BIVARIATE BEST FIRST SEARCHES TO PROCESS CATEGORY BASED QUERIES IN A GRAPH FOR TRIP PLANNING APPLICATIONS IN TRANSPORTATION

QIFENG LU

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

PHILOSOPHY OF SCIENCE
in
Civil Engineering

Dr. Kathleen Hancock, Chair
Dr. Randy L. Dymond
Dr. Shinya Kikuchi
Dr. Chang-Tien Lu
Dr. Scott F. Midkiff

February 25, 2009
Alexandria, Virginia

Keywords: Bivariate, multivariate, best first search, category based queries, trip planning, path, heuristic, optimization, consistent, admissible, L#, C*, O*, C*-P, O*-MST, O*-SCDMST, C*-Dijkstra, O*-Dijkstra, O*-Greedy, OSTQ, CSTQ, graph, state graph, state graph space, logistics, transportation network, Geographic Information System, Geographic Information Science

BIVARIATE BEST FIRST SEARCHES TO PROCESS CATEGORY BASED QUERIES IN A GRAPH FOR TRIP PLANNING APPLICATIONS IN TRANSPORTATION

Qifeng Lu

ABSTRACT

With the technological advancement in computer science, Geographic Information Science (GIScience), and transportation, more and more complex path finding queries including category based queries are proposed and studied across diverse disciplines. A category based query, such as Optimal Sequenced Routing (OSR) queries and Trip Planning Queries (TPQ), asks for a minimum-cost path that traverses a set of categories with or without a predefined order in a graph. Due to the extensive computing time required to process these complex queries in a large scale environment, efficient algorithms are highly desirable whenever processing time is a consideration. In Artificial Intelligence (AI), a best first search is an informed heuristic path finding algorithm that uses domain knowledge as heuristics to expedite the search process. Traditional best first searches are single-variate in terms of the number of variables to describe a state, and thus not appropriate to process these queries in a graph. In this dissertation, 1) two new types of category based queries, Category Sequence Traversal Query (CSTQ) and Optimal Sequence Traversal Query (OSTQ), are proposed; 2) the existing single-variate best first searches are extended to multivariate best first searches in terms of the state specified, and a class of new concepts--state graph, sub state graph, sub state graph space, local heuristic, local admissibility, local consistency, global heuristic, global admissibility, and global consistency--is

introduced into best first searches; 3) two bivariate best first search algorithms, C* and O*, are developed to process CSTQ and OSTQ in a graph, respectively; 4) for each of C* and O*, theorems on optimality and optimal efficiency in a sub state graph space are developed and identified; 5) a family of algorithms including C*-P, C-Dijkstra, O*-MST, O*-SCDMST, O*-Dijkstra, and O*-Greedy is identified, and case studies are performed on path finding in transportation networks, and/or fully connected graphs, either directed or undirected; and 6) O*-SCDMST is adopted to efficiently retrieve optimal solutions for OSTQ using network distance metric in a large transportation network.

ACKNOWLEDGEMENTS

I am extremely grateful to my advisor Dr. Kathleen Hancock for her constant support, guidance, and encouragement to help me make through the Ph.D. journey with my best efforts.

I would like to thank Dr. Lu and his Spatial Data Management research group for constantly providing feedbacks and challenging me in research to improve the research quality and make it robust and sound.

I was touched by Dr. Midkiff's earnest advisory manner and impressed by his comprehensive and high-standard advice.

I also thank Dr. Dymond and Dr. Kikuchi for their invaluable and detailed suggestions and mentoring, and accompanying me through the whole Ph.D. process.

My thanks due are to Virginia Tech staff, Lindy Cranwell of Blacksburg, Marija Telbis-Forster of Falls Church and Angela King of Alexandria campuses for being so supportive and cooperative.

DEDICATIONS

I would like to specially dedicate my work to my wife, Yumei Wang, who has been supporting, trusting, encouraging, and accompanying me through my whole Ph.D. journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ACS	Ant Colony System
AI	Artificial Intelligence
ANSE	Average Number of States Expanded
AOD	Average Obtained Distance
APT	Average Process Time
ARNS	Average Relative Number of States expanded
ARPT	Average Relative Process Time
AORC	Average Optimal Route Cost
ASD	Average Shortest Distance
C*	A bivariate best first search to process CSTQ
C*-P	An instance of C* that uses g-p as the global heuristics
C*-Dijkstra	An instance of C* whose heuristics are 0
CC	Category Constraint
CD	C*-Dijkstra
CSTQ	Category Sequence Traversal Query
ES	Exhaustive Search
g-G	A graph containing vertices that are neither points of interest, nor the given origin or the given destination.
GPS	Global positioning System
f(n)	The function to estimate the promise of a vertex n to be expanded
f(s)	The function to estimate the promise of a state s to be expanded
g-p	A global heuristic used by C*-P

GIS	Geographic Information System
GMST	Generalized Minimum Spanning Tree
IRNN	In-Route Nearest Neighbor
MaxAC-CD	Maximum Additional Cost by C*-Dijkstra
MaxANS-CD	Maximum Additional Number of States expanded by C*-Dijkstra
MaxPT	Maximum Process Time
MaxRNS	Maximum Relative Number of States expanded
MaxRPT	Maximum Relative Process Time
MinAC-CD	Minimum Additional Cost by C*-Dijkstra
MinANS-CD	Minimum Additional Number of States expanded by C*-Dijkstra
MinRNS	Minimum Relative Number of States expanded
MinPT	Minimum Process Time
MinRPT	Minimum Relative Process Time
MDA	Minimum Distance Algorithm
MRPSR	Multi-Rule Partial Sequenced Route
MST	Minimum Spanning Tree
MTNN	Multi-Type Nearest Neighbor
NN	Nearest Neighbor
NNB	Nearest Neighbor Based
NOC	Number of Categories
NOEC	Number of Objects in Each Category
NP	Non-deterministic Polynomial
NPoI	Number of Points of Interest

O*	A bivariate best first search to process OSTQ
O*-MST	O*-Minimum Spanning Tree
O*-SCDMST	O*- Semi-Connected Directed Minimum Spanning Tree
OD	O*-Dijkstra
OG	O*-Greedy
OSR	Optimal Sequenced Route
OSTQ	Optimal Sequence Traversal Query
PLUB	Page-Level Upper Bound
RLORD	An algorithm to process OSR query in a spatial database
RPT	Relative Process Time
SCDST	Semi-Connected Directed Spanning Tree
SCDMST	Semi-Connected Directed Minimum Spanning Tree
SG	State graph
SSG	Sub state graph
SSGS	Sub state graph space
TS	Tabu Search
TPQ	Trip Planning Query
TSP	Traveling Salesman Problem

LIST OF DEFINITIONS

Term	Brief Definition	Detailed Reference
δ graph	A graph whose edge cost is never negative	P20, line 42
Average Number of States Expanded	The average number of expanