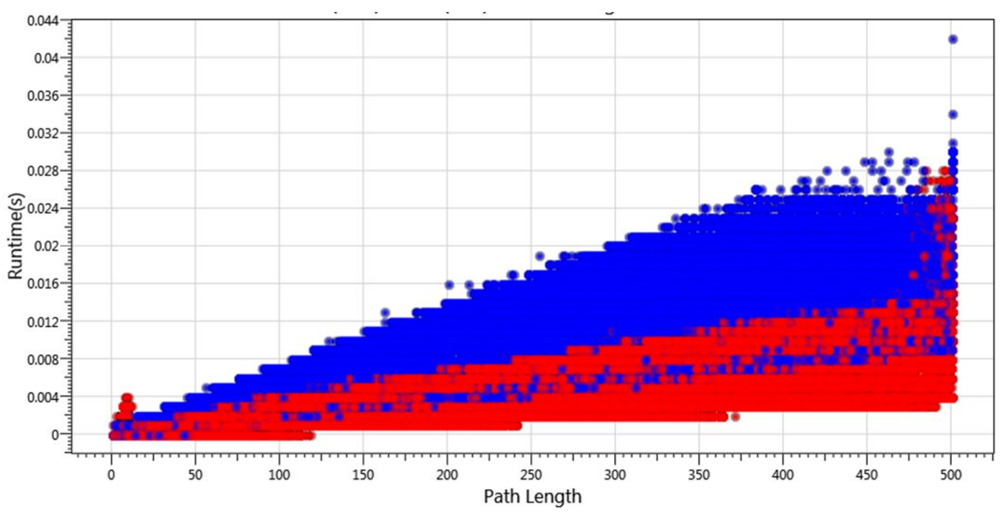


Figure 29 Graph demonstrating a higher runtime for ALT (Blue) compared to ALP (Red) as the length of the paths grow

of the reduced number of operations being performed at each visited vertex during the search, as illustrated in Figure 30.

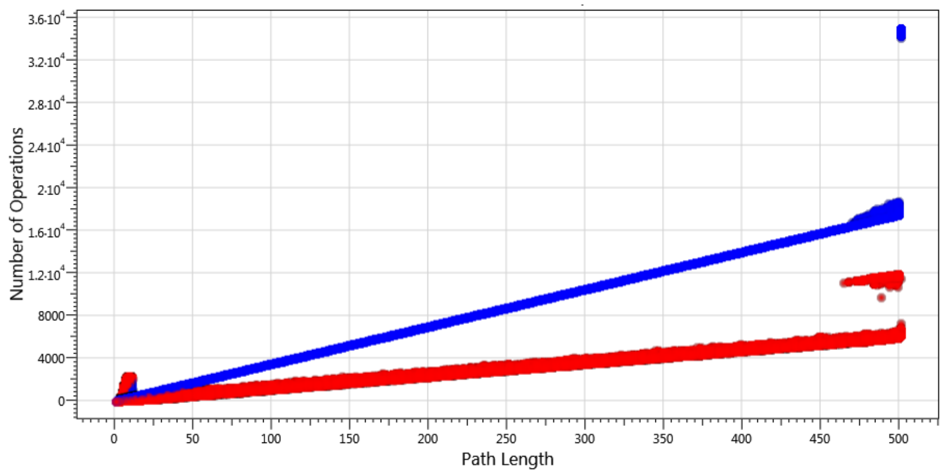


Figure 30 Graph demonstrating a higher number of operations for ALT (Blue) compared to ALP (Red) as the length of the paths grow. This corresponds to the runtime graphic on the previous page

ALT vs. ALP: Equal Landmarks

We proved, in the previous chapter, that ALT has better estimates over the same set of landmarks. In this set of trials, we look at ALT's shortest path preprocessing behavior when using the set of landmarks chosen by ALP. In other words, this set of trials was performed to see if the landmark selection techniques that were developed for ALP could be beneficial for ALT in the future. Just as in previous trials, we select 1000 source-target vertex pairs using uniform random distribution. Next, we preprocess ALP, establish its landmarks, and then use these landmarks to establish the data structure for both ALP and ALT. We then run the 1000 queries under the ALT and ALP heuristics to demonstrate ALT's behavior when using the same landmark set as ALP. We iterate through this process and work our way down the Louvain dendrogram to understand behavior as the number of landmarks grow. The figures below display the resulting data.

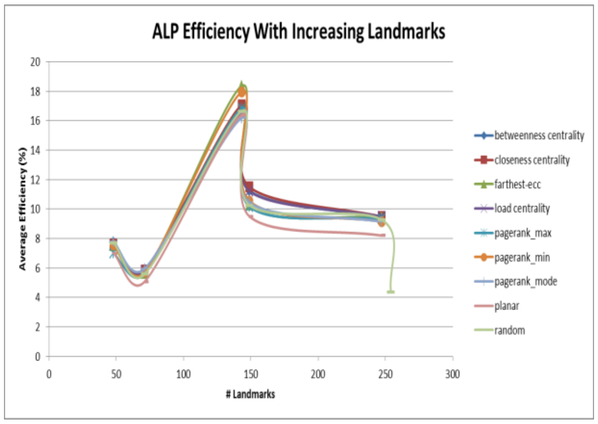


Figure 31 ALP Preprocessing in ALT: ALP #Landmarks vs Average Efficiency

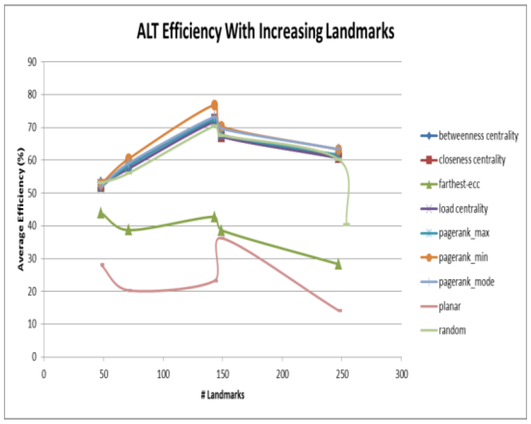


Figure 32 ALP Preprocessing in ALT: ALT #Landmarks vs Average Efficiency

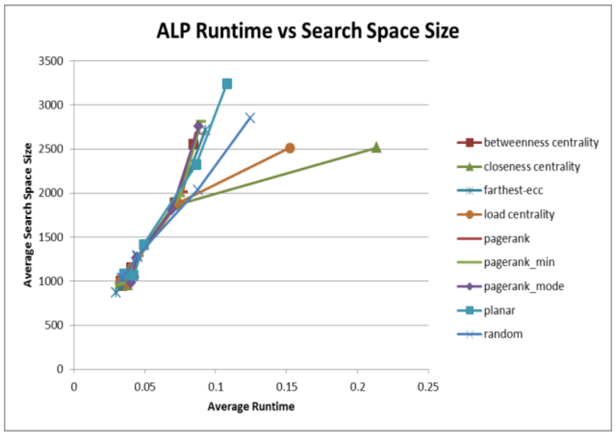


Figure 33 ALP Preprocessing in ALT: ALP Average Runtime vs Search Space Size

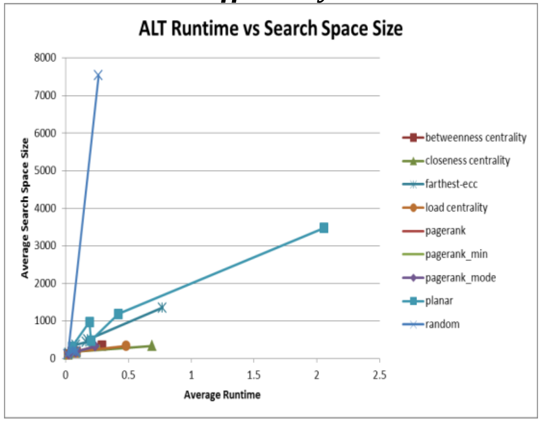


Figure 34 ALP Preprocessing in ALT: ALT Average Runtime vs Search Space Size

In Figure 31 and Figure 32, we see that ALT maintains its high efficiencies when leveraging ALP landmark selection. However, *Planar* and *Farthest-ecc* demonstrate significant drops in efficiency for ALT. This is not surprising for Planar. However, ALP's version of *Farthest* landmarks selection does not serve the ALT algorithm well. In Figure 33 and Figure 34, the efficiency gap is even more noticeable, as the average search space sizes for Planar and *Farthest-ECC* have outlier data points for ALT. The centrality measure-based landmark selection in each of these seems to maintain the efficiencies of ALT. Each of the centrality measures are computed very quickly in ALP. Therefore, they are viable candidates to speed up ALT landmark selection, though the bulk of ALT's preprocessing time comes from growing its shortest path trees from each landmark.

ALT vs. ALP: Fixed-Memory

An issue with using ALP preprocessing for ALT is defining the appropriate number of landmarks to use. As seen in each set of trials and experiments, the triangle inequality normally yields tighter lower bounds than quadrilateral inequalities over the same set and number of landmarks. Varying the used landmark selection technique helps ALP. However, throughout the vast majority of trials discussed thus far, it has not resulted in a better estimate for A* over ALT. Nonetheless, our dual landmark heuristic for ALP **can** outperform ALT when analyzing the same graph by using a **greater number of landmarks**. In this final set of trials, we simulated the use of the dual landmark ALP heuristic against the ALT heuristic in a hardware environment with fixed memory requirements. We allowed both preprocessing algorithms to use the most landmarks possible in the environment and compared their performance.

Simulating a fixed memory hardware environment for the heuristics was done by specifying upper bounds for the number of distance labels stored by the data structure. The following upper bounds for number of data labels stored were used:

- 250,000
- 500,000
- 1,000,000
- 2,500,000
- 5,000,000
- 10,000,000
- 25,000,000
- 100,000,000

Each graph in this set of trials uses six of these levels depending on the size of the graph. For each trial, the partitioning of the graph was performed with parameters such that the following was true for the landmark set L and any of these upper bounds U under the ALP environment:

$$|L|^2 + 2 \times |V| \leq U \tag{83}$$

Recall that the number of vertices is multiplied by two here because the distances *to* and *from* each landmark need to be stored for each subgraph in order to accurately compute the heuristic for directed graphs. For the landmark set L' in the ALT environment, the following requirement had to be met:

$$2 \times |L'| \times |V| \leq U \tag{84}$$

Once again, multiplication by two accounts for the fact that ALT has to store the distances to and from each landmark in order to compute the heuristic for directed graphs. We used these constraints to simulate a fixed-memory environment for ALT and

ALP. We perform each run by using optimized random landmark selection (*random-opt*). This is done for two reasons: First, it is done to support a general scenario in which we must decide whether to apply ALT or ALP, not knowing if the capabilities for complex mathematical functions such as eigenvector centrality measurement are available in the real-world environment in which we are operating. Second, the goal of this set of trials is to demonstrate the impact of number of landmarks, not selection strategies. Appropriate selection strategies for both ALT and ALP would result in choosing many of the same landmarks. Both theory and trials have shown that over the same set of landmark, ALT heuristics nearly always out-estimate ALP heuristics.

The Louvain method used throughout experimentation has the drawback that the number of partitions that it produces cannot be fixed. It simply forms a dendrogram at which each level can be used to signify community structure in a way that optimizes community modularity. Because of this, we hypothesized that relying on the levels of partitioning granted by the Louvain method for the levels of fixed memory described above can be a sub-optimal solution to a path planning implementation. Nonetheless, it is still a computationally low-cost method of partitioning that can be applied to many devices with small fixed memory.

However, it is also beneficial to understand ALP's behavior in this fixed-memory environment when it can maximize its number of landmarks. Therefore, we use two different partitioning algorithms for characterizing ALP's behavior in a fixed-memory environment. The first of which is the Louvain method, in which we choose the highest possible level of the resulting dendrogram that produces a number of partitions (which is equal to the number of landmarks) closest to the fixed-memory upper bound for ALP.

This allows for good coverage of landmarks but does not allow ALP to reach its maximum number of landmarks in the fixed-memory environment. To do that, we use a second partitioning scheme that starts with the partitions of the first level of the Louvain method dendrogram. Recall from Chapter 1 that the first level of partitioning yields the maximum modularity score for an input graph. Then, let $\boldsymbol{L}$ be the desired number of landmarks and $\boldsymbol{M}$ the number of partitions at the first level of the Louvain method dendrogram. Then, for the subgraph induced by each partition, another community detection method, called *walktrap community detection*, is applied that allows us to specify the number of communities to be fixed (Pons & Latapy, 2005). This method, based on the notion that short random walks should tend to stay in the same community, produces a dendrogram that can be cut to represent a desired number of partitions. This is done by replaying merges of the dendrogram from the beginning until the membership vector has exactly the desired number of communities, or until there are no more merges. The number of communities for each partition is fixed as follows:

$$\#Communities\ per\ Partition = \left\lfloor \frac{\boldsymbol{L}}{\boldsymbol{M}} \right\rfloor \tag{85}$$

This is true for all but the largest community, which is partitioned into $\left\lfloor \frac{\boldsymbol{L}}{\boldsymbol{M}} \right\rfloor + \boldsymbol{L}\ mod\ \boldsymbol{M}$ communities.[21]

Here, we break down a run of four road graphs in this environment that were studied the most over the dissertation effort, in their entirety. For each graph, we ran 1000 shortest path queries using the same source-target pairs selected over a uniform random distribution. We capture the average search space[22], error, and runtime (in seconds) for

[21] For each of these trials, the walktrap community detection implementation's step parameter is set to 10.
[22] We can simply use the search space here as we are not comparing runs between the graphs.

runs at each memory bound. Each graph is analyzed using both partitioning methods described above. First, we analyze runs for one of our most tested graphs, a graph of Washington DC:

Graph	Nodes	Edges	Transitivity	Average Clustering	Density
Washington DC	9522	29639	0.046936	3.919E-2	3.272E-4

First, we analyze the graph in the fixed memory environment using the Louvain algorithm. The table below shows the parameters of the run and the result data.

Dataset	Memory	Heuristic	# Landmarks	# Labels	Avg Search Space	Avg Error	Avg Runtime (s)
Washington DC	2.50E+05	ALP	138	28566	1571.1379	29.81112%	3.5428E-02
Washington DC	2.50E+05	ALT	13	9691	350.5776	4.54692%	6.8026E-03
Washington DC	5.00E+05	ALP	628	403906	1193.6679	26.73488%	2.6641E-02
Washington DC	5.00E+05	ALT	26	10198	278.5031	3.87763%	1.6158E-02
Washington DC	1.00E+06	ALT	52	12226	208.7736	2.61185%	1.9735E-02
Washington DC	2.50E+06	ALT	105	20547	153.1421	2.01167%	4.3317E-02
Washington DC	5.00E+06	ALT	262	78166	110.9219	0.99270%	9.5491E-02
Washington DC	1.00E+07	ALP	2856	8166258	1885.1978	55.36047%	3.7359E-02
Washington DC	1.00E+07	ALT	525	285147	90.4394	0.64338%	2.1970E-01

Table 17 Washington DC Fixed-Memory Performance of ALT vs ALP (Louvain)[23]

Figure 35 and Figure 36 highlight the average search space and runtime of these runs.[24] ALT has better average error and search space size than ALP landmark selection while ALP boasts better average runtimes than ALT for the larger memory queries. This is expected due to the number of arithmetic operations performed at each vertex. We also see that increasing the number of landmarks in this case does not necessarily mean an increase in ALP's algorithmic performance (in terms of search space size).

Practical implementations of ALT suffer from the fact that they have to explore the space of maximum lower bounds in order to compute its heuristic upon visiting every node. Even exhausting Python's latest available optimizations, this is still a hindrance for

[23] ALP is restricted from executing at the 1E6, 2.5E6, and 5E6 fixed memory bounds because the Louvain algorithm dendrogram only had partitioning suitable for bounds lower than that. Therefore, ALP data, for comparison, is the next lowest bound.
[24] In this section, for each set of runs, the corresponding figure for Fixed Memory vs Average Error can be found in Appendix C.

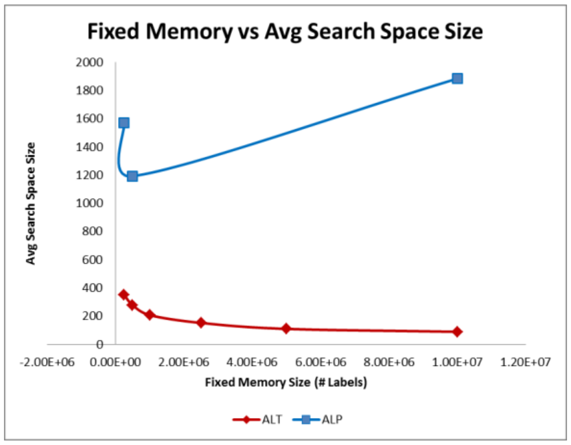


Figure 35 Washington DC Fixed Memory vs Average Search Space Size

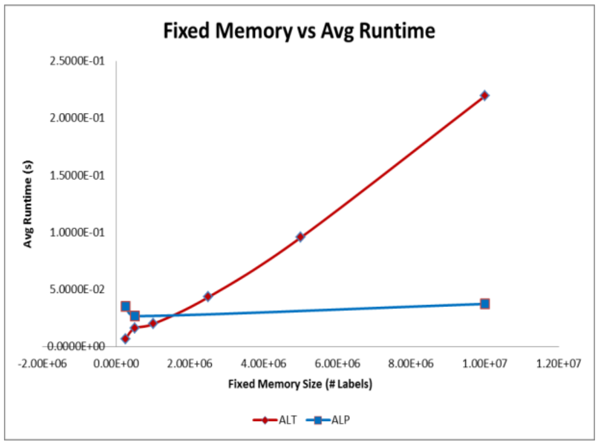


Figure 36 Washington DC Fixed Memory vs Average Runtime

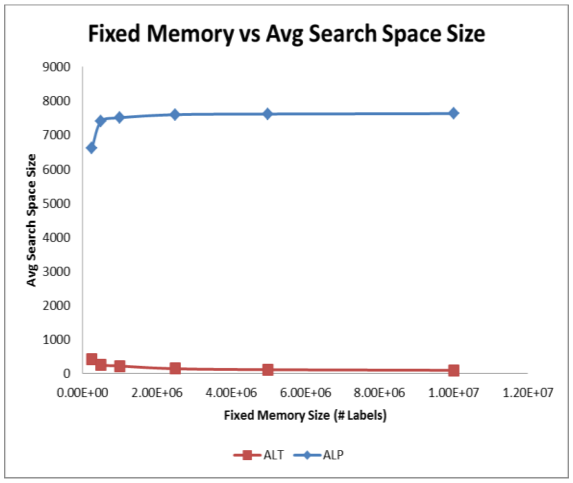
ALT. However, much to our chagrin, in this scenario, ALT still outperforms ALP in terms of average search space, average approximation error, and average runtime.

We see several of the categorized memory bounds that do not have data for ALP. This is due to restrictions on Louvain method partitioning. In this run, ALP is restricted from executing at the 1E6, 2.5E6, and 5E6 fixed memory bounds because the Louvain algorithm dendrogram only had partitioning suitable for bounds lower than that. Below are results of the graph using the partitioning of the combined Louvain and walktrap community algorithm.

Dataset	Memory	Heuristic	# Landmarks	# Labels	Avg Search Space	Avg Error	Avg Runtime (s)
Washington DC	2.500E+05	ALP	480	239922	6606.8252	69.2098%	9.4341E-02
Washington DC	2.500E+05	ALT	13	9691	417.7007	6.2752%	8.6301E-03
Washington DC	5.000E+05	ALT	689	484243	7400.9269	55.0702%	1.0007E-01
Washington DC	5.000E+05	ALT	26	10198	255.1752	3.4808%	8.3438E-03
Washington DC	1.000E+06	ALP	978	966006	7506.3653	52.9431%	1.0037E-01
Washington DC	1.000E+06	ALP	52	12226	220.4454	3.1801%	1.1773E-02
Washington DC	2.500E+06	ALP	1539	2378043	7596.7606	51.1063%	1.0487E-01
Washington DC	2.500E+06	ALT	131	26683	140.9479	1.5360%	2.0862E-02
Washington DC	5.000E+06	ALP	2141	4593403	7615.4187	51.0071%	1.0571E-01
Washington DC	5.000E+06	ALT	262	78166	113.3003	1.0035%	3.8670E-02
Washington DC	1.000E+07	ALP	2974	8854198	7627.7683	50.4975%	1.0593E-01
Washington DC	1.000E+07	ALT	525	285147	94.5616	0.6249%	1.1075E-01

Table 18 Washington DC Fixed-Memory Performance of ALT vs ALP (Louvain/Walktrap)

Figure 37 and Figure 38 highlight the average search space and runtime of these runs. The first recognizable impact of the use of the combined Louvain/Walktrap community detection method is the significantly larger search space, error, and runtime used by ALP at all levels. The second is that we do see the average search space increasing as the number of landmarks increases.



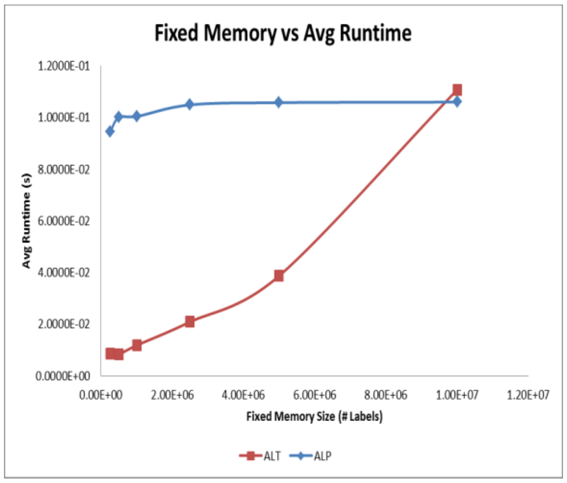


Figure 37 Washington DC Fixed Memory vs Average Search Space Size (Louvain/Walktrap)

Figure 38 Washington DC Fixed Memory vs Average Runtime (Louvain/Walktrap)

The next graph of New Mexico indicates whether or not this behavior is consistent:

Graph	**Nodes**	**Edges**	**Transitivity**	**Average Clustering**	**Density**
New Mexico (subgraph)	21,866	70,867	0.059988	0.04285	0.00011829

The following table shows the parameters of running ALT and ALP on the graph when partitioned with the Louvain method:

Dataset	Memory	Heuristic	# Landmarks	# Labels	Avg Search Space	Avg Error	Avg Runtime (s)
New Mexico	2.50E+05	ALP	401	182667	3399.1715	25.2513%	5.9344E-02
New Mexico	2.50E+05	ALT	5	21891	2207.8979	16.6621%	2.7334E-02
New Mexico	5.00E+05	ALT	11	21987	1613.1371	9.9733%	3.5869E-02
New Mexico	1.00E+06	ALT	22	22350	856.8704	4.9121%	2.9102E-02
New Mexico	2.50E+06	ALP	1554	2436782	2540.1837	23.8242%	4.5226E-02
New Mexico	2.50E+06	ALT	57	25115	544.3954	2.7099%	3.1163E-02
New Mexico	5.00E+06	ALT	114	34862	416.4034	2.0523%	6.4603E-02

Table 19 New Mexico Fixed-Memory Performance of ALT vs ALP (Louvain)

Once again, we see the average search space size of queries for dual-landmark ALP being much larger than that of ALT, with an average approximation error that is embarrassingly higher. And we can only run ALP twice under this configuration. This time, performance is even worse for ALP when it comes to runtime, as shown in the figures below.

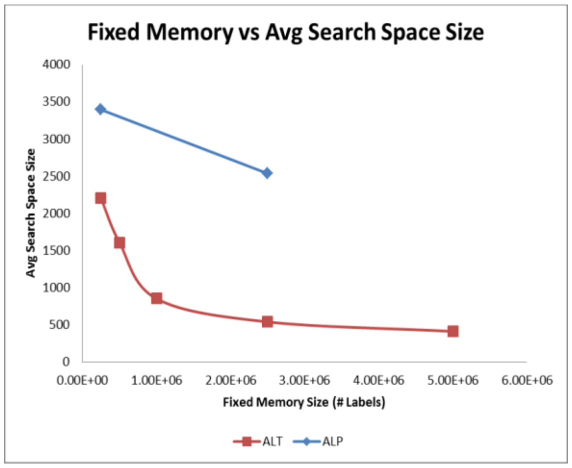


Figure 39 New Mexico Fixed Memory vs Average Search Space Size

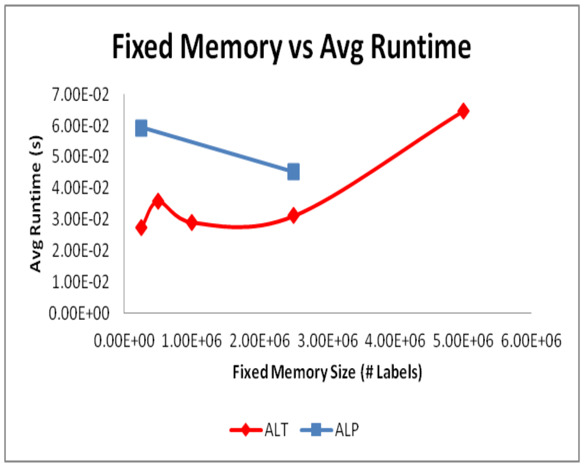


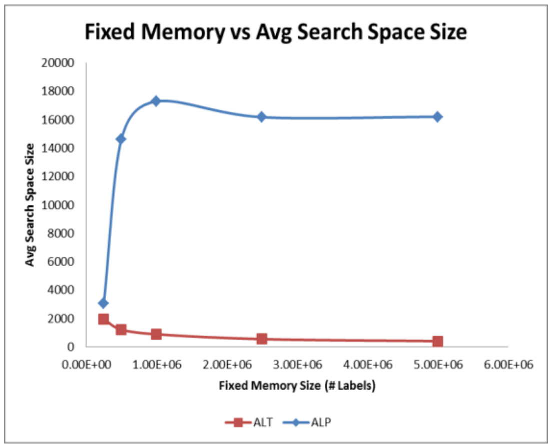
Figure 40 New Mexico Fixed Memory vs Average Runtime

The table below details the results of running the combined Louvain/Walktrap method on New Mexico:

Dataset	Memory	Heuristic	# Landmarks	# Labels	Avg Search Space	Avg Error	Avg Runtime (s)
New Mexico	2.50E+05	ALP	405	185891	3067.7518	21.4430%	4.4960E-02
New Mexico	2.50E+05	ALT	5	21891	1940.958	15.8205%	2.5976E-02
New Mexico	5.00E+05	ALP	675	477491	14615.2136	65.4098%	2.1472E-01
New Mexico	5.00E+05	ALT	11	21987	1209.9349	7.3536%	2.4114E-02
New Mexico	1.00E+06	ALP	933	892355	17284.5399	56.9401%	2.3689E-01
New Mexico	1.00E+06	ALT	22	22350	886.4815	5.4194%	2.3737E-02
New Mexico	2.50E+06	ALP	1563	2464835	16170.4717	57.7529%	2.2554E-01
New Mexico	2.50E+06	ALT	57	25115	544.3954	2.7099%	3.1163E-02
New Mexico	5.00E+06	ALP	2206	4888302	16195.4176	57.1753%	2.2700E-01
New Mexico	5.00E+06	ALT	114	34862	399.3674	2.0399%	4.4304E-02

Table 20 New Mexico Fixed-Memory Performance of ALT vs ALP (Louvain/Walktrap)

The behavior for the combined Louvain/Walktrap community detection algorithm is consistent. As further illustrated in the figures below, this experimentally verifies that the partitioning of the input graph can impact ALP, which was evident previously given that landmark selection is significant to optimization.



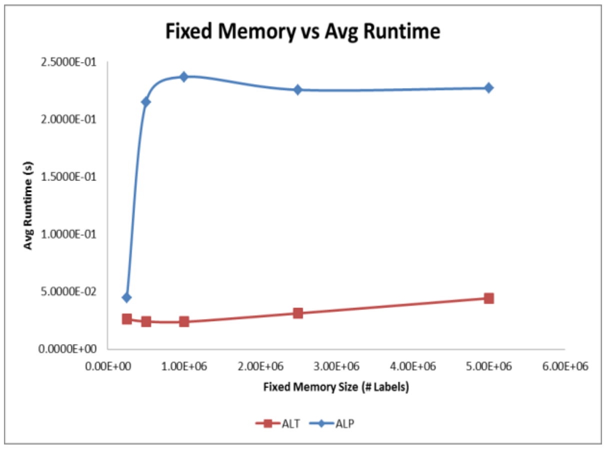


Figure 41 New Mexico Fixed Memory vs Average Search Space Size (Louvain/Walktrap)

Figure 42 New Mexico Fixed Memory vs Average Runtime (Louvain/Walktrap)

We now move onto a much larger graph than these first two that truly demonstrates the utility of dual-landmark ALP. Instead of taking subgraphs, the next graphs are entire graphs of a geographical region.

The following are details of the graph representing the full roadmap of New York City (NYC):

Graph	Nodes	Edges	Transitivity	Average Clustering	Density
New York City	264,328	730,012	0.025438	0.020772	0.000010448

The following table shows the parameters of the run and the result data under Louvain method partitioning:

Dataset	Memory	Heuristic	# Landmarks	Avg Search Space	Avg Error	Avg Runtime (s)
New York City	**1.00E+06**	**ALP**	**427**	**42821**	**27.11%**	**7.3090E-01**
New York City	1.00E+06	ALT	1	78460	57.10%	9.8260E-01
New York City	**2.50E+06**	**ALP**	**942**	**35003**	**21.73%**	**6.0750E-01**
New York City	2.50E+06	ALT	4	40943	27.68%	6.4930E-01
New York City	5.00E+06	ALT	9	28827	14.43%	6.0200E-01
New York City	1.00E+07	ALT	18	48060	14.67%	1.3790E+00
New York City	**2.50E+07**	**ALP**	**3934**	**18975**	**12.81%**	**5.1060E-01**
New York City	2.50E+07	ALT	47	40189	17.29%	2.0390E+00
New York City	1.00E+08	ALT	189	81338	13.52%	1.1060E+02

Table 21 New York City Fixed-Memory Performance of ALT vs ALP (Louvain)

Highlighted in red, for this run, are the levels of fixed memory in which dual landmark ALP has a smaller average search space, smaller runtime, and smaller average approximation error than ALT. Seen in Figure 43 and Figure 44, the difference in search

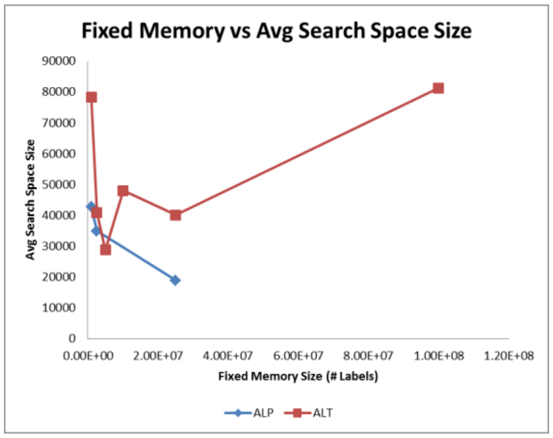


Figure 43 New York City Fixed Memory vs Average Search Space Size

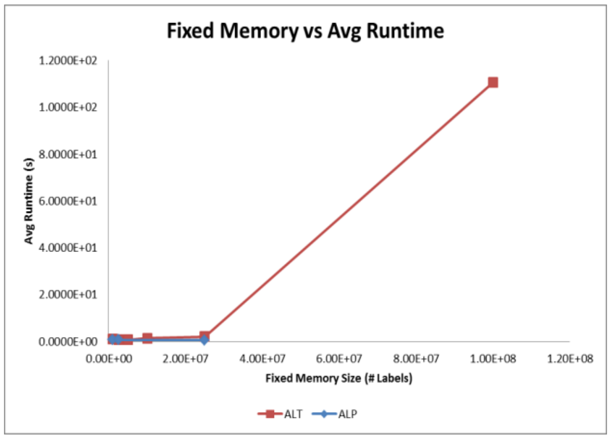


Figure 44 New York City Fixed Memory vs Average Runtime

space is substantial here. Even more notably, ALT experiences a sharp increase in both search size and runtime as the number of landmarks increase. This is the first research result that demonstrates ALP's dominance in a fixed-memory environment.

For the NYC graph, ALT has been limited to as low as a single landmark in our 5E4 upper bound memory configuration[25]. In that scenario, ALT loses out. Of note, even its average runtime, which depends not only on the search space size but on the number of arithmetic operations that occur for each node visit, is still worse for ALT in this scenario. This becomes very apparent for the 1E8 upper bound result, where the query runtimes were on the minute scale. ALT runtime simply becomes impractical when leveraging that many landmarks because it has to compute the triangle inequality for all landmarks at every visited node. As the amount of allowable memory grows, ALT does begin to algorithmically perform better, averaging a smaller search space, but does not catch up with dual landmark ALP.

We take a single scenario for the NYC graph, when the memory is limited to 2.5E6 labels. We separate the lengths of paths for queries on this graph into five different classes and attempt to understand the difference between ALT and ALP for estimations at these ranges. Table 22 details the average query search space, runtime, and approximation error for each of these path classes.

[25] Nothing could be executed in our lowest memory configuration at this point because it is smaller than the number of nodes in the graph.

Heuristic	**Average Search Space**	**Average Runtime (s)**	**Average Approx. Error**	**Path Class**
ALP	5,984.9231	0.095420387	0.310469725	0-200
ALT	4,941.0321	0.078824321	0.23462107	0-200
ALP	21,469.9707	0.365463074	0.215048211	200-400
ALT	22,222.9149	0.375098283	0.186812221	200-400
ALP	46,883.8047	0.815239528	0.184229549	400-600
ALT	56,051.3401	0.94817963	0.245895748	400-600
ALP	70,101.7847	1.236932435	0.164547937	600-800
ALT	86,528.5764	1.468556752	0.299505343	600-800
ALP	97,784.4615	1.707975973	0.18256267	800-
ALT	12,9913.6538	2.155318469	0.482575857	800-

Table 22 ALP's dominance of ALT over Large Path Lengths for 2.5E6 Data Label Upper Bound

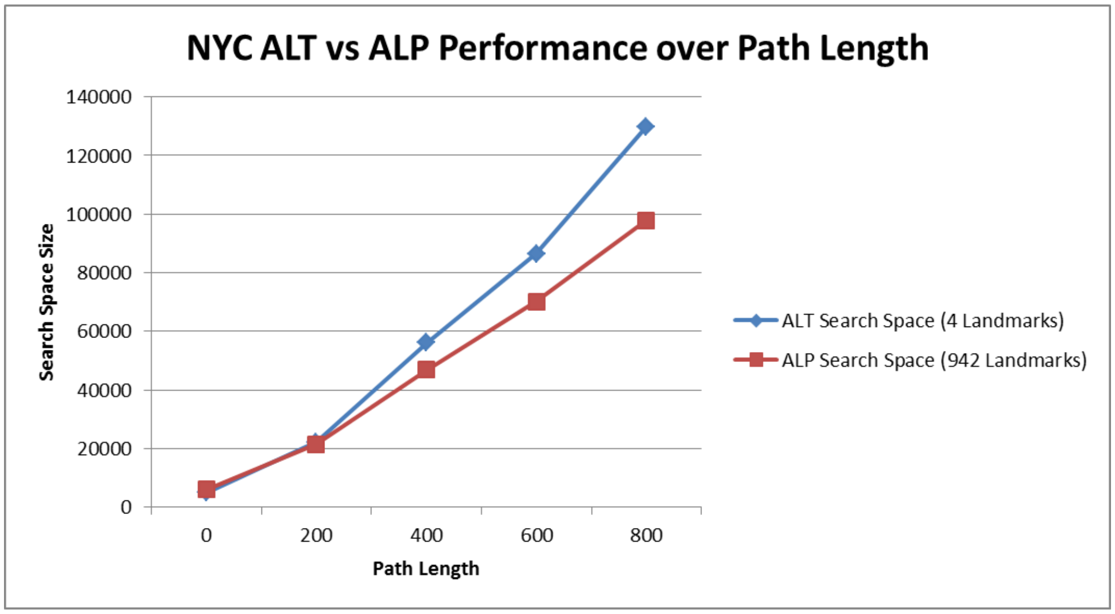


Figure 45 Performance of ALT vs ALP for 2.5M Data Labels

We see for smaller path lengths, the two algorithms are on par with each other, with ALT actually outperforming ALP for path lengths of 0-200. Beyond that range, ALP has better estimates than ALT. Figure 45 is a clear illustration of the delta in search space size between the two algorithms as the path lengths get larger.

The ability to outperform ALT in this scenario under a Louvain method partitioning demonstrates ALP's utility. Just as with the previous two graphs, this graph was run under the combined Louvain/Walktrap partitioning:

Dataset	Memory	Heuristic	# Landmarks	# Labels	Avg Search Space	Avg Error	Avg Runtime (s)
New York City	1.00E+06	ALP	686	734924	44626.7177	26.5428%	7.8985E-01
New York City	1.00E+06	ALT	1	264329	80723.4855	53.5135%	1.1611E+00
New York City	2.50E+06	ALP	1404	2235544	41409.3974	24.4544%	7.4700E-01
New York City	2.50E+06	ALT	4	264344	38028.7618	19.3056%	6.4815E-01
New York City	5.00E+06	ALP	2114	4733324	181830.958	82.5024%	3.2057E+00
New York City	5.00E+06	ALT	9	264409	71152.4154	30.8865%	1.5934E+00
New York City	1.00E+07	ALP	3077	9732257	182189.99	81.9870%	2.9852E+00
New York City	1.00E+07	ALT	18	264652	53287.5345	18.6009%	1.5012E+00
New York City	2.50E+07	ALP	4945	24717353	204581.2643	98.2239%	3.3681E+00
New York City	2.50E+07	ALT	47	266537	78984.6877	38.7101%	4.2289E+00
New York City	1.00E+08	ALT	7027	49643057	210180.5084	99.0932%	6.6567E+00
New York City	1.00E+08	ALT	189	273164	81244.2647	30.2203%	8.1485E+00

Table 23 New York City Fixed-Memory Performance of ALT vs ALP (Louvain/Walktrap)

The average runtime shown in the figure below is telling of the ability of ALP to outperform ALT even when it visits more nodes for large graphs.

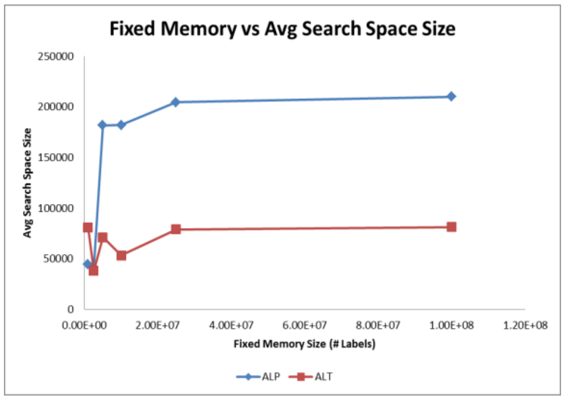


Figure 46 New York City Fixed Memory vs. Average Search Space Size (Louvain/Walktrap)

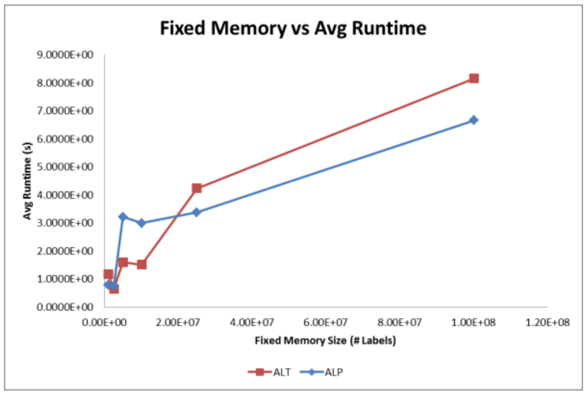


Figure 47 New York City Fixed Memory vs. Average Runtime (Louvain/Walktrap)

This is not a phenomenon. We take a look at the next largest graph in our dataset to further validate this finding.

Graph	Nodes	Edges	Transitivity	Average Clustering	Density
San Francisco Bay	321,258	794,788	0.02225	0.016565	0.000007701

In the following table, we see similar results when ALT and ALP go head to head in this graph:

Dataset	Memory	Heuristic	# Landmarks	Avg Search Space	Avg Error	Avg Runtime (s)
San Francisco Bay	1.0E+08	ALT	155	4701	1.77%	4.667E+00
San Francisco Bay	5.0E+07	ALP	4984	20534	9.53%	1.722E+00
San Francisco Bay	5.0E+07	ALT	77	6518	2.14%	1.256E+00
San Francisco Bay	2.5E+07	ALT	38	9960	3.55%	1.107E+00
San Francisco Bay	1.0E+07	ALT	15	15983	5.33%	1.193E+00
San Francisco Bay	5.0E+06	ALT	7	26016	11.49%	1.541E+00
San Francisco Bay	2.5E+06	ALP	1185	33444	14.31%	5.656E-01
San Francisco Bay	2.5E+06	ALT	3	45357	18.83%	6.392E-01

Table 24 San Francisco Fixed-Memory Performance of ALT vs ALP (Louvain)

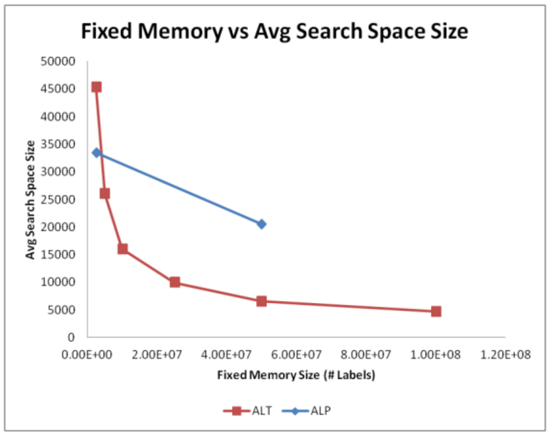


Figure 48 San Francisco Bay Fixed Memory vs. Average Search Space Size

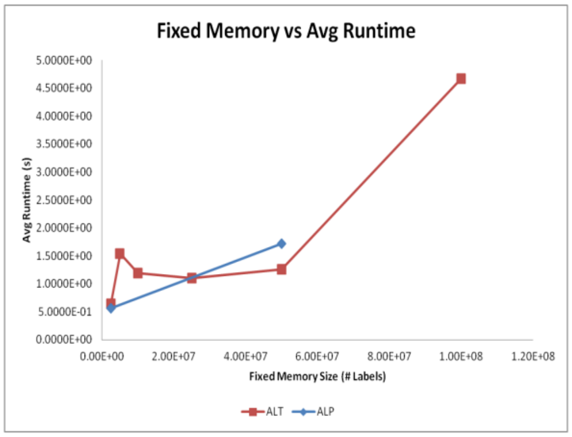


Figure 49 San Francisco Bay Fixed Memory vs. Average Runtime

Here, ALT beats ALP's search space size at the 5E6, 1E7, 2.5E7, 5E7, and 1E8 upper bounds. ALT also has comparable runtimes. This, however, could be attributed to the Louvain method's partitioning restrictions on number of landmarks used. This limited the number of runs that were performed. However, looking at the first four levels of fixed memory, we see that the combined partitioning method does not do better:

Dataset	Memory	Heuristic	# Landmarks	# Labels	Avg Search Space	Avg Error	Avg Runtime (s)
San Francisco	1.00E+07	ALP	3059	9678739	107292.2482	0.00676206	0.874311631
San Francisco	1.00E+07	ALT	15	321483	13753.011	0.08387688	0.049901005
San Francisco	5.00E+06	ALP	2087	4676827	107596.2693	0.0066022	0.876781686
San Francisco	5.00E+06	ALT	7	321307	25038.7648	0.05575716	0.114934876
San Francisco	2.50E+06	ALP	1362	2176302	108738.7758	0.00654484	0.882508475
San Francisco	2.50E+06	ALT	3	321267	39706.4424	0.03150861	0.196388122
San Francisco	1.00E+06	ALP	597	677667	108462.5846	0.00624014	0.880367487
San Francisco	1.00E+06	ALT	1	321259	87979.2272	0.01357608	0.500051364

Table 25 San Francisco Bay Fixed-Memory Performance of ALT vs ALP (Louvain/Walktrap)

At these levels, ALP is simply outmatched and is more comparable to Dijkstra's. The figures below illustrate the significance of partitioning for ALP.

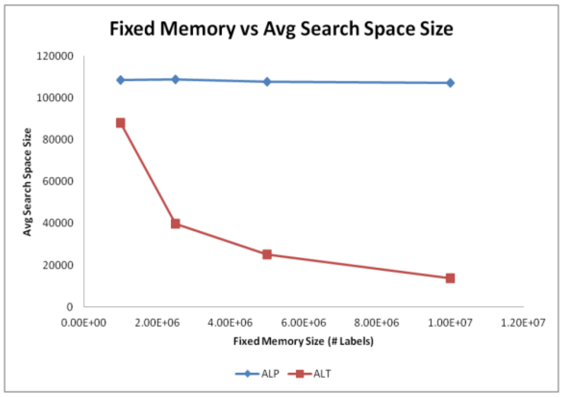


Figure 50 San Francisco Bay Fixed Memory vs. Average Search Space Size (Louvain/Walktrap)

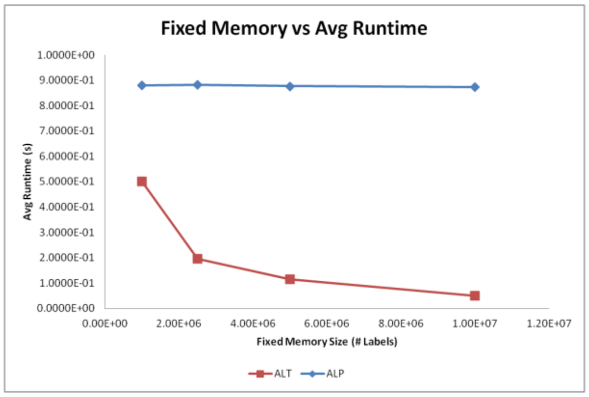


Figure 51 San Francisco Bay Fixed Memory vs. Average Runtime (Louvain/Walktrap)

Because ALP did not outperform ALT in all contexts of the last scenario, we look at a smaller run of one final dataset.

Graph	Nodes	Edges	Transitivity	Average Clustering	Density
Colorado	435,550	1,042,104	0.02518184	0.017235	0.0000054933

The following is a table of our results from analysis of Colorado:

Dataset	Memory	Heuristic	# Landmarks	Avg Search Space	Avg Error	Avg Runtime (s)
Colorado	2.5E+07	ALT	28	6.461E+04	8.52%	9.441203811
Colorado	1.0E+07	ALT	11	6.369E+04	11.68%	5.643167405
Colorado	**5.0E+06**	**ALP**	**1886**	**4.580E+04**	**14.14%**	**3.166834619**
Colorado	5.0E+06	ALT	5	4.930E+04	13.01%	3.245015286
Colorado	**2.5E+06**	**ALP**	**1132**	**5.275E+04**	**17.19%**	**3.406351286**
Colorado	2.5E+06	ALT	2	6.791E+04	22.83%	3.937390599

We see enough data in the Colorado result to verify our claim. ALP can outperform ALT when analyzing large graphs[26] in a fixed-memory environment. However, it can fall prey to the constraint that it is restricted to one landmark per partition. We address this constraint a bit more with a suggestion for future research in Chapter 5.

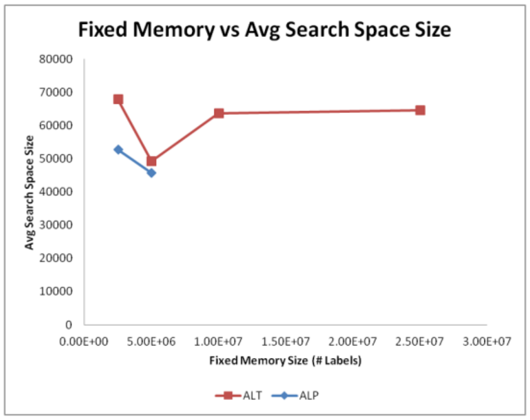


Figure 52 Colorado Fixed Memory vs. Average Search Space Size (Louvain)

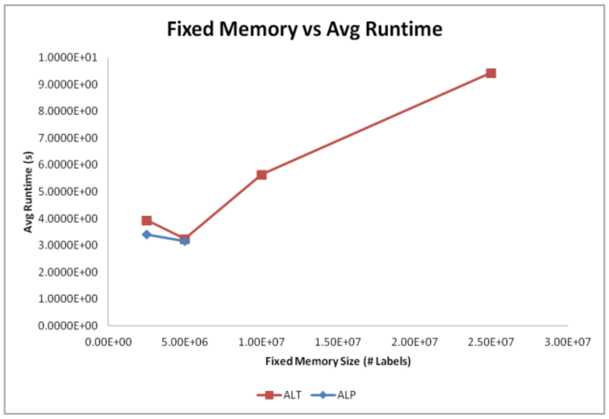


Figure 53 Colorado Fixed Memory vs. Average Runtime (Louvain)

Overall, this behavior for ALP against ALT is quite consistent for large graphs and has been seen in numerous test trials conducted outside of this fixed-memory experiment. The following figure summarizes, for all trials performed on real road graphs in all experiments, using random landmark selection, where ALP performs equally to or

[26] > ~1E5 Vertices, 5E5 Edges

better than ALT in terms of search space. The figure shows the percentage of queries for each graph of a given size in which ALP has equal or better performance.

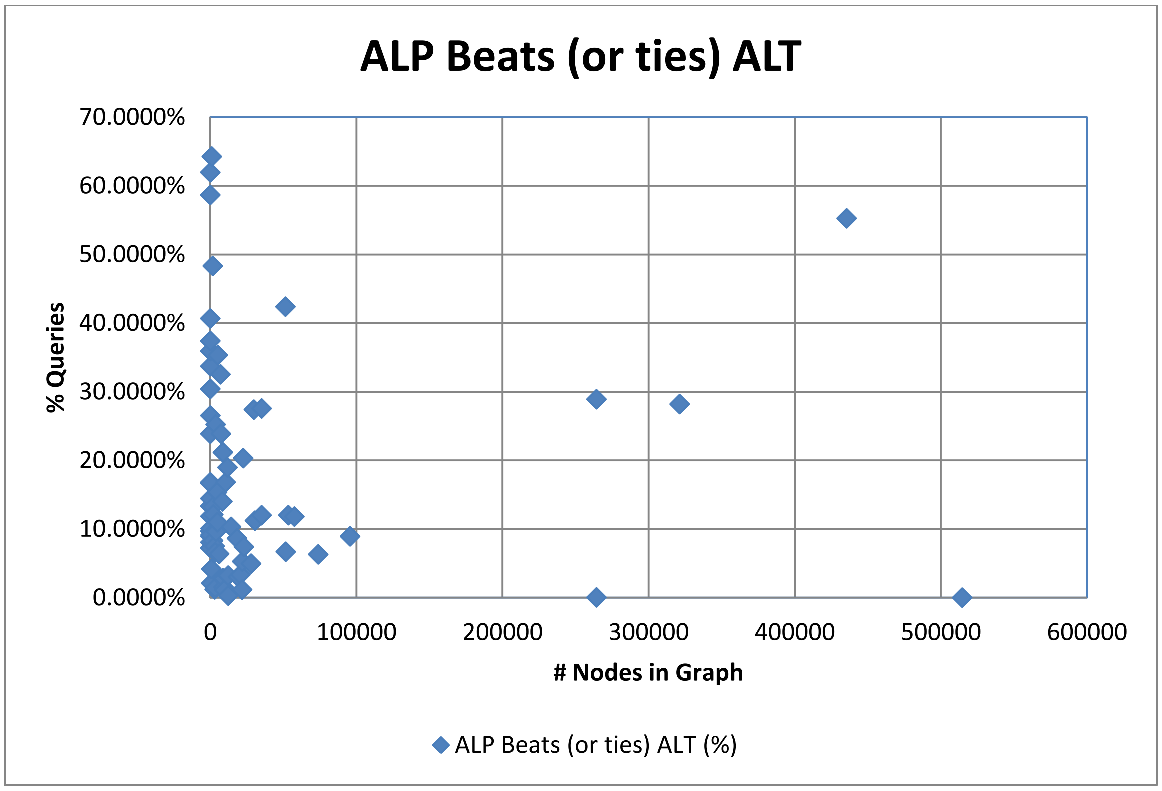


Figure 54 Percentage Of Queries in Which ALP has Equal or Better performance than ALT

Overall, this research result directly addresses the problem statement stated in Chapter 1 of this dissertation. We have shown that in a fixed-memory environment, ALP can outperform ALT on larger graphs with appropriate partitioning. We discuss what this appropriate partitioning requirement could be in the next section.

Findings

The intent of this section is to synthesize and discuss the results of data analysis in light of the research questions, literature review, and methodology laid out in the first three chapters. We note again here, that the novel feature of ALP is that it is a ***practical*** landmark-based heuristic, requiring significantly less storage space and computational time to preprocess its data structure while speeding up shortest path search. Here, we highlight patterns and themes that support this claim while also highlighting any ambiguities and inconsistencies that could leave the claim to question. Each subsection is broken down by a key observation of the behavior of ALP.

Key Observations: Greater Landmark Set Density Allows ALP to Outperform ALT

This is the primary finding of the research. While landmark selection algorithms have an effect on overall query performance, the density of the landmark set comparative to the size of the graph are the key factors that allow ALP to outperform ALT. The triangle inequality simply yields a tighter bound than the quadrilateral inequality for the path metric over the same or even a similar landmark set. Even for the metric space-based inequalities such as the one derived from the four-point condition, the triangle inequality is a simpler, stronger approach to achieving a lower bound. From the practical perspective, however, the final results of Experiment 2 show that ALT can suffer from its large space complexity in a real application scenario.

Using results from experimentation, we characterize this activity in terms of tradeoff for real graphs, here. For ALT and ALP, Figure 55 is a 3D logarithmic plot that illustrates the relationships between the number of nodes in the graph, number of landmarks used by each algorithm, and the tradeoff measurement described in the previous section for all trials run using real road graphs[27]. The plot shows a greater number of trials with ALT that demonstrate higher tradeoff values than trials of ALP. This trend continues to grow as the number of nodes in the graph gets larger. It also shows, in these instances, ALP's ability to use more landmarks with smaller tradeoff.

[27] This was done using trials in which ALT and ALP executed the same number of queries.

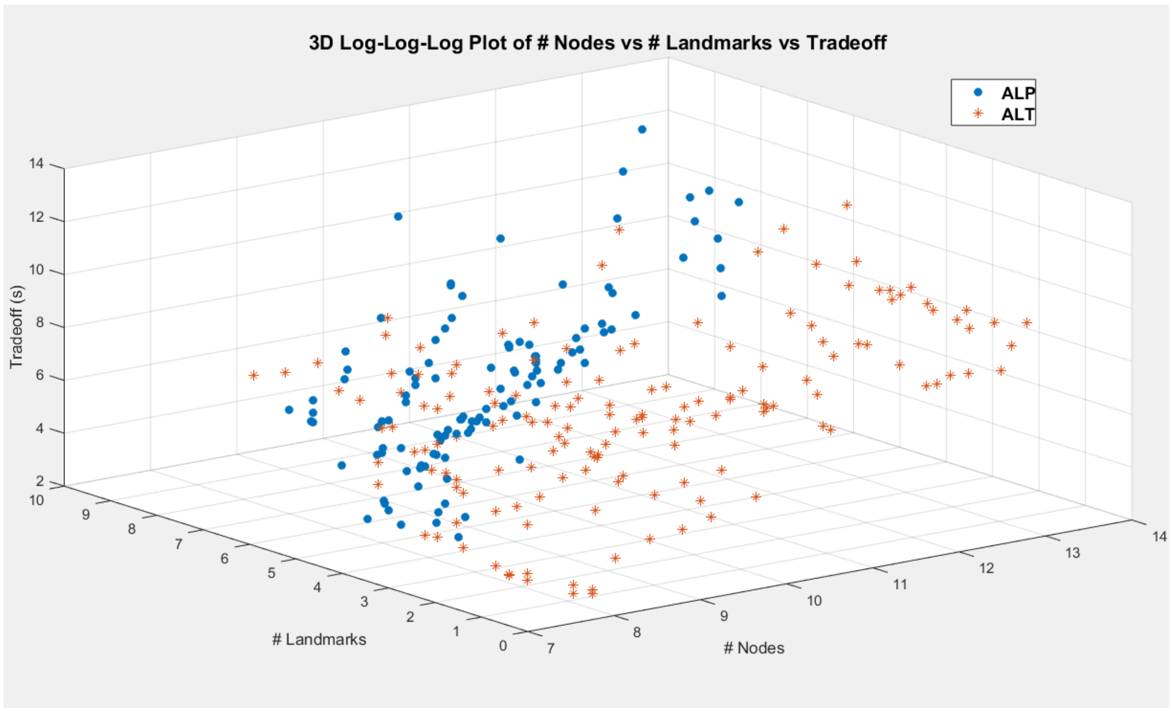


Figure 55 Log Plot of #Nodes vs # Landmarks vs Tradeoff for road graph trials shows worse tradeoff using less landmarks with ALT

For smaller graph datasets such as the Washington DC graph or the Rome graph from the experimental dataset, the benefit of landmark density for ALP will rarely aid it against ALT. The result data shows that this behavior is quite consistent, regardless of landmark selection. The only benefit ALT truly has when the number of nodes and edges in the graph grow as they do in Experiment 2 is the flexibility of the number of landmarks that it can choose. And recall, ALP's restriction in that regard is not a fundamental property of the algorithm, as the partitioning information simply serves as input. While we did not use a community detection algorithm that forms partitions that outperforms ALP results for Louvain partitions in our experiments, future research can focus on identifying the key properties of partitioning methods that optimize the choice of landmarks.

Key Observations: ALP Performance Gain Converges for Smaller Landmark Shortest Path Trees

A key observation during experimentation was that the average efficiency of shortest path queries almost always grows when the number of landmarks is increased. However, as shown in Figure 23, we see that performance gain tends to converge as the number of landmarks increases. The efficiency of a query in the ALP algorithm is dependent on the ALP estimate. The closer the estimate is to its actual distance (without overestimating) the better the estimate. To observe ALP's behavior in an environment with increasing landmarks, let us first look again at its heuristic estimates:

$$\pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v) = |d(l_1,v) - d(l_2,l_1)| - d(t,l_2) \tag{86}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,2\rangle}(v) = |d(l_1,v) - d(t,l_2)| - d(l_2,l_1) \tag{87}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,3\rangle}(v) = |d(l_2,l_1) - d(t,l_2)| - d(l_1,v) \tag{88}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,4\rangle}(v) = |d(l_1,v) - d(t,l_1)| \tag{89}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,5\rangle}(v) = |d(l_2,v) - d(t,l_2)| \tag{90}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,6\rangle}(v) = \frac{|d(l_1,v) - d(l_2,l_1)| \cdot |d(l_2,l_1) - d(t,l_2)| - d(l_1,v)\cdot d(t,l_2)}{d(l_2,l_1)} \tag{91}$$

$$\pi^{\boldsymbol{DL}}_{\langle t,7\rangle}(v) = |d(l_1,v) - d(l_2,l_1)| + |d(t,l_2) - d(l_2,l_1)| - d(l_2,l_1) \tag{92}$$

Now, we define the behavior of this heuristic when the number of landmarks increases. An increase in landmarks inherently means a decrease in the distances between all landmarks ($d(l_2,l_1)$). For the distances between landmarks and their vertices, the overall distances either stay the same or decrease. When vertices $v,t \in V$ do not share the same landmark, the following occurs:

$$\lim_{d(l_2,l_1)\to 0^+} \pi^{\boldsymbol{DL}}_{\langle t,1\rangle}(v) = d(l_1,v) - d(t,l_2) \tag{93}$$

$$\lim_{d(l_2,l_1)\to 0^+} \pi^{\boldsymbol{DL}}_{\langle t,2\rangle}(v) = |d(l_1,v) - d(t,l_2)| \tag{94}$$

$$\lim_{d(l_2,l_1)\to 0^+} \pi^{\boldsymbol{DL}}_{\langle t,3\rangle}(v) = d(t,l_2) - d(l_1,v) \tag{95}$$

$$\lim_{d(l_2,l_1)\to 0^+} \pi^{\boldsymbol{DL}}_{\langle t,6\rangle}(v) = -d(t,l_2) - d(l_1,v) \tag{96}$$

$$\lim_{d(l_2,l_1)\to 0^+} \pi^{\boldsymbol{DL}}_{\langle t,7\rangle}(v) = d(l_1,v) + d(t,l_2) \tag{97}$$

Because the shortest path graph is a metric space, $d(l_2, l_1)$ will never be negative, by definition. Therefore, in a weighted graph, we characterize the limit of the heuristic function as $d(l_2, l_1)$ approaches 0 from the right. Based on the above limits, as the number of landmarks increase, we can characterize the heuristic estimates as the search approaches the target as follows:

$$\lim_{d(l_2,l_1)\to 0^+} \pi^{\boldsymbol{DL}}(v) = d(l_1,v) + d(t,l_2) \tag{98}$$

Note that this characterizes the ALP heuristic as the number of landmarks increase and simply as the search nears the target. However, this limit at zero is still equal to the triangle inequality.

In the truly random (non-optimized) landmark selection case, because preprocessing is actually faster with smaller clusters and there is not a significant impact on preprocessing time for using more landmarks, the more landmarks that can be used to cover the graph, the better. However, we must be careful to cover the expensive preprocessing cost of computing the distance between all landmarks. Hypothetically, if all landmark nodes existed at an appropriate position on the graph border, this could result in growing out the full SPT for preprocessing time. Our results show that selectively choosing a moderate number of landmarks can result in optimal measurements across the board.

But what is this moderate number? While structural graph properties can play a significant role in the average efficiency on queries in a graph, the average efficiency

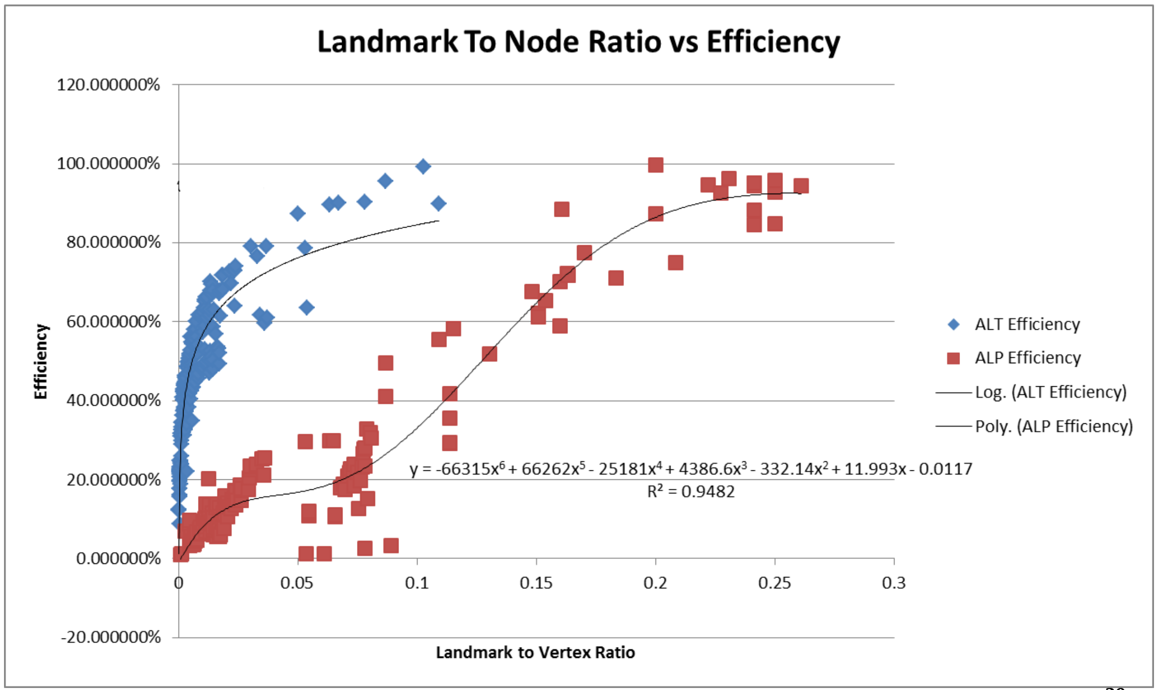


Figure 56 Plot of Landmark to Vertex Ratio vs Average Efficiency for 200 Trials[29]

increase created by increasing the number of chosen landmarks is strongly correlated to the number of vertices. In the context of landmark-to-node ratio, Figure 56 represents the average efficiency over 200 trials.[28] For ALT, we see a sharp increase as the number of vertices increase, maxing out in efficiency when approximately 10% of the vertices are chosen to be landmarks.

For ALP, we see this number is about at 25%. For ALT, it is difficult to tell from the acquired data precisely where its efficacy ends before hitting the 100% efficiency limit. The ALP trendline is approximately characterized by a sextic function, with an R-squared value of 0.9482[30]. We can analyze this function's derivative to get a sense of

[28] Each trial had 1000 queries.

[29] The line drawn for ALT is a very loose approximation of the data. Of linear, polynomial, and log scale, it was, however, the log scale was the best fit for the data that we had on hand. Nonetheless, we cannot make as adequate of assumptions about ALT based on this equation as we can about ALP based on the polynomial. Hence, no equation is featured in the image.

[30] The R-squared value grew as the degree of the polynomial grew.

when gains in efficiency begin to decrease as the number of landmarks grows. As seen in the figure, for ALP, let

$$f(x) = -66315x^6 + 66262x^5 - 25181x^4 + 4386.6x^3 - 332.14x^2 + 11.993x - 0.0117 \quad (99)$$

Where x equals the landmark-to-vertex ratio and $f(x)$ equals the average efficiency of trials with that landmark-to-vertex ratio. The derivative is then defined as

$$f'(x) = -397890x^5 + 331310x^4 - 100724x^3 + 13159.8x^2 - 664.28x + 11.993 \quad (100)$$

The only real root of this function's derivative is at 0.2567, where the function itself begins to have 100% efficiency scores[31]. The heuristic's efficiency cannot grow beyond 100% because it is admissible. Therefore, it makes sense that it would have a slope of zero once average efficiency becomes 100%. Finding a moderate number of landmarks to choose for preprocessing ALP requires, however, looking at the second derivative:

$$f''(x) = -1989450x^4 + 1325240x^3 - 302172x^2 + 26319.6x - 664.28 \quad (101)$$

The zeros for $f''(x)$ are 0.04199 and 0.12763. At these values for landmark-to-vertex ratio, the rate of increase of efficiency increase creeps to zero, which is very apparent in the graph. In other words, only an ordinary increase in efficiency will occur at these points.

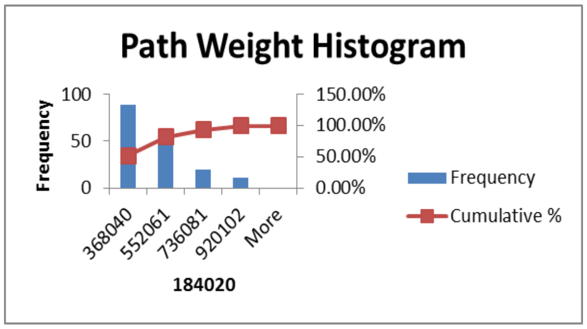
It should be noted that the analysis of the sextic equation provides a good approximation for these data collected over the course of experimentation. The polynomial of degree six was used because it had a significant R-squared value and was the lowest degree polynomial with real roots for both its first and second derivatives. The first five polynomials either had first or second derivatives close to this one. These roots

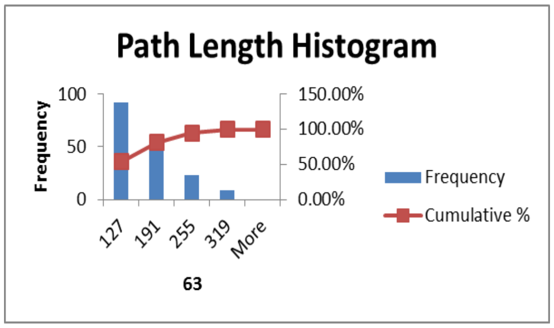
[31] Solved using Newton's Method

appear to be correct in terms of understanding the lull in efficiency gain after choosing a certain number of landmarks in the data.

Key Observations: Better Tradeoff through BFS during landmark selection

Prior to labeling for distributed embedding, a speedup in preprocessing time was achieved by leveraging breadth-first search (BFS) as opposed to Dijkstra's shortest path algorithm for path weights during the landmark selection. When dealing with weighted graphs, we cannot use a BFS measurement for the actual labeling of graph vertices. However, treating the graph as unweighted when selecting the landmarks produces strong results, as they give a rough estimate of actual path cost. Often, particularly for road graph datasets, the path length can act as a (somewhat) rough estimate of the distance. In the figures below, the path length and path weight histograms for an NYC road graph dataset take on roughly the same structure.





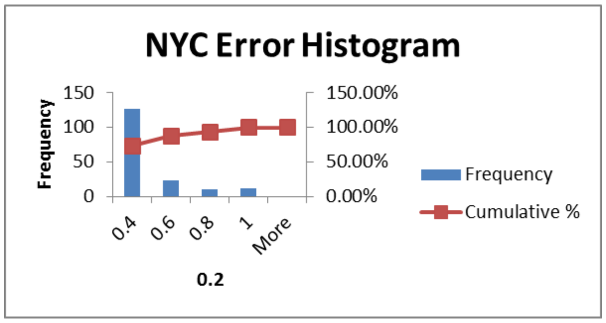


Table 26 NYC Path Histograms

The use of BFS is quick, granting better tradeoff in practical applications. Therefore, to speed up ALP's farthest and planar landmark selection, we use a BFS algorithm to identify farthest nodes or to compute the distance between coordinates. This is the same strategy that the originators of ALT used to improve *farthest*, creating *farthest-d*. This paradigm should be used when developing future landmark selection techniques and is, of course, subject to the application of the graph and shortest path search.

Key Observations: ALP Performance behaviors are consistent with ALT, except for tradeoff.

After a certain point, a higher landmark-to-node ratio has insignificant efficiency increases for ALP. Therefore, its true benefit is speeding up preprocessing, handling larger graph datasets, and faster practical implementations due to its ability to make fewer computations at each node. Outside of this, the heuristic's behavior changes similarly to ALT with respect to graph structure and algorithm parameters.

- **Both algorithms see performance increases as the number of landmarks grows.** Both heuristics demonstrate performance increases over a larger set of landmarks, with the increase in performance being capped by the ratio of number of landmarks to number of vertices.
- **Both algorithms show landmark selection's utility is simply to optimize efficiency within a set of bounds.** However, dramatic efficiency increases are not seen by varying landmark selection. As shown earlier, dramatic increases are guided much more by the number of landmarks. Though, clear optimality can be found at the ceiling of a roughly 4% efficiency window for both ALP and ALT.

- **Both algorithms have similar correlation between graph transitivity and query efficiency**. Both algorithms exhibit a straightforward relationship to this property of the graph structure. Transitivity (and like its close property, clustering coefficient) measures the relative frequency of triangles in the graph. Given that both ALT and ALP heuristics are heavily dependent on the triangle inequality, it makes sense that they are both influenced by the measure of triangles in the graph, at both extremes of the measure.

The second bullet point drives home a strong point. The primary, practical use case for the ALP algorithm is for landmark-based heuristic search in large graphs. The data show that when the graphs grow in size, both ALT and ALP experience a decrease in average efficiency.

In relation to the third bullet, another factor that can shift the behavior of ALP to outperform ALT is the length of the path being queried. Smaller values for transitivity and average clustering coefficient typically correlate to longer paths in the graph. However, the inverse is not necessarily true. Large paths could simply imply a large graph. Figure 57 illustrates the average approximation error for queries performed over all trials for ALP and ALT at given path lengths. Both algorithms have fairly similar theoretical performance as the path lengths grow larger, with the average approximation error approaching zero. And as noted earlier and illustrated in Figure 29, ALT begins to experience greater runtimes than ALP as path lengths become larger. This performance over large path lengths may be the second largest benefit of the ALP heuristic[32]. However, if a method could be created to implement ALT such that it could use a subset of its landmarks to compute its heuristic while maintaining a tighter lower bound, it could

[32] With the largest benefit being the drastic reduction in space complexity.

see better runtime performance that ALP. This method would inherently not be ALT, but a new class of algorithms that can get close to ALT’s approximations while reducing its memory requirements.

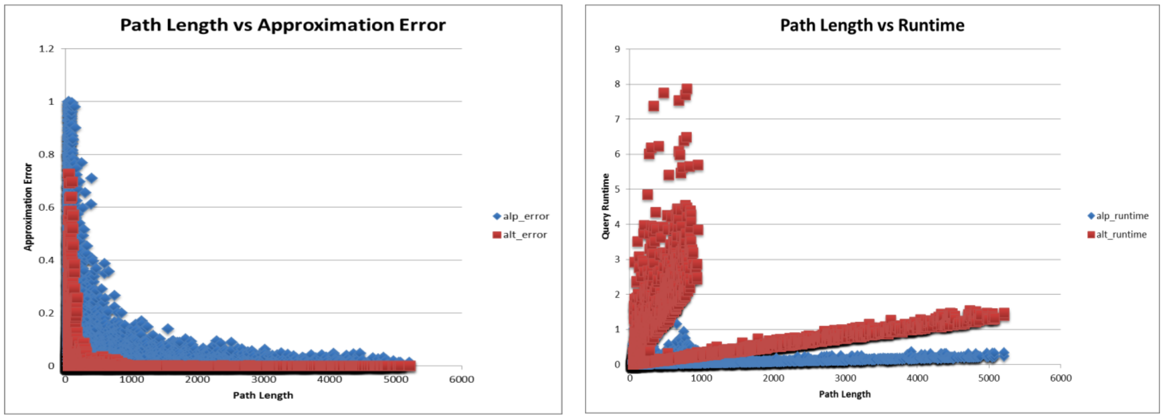


Figure 57 ALT Experiences Better Runtimes and better Approximation Error while ALP Experiences Better Runtimes over Growing Path Length

Despite the algorithms’ similarities, the tradeoff and query runtime (if the number of landmarks scale along with the graph size) of ALT is not practical in many use cases for graphs of size V5 and up. The plots below take the two key variables for computing tradeoff and illustrate them for all trials of 1000 queries.[33] Note the drastic difference in the scales for each plot. We see that the ALP graph most closely follows a quadratic polynomial function whereas the ALT graph follows more of a power law (albeit with a somewhat low R-squared value). The trials of this experiment demonstrate that ALP typically has both a lower preprocessing time and a lower rate of increasing tradeoff over all collected data.

[33] Intentionally excluded are the synthetic graph trials. The shapes of those graphs would skew this picture.

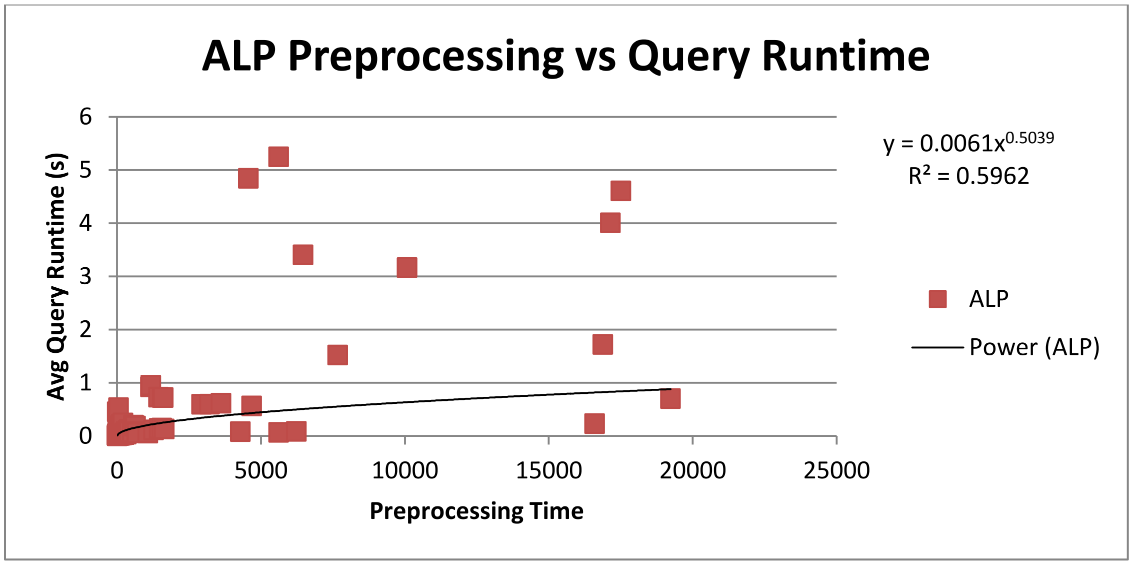


Figure 58 ALP Preprocessing vs Query Runtime for Trials of 1000 Queries on Real Road Graphs

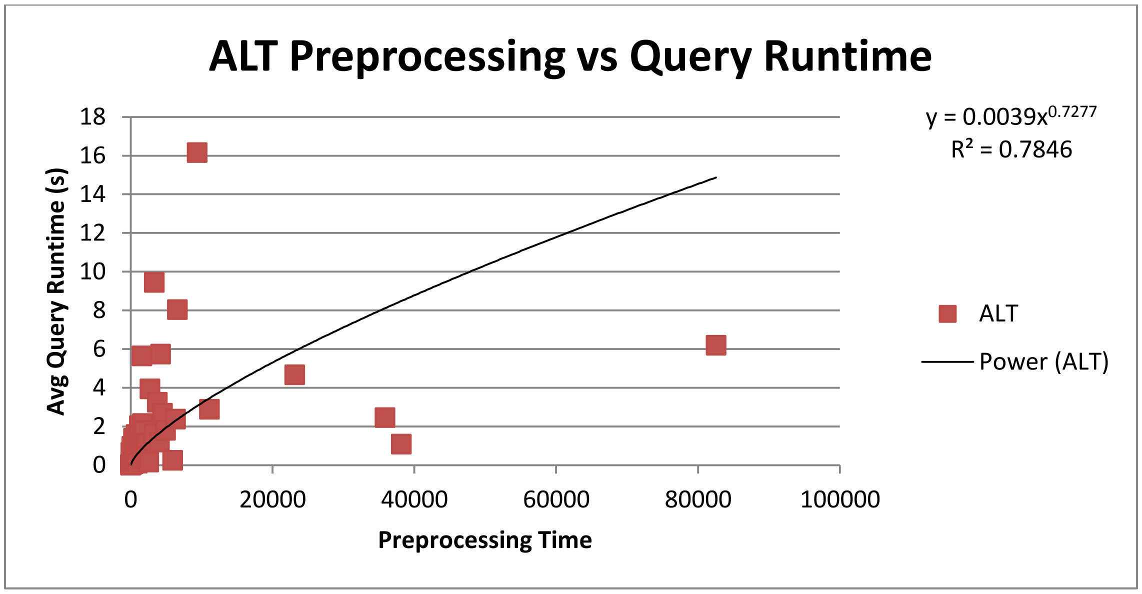


Figure 59 ALT Preprocessing vs. Query Runtime for Trials of 1000 Queries on Real Road Graphs

In practical use cases, such as when road graphs are loaded for temporary path query sessions, ALP serves much higher utility than ALT both for preprocessing time and runtime in a normal computing environment. The four bar graphs below illustrates the degree to which ALP presents a better overall tradeoff and average preprocessing time for all real road graph trials studied in this dissertation. They also further drive home the notion that ALP has greater utility in larger graphs in a normal compute environment. While runtime is a machine-dependent and implementation-dependent metric, the result data described in this chapter demonstrate ALP outperforming ALT for a straightforward Python implementation in large graphs.

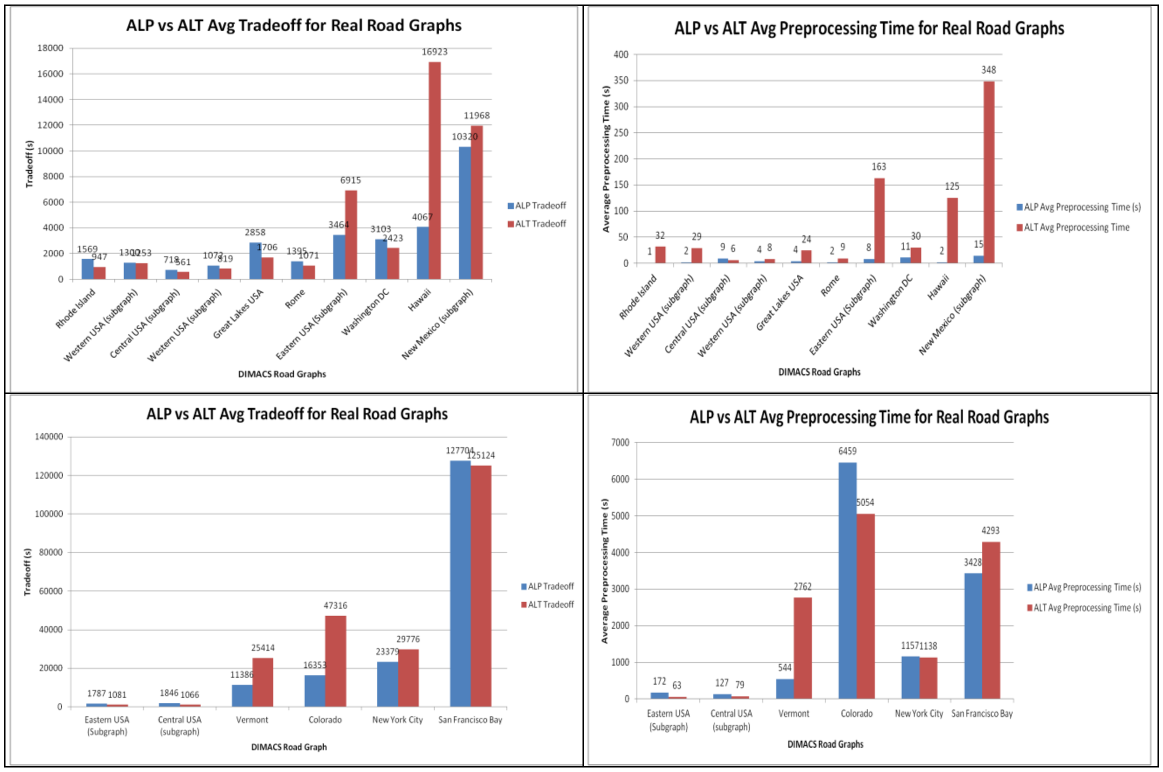


Figure 60 Four Charts demonstrating Overall Tradeoff for ALP vs ALT Trials on Real Road Graphs

Summary

We have evaluated the performance bounds and landmark selection algorithms for ALP, as well as its performance in comparison to ALT. We have successfully demonstrated and given justification for ALP having stronger performance in a fixed memory environment over larger graphs. We end this chapter by summarizing the answers to the first two research questions proposed in Chapter 1.

What landmark selection techniques theoretically fit best with ALP?

The landmark selection methods that were used for experimentation demonstrate approximately a four percent range of efficiency. At the scale of millions of vertices, this becomes significant. The average preprocessing time and efficiency over ALP trials is displayed in Table 27. Katz centrality is explicitly excluded from this table because of its inability to converge on some larger graphs. PageRank, load, and closeness centrality worked best with ALP, providing consistent efficiency across datasets while supporting. In certain trials, particularly in larger graphs, betweenness centrality also provided sufficient speedups, as well. Random can provide sufficient speedups, but is clearly non-deterministic. In this table, optimized random has fairly high efficiency because it has been computed in every trial, even on graphs where average efficiency is quite high. In general, centrality measure-based landmark selection has much better tradeoff as it informs the heuristic of the graph structure while efficiently identifying landmarks and growing shortest path trees. This type of selection is trivial to compute and could provide for the fastest form of preprocessing to achieve a speedup over Dijkstra's algorithm. Future research will demonstrate the benefits and detriments for both the use of more centrality measures for preprocessing and the use of max, min, and mode for the vectors

Landmark Selection	Average Preprocessing Time	Average Efficiency
Optimized random	321.24661469117	0.29315061
Farthest-d	1335.61578881421	0.24726657
Planar	37.05859077324	0.25356666
Betweenness Centrality	57.55697033005	0.25227837
PageRank (Max)	119.04858074428	0.29675092
PageRank (Mode)	67.41231769577	0.28929956
PageRank (Min)	52.29438178512	0.28659692
Closeness Centrality	52.30552490364	0.28664093
Load Centrality	62.45711426451	0.29073376
Farthest (Eccentricity)	499.05140597165	0.29271267

Table 27 Average Preprocessing and Efficiency for ALP Landmark Selection over All Trials

produced by these centrality measures to select landmarks.

What are the ideal characteristics for landmark shortest path trees? In other words, how much preprocessing and memory is required for ALP to maintain its key benefits?

In terms of the ideal characteristics of landmark shortest path trees, the ALP heuristic provides more efficient search over a larger number of landmarks. The use of a larger number of landmarks implies smaller shorter path trees and larger knowledge of the overall graph, as the distances between all landmarks must be recorded. The optimal properties of the shortest path tree require larger paths between the landmarks with short paths between the vertices owned by the landmark and the landmark itself. The data show that this provides the most optimal ALP estimates. This is also what allows ALP to outperform ALT in the fixed-memory environment. The opposite can also be true (rarely), where the trees create small paths between the landmarks and significantly large paths between the vertices owned by the landmarks. This second scenario happens only for a few instances, which is why ALP systematically shows improvement over a growing number of landmarks.

The informal answer to this second question is that ALP requires significantly less preprocessing than that of ALT. However, the more memory ALP uses, the more on par it can be with algorithms such as ALT. Because ALP's data structure is typically so small, the number of landmarks used can often be chosen liberally. The time that it takes to grow ALT shortest path trees for each landmark over the entire graph is astounding as the graphs grow. Meanwhile, ALP maintains fairly consistent preprocessing times. The bottleneck in preprocessing for larger number of landmarks in ALP only comes from computing the distances between landmarks, meaning that partitioning with too much fidelity can result in preprocessing times similar to ALT. However, most trials demonstrated significantly smaller preprocessing time for ALP in comparison to ALT. In the context of memory, ALT consistently filled up memory and tapped into swap space for graphs of over 40,000 nodes. This will vary for different implementations, as NetworkX objects turned out to be large and clunky. As will be discussed further in the next chapter, a full C/C++ implementation, and not simply the Cython conversion, should be used in the future to compare both algorithms.

How does the algorithm behave as the number of landmarks used to guide the search increases?

This correlates strongly with the previous question. ALP always experiences a performance increase over a larger number of landmarks, reaching 100% efficiency for our trials when the landmarks make up 25.6% of the graph and suffering smaller gains in efficiency after 4.2% and 12.8%. Once again, these cutoff points are for the structure of our datasets and the fact that this many landmarks can be used during preprocessing is a testament to ALP's benefits. ALP can handle a larger number of landmarks without significant increase to preprocessing time, whereas ALT preprocessing time grows substantially. This property makes ALP a more feasible preprocessing algorithm than ALT and other similar algorithms in fixed-memory environments, such as embedded systems.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

In this chapter, we interpret, examine, and qualify the results of the investigation and draw inferences from them.

Conclusions

In this dissertation, we identified a heuristic for A* that leverages a data structure of size $\theta(|\boldsymbol{L}|^2 + |V|)$ as opposed to ALT's $\theta(|\boldsymbol{L}| \cdot |V|)$. This data structure is formed through a new embedding process, which only requires growing and storing the distances of a shortest path tree for a subgraph (graph partition) owned by a landmark. With this type of embedding, the new heuristic for A* search, dubbed ALP, leverages polygon inequalities to estimate the distance from a vertex to the search goal. This dissertation primarily used quadrilateral inequalities to guide A* search. We experimentally tested the performance bounds of this heuristic, multiple landmark selection techniques based on those of ALT, as well as new techniques that leverage the structure of the partition, and trials that compare the heuristic directly to ALT over the same datasets in a fixed-memory environment. Through experimentation and theory, we have identified the key parameters, bounds, and behaviors of the algorithm in the context of road graphs and synthetic graph data structures.

Implications

We have identified each theoretical scenario in which ALP's heuristic function can give a better estimate of the distance to an A* search goal than ALT's. We have established that ALP typically outperforms ALT when analyzing larger graphs in a fixed-memory environment due to ALP's ability to leverage more landmarks. We also established that in cases in which the ALT heuristic has greater average estimates than the ALP dual landmark heuristic, ALP can still computationally outperform ALT and Dijkstra's algorithm can potentially outperform A* using either ALP or ALT. The fact that Dijkstra's algorithm can computationally outperform both of these methods as graphs scale should serve as a cautionary example for other methods of shortest path preprocessing. Too many computations at a particular vertex can mean a substantial decrease in ***practical*** performance on average, even with significant theoretical performance.

One more open-ended research question has not been answered: *In what ways can this be applied to path planning? What real-world applications exist for ALP that were previously impractical to solve with ALT?* Experimentation with ALP in comparison to ALT led us towards an answer to this question. First, ALT in the Python NetworkX environment created an extremely high memory cost. For larger graphs, this cost often came without significant speedup to Dijkstra's (though still more algorithmically more efficient than ALP). If coded using a lower-level language in a smaller environment, such as a C++ program for a Raspberry Pi (Halfacree & Upton, 2012), ALT would still be an infeasible heuristic for A* for graphs on the order of tens of thousands of vertices. ALP now makes operations in such an environment possible. Even if a device could handle

ALT in that environment, if the device were processing graphs on the order of hundreds of thousands of nodes, the experiments in this dissertation allow us to conclude that ALP would outperform ALT in terms of runtime efficiency (and still in terms of memory).

Prior to this research, forming graphs based on collected data and running analytics such as shortest path queries would be infeasible for graphs above such a threshold, as the search space would grow too high for Dijkstra's algorithm and the memory requirement would grow too high for ALT. Now, A* has a class of algorithms for heuristic estimation that require neither the massive search space size of Dijkstra nor the massive data structures of algorithms such as ALT. It even has the capacity to store less information than algorithms such as PCD, which were created to reduce search space size. In the real world, ALP can enable smaller, memory-limited devices without constant internet or local network connection to efficiently navigate paths in large graph datasets. Note that distributed embedding is the real memory-reducing property, here. Much of ALP's benefits over ALT are derived from the fact that ALP can leverage the distributed embedding environment while ALT cannot.

ALP's benefit can reduces the requirement of energy required to power small-scale devices that have to perform path computation on graphs. Such localized navigation planning can allow for more intelligent planning to occur in denied areas such as space or military domains. Also, ALP can be a reasonable algorithm to use in cloud computing, when a large graph dataset is updated periodically but would benefit from a speedup to Dijkstra. Depending on the period of time between graph changes, ALP provides a reasonable preprocessing time to speed up shortest path queries in this scenario.

Recommendations

In this dissertation, experiments were initially conducted on diverse classes of synthetic networks and then the focus turned to road networks. The next step in characterizing ALP would be to further explore ALP's behavior in comparison to ALT's on a broader range of graphs. This broader dataset should contain graphs possessing particular characteristics such that more comparative information about both ALT and ALP can be gleaned. Further, the other algorithms mentioned in Chapter 2 for preprocessing shortest path queries should be run on this broader set of graphs as well to identify similarities and differences among algorithms, as well as identifying where they have optimal utility.

In these studies, ALP uses only the basic quadrilateral inequalities derived from triangle inequalities as well as Ptolemy's inequality and the Four-Point inequality[34]. Future research can include the use and selection of varying heuristics for special quadrilaterals along with that of other polygons induced on the graph. Such research would address the difficult problem of extracting information such as angle and inscribed shapes before the heuristic could be computed. Future theoretical research could also contribute to automated methods of deriving these inequalities for higher-sided shapes.

Also, we know that quadrilateral inequality bounds are not typically tighter than triangle inequality bounds for more moderate size graphs. However, they consistently showed performance on par with the triangle inequality over larger path lengths. This meant that the inefficiency often stemmed from the visited vertex and target sharing the same landmark during the search. At that point, the search becomes equivalent to ALT with one (very close) landmark. Once the search reaches that point, it is obvious that

[34] Conditionally

ALT would outperform ALP as long as ALT is using more than one landmark. Performance can be increased by allowing multiple landmarks within a subgraph, such that once the visited vertex and target do fall within the same partition, they can execute a more efficient version of ALT.

The ALP class of algorithms differs in behavior from the ALT class of algorithms because of ALPs lower asymptotic space complexity (i.e., distributed landmark embedding). These properties change the average expected computational performance of PPSP queries for each landmark selection technique. Because of this, the ALP paradigm may speed up other algorithms that leverage the triangle inequality. One example comes from identifying duplicate strings and objects in XML databases (Weis & Naumann, 2004). Specifically, because pairwise calculations of all string tokens in a dataset need to be performed to accurately identify duplicate strings, expensive edit distance calculations for this type of query are infeasible for larger datasets. Instead, a series of filters are typically applied to these string token pairs to drastically reduce the total number of edit distance calculations required. A new class of filters could be created that actively use information about relationships between other string tokens in the corpus to significantly reduce the required number of candidate pairs for comparison. The new filters would rely on the generalized polygon inequality to bound the possibility of chunks of data to be candidate duplicates. Identifying the use of other geometric inequalities in this manner could provide previously unforeseen benefits to such algorithms.

Before such a thing could be studied, however, we note that one of the limitations of the experiments in this dissertation is that we assume some partitioning of the graph as an input parameter to ALP when forming its data structure. For utility, another class of experiments would be to start with as many landmarks as each method allows (where memory is bounded) and then, in the case of ALP, to grow classes of the partition around each of the landmarks. This would provide maximum utility in these other application spaces.

Summary

Modern navigation planning requires the ability to regularly compute the shortest path between two points in massive road networks. In such cases, preprocessing algorithms are used to increase the performance of shortest path queries. Many such algorithms require heavy upfront computation and storage. Few algorithms concern themselves with the space complexity required to aid queries. The problem that this research addresses is that modern shortest path preprocessing algorithms have space and preprocessing time requirements for large-scale graphs that are impractical for resource-limited devices.

ALT describes a preprocessing technique for shortest path queries that, prior to query time, chooses a relatively small number of landmark nodes in a graph and computes the distances between all vertices and these landmarks, allowing the A* algorithm to leverage the triangle inequality during search queries. The algorithm works as follows: For a simple graph G with vertices $, l, C \in V$, where l is a landmark vertex chosen beforehand, the shortest path distances between each vertex serves as a distance metric, allowing the graph to form a metric space. Therefore, for the distances between vertices $A, l, C \in V$, the following reverse triangle inequality holds:

$$|d(A,l) - d(l,C)| \leq d(A,C) \tag{1}$$

ALT uses this inequality to create a heuristic estimate for A* upon a visit to vertex A. By computing and storing the values between each chosen landmark and all vertices in the graph apriori, this lower bound is computed for each chosen landmark vertex l. The maximum of these lower bounds is the ALT heuristic function's value, denoted as $\pi^{\boldsymbol{L}}$. By using information about multiple landmarks, new lower bounds can be computed from either generalized polygon inequalities or inequalities specific to any shape embedded within the graph. The use of these new lower bounds as a heuristic has resulted in a new class of algorithms called ALP, for A*, Landmarks, and Polygon Inequalities. The base case for this class of algorithms is the heuristic used for the ALT algorithm. Here, we demonstrate that polygon inequalities for quadrilaterals can also be used to establish the lower bounds for shortest path queries in a graph. The following reverse quadrilateral inequalities hold for a graph G with source and target vertices $s, t \in V$ and chosen landmarks $l_1, l_2 \in V$:

$d(s,t) \geq \lvert d(l_1,s) - d(l_2,l_1) \rvert - d(t,l_2)$	**Reverse Quadrilateral Inequalities**
$d(s,t) \geq \lvert d(l_1,s) - d(t,l_2) \rvert - d(l_2,l_1)$	
$d(s,t) \geq \lvert d(l_2,l_1) - d(t,l_2) \rvert - d(l_1,s)$	
$d(s,t) \geq \lvert d(l_1,s) - d(t,l_2) \rvert$	***l_1=l_2***
$d(s,t) \geq \lvert d(l_1,s) - d(t,l_2) \rvert$	***l_1=l_2***
$d(s,t) \geq \dfrac{\lvert d(l_1,s) - d(l_2,l_1) \rvert \cdot \lvert d(l_2,l_1) - d(t,l_2) \rvert - d(l_1,s) \cdot d(t,l_2)}{d(l_2,l_1)}$	**Ptolemy's Inequality**
$d(s,t) \geq \lvert d(l_1,s) - d(l_2,l_1) \rvert + \lvert d(t,l_2) - d(l_2,l_1) \rvert - d(l_2,l_1)$	**Four-Point Condition**

The first five are derived from the triangle inequality as applied to quadrilaterals. A potential problem with these inequalities is that they have ability to generate negative lower bound estimates. However, because multiple points are used, a varying set of inequalities can be generated to estimate distances. When attempting to estimate lower bounds using ALP, other inequalities should be considered such that the highest possible estimate can be used. We use the sixth and seventh equation, derived from Ptolemy's inequality and the Four-Point condition on metric spaces, respectively, as a concrete example for the dual landmark case. Just as with ALT, the maximum over the set of these lower bounds are used to tighten the lower bound for the distance between two vertices. We denote the maximum of the six equations for ALP as $\pi^{\boldsymbol{DL}}$, ALP's dual-landmark heuristic for A*. The following describes $\pi^{\boldsymbol{DL}}$ as a heuristic:

- $\pi^{\boldsymbol{DL}}$ is an admissible heuristic for A*.
- Using distributed embedding, $\pi^{\boldsymbol{DL}}$ is not consistent.
- $\pi^{\boldsymbol{DL}}$ does not dominate $\pi^{\boldsymbol{L}}$ over the same set of landmarks.
- $\pi_t^{\boldsymbol{L}}$ does not dominate $\pi_t^{\boldsymbol{DL}}$ over different landmark sets.

ALP's data structure can exhibit a space complexity of $\theta(|\boldsymbol{L}|^2 + |V|)$ (as opposed to ALT's $\theta(|\boldsymbol{L}|^2 \cdot |V|)$ using the following technique, called *distributed embedding*. With a partitioned graph as input, the dual landmark approach identifies a single landmark within each partition and computes a shortest path tree for the subgraph induced by each chosen landmark's graph partition. Each vertex in the graph is labeled with an identifier, signifying its landmark partition and the distance to and from its corresponding landmark. Any of the landmark selection methods for ALT can be used for the subgraph induced by the graph partition to select an optimal set. The final step of this process is a shortest path calculation between the selected landmarks. This is achieved by a Dijkstra's shortest path tree computation from each landmark that has a stopping condition of all landmarks being in the tree. Allowing each landmark in the graph to access only a subgraph limits the size of the data structure used at query time, significantly reduces the preprocessing time, and bounds the number of operations performed to compute the heuristic.

We implemented both the ALP and ALT algorithms in a Python 2.7 environment with the aid of the NetworkX 1.9 library. For larger graphs, we enhanced this implementation using Cython and GCC optimizations. We used this environment to implement the following ALT-based landmark selection techniques for ALP:

- *random* and *random-p*
    - Simple random landmark selection and randomly selected vertices over a series of trials, respectively
- *farthest-d* and *farthest-ecc*
    - Choosing the farthest landmarks from the current set of landmark vertices and choosing the landmark in each cluster with highest eccentricity, respectively
- *planar*
    - Choose landmarks on the periphery of their respective subgraph
- *betweenness*
    - Choose landmarks with the highest betweenness centrality in their subgraph

Each of these techniques were used within each subgraph to identify a single landmark within the subgraph to add to the overall set. During development, it was noticed that good landmark selection for ALP is focused on computations made within the subgraph. That combined with the ability to trivially compute centrality measures for a subgraph allowed us to also create landmark selection techniques similar to *betweenness* for the following types of centrality measures:

- PageRank
- Load
- Katz
- (Vertex) Closeness

For PageRank, the maximum, minimum, and mode vertices were trialed to identify which would provide optimal results.

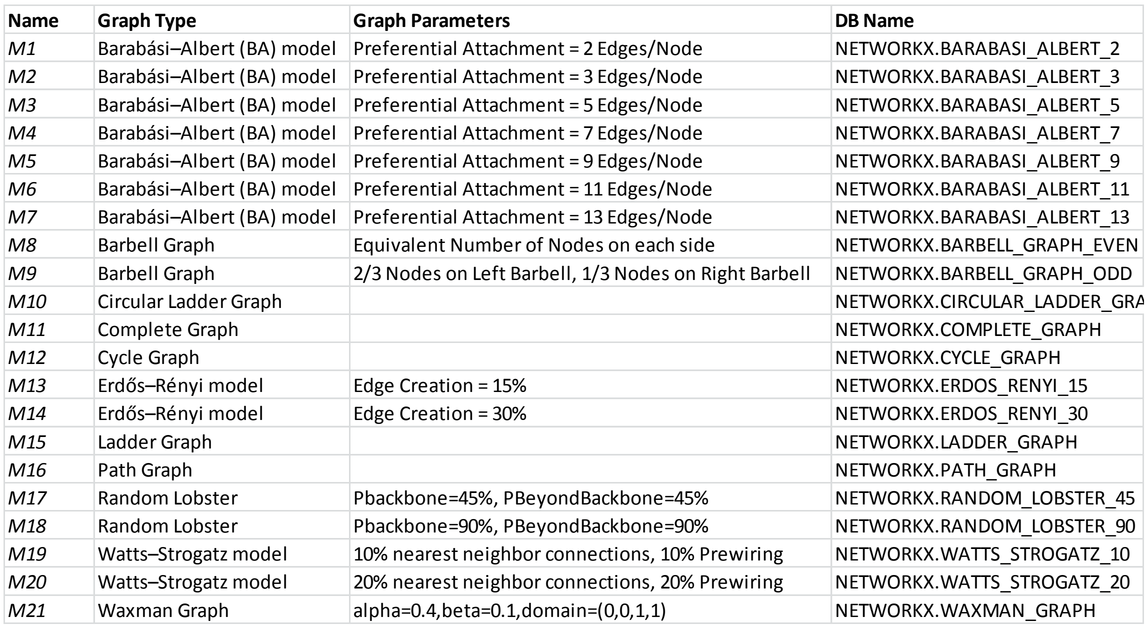
During experimentation, we ran thousands of trials for ALP with different road graph and synthetic graph datasets to characterize its behavior, comprehend its performance bounds, compare landmark selection methods, and understand how it compares to ALT. We see that graph transitivity and average clustering coefficients are strong factors in the efficiency of ALP, much like other search algorithms. More importantly, we see that it has significantly high performance over large path lengths, allowing the ALP heuristic to outperform the ALT heuristic. Further, as the number of landmarks for ALP grows, its efficiency increases. Though, gains in performance start to become fairly constant after the ratio of number of landmarks to vertices grows beyond a certain point. In terms of landmark selection, we see that centrality measure-based landmark selection provides a trivial method to select landmarks based on a graph partition's structure and has strong performance in the ALP environment. We also showed that varying amongst the type of landmark selection techniques proposed here results in a 4% difference in average query efficiency. In each of the runs against ALT, we see that ALP’s behavior varies in similar ways to ALT, with ALT simply providing a better estimate on average. The two algorithms behave similarly in the context of graph structure and size, but not in terms of number of chosen landmarks. ALT can reach 100% efficiency scores with fewer landmarks than ALP. However, performing preprocessing and storing a data structure for graphs that have nodes that are more than in the tens of thousands requires significant resources. Finally, in a fixed-memory environment, simulating a small or embedded system with limited resources, ALP heuristics outperformed ALT as the size of the graphs grew. On the order of hundreds of thousands of vertices, ALP was able to leverage denser landmark sets to make better heuristic estimates than ALT. Further, ALP’s preprocessing time requirements grew more slowly than ALT’s as the number of landmarks grew. Because of this, ALP is a more practical algorithm that can be used for a variety of applications when preprocessing is an option.

Appendices

Appendices contain all research instruments used, as well as any relevant additional materials such as sample interview transcripts, sample coding schemes, summary charts, and so forth. Each item that is included as an appendix is given a letter or number and listed in the table of contents.

Appendix A: Graphs and Applied Mathematics Concepts

Recall that the following graphs were used for experimentation. Below the table are their definitions and sources on their origin:



Name	Graph Type	Graph Parameters	DB Name
M1	Barabási–Albert (BA) model	Preferential Attachment = 2 Edges/Node	NETWORKX.BARABASI_ALBERT_2
M2	Barabási–Albert (BA) model	Preferential Attachment = 3 Edges/Node	NETWORKX.BARABASI_ALBERT_3
M3	Barabási–Albert (BA) model	Preferential Attachment = 5 Edges/Node	NETWORKX.BARABASI_ALBERT_5
M4	Barabási–Albert (BA) model	Preferential Attachment = 7 Edges/Node	NETWORKX.BARABASI_ALBERT_7
M5	Barabási–Albert (BA) model	Preferential Attachment = 9 Edges/Node	NETWORKX.BARABASI_ALBERT_9
M6	Barabási–Albert (BA) model	Preferential Attachment = 11 Edges/Node	NETWORKX.BARABASI_ALBERT_11
M7	Barabási–Albert (BA) model	Preferential Attachment = 13 Edges/Node	NETWORKX.BARABASI_ALBERT_13
M8	Barbell Graph	Equivalent Number of Nodes on each side	NETWORKX.BARBELL_GRAPH_EVEN
M9	Barbell Graph	2/3 Nodes on Left Barbell, 1/3 Nodes on Right Barbell	NETWORKX.BARBELL_GRAPH_ODD
M10	Circular Ladder Graph		NETWORKX.CIRCULAR_LADDER_GRA
M11	Complete Graph		NETWORKX.COMPLETE_GRAPH
M12	Cycle Graph		NETWORKX.CYCLE_GRAPH
M13	Erdős–Rényi model	Edge Creation = 15%	NETWORKX.ERDOS_RENYI_15
M14	Erdős–Rényi model	Edge Creation = 30%	NETWORKX.ERDOS_RENYI_30
M15	Ladder Graph		NETWORKX.LADDER_GRAPH
M16	Path Graph		NETWORKX.PATH_GRAPH
M17	Random Lobster	Pbackbone=45%, PBeyondBackbone=45%	NETWORKX.RANDOM_LOBSTER_45
M18	Random Lobster	Pbackbone=90%, PBeyondBackbone=90%	NETWORKX.RANDOM_LOBSTER_90
M19	Watts–Strogatz model	10% nearest neighbor connections, 10% Prewiring	NETWORKX.WATTS_STROGATZ_10
M20	Watts–Strogatz model	20% nearest neighbor connections, 20% Prewiring	NETWORKX.WATTS_STROGATZ_20
M21	Waxman Graph	alpha=0.4,beta=0.1,domain=(0,0,1,1)	NETWORKX.WAXMAN_GRAPH

1. Barabási–Albert model (Zadorozhnyi & Yudin, 2012) – random scale free graph using a preferential attachment mechanism
2. Barbell Graph (Ghosh, Boyd, & Saberi, 2008) – simple graph obtained by connecting two copies of a complete graph by a bridge (path)

3. Circular Ladder Graph (Ghosh et al., 2008) – graph corresponding to the skeleton of an *n*-prism
4. Complete graph (Alspach, Bermond, & Sotteau, 1990)– graph in which each pair of graph vertices is connected by an edge
5. Cycle graph (Gross & Yellen, 2005) – a graph containing a single cycle through all nodes
6. Erdős–Rényi graph (Erdős & Rényi, 1959) – Random graph in which all pairs of vertices share an edge with a common probability
7. Ladder Graph (Noy & Ribó, 2004) – A planar undirected graph obtained as the Cartesian product of two path graphs, one of which has only one edge
8. Path Graph (Gross & Yellen, 2005) – A tree containing only vertices of degree 2 and 1
9. Random Lobster Graph (Golomb & Lushbaugh, 1996) – A tree in which the removal of leaf nodes leaves a tree in which every vertex is either on the central stalk or one edge away from the central stalk known as a caterpillar graph
10. Watts-Strogatz Graph (Watts & Strogatz, 1998)- Random graph formed with small world properties, such as short path lengths and high clustering coefficients

Appendix B: Data Description

This section of the appendix hosts the description of data used collected during experimentation. The following series of tables is the data dictionary for the dissertation MySQL database.

alt_alp_comparison_trials

Table comments: Table connecting Trial IDs, Experiment IDs, and Graph IDs

Column	**Type**	**Null**	**Default**	**Comments**
trial_id *(Primary)*	int(11)	No		Trial ID
experiment_id	int(11)	No		Experiment ID
graph_id	int(11)	Yes	*NULL*	Graph ID

Indexes

Keyname	**Type**	**Unique**	**Packed**	**Column**	**Cardinality**	**Collation**	**Null**	**Comment**
PRIMARY	BTREE	Yes	No	trial_id	32362	A	No	
fk_graph_id_idx	BTREE	No	No	graph_id	3236	A	Yes	
fk_experiment_id_idx	BTREE	No	No	experiment_id	42	A	No	

embedding_techniques

Table comments: Descriptions of landmark selection techniques

Column	**Type**	**Null**	**Default**	**Comments**
et_id *(Primary)*	int(11)	No		Embedding method ID

description	varchar(45)	No		Description of Embedding Method

Indexes

Keyname	**Type**	**Unique**	**Packed**	**Column**	**Cardinality**	**Collation**	**Null**	**Comment**
PRIMARY	BTREE	Yes	No	et_id	13	A	No	

error

Table comments: Table of Approximation Error for Each Query

Column	**Type**	**Null**	**Default**	**Comments**
query_id	int(11)	No		Query ID
error	decimal(30,15)	No		Initial Approximation Error for search

Indexes

Keyname	**Type**	**Unique**	**Packed**	**Column**	**Cardinality**	**Collation**	**Null**	**Comment**
query_fk_idx	BTREE	No	No	query_id	3224515	A	No	

experiments

Table comments: Table of experiments

Column	**Type**	**Null**	**Default**	**Comments**
experiment_id *(Primary)*	int(11)	No		Experiment ID
description	varchar(250)	Yes	*NULL*	Description of Experiment
start_time	datetime	Yes	*NULL*	Experiment Time (US Eastern

				Standard Time)
result	varchar(10)	Yes	*NULL*	SUCCESS OR FAILURE

Indexes

Keyname	**Type**	**Unique**	**Packed**	**Column**	**Cardinality**	**Collation**	**Null**	**Comment**
PRIMARY	BTREE	Yes	No	experiment_id	741	A	No	

graphs

Table comments: Table of the graphs used for experimentation

Column	**Type**	**Null**	**Default**	**Comments**
graph_id *(Primary)*	int(11)	No		Graph ID
directed	bit(1)	No		nx.is_directed
num_nodes	int(11)	Yes	*NULL*	Number of Nodes in the graph
num_edges	int(11)	Yes	*NULL*	Number of Edges in the graph
estrada_index	decimal(60, 30)	Yes	*NULL*	Estrada Index of the graph
is_chordal	bit(1)	Yes	*NULL*	Whether or not the graph has chordal structure
largest_clique_size	int(11)	Yes	*NULL*	nx.graph_clique_number
num_max_cliques	int(11)	Yes	*NULL*	nx.graph_number_of_cliques(g)
transitivity	decimal(20, 15)	Yes	*NULL*	Transitivity of graph structure
average_clustering	decimal(20, 15)	Yes	*NULL*	Average Clustering of the graph
average_node_connectivity	decimal(20, 15)	Yes	*NULL*	nx.average_node_connectivity(g)
edge_connectivity	int(11)	Yes	*NULL*	nx.edge_connectivity(g)
node_connectivity	int(11)	Yes	*NULL*	nx.node_connectivity(g)

diameter	int(11)	Yes	*NULL*	nx.diameter(g)
size_periphery	int(11)	Yes	*NULL*	Number of nodes with eccentricity equal to the diameter len(nx.periphery(g))
is_eulerian	bit(1)	Yes	*NULL*	nx.is_eulerian(g)
average_shortest_path_length	decimal(20, 16)	Yes	*NULL*	Average length of shortest paths in the graph
num_connected_double_edge_ swaps	int(11)	Yes	*NULL*	Number of successful double edge swaps where the number of swaps is set to the number of edges in the graph: nx.connected_double_edge_s wap(g, num_edges)
is_tree	bit(1)	Yes	*NULL*	Whether or not the graph is a tree
density	decimal(20, 17)	Yes	*NULL*	Density of the graph
graph_name	varchar(250)	Yes	*NULL*	What data does the graph represent? (e.g. NYC, San Francisco)

Indexes

Keyname	**Type**	**Unique**	**Packed**	**Column**	**Cardinality**	**Collation**	**Null**	**Comment**
PRIMARY	BTREE	Yes	No	graph_id	3045	A	No	

heuristics

Table comments: Table of A* heuristics

Column	**Type**	**Null**	**Default**	**Comments**
heuristic_id *(Primary)*	int(11)	No		Heuristic ID
description	varchar(15)	No		Description of Heuristic (e.g. ALT, ALP, Dijkstra)

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	heuristic_id	12	A	No	

preprocessing

Table comments: Stores preprocessing information about the trial run for each heuristic used

Column	Type	Null	Default	Comments
preprocessing_id *(Primary)*	int(11)	No		Preprocessing ID
trial_id	int(11)	No		Trial ID
heuristic_id	int(11)	Yes	*NULL*	Heuristic ID
graph_id	int(11)	Yes	*NULL*	Graph ID
preprocessing_time	decimal(20,7)	Yes	*NULL*	Total time for preprocessing (Landmark Selection + Shortest Path Tree Growth)

Indexes

Keyname	Type	Unique	Packed	Column	Cardinality	Collation	Null	Comment
PRIMARY	BTREE	Yes	No	preprocessing_id	16701	A	No	
fk_graph_id_idx	BTREE	No	No	graph_id	1670	A	Yes	
fk_heuristic_id_idx	BTREE	No	No	heuristic_id	16	A	Yes	

query

Table comments: Table of shortest path queries. Each row is a single source-target PPSP query

Column	**Type**	**Null**	**Default**	**Comments**
query_id *(Primary)*	int(11)	No		Query ID
trial_id	int(11)	No		Trial ID
heuristic_id	int(11)	No		Heuristic ID
embedding_method	int(11)	Yes	*NULL*	Landmark Selection Technique
source	int(11)	No		Source vertex
target	int(11)	No		Target Vertex
path_length	int(11)	No		Number of vertices traversed
num_landmarks	int(11)	Yes	*NULL*	Number of Landmarks
runtime	decimal(14,7)	No		Runtime
search_space_size	int(11)	No		Search Space Size
num_operations	int(20)	No		number of arithmetic operations executed for this query
total_estimates	int(20)	No		Total estimates made. (Should be equal to the number of visits)
path_weight	decimal(30,10)	Yes	*NULL*	Actual path cost of shortest path query

Indexes

Keyname	**Type**	**Unique**	**Packed**	**Column**	**Cardinality**	**Collation**	**Null**	**Comment**
PRIMARY	BTREE	Yes	No	query_id	15213235	A	No	
fk_heuristic_id_idx	BTREE	No	No	heuristic_id	18	A	No	
fk_embedding_method_idx	BTREE	No	No	embedding_method	18	A	Yes	
fk_trial_id_idx	BTREE	No	No	trial_id	16428	A	No	

Appendix C: Supplemental Experiment Data

In this section of the appendix, we attach extra results of interests that further support the claims made in this dissertation. This section also provides more detailed data regarding the experiments of Chapter 4. While these details were not critical in proving our claims and answering the research questions, they do further characterize the ALP algorithm in the context of the ALT algorithm and could prove useful in future research.

Experiment 1 Extension: Graph Efficiency vs Structure

The following is a table of the average efficiency of queries at each graph scale.

Graph Category	Algorithm	# Queries	Efficiency
V1	ALP	1058124	0.35639541
V1	Dijkstra	1068912	0.22648232
V2	ALP	880766	0.46972808
V2	Dijkstra	890455	0.20704487
V3	ALP	873815	0.31745607
V3	Dijkstra	629171	0.08550012
V4	ALP	109302	0.23096096
V4	Dijkstra	182254	0.12391506
V5	ALP	253818	0.11325759
V5	Dijkstra	267314	0.02320272
V7	ALP	19073	0.03724294
V7	Dijkstra	16287	0.00320424
E1	ALP	384815	0.37774329
E1	Dijkstra	321820	0.33719738
E2	ALP	1091292	0.55122999
E2	Dijkstra	1133013	0.24995331
E3	ALP	1009983	0.28840927
E3	Dijkstra	830083	0.0882232
E4	ALP	349616	0.11146852
E4	Dijkstra	351080	0.06686537
E5	ALP	305159	0.12243344
E5	Dijkstra	358145	0.04312324
E6	ALP	16481	0.19287126
E6	Dijkstra	17486	0.09854249
E7	ALP	36055	0.14320232

Graph Category	Algorithm	# Queries	Efficiency
E7	Dijkstra	41269	0.08432151

Table 28 Average Efficiency of Queries at Each Graph Scale

One other measurement that was used to measure ALP performance involves using performance of the Dijkstra's shortest path algorithm as an basis for runtime measurement. At every graph scale, both ALP and ALT have speedups over Dijkstra. For each vertex and edge scale, we divide the average efficiency of ALP with A* runs by the average efficiency of Dijkstra runs to establish a *Vertex Efficiency Multiplier* and an *Edge Efficiency Multiplier*, respectively. Figure 61 and Figure 62 illustrate the efficiency of ALP over basic Dijkstra's for the graph scales noted in Figure 17.

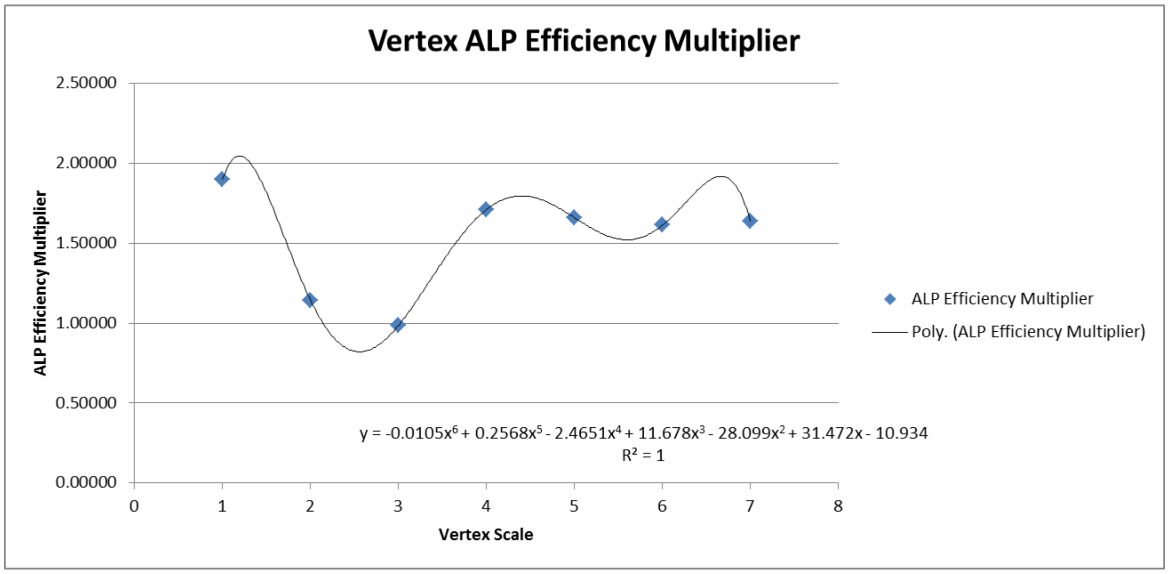


Figure 61 Efficiency Multipliers for Vertex Scales[35]

[35] The two equations noted in the figure are anecdotal and will always differ as graph structures vary. Simply, these are the equations derived for these runs. Nonetheless, the methods of deriving them may be useful in determining whether to use ALP or not for similar graphs.

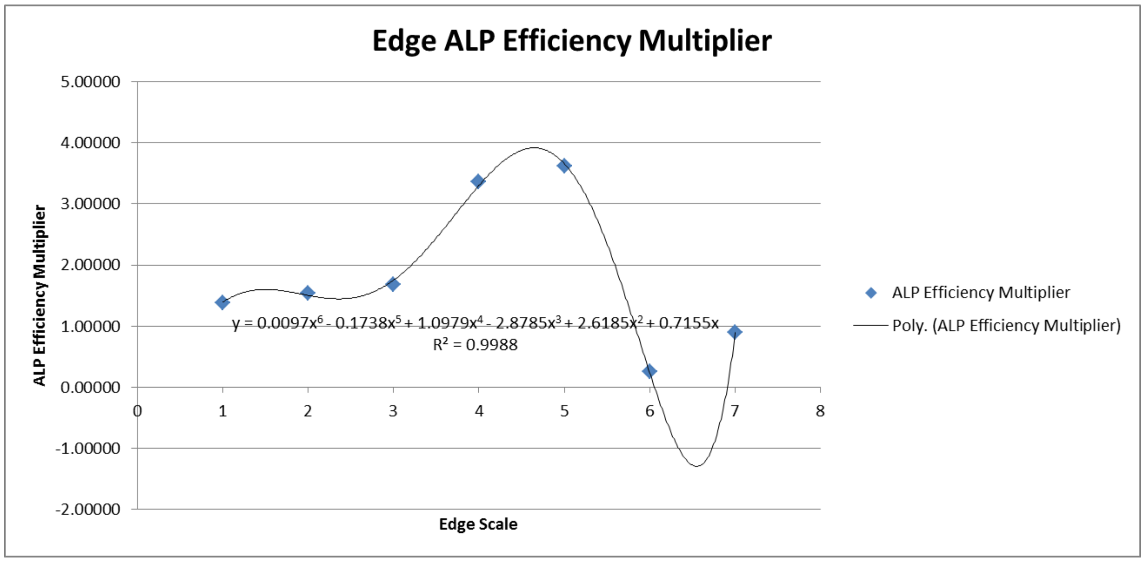


Figure 62 Efficiency Multiplier for Edge Scales

Experiment 1 Extension: Varying Graph Structure Trials

For real road datasets, we take the largest directed subgraph of the dataset and also execute 1000 queries on 1000 random source-target vertex pairs. The real graphs fell into all vertex classes except for V2 and V6. Each edge class was used during this experiment. Just as with synthetic graphs, for each real road graph dataset, for communities derived from each hierarchy level, we analyze the efficiency of all queries run on ALP with optimized random landmark selection. We set a maximum number of communities and inherently, a maximum number of landmarks, to 2500. This maximum allowed for querying enough graph variants such that trends could be confirmed. Table 29 and Figure 63 describe the number of runs and average ALP efficiency for each graph.

To perform more trials, we used subgraphs of each of the datasets. In Table 29, the names of the graph datasets are suffixed with their number of vertices and number of edges. Some graphs were run as both undirected and directed graphs during experimentation.[36] We see, here, that directed graphs have higher average efficiency in ALP, as do smaller graphs.

Name	#Nodes	#Edges	# Queries	Avereage Efficiency
Washington DC	9522	14832	23993	0.02035347
NYC (Undirected)	264346	365050	19073	0.03724294
Rhode Island	53288	68496	6990	0.03760536
Rome (Undirected)	3353	4831	27919	0.04239815
United States (Eastern)	35103	42902	3543	0.04337649
United States (Eastern)	49404	57960	2997	0.04827491
Vermont	95671	105124	1998	0.05703919
United States (Western)	28652	36906	3996	0.06557798
United States (Western)	51447	62272	2997	0.07205112
Great Lakes	34198	42957	3996	0.083298
Luxembourg	84136	85579	193294	0.08340697
United States (Western)	13499	17421	3996	0.08795465
United States (Eastern)	24728	30000	3996	0.09198804
New Mexico	29381	33476	3996	0.11281136
Great Lakes	11773	15861	3996	0.11348083
United States (Eastern)	13816	16819	2997	0.12692009
United States (Eastern)	29796	32528	4041	0.12784286
New Mexico	28115	32736	3996	0.13894572
United States (Central)	11584	13188	2997	0.13962499
New Mexico	15221	17919	3996	0.14147232
Hawaii	9237	10711	5994	0.14916109
United States (Western)	8294	9851	3001	0.16377774
Great Lakes	3700	4483	2997	0.17008902
United States (Eastern)	5573	6391	2997	0.17019366
United States (Central)	5327	6121	2997	0.18771198
United States (Central)	9549	10677	2997	0.19655155
United States (Central)	7276	7856	2997	0.21757875
United States (Central)	5422	6105	2997	0.223868
Rome (Directed)	3353	4831	614089	0.3252697

Table 29 Real Road Graph Shortest Path Average Query Efficiency

[36] Real road graphs are directed graphs unless otherwise specified.

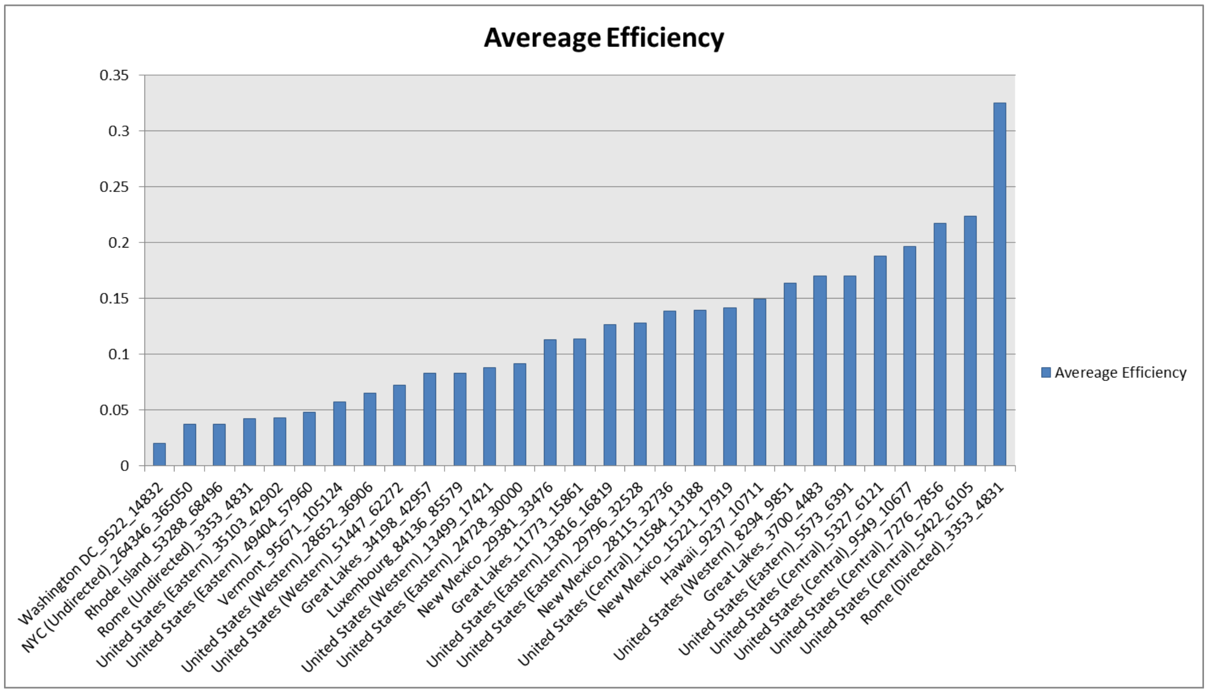


Figure 63 Average Efficiency for Real Road Graphs

Experiment 1 Extension: Landmark Selection Trials

We pull samples from the landmark selection series of trials and plot it in Figure 64 and Figure 65 to demonstrate the behavior of each trial with respect to a trial's average query distance and the three metrics. We once again confirm a small difference in efficiency between the most efficient landmark selection technique (in this case, *farthest-ecc*) and the least efficient (*planar*).

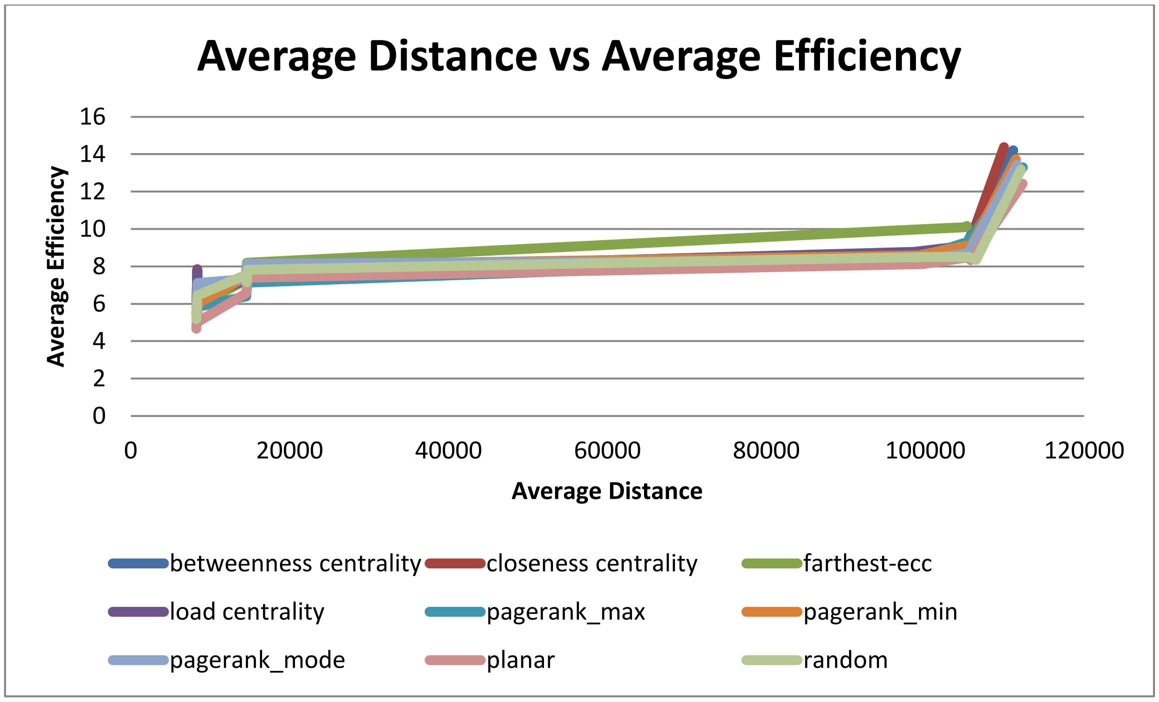


Figure 64 Average Distance vs Efficiency

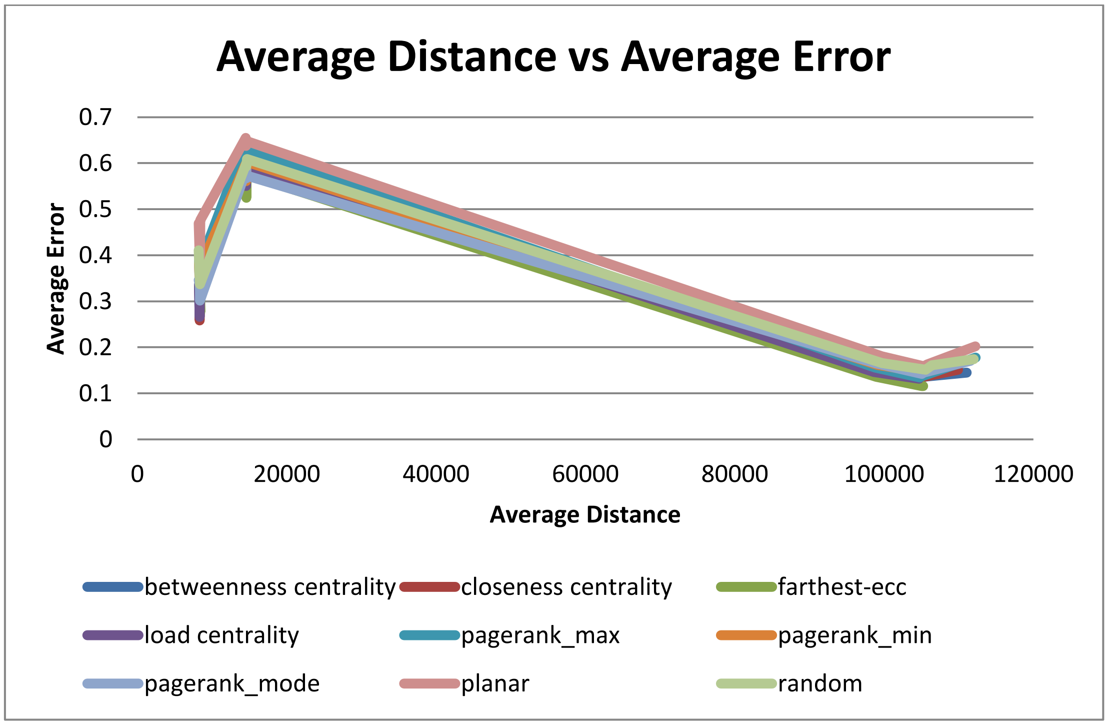


Figure 65 Average Distance vs Average Error

Experiment 2 Extension: ALT vs. ALP: Graph Structure and Landmark Selection

We began Experiment 2 with a small set of trials involving measurement of ALT's performance against ALP's performance over four synthetic graph structures using ALT-based landmark selection techniques. Using a small set of graph structures comprised of the graphs that have significantly variable behavior under different parameters, optimized random, farthest-d, planar, and betweenness centrality landmark selection were performed on each graph. For the synthetic graphs, the following graphs were used:

- Barabási Albert Graph with 3 edges per vertex
- Barabási Albert Graph with 7 edges per vertex
- Erdős–Rényi Graph with 15% Edge Creation
- Watts-Strogatz Model with 10% Nearest Neighbor

Each of these graphs were created for scales V1, V2, and V4 by starting with 100 nodes and multiplying the nodes by 10 until we got to 10000. Each of the figures below illustrates the dramatic difference in average efficiency between ALT and ALP for varied graph structures. We used the four implemented types of landmark selection for ALT and their ALP equivalent for embedding.[37] The runs with maximum efficiency are highlighted in the illustration. During analysis, the structure of the graph did not have a significant impact for graphs at these scales. However, for these scales, it is obvious that ALT is the more efficient algorithm to use, with average efficiency scores as large as ten times that of ALP.

[37] *Optimized Random, Farthest-D, Planar, Betweenness Centrality*

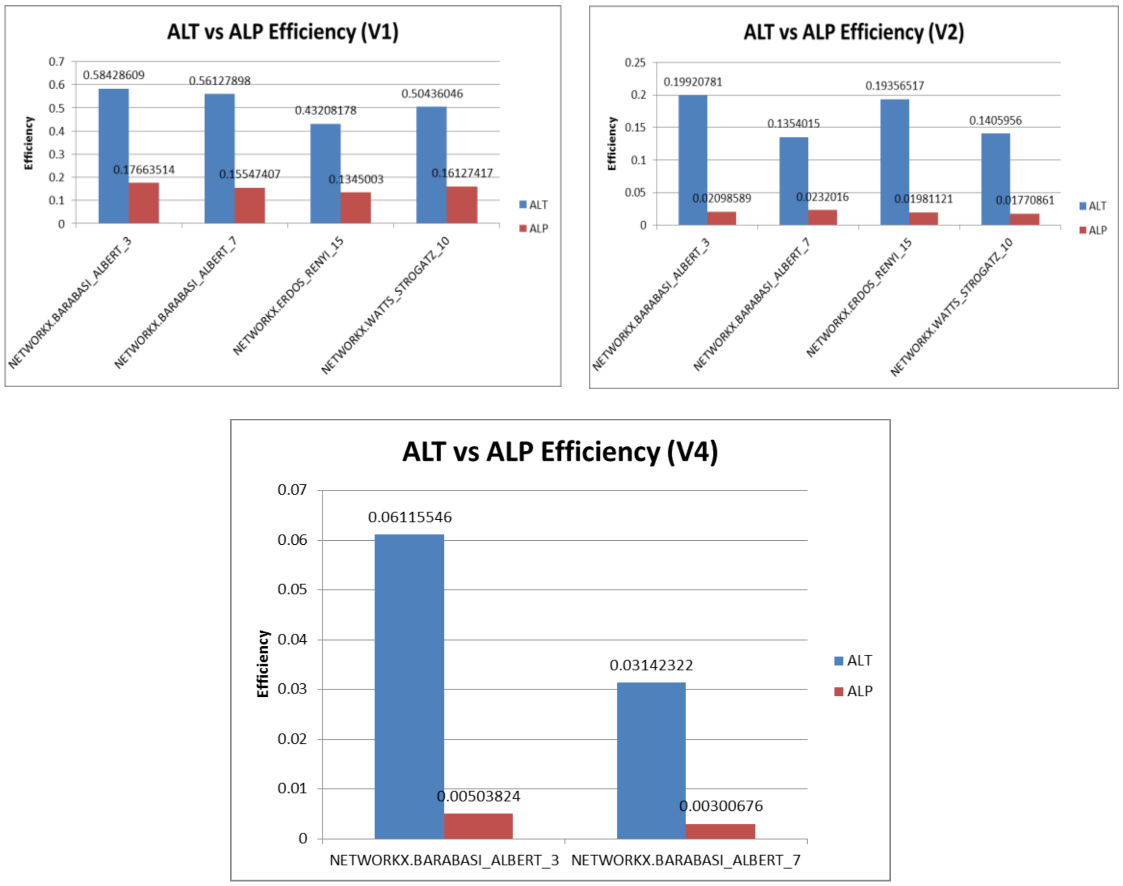


Figure 66 ALT vs ALP: Significant Difference in Efficiency for Graphs of size V1, V2, and V4

Size seems to have more of an impact on the difference in efficiency than structure. This is because ALT and ALP are based on the same kind of geometric inequalities. Therefore, they behave similarly over different graph structures.

In the next set of trials, we highlight the differences in performance for each type of landmark selection in real graphs. First, we run each landmark selection technique that is native to ALT (*random*, *farthest-d*, *planar*, and *betweenness centrality*) for both ALT and ALP, respectively. In Table 30, we take a look at an exemplar of the dramatic difference in preprocessing between ALT and ALP. The results highlight preprocessing times for a dataset representing a subset of the United States Eastern seaboard. The goal was to identify and label 389 landmarks. The preprocessing time for ALT for this ~30,000 vertex graph was almost five times that of ALP, at best, for the ALT preprocessing techniques. This, along with the results above, is a clear demonstration, that with straightforward implementations, ALT is a heuristic that is simple to run on smaller graphs (<V4), but begins to lose its utility in comparison to ALP at a certain scale. Meanwhile, as shown in our previous experiments, ALP's utility, in the context of tradeoff, improves for larger graphs. As stated earlier, the vast difference in preprocessing time is obvious from the methodology.

Heuristic	Landmark Selection	Time (s)
ALT	Random	471.0266
ALP	Random	55.49462
ALT	Planar	942.0947
ALP	Planar	69.54433
ALT	Betweenness Centrality	964.592
ALP	Betweenness Centrality	88.71064

Table 30 ALT vs ALP Preprocessing

Because of this, the figures below demonstrate the utility of ALP in comparison to ALT. ALP should be used for larger graphs, barring restrictions on application. We observe data taken from 291 combinations of graph types and landmark selection methods for ALP and compare it to 109 that were run for ALT.[38] The runtimes for each data point was measured for 1,000 queries. ALT exhibits such high preprocessing times that the total time for its trial runs significantly exceeds that of ALT's after about 7,500 nodes or 15,000 edges. The values in the charts below are on a log scale. ALT commonly suffers from having larger tradeoff values, due to its significantly long preprocessing times.

[38] It was infeasible to run as many ALT trials, particularly when it came to larger graphs, because of ALT's preprocessing times and significant memory requirements. Therefore, we leverage a scatter plot to make the comparisons in this section apparent.

Finally, among these trials, we identify three graphs from which to further analyze landmark selection. Each of these graphs were run with 1000 queries for ALT and ALP after using each landmark selection method, using the same number of landmarks, but their own individual landmark selection. In Figure 69, we see that the average efficiency of each of these three graphs under each landmark selection stays fairly the same, with the exclusion of planar and Farthest-D for ALT. In particular, we see orders of magnitude difference between ALT and ALP, in terms of efficiency. In Figure 70 and Figure 71, we see orders of magnitude difference for preprocessing time, as well, as ALT takes a significant amount of time to compute its shortest path trees. The preprocessing time bar chart is at the log scale, as the preprocessing times scale exponentially for ALT as the graph grows. We see that *Planar* and *Farthest-ecc* demonstrate the worst tradeoffs for the larger New Mexico graph, but not for the smaller graphs. Overall, the tradeoff for ALT grows to be significantly worse than that of ALP, over larger graphs, regardless of landmark selection.[39] In comparison with ALP, we see that the efficiencies across landmark selection techniques are roughly the same at each graph, regardless of landmark selection. This is because landmark selection for ALP is guided significantly influenced by the partitioning of the graph.

[39] A bar chart of these tradeoffs can be found in the backmatter.

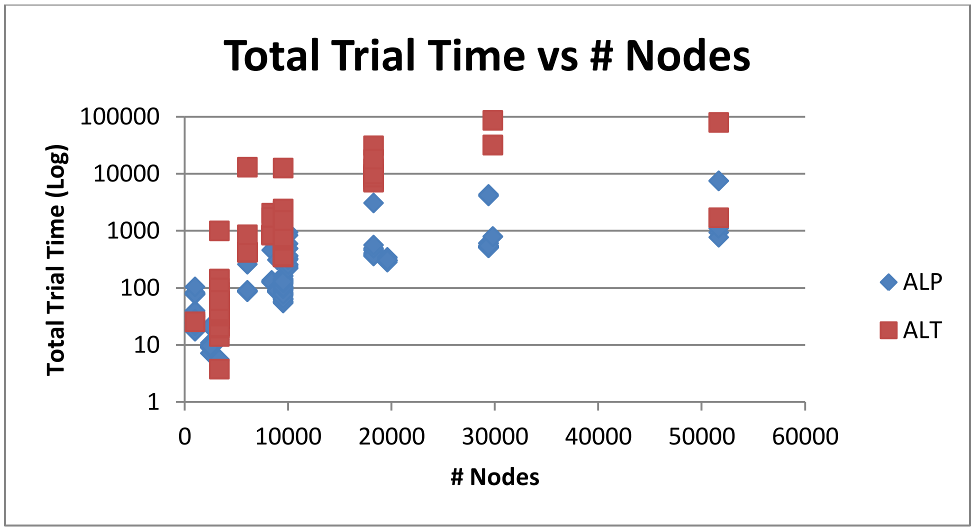


Figure 67 ALT vs ALP: Total Trial Time for Increasing Nodes

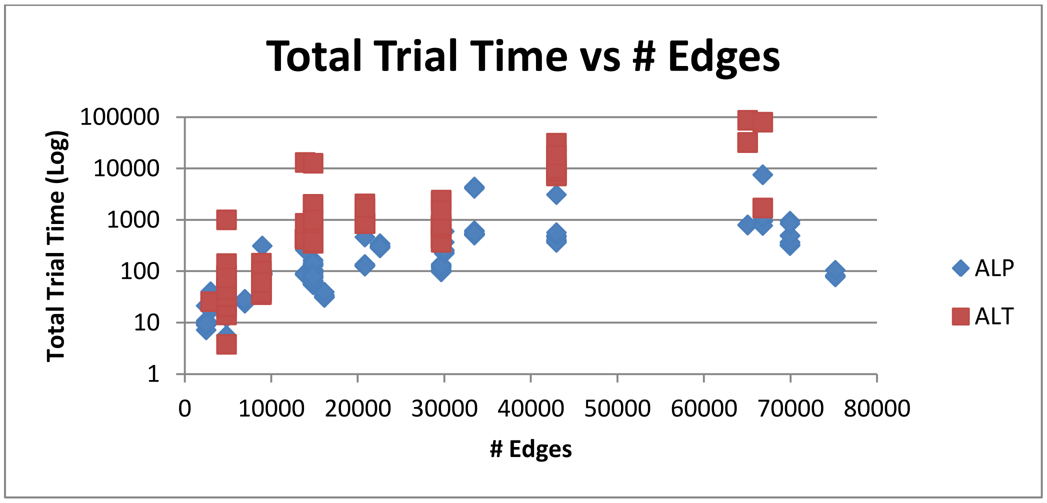


Figure 68 ALT vs ALP: Total Trial Time for Increasing Edges

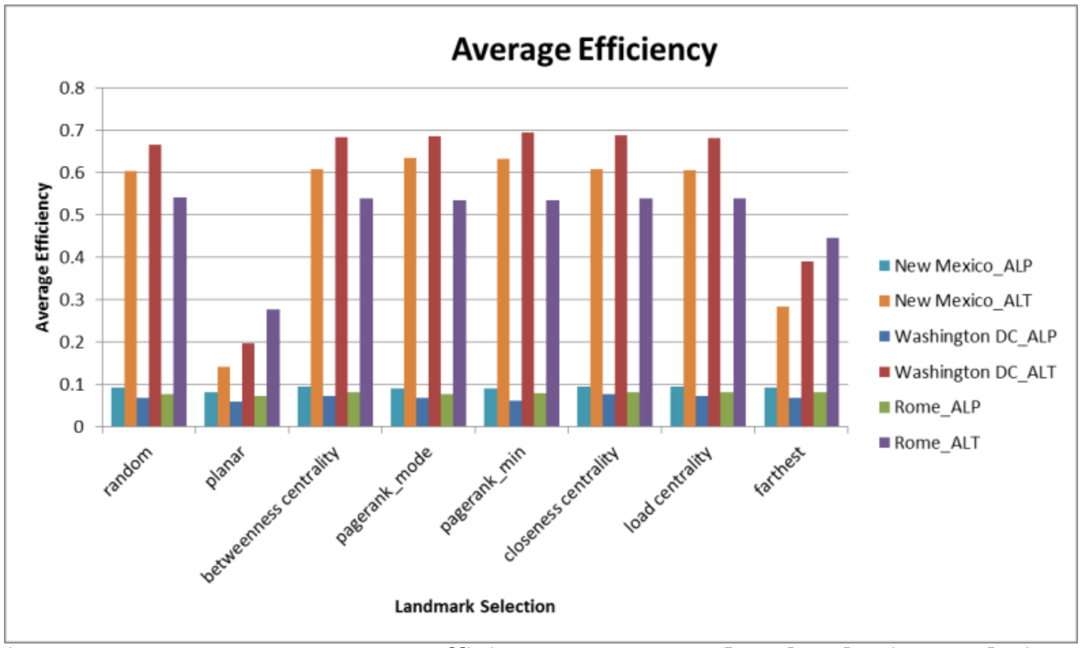


Figure 69 ALT vs ALP: Average Efficiency among Landmark Selection Techniques using the Same Number of Landmarks

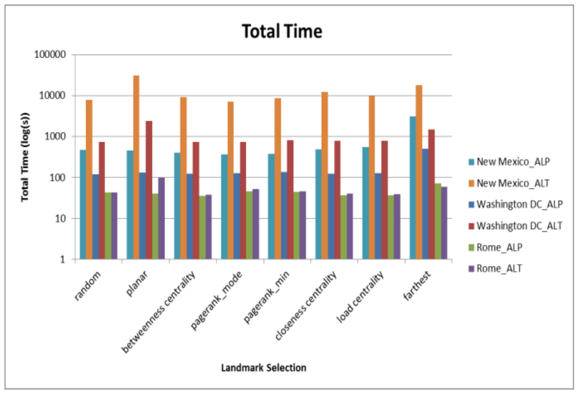


Figure 70 ALT vs. ALP: Total Times for Each Landmark selection Technique with the Same Number of Landmarks

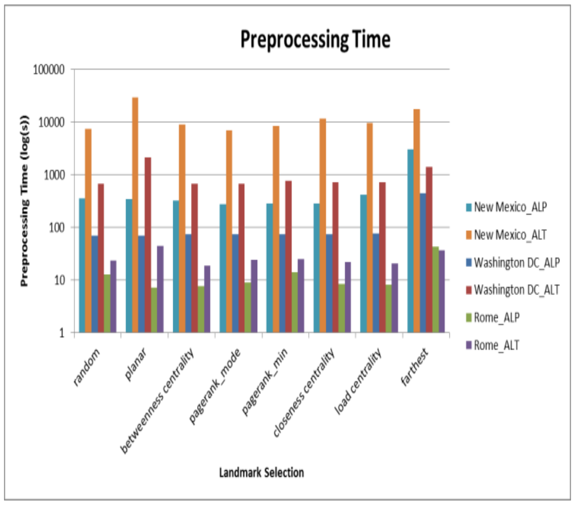


Figure 71 ALT vs. ALP: Preprocessing Times for Each Landmark selection Technique with the Same Number of Landmarks

Detailed Graph Performance Measurements

This section enumerates graph performance for each graph structure at the scales defined in Chapter 4. This section should be used to answer any further questions about the capabilities of ALP. More data concerning these runs can be found in the ALP dataset (available upon request). Efficiency is multiplied by 100 in these data. Tables spanning more than one page have a caption located at the beginning of the table.

Name	# Landmarks	Average Runtime	Average Search Space Size	Efficiency	# Nodes	# Edges	Density	Average Path Length	Average Clustering
NETWORKX.RANDOM_LOBSTER_45	10	0.000511224	19.072	64.44249	60	59	0.033333	11.485	0
NETWORKX.RANDOM_LOBSTER_45	8	0.000535808	20.375	63.40044	60	59	0.033333	11.578	0

Table 31 V1 Synthetic Graphs Performance and Structure

Table 32 V3 Synthetic Graph Structure

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering
NETWORKX.WAXMAN_GRAPH	10	1500	36473	0.03244	0	37204	0.0786406	0.082064328
NETWORKX.WAXMAN_GRAPH	5	2000	66454	0.03324	0	82805	0.0793045	0.08278744
NETWORKX.WAXMAN_GRAPH	5	4000	264030	0.03301	0	565746	0.0791687	0.082595149
NETWORKX.RANDOM_LOBSTER_90	212	1223	1222	0.00164	1	1222	0	0
NETWORKX.RANDOM_LOBSTER_90	95	1223	1222	0.00164	1	1222	0	0

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering
NETWORKX.RANDOM_LOBSTER_90	44	1223	1222	0.00164	1	1222	0	0
NETWORKX.RANDOM_LOBSTER_90	39	1223	1222	0.00164	1	1222	0	0
NETWORKX.RANDOM_LOBSTER_90	343	2088	2087	0.00096	1	2087	0	0
NETWORKX.RANDOM_LOBSTER_90	159	2088	2087	0.00096	1	2087	0	0
NETWORKX.RANDOM_LOBSTER_90	75	2088	2087	0.00096	1	2087	0	0
NETWORKX.RANDOM_LOBSTER_90	44	2088	2087	0.00096	1	2087	0	0
NETWORKX.RANDOM_LOBSTER_90	434	2613	2612	0.00077	1	2612	0	0
NETWORKX.RANDOM_LOBSTER_90	200	2613	2612	0.00077	1	2612	0	0
NETWORKX.RANDOM_LOBSTER_90	95	2613	2612	0.00077	1	2612	0	0
NETWORKX.RANDOM_LOBSTER_90	52	2613	2612	0.00077	1	2612	0	0
NETWORKX.RANDOM_LOBSTER_45	308	1528	1527	0.00131	1	1527	0	0
NETWORKX.RANDOM_LOBSTER_45	143	1528	1527	0.00131	1	1527	0	0
NETWORKX.RANDOM_LOBSTER_45	65	1528	1527	0.00131	1	1527	0	0
NETWORKX.RANDOM_LOBSTER_45	40	1528	1527	0.00131	1	1527	0	0
NETWORKX.PATH_GRAPH	40	1500	1499	0.00133	1	1499	0	0
NETWORKX.PATH_GRAPH	499	2000	1999	0.001	1	1999	0	0
NETWORKX.PATH_GRAPH	249	2000	1999	0.001	1	1999	0	0
NETWORKX.PATH_GRAPH	124	2000	1999	0.001	1	1999	0	0
NETWORKX.PATH_GRAPH	61	2000	1999	0.001	1	1999	0	0
NETWORKX.PATH_GRAPH	499	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	499	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	499	4000	3999	0.0005	1	3999	0	0

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering
NETWORKX.PATH_GRAPH	249	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	249	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	249	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	124	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	124	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	124	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	64	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	64	4000	3999	0.0005	1	3999	0	0
NETWORKX.PATH_GRAPH	64	4000	3999	0.0005	1	3999	0	0
NETWORKX.LADDER_GRAPH	499	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	499	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	499	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	499	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	499	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	249	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	249	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	249	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	249	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	249	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	249	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	124	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	124	2000	2998	0.0015	0	2998	0	0

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering
NETWORKX.LADDER_GRAPH	124	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	124	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	124	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	124	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	61	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	61	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	61	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	61	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	61	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	61	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	33	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	33	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	33	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	33	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	33	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	33	2000	2998	0.0015	0	2998	0	0
NETWORKX.LADDER_GRAPH	36	3000	4498	0.001	0	4498	0	0
NETWORKX.LADDER_GRAPH	499	4000	5998	0.00075	0	5998	0	0
NETWORKX.LADDER_GRAPH	249	4000	5998	0.00075	0	5998	0	0
NETWORKX.LADDER_GRAPH	124	4000	5998	0.00075	0	5998	0	0
NETWORKX.LADDER_GRAPH	62	4000	5998	0.00075	0	5998	0	0
NETWORKX.CYCLE_GRAPH	499	2000	2000	0.001	0	2000	0	0

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering
NETWORKX.CYCLE_GRAPH	249	2000	2000	0.001	0	2000	0	0
NETWORKX.CYCLE_GRAPH	124	2000	2000	0.001	0	2000	0	0
NETWORKX.CYCLE_GRAPH	62	2000	2000	0.001	0	2000	0	0
NETWORKX.CYCLE_GRAPH	499	4000	4000	0.0005	0	4000	0	0
NETWORKX.CYCLE_GRAPH	249	4000	4000	0.0005	0	4000	0	0
NETWORKX.CYCLE_GRAPH	124	4000	4000	0.0005	0	4000	0	0
NETWORKX.CYCLE_GRAPH	63	4000	4000	0.0005	0	4000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	249	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	249	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	124	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	124	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	62	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	62	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	32	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	32	2000	3000	0.0015	0	3000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	499	4000	6000	0.00075	0	6000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	249	4000	6000	0.00075	0	6000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	124	4000	6000	0.00075	0	6000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	124	4000	6000	0.00075	0	6000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	62	4000	6000	0.00075	0	6000	0	0
NETWORKX.CIRCULAR_LADDER_GRAPH	62	4000	6000	0.00075	0	6000	0	0
NETWORKX.BARBELL_GRAPH_ODD	12	1665	443224	0.31995	1	336	0.9999943	0.799996393

Name	# Landm arks	# Nodes	# Edges	Density	Ch ord al	# Max Cliques	Transitivity	Average Clustering
NETWORKX.BARBELL_GRAPH_ODD	7	1665	443224	0.31995	1	336	0.9999943	0.799996393
NETWORKX.BARBELL_GRAPH_ODD	4	1665	443224	0.31995	1	336	0.9999943	0.799996393
NETWORKX.BARBELL_GRAPH_ODD	3	1665	443224	0.31995	1	336	0.9999943	0.799996393
NETWORKX.BARBELL_GRAPH_ODD	7	3332	1776223	0.32007	1	669	0.9999986	0.800119147
NETWORKX.BARBELL_GRAPH_ODD	4	3332	1776223	0.32007	1	669	0.9999986	0.800119147
NETWORKX.BARBELL_GRAPH_ODD	3	3332	1776223	0.32007	1	669	0.9999986	0.800119147
NETWORKX.BARBELL_GRAPH_EVEN	17	1500	250001	0.22237	1	503	0.9999879	0.666661333
NETWORKX.BARBELL_GRAPH_EVEN	9	1500	250001	0.22237	1	503	0.9999879	0.666661333
NETWORKX.BARBELL_GRAPH_EVEN	5	1500	250001	0.22237	1	503	0.9999879	0.666661333
NETWORKX.BARBELL_GRAPH_EVEN	3	1500	250001	0.22237	1	503	0.9999879	0.666661333
NETWORKX.BARBELL_GRAPH_EVEN	17	3000	1000001	0.2223	1	1003	0.999997	0.666665333
NETWORKX.BARBELL_GRAPH_EVEN	9	3000	1000001	0.2223	1	1003	0.999997	0.666665333
NETWORKX.BARBELL_GRAPH_EVEN	5	3000	1000001	0.2223	1	1003	0.999997	0.666665333
NETWORKX.BARBELL_GRAPH_EVEN	3	3000	1000001	0.2223	1	1003	0.999997	0.666665333
NETWORKX.BARABÁSI_ALBERT_9	9	1500	13419	0.01194	0	11104	0.0366696	0.040707013
NETWORKX.BARABÁSI_ALBERT_9	12	2000	17919	0.00896	0	14960	0.0311779	0.037446104
NETWORKX.BARABÁSI_ALBERT_9	8	2000	17919	0.00896	0	14960	0.0311779	0.037446104
NETWORKX.BARABÁSI_ALBERT_9	15	4000	35919	0.00449	0	31263	0.0180052	0.02119164
NETWORKX.BARABÁSI_ALBERT_9	10	4000	35919	0.00449	0	31263	0.0180052	0.02119164
NETWORKX.BARABÁSI_ALBERT_7	8	1500	10451	0.0093	0	8714	0.0305568	0.039336318
NETWORKX.BARABÁSI_ALBERT_7	17	2000	13951	0.00698	0	11938	0.0232755	0.028426648
NETWORKX.BARABÁSI_ALBERT_7	11	2000	13951	0.00698	0	11938	0.0232755	0.028426648

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering
NETWORKX.BARABÁSI_ALBERT_7	10	2000	13951	0.00698	0	11938	0.0232755	0.028426648
NETWORKX.BARABÁSI_ALBERT_7	18	2500	17451	0.00559	0	15102	0.0199917	0.023478844
NETWORKX.BARABÁSI_ALBERT_7	10	2500	17451	0.00559	0	15102	0.0199917	0.023478844
NETWORKX.BARABÁSI_ALBERT_7	37	4000	27951	0.00349	0	24927	0.0144269	0.018219935
NETWORKX.BARABÁSI_ALBERT_7	13	4000	27951	0.00349	0	24927	0.0144269	0.018219935
NETWORKX.BARABÁSI_ALBERT_7	12	4000	27951	0.00349	0	24927	0.0144269	0.018219935
NETWORKX.BARABÁSI_ALBERT_6	11	1500	8964	0.00797	0	7752	0.0253886	0.029550621
NETWORKX.BARABÁSI_ALBERT_5	11	1500	7475	0.00665	0	6492	0.0228157	0.031865762
NETWORKX.BARABÁSI_ALBERT_5	46	2000	9975	0.00499	0	8904	0.0171176	0.022560463
NETWORKX.BARABÁSI_ALBERT_5	13	2000	9975	0.00499	0	8904	0.0171176	0.022560463
NETWORKX.BARABÁSI_ALBERT_5	11	2000	9975	0.00499	0	8904	0.0171176	0.022560463
NETWORKX.BARABÁSI_ALBERT_5	44	4000	19975	0.0025	0	18451	0.0106202	0.014365793
NETWORKX.BARABÁSI_ALBERT_5	14	4000	19975	0.0025	0	18451	0.0106202	0.014365793
NETWORKX.BARABÁSI_ALBERT_4	14	1500	5984	0.00532	0	5446	0.0168498	0.023264593
NETWORKX.BARABÁSI_ALBERT_3	82	2000	5991	0.003	0	5648	0.0091858	0.015717906
NETWORKX.BARABÁSI_ALBERT_3	76	2000	5991	0.003	0	5619	0.011163	0.018371522
NETWORKX.BARABÁSI_ALBERT_3	22	2000	5991	0.003	0	5619	0.011163	0.018371522
NETWORKX.BARABÁSI_ALBERT_3	19	2000	5991	0.003	0	5619	0.011163	0.018371522
NETWORKX.BARABÁSI_ALBERT_3	19	2000	5991	0.003	0	5648	0.0091858	0.015717906
NETWORKX.BARABÁSI_ALBERT_3	17	2000	5991	0.003	0	5648	0.0091858	0.015717906
NETWORKX.BARABÁSI_ALBERT_3	121	4000	11991	0.0015	0	11516	0.0059219	0.010684208
NETWORKX.BARABÁSI_ALBERT_3	26	4000	11991	0.0015	0	11516	0.0059219	0.010684208

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering
NETWORKX.BARABÁSI_ALBERT_3	20	4000	11991	0.0015	0	11516	0.0059219	0.010684208
NETWORKX.BARABÁSI_ALBERT_2	23	1500	2996	0.00266	0	2912	0.0053115	0.01241446
NETWORKX.BARABÁSI_ALBERT_2	18	2500	4996	0.0016	0	4880	0.0045243	0.009747668
NETWORKX.BARABÁSI_ALBERT_2	10	2500	4996	0.0016	0	4880	0.0045243	0.009747668
NETWORKX.BARABÁSI_ALBERT_13	11	2000	25831	0.01292	0	21857	0.0398079	0.043506926
NETWORKX.BARABÁSI_ALBERT_13	8	4000	51831	0.00648	0	44434	0.0237987	0.025350993
NETWORKX.BARABÁSI_ALBERT_11	8	2000	21879	0.01094	0	18138	0.0355809	0.038044008
NETWORKX.BARABÁSI_ALBERT_11	8	4000	43879	0.00549	0	37727	0.0211312	0.023212327

Table 33 V3 Synthetic Graph Performance

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.WAXMAN_GRAPH	10	0.055467024	746.94736	3.1912	1.160282	1500	36473
NETWORKX.WAXMAN_GRAPH	5	0.094738145	975.8488	3.2813	1.02974	2000	66454
NETWORKX.WAXMAN_GRAPH	5	0.269477554	2002.1622	3.1652	0.497598	4000	264030
NETWORKX.RANDOM_LOBSTER_90	212	0.011926244	367.232	156.495	43.8782	1223	1222
NETWORKX.RANDOM_LOBSTER_90	95	0.010520858	394.297	157.545	40.96436	1223	1222

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.RANDOM_LOBSTER_90	44	0.009802136	411.273	157.187	39.09726	1223	1222
NETWORKX.RANDOM_LOBSTER_90	39	0.010022001	424.471	162.286	38.50999	1223	1222
NETWORKX.RANDOM_LOBSTER_90	343	0.014707763	619.9289	255.5596	42.53912	2088	2087
NETWORKX.RANDOM_LOBSTER_90	159	0.019793989	679.998	258.079	38.48101	2088	2087
NETWORKX.RANDOM_LOBSTER_90	75	0.017555335	691.057	252.995	36.86849	2088	2087
NETWORKX.RANDOM_LOBSTER_90	44	0.017395397	713.825	257.887	36.60131	2088	2087
NETWORKX.RANDOM_LOBSTER_90	434	0.019393084	764.998	322.7928	43.54557	2613	2612
NETWORKX.RANDOM_LOBSTER_90	200	0.027931002	863.405	333.934	39.1759	2613	2612
NETWORKX.RANDOM_LOBSTER_90	95	0.023308151	838.624	314.181	37.52181	2613	2612
NETWORKX.RANDOM_LOBSTER_90	52	0.02357689	907.856	335.812	37.04381	2613	2612
NETWORKX.RANDOM_LOBSTER_45	308	0.017333188	467.88	305.821	65.74632	1528	1527
NETWORKX.RANDOM_LOBSTER_45	143	0.013820886	484.45	305.462	62.95169	1528	1527
NETWORKX.RANDOM_LOBSTER_45	65	0.01266592	504.59	308.111	61.01867	1528	1527
NETWORKX.RANDOM_LOBSTER_45	40	0.012804669	543.152	323.475	59.26136	1528	1527
NETWORKX.PATH_GRAPH	40	0.010638514	521.24844	506.2713	94.76616	1500	1499
NETWORKX.PATH_GRAPH	499	0.017287046	669.024	667.8699	99.39805	2000	1999
NETWORKX.PATH_GRAPH	249	0.025150551	669.485	666.913	98.908	2000	1999
NETWORKX.PATH_GRAPH	124	0.021305844	661.817	655.965	97.85148	2000	1999
NETWORKX.PATH_GRAPH	61	0.01987497	663.818	650.948	96.05947	2000	1999
NETWORKX.PATH_GRAPH	499	0.028369162	1336.3964	1333.667	99.17086	4000	3999
NETWORKX.PATH_GRAPH	499	0.029679637	1305.2162	1302.907	99.33023	4000	3999

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.PATH_GRAPH	499	0.033669614	1326.5696	1323.772	99.17364	4000	3999
NETWORKX.PATH_GRAPH	249	0.035472603	1364.033	1358.958	98.86861	4000	3999
NETWORKX.PATH_GRAPH	249	0.028614471	1351.7257	1345.975	98.77093	4000	3999
NETWORKX.PATH_GRAPH	249	0.029254625	1316.1942	1310.505	98.63464	4000	3999
NETWORKX.PATH_GRAPH	124	0.035543398	1311.792	1300.269	97.75051	4000	3999
NETWORKX.PATH_GRAPH	124	0.03947176	1290.081	1278.347	97.71102	4000	3999
NETWORKX.PATH_GRAPH	124	0.03533038	1361.564	1349.446	97.80879	4000	3999
NETWORKX.PATH_GRAPH	64	0.033654166	1380.431	1355.614	96.14231	4000	3999
NETWORKX.PATH_GRAPH	64	0.035708795	1383.649	1360.311	96.46316	4000	3999
NETWORKX.PATH_GRAPH	64	0.039881881	1396.295	1374.016	96.54972	4000	3999
NETWORKX.LADDER_GRAPH	499	0.015652372	668.5125	347.026	52.32163	2000	2998
NETWORKX.LADDER_GRAPH	499	0.015878414	668.1942	346.4234	52.40885	2000	2998
NETWORKX.LADDER_GRAPH	499	0.01594825	660.3724	342.5345	52.39372	2000	2998
NETWORKX.LADDER_GRAPH	499	0.015056354	643.7708	331.3944	52.05124	2000	2998
NETWORKX.LADDER_GRAPH	499	0.015866916	649.2713	336.7698	52.29019	2000	2998
NETWORKX.LADDER_GRAPH	249	0.016466344	677.8158	347.4094	51.36999	2000	2998
NETWORKX.LADDER_GRAPH	249	0.015905344	682.7668	349.3984	51.52546	2000	2998
NETWORKX.LADDER_GRAPH	249	0.015368899	629.2993	322.2212	51.43448	2000	2998
NETWORKX.LADDER_GRAPH	249	0.015351082	660.7247	338.5806	51.49247	2000	2998
NETWORKX.LADDER_GRAPH	249	0.023341279	675.34	346.425	51.32334	2000	2998
NETWORKX.LADDER_GRAPH	249	0.015472122	656.1041	336.4244	51.5769	2000	2998

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.LADDER_GRAPH	124	0.019176952	653.731	331.218	50.46436	2000	2998
NETWORKX.LADDER_GRAPH	124	0.015224438	656.8168	332.5235	50.57294	2000	2998
NETWORKX.LADDER_GRAPH	124	0.015373114	642.4394	327.4484	51.10467	2000	2998
NETWORKX.LADDER_GRAPH	124	0.015874915	683.4154	346.6436	50.63131	2000	2998
NETWORKX.LADDER_GRAPH	124	0.015763692	661.8488	335.5536	50.82275	2000	2998
NETWORKX.LADDER_GRAPH	124	0.014814325	642.2553	324.7367	50.39013	2000	2998
NETWORKX.LADDER_GRAPH	61	0.016403922	670.3964	339.8619	50.48532	2000	2998
NETWORKX.LADDER_GRAPH	61	0.015590305	672.7317	336.6567	49.63934	2000	2998
NETWORKX.LADDER_GRAPH	61	0.018256605	673.851	337.32	49.98553	2000	2998
NETWORKX.LADDER_GRAPH	61	0.015467196	663.2833	332.011	49.87475	2000	2998
NETWORKX.LADDER_GRAPH	61	0.016246931	687.8328	346.3994	50.1129	2000	2998
NETWORKX.LADDER_GRAPH	61	0.015091519	647.7447	325.4464	49.80288	2000	2998
NETWORKX.LADDER_GRAPH	33	0.015736654	663.7848	326.4334	48.83879	2000	2998
NETWORKX.LADDER_GRAPH	33	0.015297337	652.3133	325.3143	49.51195	2000	2998
NETWORKX.LADDER_GRAPH	33	0.015966044	669.5896	333.1752	49.15448	2000	2998
NETWORKX.LADDER_GRAPH	33	0.016450556	716.0691	350.0961	48.52898	2000	2998
NETWORKX.LADDER_GRAPH	33	0.018578129	675.866	336.553	49.36814	2000	2998
NETWORKX.LADDER_GRAPH	33	0.015419692	657.2593	327.1622	49.76974	2000	2998
NETWORKX.LADDER_GRAPH	36	0.045794711	1495.97016	10.5776	1.693635	3000	4498
NETWORKX.LADDER_GRAPH	499	0.036878761	1297.1632	663.9159	51.32118	4000	5998
NETWORKX.LADDER_GRAPH	249	0.03991301	1360.9139	691.3223	50.74776	4000	5998

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.LADDER_GRAPH	124	0.04846504	1359.492	688.955	50.42764	4000	5998
NETWORKX.LADDER_GRAPH	62	0.043796067	1317.419	661.973	49.60765	4000	5998
NETWORKX.CYCLE_GRAPH	499	0.012621767	507.8699	506.5896	99.30139	2000	2000
NETWORKX.CYCLE_GRAPH	249	0.021090964	509.506	503.078	98.5578	2000	2000
NETWORKX.CYCLE_GRAPH	124	0.015925891	494.518	486.108	97.6634	2000	2000
NETWORKX.CYCLE_GRAPH	62	0.01602904	528.599	509.512	95.82431	2000	2000
NETWORKX.CYCLE_GRAPH	499	0.022196118	1001.5726	998.7988	99.24962	4000	4000
NETWORKX.CYCLE_GRAPH	249	0.022201269	1003.2262	993.4655	98.44171	4000	4000
NETWORKX.CYCLE_GRAPH	124	0.029588307	1054.772	1039.059	97.74112	4000	4000
NETWORKX.CYCLE_GRAPH	63	0.028494363	1048.29	1020.792	96.06976	4000	4000
NETWORKX.CIRCULAR_LADDER_GRAPH	249	0.018850444	491.979	250.868	51.0116	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	249	0.0064497	221.25	113.75	52.7325	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	124	0.015534152	490.107	245.821	50.48463	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	124	0.013278003	505.2773	254.1221	50.64453	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	62	0.014286425	511.221	250.097	49.21524	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	62	0.012738555	504.5926	248.6366	49.79435	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	32	0.014873145	523.682	251.116	48.69981	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	32	0.016378737	518.1221	251.3544	48.78238	2000	3000
NETWORKX.CIRCULAR_LADDER_GRAPH	499	0.027456708	984.8619	504.5165	51.42	4000	6000
NETWORKX.CIRCULAR_LADDER_GRAPH	249	0.027394978	1001.0851	504.3844	50.51295	4000	6000
NETWORKX.CIRCULAR_LADDER_GRAPH	124	0.031709929	1017.4955	507.4324	50.14378	4000	6000

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.CIRCULAR_LADDER_GRAPH	124	0.036367474	975.865	488.003	50.17864	4000	6000
NETWORKX.CIRCULAR_LADDER_GRAPH	62	0.035299103	1021.284	500.165	49.06832	4000	6000
NETWORKX.CIRCULAR_LADDER_GRAPH	62	0.027559318	1013.1371	502.2913	49.28862	4000	6000
NETWORKX.BARBELL_GRAPH_ODD	12	0.267714257	741.6597	162.4895	21.01702	1665	443224
NETWORKX.BARBELL_GRAPH_ODD	7	0.256901449	769.1902	170.6597	20.77086	1665	443224
NETWORKX.BARBELL_GRAPH_ODD	4	0.252819649	762.8448	170.0631	21.12617	1665	443224
NETWORKX.BARBELL_GRAPH_ODD	3	0.256378367	768.3303	174.025	22.0402	1665	443224
NETWORKX.BARBELL_GRAPH_ODD	7	1.014840501	1533.1672	335.7177	20.49526	3332	1776223
NETWORKX.BARBELL_GRAPH_ODD	4	1.072889113	1474.2082	319.4264	20.52234	3332	1776223
NETWORKX.BARBELL_GRAPH_ODD	3	1.011720556	1514.2152	333.014	21.1022	3332	1776223
NETWORKX.BARBELL_GRAPH_EVEN	17	0.127165421	643.4855	241.7477	36.80949	1500	250001
NETWORKX.BARBELL_GRAPH_EVEN	9	0.120824437	620.4905	234.4164	36.08183	1500	250001
NETWORKX.BARBELL_GRAPH_EVEN	5	0.123599628	651.3333	252.2983	37.71891	1500	250001
NETWORKX.BARBELL_GRAPH_EVEN	3	0.122485971	636.7167	244.4484	37.8418	1500	250001
NETWORKX.BARBELL_GRAPH_EVEN	17	0.496028897	1293.2833	494.8488	37.32289	3000	1000001
NETWORKX.BARBELL_GRAPH_EVEN	9	0.494851285	1278.1832	491.038	36.82193	3000	1000001
NETWORKX.BARBELL_GRAPH_EVEN	5	0.494510013	1304.9019	479.3433	35.00984	3000	1000001
NETWORKX.BARBELL_GRAPH_EVEN	3	0.482507466	1276.0731	492.4555	38.83312	3000	1000001
NETWORKX.BARABÁSI_ALBERT_9	9	0.038634122	754.8238	3.7077	1.431025	1500	13419
NETWORKX.BARABÁSI_ALBERT_9	12	0.06077012	964.1592	3.7487	0.943123	2000	17919
NETWORKX.BARABÁSI_ALBERT_9	8	0.060346278	979.985	3.7728	1.108929	2000	17919

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.BARABÁSI_ALBERT_9	15	0.106309873	2012.8979	3.9179	0.697768	4000	35919
NETWORKX.BARABÁSI_ALBERT_9	10	0.105891223	1878.3243	3.8869	0.62009	4000	35919
NETWORKX.BARABÁSI_ALBERT_7	8	0.035828095	751.21442	3.8348	1.545141	1500	10451
NETWORKX.BARABÁSI_ALBERT_7	17	0.056699913	995.979	3.9049	1.198619	2000	13951
NETWORKX.BARABÁSI_ALBERT_7	11	0.056671879	1008.1992	3.9189	1.085385	2000	13951
NETWORKX.BARABÁSI_ALBERT_7	10	0.055639405	1015.1582	3.9399	1.134895	2000	13951
NETWORKX.BARABÁSI_ALBERT_7	18	0.054930805	1284.8028	4.002	1.133964	2500	17451
NETWORKX.BARABÁSI_ALBERT_7	10	0.054646077	1229.2412	4	0.817167	2500	17451
NETWORKX.BARABÁSI_ALBERT_7	37	0.099659625	1935.2633	4.0791	0.614585	4000	27951
NETWORKX.BARABÁSI_ALBERT_7	13	0.097212028	1904.7427	4.0811	0.698729	4000	27951
NETWORKX.BARABÁSI_ALBERT_7	12	0.100160735	1988.2442	4.1301	0.843133	4000	27951
NETWORKX.BARABÁSI_ALBERT_6	11	0.034587752	770.37658	3.984	1.479355	1500	8964
NETWORKX.BARABÁSI_ALBERT_5	11	0.032678317	752.50552	4.0911	1.561642	1500	7475
NETWORKX.BARABÁSI_ALBERT_5	46	0.053001714	1015.1071	4.1872	1.198969	2000	9975
NETWORKX.BARABÁSI_ALBERT_5	13	0.053475886	1048.4885	4.2533	1.042122	2000	9975
NETWORKX.BARABÁSI_ALBERT_5	11	0.053401707	1035.8649	4.2212	1.123814	2000	9975
NETWORKX.BARABÁSI_ALBERT_5	44	0.09360658	1999.3093	4.4084	0.477497	4000	19975
NETWORKX.BARABÁSI_ALBERT_5	14	0.090803667	1918.5666	4.4034	0.618068	4000	19975
NETWORKX.BARABÁSI_ALBERT_4	14	0.030932879	782.5971	4.3704	1.390721	1500	5984
NETWORKX.BARABÁSI_ALBERT_3	82	0.043339349	958.0591	4.7918	1.348218	2000	5991
NETWORKX.BARABÁSI_ALBERT_3	76	0.05455953	1026.203	4.792	1.29832	2000	5991

Name	# Landmarks	Avg Runtime	Avg Search Space	Average Path Length	Efficiency	# Nodes	# Edges
NETWORKX.BARABÁSI_ALBERT_3	22	0.052871013	1079.852	4.771	1.16898	2000	5991
NETWORKX.BARABÁSI_ALBERT_3	19	0.051465703	1024.67	4.723	1.27499	2000	5991
NETWORKX.BARABÁSI_ALBERT_3	19	0.058855596	1023.326	4.773	1.26934	2000	5991
NETWORKX.BARABÁSI_ALBERT_3	17	0.050921925	948.402	4.801	1.36665	2000	5991
NETWORKX.BARABÁSI_ALBERT_3	121	0.077737156	1927.7067	5.009	0.632593	4000	11991
NETWORKX.BARABÁSI_ALBERT_3	26	0.077238052	2025.7207	5.021	0.834655	4000	11991
NETWORKX.BARABÁSI_ALBERT_3	20	0.075645678	1941.0761	5.002	0.606286	4000	11991
NETWORKX.BARABÁSI_ALBERT_2	23	0.026608852	802.44324	5.4685	1.80191	1500	2996
NETWORKX.BARABÁSI_ALBERT_2	18	0.040212441	1155.7618	5.4184	1.522372	2500	4996
NETWORKX.BARABÁSI_ALBERT_2	10	0.041262981	1220.4605	5.5305	1.084384	2500	4996
NETWORKX.BARABÁSI_ALBERT_13	11	0.067197296	986.6757	3.5866	1.063153	2000	25831
NETWORKX.BARABÁSI_ALBERT_13	8	0.122725864	1941.4254	3.7508	0.550881	4000	51831
NETWORKX.BARABÁSI_ALBERT_11	8	0.067848633	1011.4715	3.7017	0.779419	2000	21879
NETWORKX.BARABÁSI_ALBERT_11	8	0.117220611	1862.7187	3.7978	0.687658	4000	43879

Name	# Landmarks	Average Runtime	Average Search Space Size	Efficiency	# Nodes	# Edges
Great Lakes	267	0.015718917	498.3053	21.93133	3700	4483
Great Lakes	97	0.022094637	753.3323	14.17775	3700	4483
Great Lakes	86	0.021475039	724.6547	14.91763	3700	4483
Rome	299	0.041006723	1211.8981	3.321693	3353	4831
Rome	262	0.044333826	1310.9817	2.689657	3353	4831
Rome	183	0.023491724	856.8128	10.23698	3353	4831
Rome	58	0.031587066	1222.4304	6.980097	3353	4831
Rome	48	0.031979783	1255.113367	6.564847	3353	4831

Table 34 V3 Real Graph Performance

Name	# Landmarks	# Nodes	# Edges	Directed	Density	Chordal	Largest Clique Size	# Max Cliques	Transitivity	Average Clustering	Average Path Length
Great Lakes	267	3700	4483	1	0.000655108	0	3	4375	0.021273901	0.014108108	76.9249
Great Lakes	97	3700	4483	1	0.000655108	0	3	4375	0.021273901	0.014108108	74.0591
Great Lakes	86	3700	4483	1	0.000655108	0	3	4375	0.021273901	0.014108108	74.0861
Rome	299	3353	4831	0	0.000859665	0	3	4571	0.037358491	0.030271399	12.2581
Rome	262	3353	4831	0	0.000859665	0	3	4571	0.037358491	0.030271399	12.2552
Rome	183	3353	4831	0	0.000859665	0	3	4571	0.037358491	0.030271399	40.6727
Rome	58	3353	4831	0	0.000859665	0	3	4571	0.037358491	0.030271399	38.9046
Rome	48	3353	4831	0	0.000859665	0	3	4571	0.037358491	0.030271399	40.4484

Table 35 V3 Real Graph Structure

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	Largest Clique Size	# Max Clic	Transitivity	Average Clustering	Average Path Length
NETWORKX.PATH_GRAPH	2499	10000	9999	0.0002	1	2	9999	0	0	3329.7628
NETWORKX.PATH_GRAPH	1249	10000	9999	0.0002	1	2	9999	0	0	3399.5495
NETWORKX.PATH_GRAPH	624	10000	9999	0.0002	1	2	9999	0	0	3264.0851
NETWORKX.PATH_GRAPH	311	10000	9999	0.0002	1	2	9999	0	0	3414.4985
NETWORKX.PATH_GRAPH	155	10000	9999	0.0002	1	2	9999	0	0	3243.026
NETWORKX.PATH_GRAPH	81	10000	9999	0.0002	1	2	9999	0	0	3332.7828
NETWORKX.LADDER_GRAPH	499	8000	11998	0.000375	0	2	11998	0	0	1335.1171
NETWORKX.LADDER_GRAPH	249	8000	11998	0.000375	0	2	11998	0	0	1291.3674
NETWORKX.LADDER_GRAPH	124	8000	11998	0.000375	0	2	11998	0	0	1268.1141
NETWORKX.LADDER_GRAPH	64	8000	11998	0.000375	0	2	11998	0	0	1385.6597
NETWORKX.LADDER_GRAPH	155	20000	29998	0.00015	0	2	29998	0	0	3312.9271
NETWORKX.CYCLE_GRAPH	79	10000	10000	0.0002	0	2	10000	0	0	2558.7227
NETWORKX.CIRCULAR_LADDER_GRAPH	499	8000	12000	0.000375	0	2	12000	0	0	1004.1231
NETWORKX.CIRCULAR_LADDER_GRAPH	249	8000	12000	0.000375	0	2	12000	0	0	984.2793
NETWORKX.CIRCULAR_LADDER_GRAPH	124	8000	12000	0.000375	0	2	12000	0	0	1003.6917
NETWORKX.CIRCULAR_LADDER_GRAPH	63	8000	12000	0.000375	0	2	12000	0	0	977.6466
NETWORKX.CIRCULAR_LADDER_GRAPH	156	20000	30000	0.00015	0	2	30000	0	0	2521.1341
NETWORKX.BARABASI_ALBERT_9	13	10000	89919	0.001799	0	7	82326	0.008592076	0.010003391	4.1762
NETWORKX.BARABASI_ALBERT_9	9	10000	89919	0.001799	0	7	82057	0.008860695	0.010512408	4.1301
NETWORKX.BARABASI_ALBERT_6	1249	10000	59964	0.001199	0	5	56615	0.005916068	0.007714927	4.4755
NETWORKX.BARABASI_ALBERT_6	624	10000	59964	0.001199	0	5	56615	0.005916068	0.007714927	4.4825
NETWORKX.BARABASI_ALBERT_6	311	10000	59964	0.001199	0	5	56615	0.005916068	0.007714927	4.4685
NETWORKX.BARABASI_ALBERT_6	155	10000	59964	0.001199	0	5	56615	0.005916068	0.007714927	4.4885
NETWORKX.BARABASI_ALBERT_6	81	10000	59964	0.001199	0	5	56615	0.005916068	0.007714927	4.4775
NETWORKX.BARABASI_ALBERT_6	15	10000	59964	0.001199	0	6	56468	0.005944305	0.007894655	4.4675
NETWORKX.BARABASI_ALBERT_6	15	10000	59964	0.001199	0	6	56613	0.005733517	0.00763192	4.5095
NETWORKX.BARABASI_ALBERT_5	20	10000	49975	0.001	0	5	47683	0.00497716	0.007111235	4.6486

Table 36 V4 Synthetic Graph Structure

Name	# Landmarks	Average Runtime	Average Search Space Size	Efficiency	# Nodes	# Edges
NETWORKX.PATH_GRAPH	2499	0.068842525	3331.04	99.881982	10000	9999
NETWORKX.PATH_GRAPH	1249	0.068173933	3402.3964	99.62037	10000	9999
NETWORKX.PATH_GRAPH	624	0.065755312	3269.7898	99.571201	10000	9999
NETWORKX.PATH_GRAPH	311	0.068494629	3426.044	98.857267	10000	9999
NETWORKX.PATH_GRAPH	155	0.065067191	3269.1261	98.071071	10000	9999
NETWORKX.PATH_GRAPH	81	0.067794454	3384.0861	96.816727	10000	9999
NETWORKX.LADDER_GRAPH	499	0.067978507	2629.1602	50.795115	8000	11998
NETWORKX.LADDER_GRAPH	249	0.067041001	2543.3233	50.731401	8000	11998
NETWORKX.LADDER_GRAPH	124	0.064860955	2519.9149	50.208258	8000	11998
NETWORKX.LADDER_GRAPH	64	0.071977737	2782.0851	49.495866	8000	11998
NETWORKX.LADDER_GRAPH	155	0.367831638	6582.944	49.898784	20000	29998
NETWORKX.CYCLE_GRAPH	79	0.054701921	2649.7297	95.639289	10000	10000
NETWORKX.CIRCULAR_LADDER_GRAPH	499	0.049047699	1970.8589	51.045996	8000	12000
NETWORKX.CIRCULAR_LADDER_GRAPH	249	0.048405474	1955.1672	50.311622	8000	12000
NETWORKX.CIRCULAR_LADDER_GRAPH	124	0.049360576	1999.5556	49.863143	8000	12000
NETWORKX.CIRCULAR_LADDER_GRAPH	63	0.048025332	1992.5566	49.283143	8000	12000
NETWORKX.CIRCULAR_LADDER_GRAPH	156	0.124017879	5041.006	49.871892	20000	30000
NETWORKX.BARABASI_ALBERT_9	13	0.272763037	5277.8098	0.194354	10000	89919
NETWORKX.BARABASI_ALBERT_9	9	0.288305324	4895.957	0.242883	10000	89919
NETWORKX.BARABASI_ALBERT_6	1249	0.24778024	4545.4855	0.368408	10000	59964
NETWORKX.BARABASI_ALBERT_6	624	0.254611392	4790.1572	0.297167	10000	59964
NETWORKX.BARABASI_ALBERT_6	311	0.251446031	4811.1341	0.328208	10000	59964
NETWORKX.BARABASI_ALBERT_6	155	0.241420774	4977.6687	0.242092	10000	59964
NETWORKX.BARABASI_ALBERT_6	81	0.249340214	4918.8298	0.266587	10000	59964
NETWORKX.BARABASI_ALBERT_6	15	0.238799786	5082.4484	0.421201	10000	59964
NETWORKX.BARABASI_ALBERT_6	15	0.313661984	5216.018	0.306567	10000	59964
NETWORKX.BARABASI_ALBERT_5	20	0.317060164	5105.044	0.222853	10000	49975

Table 37 V4 Synthetic Graph Performance

Name	# Landmar	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivit	Average Clustering	Average Path Length
United States (Western)	639	8294	9851	0.000286	0	9225	0.05882	0.035905474	116.9329
United States (Western)	197	8294	9851	0.000286	0	9225	0.05882	0.035905474	121.7097
United States (Western)	156	8294	9851	0.000286	0	9225	0.05882	0.035905474	115
United States (Western)	1069	13499	17421	0.000191	0	17140	0.013094	0.010790923	137.2022
United States (Western)	256	13499	17421	0.000191	0	17140	0.013094	0.010790923	137.8188
United States (Western)	127	13499	17421	0.000191	0	17140	0.013094	0.010790923	141.5676
United States (Western)	123	13499	17421	0.000191	0	17140	0.013094	0.010790923	143.1491
Great Lakes	867	11773	15861	0.000229	0	15546	0.014845	0.012531499	139.6707
Great Lakes	220	11773	15861	0.000229	0	15546	0.014845	0.012531499	141.3303
Great Lakes	121	11773	15861	0.000229	0	15546	0.014845	0.012531499	143.3413
Great Lakes	120	11773	15861	0.000229	0	15546	0.014845	0.012531499	136.1922
United States (Eastern)	410	5573	6391	0.000412	0	6199	0.02804	0.017040493	89.4675
United States (Eastern)	136	5573	6391	0.000412	0	6199	0.02804	0.017040493	94.8799
United States (Eastern)	110	5573	6391	0.000412	0	6199	0.02804	0.017040493	89.9269
United States (Central)	588	7276	7856	0.000297	0	7709	0.019395	0.01019333	177.2352
United States (Central)	213	7276	7856	0.000297	0	7709	0.019395	0.01019333	181.993
United States (Central)	191	7276	7856	0.000297	0	7709	0.019395	0.01019333	175.9419
United States (Central)	413	5327	6121	0.000431	0	5803	0.048901	0.030573806	102.3323
United States (Central)	140	5327	6121	0.000431	0	5803	0.048901	0.030573806	104.6386
United States (Central)	119	5327	6121	0.000431	0	5803	0.048901	0.030573806	103.8028
New Mexico	1140	15221	17919	0.000155	0	16656	0.058933	0.0360445	222.6256
New Mexico	335	15221	17919	0.000155	0	16656	0.058933	0.0360445	217.4525
New Mexico	216	15221	17919	0.000155	0	16656	0.058933	0.0360445	215.4695
New Mexico	213	15221	17919	0.000155	0	16656	0.058933	0.0360445	218.9209
Hawaii	676	9237	10711	0.000251	0	10233	0.038371	0.023730648	194.0501
Hawaii	216	9237	10711	0.000251	0	10233	0.038371	0.023730648	194.3293
Hawaii	159	9237	10711	0.000251	0	10233	0.038371	0.023730648	193.3554
Washington DC	626	9522	14832	0.000327	0	13720	0.046936	0.039189946	73.4364
Washington DC	582	9522	14832	0.000327	0	13720	0.046936	0.039189946	12.5976
Washington DC	508	9522	14832	0.000327	0	13720	0.046936	0.039189946	12.6044
Washington DC	136	9522	14832	0.000327	0	13720	0.046936	0.039189946	74.2412
Washington DC	71	9522	14832	0.000327	0	13720	0.046936	0.039189946	74.3223

Table 38 V4 Real Graph Structure

Name	# Landmar	Average Runtime	Average Search Space Size	Efficiency	# Nodes	# Edges
United States (Western)	639	0.022174926	761.968	23.69344	8294	9851
United States (Western)	197	0.041423797	1299.01	13.51363	8294	9851
United States (Western)	156	0.038698718	1415.6724	10.47789	8294	9851
United States (Western)	1069	0.04439088	1343.0891	14.98197	13499	17421
United States (Western)	256	0.076482814	2544.6877	7.576346	13499	17421
United States (Western)	127	0.089534322	3055.4615	6.407888	13499	17421
United States (Western)	123	0.093477266	3209.8949	6.215656	13499	17421
Great Lakes	867	0.034411663	1084.1742	16.99392	11773	15861
Great Lakes	220	0.048285952	1662.1431	10.96033	11773	15861
Great Lakes	121	0.060118319	2109.6697	9.131241	11773	15861
Great Lakes	120	0.064795337	2296.5656	8.306837	11773	15861
United States (Eastern)	410	0.016992073	528.4194	23.47322	5573	6391
United States (Eastern)	136	0.02764795	910.5085	14.65439	5573	6391
United States (Eastern)	110	0.028131704	933.1602	12.93048	5573	6391
United States (Central)	588	0.023168264	785.045	30.52616	7276	7856
United States (Central)	213	0.044543578	1337.4174	17.51926	7276	7856
United States (Central)	191	0.036169253	1299.7928	17.22821	7276	7856
United States (Central)	413	0.019570923	604.3744	24.7907	5327	6121
United States (Central)	140	0.027698786	887.4935	17.41308	5327	6121
United States (Central)	119	0.032490168	1053.4885	14.10981	5327	6121
New Mexico	1140	0.044394453	1408.5806	21.49428	15221	17919
New Mexico	335	0.069501981	2330.5866	12.24492	15221	17919
New Mexico	216	0.069054206	2329.5986	11.58909	15221	17919
New Mexico	213	0.0747001	2537.2292	11.26064	15221	17919
Hawaii	676	0.041771099	1365.3969	19.60609	9237	10711
Hawaii	216	0.05561454	1971.8774	13.05845	9237	10711
Hawaii	159	0.060690221	2130.475	12.08379	9237	10711
Washington DC	626	0.03679279	1310.8859	9.577548	9522	14832
Washington DC	582	0.134296621	3655.3239	1.291211	9522	14832
Washington DC	508	0.147273851	3915.2281	1.134729	9522	14832
Washington DC	136	0.058449629	2293.5225	5.45973	9522	14832
Washington DC	71	0.064369123	2526.8468	4.663093	9522	14832

Table 39 V4 Real Graph Performance

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering	Average Path Length
NETWORKX.PATH_GRAPH	197	50000	49999	0.00004	1	49999	0	0	17150.3774
NETWORKX.PATH_GRAPH	197	50000	49999	0.00004	1	49999	0	0	16958.1474
NETWORKX.CYCLE_GRAPH	196	50000	50000	4E-05	0	50000	0	0	12240.1992
NETWORKX.CYCLE_GRAPH	196	50000	50000	4E-05	0	50000	0	0	12621.5576
NETWORKX.CIRCULAR_LADDER_GRAPH	198	100000	150000	3E-05	0	150000	0	0	12262.683
NETWORKX.BARABASI_ALBERT_5	19	50000	249975	0.0002	0	245716	0.00138186	0.00194037	5.0931
NETWORKX.BARABASI_ALBERT_4	24	50000	199984	0.00016	0	197598	0.000995137	0.001763283	5.4234
NETWORKX.BARABASI_ALBERT_2	68	50000	99996	8E-05	0	99702	0.000338713	0.001146708	6.7027
NETWORKX.BARABASI_ALBERT_2	61	50000	99996	8E-05	0	99657	0.000373794	0.001395318	6.6346

Table 40 V5 Synthetic Graph Structure

Name	# Landmarks	Average Runtime	Average Search Space Size	Efficiency	# Nodes	# Edges
NETWORKX.PATH_GRAPH	197	1.162572475	17274.7538	98.027918	50000	49999
NETWORKX.PATH_GRAPH	197	1.338624261	17073.5527	98.317513	50000	49999
NETWORKX.CYCLE_GRAPH	196	0.811346265	12383.5686	97.868078	50000	50000
NETWORKX.CYCLE_GRAPH	196	0.829821832	12766.96	97.838298	50000	50000
NETWORKX.CIRCULAR_LADDER_GRAPH	198	1.803097851	24450.5978	50.114674	100000	150000
NETWORKX.BARABASI_ALBERT_5	19	1.532056167	26169.5085	0.057668	50000	249975
NETWORKX.BARABASI_ALBERT_4	24	1.471386127	27511.977	0.093213	50000	199984
NETWORKX.BARABASI_ALBERT_2	68	1.717726896	27918.3924	0.084815	50000	99996
NETWORKX.BARABASI_ALBERT_2	61	1.673248201	28048.5936	0.084815	50000	99996

Table 41 V5 Synthetic Graph Performance

Table 42 V5 Real Graph Structure

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering	Average Path Length
United States (Western)	2165	28652	36906	8.99E-05	0	36486	0.009180182	0.00792848	128.3984
United States (Western)	460	28652	36906	8.99E-05	0	36486	0.009180182	0.00792848	131.979
United States (Western)	161	28652	36906	8.99E-05	0	36486	0.009180182	0.00792848	135.7367
United States (Western)	145	28652	36906	8.99E-05	0	36486	0.009180182	0.00792848	132.5916
United States (Western)	936	51447	62272	4.71E-05	0	57378	0.069277523	0.04312918	364.011
United States (Western)	398	51447	62272	4.71E-05	0	57378	0.069277523	0.04312918	364.4875
United States (Western)	384	51447	62272	4.71E-05	0	57378	0.069277523	0.04312918	358.7568
Great Lakes	2384	34198	42957	7.35E-05	0	42033	0.017760608	0.01452911	133.0701
Great Lakes	540	34198	42957	7.35E-05	0	42033	0.017760608	0.01452911	136.2492
Great Lakes	204	34198	42957	7.35E-05	0	42033	0.017760608	0.01452911	138.2272
Great Lakes	193	34198	42957	7.35E-05	0	42033	0.017760608	0.01452911	138.2212
United States (Eastern)	390	29796	32528	7.33E-05	0	31873	0.020404445	0.01158545	373.7355
United States (Eastern)	799	49404	57960	4.75E-05	0	56146	0.027095911	0.01774485	157.4114
United States (Eastern)	302	49404	57960	4.75E-05	0	56146	0.027095911	0.01774485	157.4935
United States (Eastern)	277	49404	57960	4.75E-05	0	56146	0.027095911	0.01774485	158.0531
United States (Eastern)	613	35103	42902	6.96E-05	0	41241	0.03231088	0.02300373	154.989
United States (Eastern)	229	35103	42902	6.96E-05	0	41241	0.03231088	0.02300373	156.0961
United States (Eastern)	210	35103	42902	6.96E-05	0	41241	0.03231088	0.02300373	151.0806
Rhode Island	917	53288	68496	4.82E-05	0	65847	0.028935623	0.02228457	207.1892
Rhode Island	306	53288	68496	4.82E-05	0	65847	0.028935623	0.02228457	206.3524
Rhode Island	255	53288	68496	4.82E-05	0	65847	0.028935623	0.02228457	203.025

Name	# Landmarks	# Nodes	# Edges	Density	Chordal	# Max Cliques	Transitivity	Average Clustering	Average Path Length
Rhode Island	254	53288	68496	4.82E-05	0	65847	0.028935623	0.02228457	204.0781
New Mexico	2246	29381	33476	7.76E-05	0	32041	0.038542474	0.02285491	235.4324
New Mexico	599	29381	33476	7.76E-05	0	32041	0.038542474	0.02285491	238.044
New Mexico	350	29381	33476	7.76E-05	0	32041	0.038542474	0.02285491	245.3554
New Mexico	343	29381	33476	7.76E-05	0	32041	0.038542474	0.02285491	234.0631
New Mexico	2161	28115	32736	8.28E-05	0	30549	0.059531971	0.03542119	249.2362
New Mexico	596	28115	32736	8.28E-05	0	30549	0.059531971	0.03542119	253.1321
New Mexico	317	28115	32736	8.28E-05	0	30549	0.059531971	0.03542119	253.9129
New Mexico	315	28115	32736	8.28E-05	0	30549	0.059531971	0.03542119	253.3904
Luxembourg	1063	84136	85579	2.42E-05	0	85361	0.003364786	0.0016822	378.0985
Luxembourg	392	84136	85579	2.42E-05	0	85361	0.003364786	0.0016822	377.5447
Luxembourg	386	84136	85579	2.42E-05	0	85361	0.003364786	0.0016822	378.4762
Luxembourg	249	84136	85579	2.42E-05	0	85361	0.003364786	0.0016822	381.6023
Luxembourg	247	84136	85579	2.42E-05	0	85361	0.003364786	0.0016822	383.8765

Table 43 V5 Real Graph Performance

Name	# Landmarks	Average Runtime	Average Search Space Size	Efficiency (%)	# Nodes	# Edges
United States (Western)	2165	0.06864359	1645.6857	12.85869	28652	36906
United States (Western)	460	0.112180543	3738.6647	5.618649	28652	36906

United States (Western)	161	0.147076599	5029.4294	3.860621	28652	36906
United States (Western)	145	0.14743869	5077.1491	3.893233	28652	36906
United States (Western)	936	0.165575983	5339.035	9.319389	51447	62272
United States (Western)	398	0.226476887	7611.0821	6.355375	51447	62272
United States (Western)	384	0.231567124	7720.3824	5.940571	51447	62272
Great Lakes	2384	0.074080025	1260.3824	17.4718	34198	42957
Great Lakes	540	0.100425051	3146.6086	7.092262	34198	42957
Great Lakes	204	0.166356931	5366.6817	4.283223	34198	42957
Great Lakes	193	0.159183911	5140.0981	4.471912	34198	42957
United States (Eastern)	390	0.120883669	4037.9265	12.78429	29796	32528
United States (Eastern)	799	0.135431556	4154.988	7.308468	49404	57960
United States (Eastern)	302	0.227272182	7173.5195	3.708939	49404	57960
United States (Eastern)	277	0.236171585	7458.1101	3.465065	49404	57960
United States (Eastern)	613	0.15333767	4256.5075	6.135425	35103	42902
United States (Eastern)	229	0.191003423	6563.6647	3.871391	35103	42902
United States (Eastern)	210	0.203058066	7046.22185	3.544912	35103	42902
Rhode Island	917	0.184251335	6427.5295	5.544675	53288	68496
Rhode Island	306	0.255614322	9257.7037	3.86979	53288	68496
Rhode Island	255	0.276409393	10001.1982	3.35997	53288	68496
Rhode Island	254	0.265858525	9658.993	3.46967	53288	68496
New Mexico	2246	0.073057754	2370.4755	18.43823	29381	33476
New Mexico	599	0.105529631	3578.7257	10.62985	29381	33476
New Mexico	350	0.137659912	4753.6026	8.000731	29381	33476
New Mexico	343	0.132616894	4527.7928	8.055736	29381	33476

New Mexico	2161	0.046677033	1440.6126	23.08611	28115	32736
New Mexico	596	0.079765107	2614.7087	12.85735	28115	32736
New Mexico	317	0.108490007	3608.3123	9.558649	28115	32736
New Mexico	315	0.102721638	3384.8158	10.07619	28115	32736
Luxembourg	1063	0.095687627	3425.1552	20.22771	84136	85579
Luxembourg	392	0.271665576	7639.6507	9.820803	84136	85579
Luxembourg	386	0.123748903	8573.2232	8.716143	84136	85579
Luxembourg	249	0.157580806	10999.7651	6.895078	84136	85579
Luxembourg	247	0.292890311	10626.5334	7.054766	84136	85579

Name	**# Landmai**	**# Nodes**	**# Edges**	**Directed**	**Density**	**Chordal**	**# Max Cliques**	**Transitivity**	**Average Clustering**	**Average Path Length**
New York City	280	264346	365050	0	1.04481E-05	0	352355	0.025446321	0.020779882	284.6637
New York City	233	264346	365050	0	1.04481E-05	0	352355	0.025446321	0.020779882	267.8894

Table 44 V7 Real Graph Structure

Name	**# Landmai**	**Average Runtime**	**Average Search Space Size**	**Efficiency**	**# Nodes**	**# Edges**
New York City	280	1.099790801	37105.43138	1.1585642	264346	365050
New York City	233	1.31172116	40166.16357	1.059572545	264346	365050

Table 45 V7 Real Graph Performance

ALT-Based Landmark Selection

Table 46 ALT-Based Landmark Selection over Synthetic Graphs

Name	**Efficiency**	**Selection**
NETWORKX.BARABÁSI_ALBERT_11	0.07332625	random
NETWORKX.BARABÁSI_ALBERT_11	0.06418928	farthest-d
NETWORKX.BARABÁSI_ALBERT_11	0.07571031	planar
NETWORKX.BARABÁSI_ALBERT_11	0.07587722	betweenness centrality
NETWORKX.BARABÁSI_ALBERT_11	0.16729419	*farthest-ecc*
NETWORKX.BARABÁSI_ALBERT_13	0.30886323	random
NETWORKX.BARABÁSI_ALBERT_13	0.3233789	farthest-d
NETWORKX.BARABÁSI_ALBERT_13	0.30883986	planar
NETWORKX.BARABÁSI_ALBERT_13	0.32916084	betweenness centrality
NETWORKX.BARABÁSI_ALBERT_3	0.10972896	random
NETWORKX.BARABÁSI_ALBERT_3	0.10473636	farthest-d
NETWORKX.BARABÁSI_ALBERT_3	0.10926481	planar
NETWORKX.BARABÁSI_ALBERT_3	0.10969018	betweenness centrality
NETWORKX.BARABÁSI_ALBERT_5	0.09167031	random
NETWORKX.BARABÁSI_ALBERT_5	0.0761458	farthest-d
NETWORKX.BARABÁSI_ALBERT_5	0.08736555	planar
NETWORKX.BARABÁSI_ALBERT_5	0.08885303	betweenness centrality

NETWORKX.BARABÁSI_ALBERT_5	0.14827956	*farthest-ecc*
NETWORKX.BARABÁSI_ALBERT_7	0.06162094	random
NETWORKX.BARABÁSI_ALBERT_7	0.05603413	farthest-d
NETWORKX.BARABÁSI_ALBERT_7	0.06396991	planar
NETWORKX.BARABÁSI_ALBERT_7	0.06365355	betweenness centrality
NETWORKX.BARABÁSI_ALBERT_7	0.17620701	*farthest-ecc*
NETWORKX.BARABÁSI_ALBERT_9	0.07793635	random
NETWORKX.BARABÁSI_ALBERT_9	0.07002625	farthest-d
NETWORKX.BARABÁSI_ALBERT_9	0.07776382	planar
NETWORKX.BARABÁSI_ALBERT_9	0.08039156	betweenness centrality
NETWORKX.BARABÁSI_ALBERT_9	0.15977675	*farthest-ecc*
NETWORKX.BARBELL_GRAPH_EVEN	0.073432	random
NETWORKX.BARBELL_GRAPH_EVEN	0.07634492	farthest-d
NETWORKX.BARBELL_GRAPH_EVEN	0.07487301	planar
NETWORKX.BARBELL_GRAPH_EVEN	0.07635513	betweenness centrality
NETWORKX.BARBELL_GRAPH_ODD	0.07456582	random
NETWORKX.BARBELL_GRAPH_ODD	0.0769721	farthest-d
NETWORKX.BARBELL_GRAPH_ODD	0.07638805	planar
NETWORKX.BARBELL_GRAPH_ODD	0.07777363	betweenness centrality
NETWORKX.CIRCULAR_LADDER_GRAPH	0.22774735	random
NETWORKX.CIRCULAR_LADDER_GRAPH	0.23844454	farthest-d
NETWORKX.CIRCULAR_LADDER_GRAPH	0.21353256	planar
NETWORKX.CIRCULAR_LADDER_GRAPH	0.26295345	betweenness centrality

NETWORKX.COMPLETE_GRAPH	0.19454333	random
NETWORKX.COMPLETE_GRAPH	0.23383333	farthest-d
NETWORKX.COMPLETE_GRAPH	0.22621	planar
NETWORKX.COMPLETE_GRAPH	0.23251667	betweenness centrality
NETWORKX.CYCLE_GRAPH	0.92052778	random
NETWORKX.CYCLE_GRAPH	0.91600154	farthest-d
NETWORKX.CYCLE_GRAPH	0.9077191	planar
NETWORKX.CYCLE_GRAPH	0.93834186	betweenness centrality
NETWORKX.CYCLE_GRAPH	0.96070461	*farthest-ecc*
NETWORKX.ERDOS_RENYI_15	0.06646538	random
NETWORKX.ERDOS_RENYI_15	0.06908302	farthest-d
NETWORKX.ERDOS_RENYI_15	0.06604975	planar
NETWORKX.ERDOS_RENYI_15	0.06698582	betweenness centrality
NETWORKX.ERDOS_RENYI_30	0.2989426	random
NETWORKX.ERDOS_RENYI_30	0.37306945	farthest-d
NETWORKX.ERDOS_RENYI_30	0.30568982	planar
NETWORKX.ERDOS_RENYI_30	0.28313647	betweenness centrality
NETWORKX.LADDER_GRAPH	0.25560181	random
NETWORKX.LADDER_GRAPH	0.24253707	farthest-d
NETWORKX.LADDER_GRAPH	0.20743344	planar
NETWORKX.LADDER_GRAPH	0.25252634	betweenness centrality
NETWORKX.LADDER_GRAPH	0.18189499	*farthest-ecc*
NETWORKX.PATH_GRAPH	0.94652043	random

NETWORKX.PATH_GRAPH	0.94830543	farthest-d
NETWORKX.PATH_GRAPH	0.93117669	planar
NETWORKX.PATH_GRAPH	0.95403131	betweenness centrality
NETWORKX.RANDOM_LOBSTER_45	0.43970513	random
NETWORKX.RANDOM_LOBSTER_45	0.5726334	farthest-d
NETWORKX.RANDOM_LOBSTER_45	0.4154184	planar
NETWORKX.RANDOM_LOBSTER_45	0.42582864	betweenness centrality
NETWORKX.RANDOM_LOBSTER_90	0.26528347	random
NETWORKX.RANDOM_LOBSTER_90	0.34019603	farthest-d
NETWORKX.RANDOM_LOBSTER_90	0.26457455	planar
NETWORKX.RANDOM_LOBSTER_90	0.24160878	betweenness centrality
NETWORKX.WATTS_STROGATZ_10	0.0857167	random
NETWORKX.WATTS_STROGATZ_10	0.08798697	farthest-d
NETWORKX.WATTS_STROGATZ_10	0.09040027	planar
NETWORKX.WATTS_STROGATZ_10	0.09307308	betweenness centrality
NETWORKX.WATTS_STROGATZ_20	0.10739018	random
NETWORKX.WATTS_STROGATZ_20	0.1075506	farthest-d
NETWORKX.WATTS_STROGATZ_20	0.11082154	planar
NETWORKX.WATTS_STROGATZ_20	0.1085017	betweenness centrality
NETWORKX.WAXMAN_GRAPH	0.19642262	random
NETWORKX.WAXMAN_GRAPH	0.21845825	farthest-d
NETWORKX.WAXMAN_GRAPH	0.1896359	planar
NETWORKX.WAXMAN_GRAPH	0.18904927	betweenness centrality

NETWORKX.WAXMAN_GRAPH	0.28171423	*farthest-ecc*

References

Aardal, K., Nemhauser, G. L., & Weismantel, R. (2005). *Handbooks in Operations Research and Management Science: Discrete Optimization*: Elsevier Science.

Alspach, B., Bermond, J. C., & Sotteau, D. (1990). Decomposition into Cycles I: Hamilton Decompositions. In G. Hahn, G. Sabidussi & R. Woodrow (Eds.), *Cycles and Rays* (Vol. 301, pp. 9-18): Springer Netherlands.

Andersen, R., Chung, F., & Lang, K. (2006). *Local Graph Partitioning using PageRank Vectors.* Paper presented at the Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '06), Washington DC.

Awasthi, A., Lechevallier, Y., Parent, M., & Proth, J. M. (2005, 13-15 Sept. 2005). *Rule based prediction of fastest paths on urban networks.* Paper presented at the Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE.

Aynaud, T. (2010). Community detection for NetworkX's documentation¶. from http://perso.crans.org/aynaud/communities/index.html

Bao, Y., Feng, G., Liu, T.-Y., Ma, Z.-M., & Wang, Y. (2006). Ranking Websites: A Probabilistic View. *Internet Mathematics, 3*(3), 295-320. doi: 10.1080/15427951.2006.10129125

Bard, J. F., Yu, G., & Arguello, M. F. (2001). Optimizing aircraft routings in response to groundings and delays. *Iie Transactions, 33*(10), 931-947. doi: 10.1080/07408170108936885

Bauer, R., Columbus, T., Katz, B., Krug, M., & Wagner, D. (2010). *Preprocessing speed-up techniques is hard*. Paper presented at the Proceedings of the 7th international conference on Algorithms and Complexity, Rome, Italy.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2011). Cython: The Best of Both Worlds. *Computing in Science and Engg., 13*(2), 31-39. doi: 10.1109/mcse.2010.118

Blondel, V. D., Guillaume, J. L., Lambiotte, R., & Lefebvre, E. (2008). Fast unfolding of communities in large networks. *Journal of Statistical Mechanics-Theory and Experiment*, 12. doi: 10.1088/1742-5468/2008/10/p10008

Bo, W., & Dong, J.-X. (2010). *The System of GPS Navigation Based on ARM Processor*. Paper presented at the Proceedings of the 2010 International Forum on Information Technology and Applications - Volume 02.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual (Web) search engine. *Computer Networks and ISDN Systems, 30*, 107-117.

Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., . . . Weiss, H. (2007). {RFC 4838, Delay-Tolerant Networking Architecture}. *IRTF DTN Research Group*. doi: citeulike-article-id:7179323

Chan, T. M. (2007). *More algorithms for all-pairs shortest paths in weighted graphs*. Paper presented at the Proceedings of the thirty-ninth annual ACM symposium on Theory of computing, San Diego, California, USA.

Chen, J., Bardes, E. E., Aronow, B. J., & Jegga, A. G. (2009). ToppGene Suite for gene list enrichment analysis and candidate gene prioritization. *Nucleic Acids Research, 37*, W305-W311. doi: 10.1093/nar/gkp427

Chen, P., Xie, H., Maslov, S., & Redner, S. (2007). Finding scientific gems with Google's PageRank algorithm. *Journal of Informetrics, 1*(1), 8-15. doi: 10.1016/j.joi.2006.06.001

Chittka, L., Geiger, K., & Kunze, J. A. N. (1995). The influences of landmarks on distance estimation of honey bees. *Animal Behaviour, 50*(1), 23-31. doi: http://dx.doi.org/10.1006/anbe.1995.0217

Costa, M., Castro, M., Rowstron, A., & Key, P. (2004, 2004). *PIC: practical Internet coordinates for distance estimation.* Paper presented at the Distributed Computing Systems, 2004. Proceedings. 24th International Conference on.

Cowen, L. J. (1999). *Compact routing with minimum stretch*. Paper presented at the Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Baltimore, Maryland, USA.

Cullum, J. K., & Willoughby, R. A. (2002). *Lanczos Algorithms for Large Symmetric Eigenvalue Computations, Vol. 1*: Society for Industrial and Applied Mathematics.

Das Sarma, A., Gollapudi, S., & Panigrahy, R. (2011). Estimating PageRank on Graph Streams. *Journal of the Acm, 58*(3). doi: 10.1145/1970392.1970397

Delling, D., Goldberg, A. V., Pajor, T., & Werneck, R. F. (2011). *Customizable route planning*. Paper presented at the Proceedings of the 10th international conference on Experimental algorithms, Crete, Greece.

Delling, D., Sanders, P., Schultes, D., & Wagner, D. (2009). *Highway Hierarchies Star* (Vol. 74).

Delling, D., & Wagner, D. (2007). *Landmark-based routing in dynamic graphs*. Paper presented at the Proceedings of the 6th international conference on Experimental algorithms, Rome, Italy.

Demetrescu, C., Goldberg, A., & Johnson, D. (2006). *Challenge Datasets* [TIGER/Line graph]. Retrieved from: http://www.dis.uniroma1.it/challenge9/download.shtml

Developers, N. (2010). NetworkX. *networkx. lanl. gov*.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik, 1*(1), 269-271. doi: citeulike-article-id:2215313

doi: 10.1007/BF01386390

Dong, Z., ZuKuan, W., Jae-Hong, K., & ShuGuang, T. (2010, 25-27 June 2010). *An optimized Dijkstra algorithm for Embedded-GIS.* Paper presented at the Computer Design and Applications (ICCDA), 2010 International Conference on.

Duan, R., Pettie, S., & Siam/Acm. (2009). Dual-Failure Distance and Connectivity Oracles. *Proceedings of the Twentieth Annual Acm-Siam Symposium on Discrete Algorithms*, 506-515.

Erdős, P., & Rényi, A. (1959). On random graphs. *Publicationes Mathematicae Debrecen, 6*, 290-297.

Euler, L. (1736). Solutio problematis ad geometriam situs pertinentis. *Graph Theory 1736-1936*. doi: citeulike-article-id:6649492

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Commun. ACM, 5*(6), 345. doi: 10.1145/367766.368168

Fortz, B., & Thorup, M. (2000, 2000). *Internet traffic engineering by optimizing OSPF weights.* Paper presented at the INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE.

Fredman, M. L., & Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM, 34*(3), 596-615. doi: 10.1145/28869.28874

Freedman, D. (1971). *Markov chains*: Holden-Day.

Freeman, L. C. (1979). Centrality in social networks conceptual clarification. *Social networks, 1*(3), 215-239.

Fuchs, F. (2010). *On Preprocessing the ALT-Algorithm.* (Master's thesis), University of Karlsruhe, Institute for Theoretical Informatics.

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008a). Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In C. C. McGeoch (Ed.), *Experimental Algorithms, Proceedings* (Vol. 5038).

Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008b). *Contraction hierarchies: faster and simpler hierarchical routing in road networks*. Paper presented at the Proceedings of the 7th international conference on Experimental algorithms, Provincetown, MA, USA.

Ghosh, A., Boyd, S., & Saberi, A. (2008). Minimizing Effective Resistance of a Graph. *SIAM Rev., 50*(1), 37-66. doi: 10.1137/050645452

Goh, K.-I., Kahng, B., & Kim, D. (2001). Universal behavior of load distribution in scale-free networks. *Physical Review Letters, 87*(27), 278701.

Goldberg, A., & Werneck, R. (2005). *Computing Point-to-Point Shortest Paths from External Memory*. Paper presented at the Proceedings of the Seventh Workshop on Algorithm Engineering and Experiments (ALENEX'05).

Goldberg, A. V., & Harrelson, C. (2005). Computing the Shortest Path: A* Search Meets Graph Theory. *Proceedings of the Sixteenth Annual Acm-Siam Symposium on Discrete Algorithms*, 156-165.

Goldberg, A. V., Kaplan, H., & Werneck, R. F. (2009). Reach for A*: Shortest Path Algorithms with Preprocessing. In C. Demetrescu, A. V. Goldberg & D. S. Johnson (Eds.), *Shortest Path Problem* (Vol. 74, pp. 93-139). Providence: Amer Mathematical Soc.

Goldberg, A. V., & Werneck, R. F. (2005). *Computing point-to-point shortest paths from external memory*. Paper presented at the Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX’05).

Goldman, R., Shivakumar, N., Venkatasubramanian, S., & Garcia-Molina, H. (1998). Proximity search in databases. In A. Gupta, O. Shmueli & J. Widom (Eds.), *Proceedings of the Twenty-Fourth International Conference on Very-Large Databases* (pp. 26-3737): Morgan Kaufmann Publishers Inc.

Golomb, S. W., & Lushbaugh, W. (1996). *Polyominoes: Puzzles, Patterns, Problems, and Packings*: Princeton University Press.

Griffith, A. (2002). *GCC: The Complete Reference*: McGraw-Hill, Inc.

Gross, J. L., & Yellen, J. (2005). *Graph Theory and Its Applications, Second Edition*: CRC Press.

Gutman, R. (2004). *Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks*. Paper presented at the Proceedings 6th Workshop on Algorithm Engineering and Experiments (ALENEX).

Halfacree, G., & Upton, E. (2012). *Raspberry Pi User Guide*: Wiley Publishing.

Han, W.-S., Lee, J., Pham, M.-D., & Yu, J. X. (2010). iGraph: a framework for comparisons of disk-based graph indexing techniques. *Proc. VLDB Endow., 3*(1-2), 449-459.

Harary, F., & Schwenk, A. J. (1979). The spectral approach to determining the number of walks in a graph. 443-449.

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, IEEE Transactions on, 4*(2), 100-107. doi: 10.1109/TSSC.1968.300136

Holdsworth, J. J., & Lui, S. M. (2009). *GPS-enabled mobiles for learning shortest paths: a pilot study*. Paper presented at the Proceedings of the 4th International Conference on Foundations of Digital Games, Orlando, Florida.

Hutchinson, D., Maheshwari, A., & Zeh, N. (2003). An external memory data structure for shortest path queries. *Discrete Appl. Math., 126*(1), 55-82. doi: 10.1016/s0166-218x(02)00217-2

Jain, S., Fall, K., & Patra, R. (2004). *Routing in a delay tolerant network* (Vol. 34): ACM.

Johnson, D. B. (1977). Efficient Algorithms for Shortest Paths in Sparse Networks. *J. ACM, 24*(1), 1-13. doi: 10.1145/321992.321993

Kamvar, S., Haveliwala, T., & Golub, G. (2004). Adaptive methods for the computation of PageRank. *Linear Algebra and Its Applications, 386*, 51-65. doi: 10.1016/j.laa.2003.12.008

Katz, L. (1953). A new status index derived from sociometric analysis. *Psychometrika, 18*(1), 39-43.

Kay, D. C. (2011). *College Geometry: A Unified Development*: Taylor & Francis.

Lauther, U. (2004). An extremely fast, exact algorithm for finding shortest paths in static networks with geographical background. *Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, 22*, 219-230.

Lin, X. H., Kwok, Y. K., & Lau, V. K. N. (2003). A genetic algorithm based approach to route selection and capacity flow assignment. *Computer Communications, 26*(9), 961-974. doi: 10.1016/s0140-3664(02)00240-2

Liu, X. M., Bollen, J., Nelson, M. L., & Van de Sompel, H. (2005). Co-authorship networks in the digital library research community. *Information Processing & Management, 41*(6), 1462-1480. doi: 10.1016/j.ipm.2005.03.012

Luo, D. J., Zhu, X. J., Wu, X. B., Chen, G. H., & Ieee. (2011). Maximizing Lifetime for the Shortest Path Aggregation Tree in Wireless Sensor Networks *2011 Proceedings Ieee Infocom* (pp. 1566-1574). New York: Ieee.

M, R. H., #246, hring, Schilling, H., Sch, B., #252, . . . Willhalm, T. (2007). Partitioning graphs to speedup Dijkstra's algorithm. *J. Exp. Algorithmics*, *11*, 2.8. doi: 10.1145/1187436.1216585

Maruhashi, K., Shigezumi, J., Yugami, N., & Faloutsos, C. (2012). *EigenSP: A More Accurate Shortest Path Distance Estimation on Large-Scale Networks*. Paper presented at the Proceedings of the 2012 IEEE 12th International Conference on Data Mining Workshops.

Maue, J. (2006). *A Goal-Directed Shortest Path Algorithm Using Precomputed Cluster Distances.* (Master's Thesis), Saarland University, Saarbr{\"u}cken. Retrieved from http://www.n.ethz.ch/~mauej/publications/maue-06.pdf

Maue, J., Sanders, P., & Matijevic, D. (2010). Goal-directed shortest-path queries using precomputed cluster distances. *J. Exp. Algorithmics*, *14*, 3.2-3.27. doi: 10.1145/1498698.1564502

Maue, J., Sanders, P., Matijevic, D., Alvarez, C., & Serna, M. (2006). Goal directed shortest path queries using precomputed cluster distances. *Experimental Algorithms, Proceedings, 4007*, 316-327.

Miao, Q. (2014). Approximate Shortest Distance Computing: A Query-Dependent Local Landmark Scheme. *IEEE Transactions on Knowledge and Data Engineering, 26*(1), 55-68.

Millman, R. S., & Parker, G. D. (1991). *Geometry: A Metric Approach with Models*: Springer.

Mises, R. V., & Pollaczek-Geiringer, H. (1929). Praktische Verfahren der Gleichungsauflösung. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik, 9*(2), 152-164. doi: 10.1002/zamm.19290090206

Newman, M. E. (2001). Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Physical review E, 64*(1), 016132.

Noy, M., & Ribó, A. (2004). Recursively constructible families of graphs. *Advances in Applied Mathematics, 32*(1–2), 350-363. doi: http://dx.doi.org/10.1016/S0196-8858(03)00088-5

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank citation ranking: Bringing order to the Web. Stanford: Stanford University.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*: Addison-Wesley Longman Publishing Co., Inc.

Pons, P., & Latapy, M. (2005). Computing Communities in Large Networks Using Random Walks. In p. Yolum, T. Güngör, F. Gürgen & C. Özturan (Eds.), *Computer and Information Sciences - ISCIS 2005* (Vol. 3733, pp. 284-293): Springer Berlin Heidelberg.

Potamias, M., Bonchi, F., Castillo, C., & Gionis, A. (2009). *Fast shortest path distance estimation in large networks*. Paper presented at the Proceedings of the 18th ACM conference on Information and knowledge management, Hong Kong, China.

Royset, J. O., Carlyle, W. M., & Wood, R. K. (2009). Routing Military Aircraft With A Constrained Shortest-Path Algorithm. *Military Operations Research, 14*(3), 31-52.

Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*: Prentice Hall Press.

Sanders, P., & Schultes, D. (2007). Engineering fast route planning algorithms. *Experimental Algorithms, Proceedings, 4525*, 23-36.

Sankaranarayanan, J., & Samet, H. (2010). Query Processing Using Distance Oracles for Spatial Networks. *Knowledge and Data Engineering, IEEE Transactions on, 22*(8), 1158-1175. doi: 10.1109/TKDE.2010.75

Sankaranarayanan, J., Samet, H., & Alborzi, H. (2009). Path oracles for spatial networks. *Proc. VLDB Endow., 2*(1), 1210-1221.

Santhosh, S. S., Sasiprabha, T., & Jeberson, R. (2010, 13-15 Nov. 2010). *BLI - NAV embedded navigation system for blind people.* Paper presented at the Recent Advances in Space Technology Services and Climate Change (RSTSCC), 2010.

Seidel, R. (1995). On the all-pairs-shortest-path problem in unweighted undirected graphs. *J. Comput. Syst. Sci., 51*(3), 400-403. doi: 10.1006/jcss.1995.1078

Shimbel, A. (1953). Structural parameters of communication networks. *The bulletin of mathematical biophysics, 15*(4), 501-507. doi: 10.1007/BF02476438

Sommer, C. (2012). Shortest-Path Queries in Static Networks.

Soundarajan, S., & Hopcroft, J. E. (2015). Use of Local Group Information to Identify Communities in Networks. *ACM Trans. Knowl. Discov. Data, 9*(3), 1-27. doi: 10.1145/2700404

Strang, G. (2007). *Computational Science and Engineering*: Wellesley-Cambridge Press.

Summerfield, M. (2013). *Python in Practice: Create Better Programs Using Concurrency, Libraries, and Patterns*: Addison-Wesley Professional.

Sun, T. L., Deng, K. Y., & Deng, J. W. (2008). *Novel numerical methods for rapid computation of PageRank*. Beijing: Publishing House Electronics Industry.

Surhone, L. M., Tennoe, M. T., & Henssonow, S. F. (2011). *Cython*: VDM Publishing.

Takes, F. W., & Kosters, W. A. (2014). *Adaptive Landmark Selection Strategies for Fast Shortest Path Computation in Large Real-World Graphs*. Paper presented at the Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 01.

Thorup, M., & Zwick, U. (2001). *Approximate distance oracles*. Paper presented at the Proceedings of the thirty-third annual ACM symposium on Theory of computing, Hersonissos, Greece.

Voevodski, K., Teng, S.-H., & Xia, Y. (2009a). Finding local communities in protein networks. *BMC Bioinformatics, 10*, 297.

Voevodski, K., Teng, S.-H., & Xia, Y. (2009b). Spectral affinity in protein networks. *BMC Systems Biology, 3*(112).

Wagner, D., Willhalm, T., & Zaroliagis, C. (2005). Geometric containers for efficient shortest-path computation. *J. Exp. Algorithmics, 10*, 1.3. doi: 10.1145/1064546.1103378

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of /`small-world/' networks. *Nature, 393*(6684), 440-442.

Weis, M., & Naumann, F. (2004). *Detecting duplicate objects in XML documents*. Paper presented at the Proceedings of the 2004 international workshop on Information quality in information systems, Paris, France.

Yussof, S., Razali, R. A., Ong Hang, S., Ghapar, A. A., & Din, M. M. (2009, 25-27 June 2009). *A Coarse-Grained Parallel Genetic Algorithm with Migration for Shortest Path Routing Problem.* Paper presented at the High Performance Computing and Communications, 2009. HPCC '09. 11th IEEE International Conference on.

Zadorozhnyi, V. N., & Yudin, E. B. (2012). Structural properties of the scale-free Barabasi-Albert graph. *Autom. Remote Control, 73*(4), 702-716. doi: 10.1134/s0005117912040091

Zakzouk, A. A. A., Zaher, H. M., & El-Deen, R. A. Z. (2010, 28-30 March 2010). *An ant colony optimization approach for solving shortest path problem with fuzzy constraints.* Paper presented at the Informatics and Systems (INFOS), 2010 The 7th International Conference on.

Zongyan, X., Haihua, L., & Ye, G. (2012, 17-19 Aug. 2012). *A Study on the Shortest Path Problem Based on Improved Genetic Algorithm.* Paper presented at the Computational and Information Sciences (ICCIS), 2012 Fourth International Conference on.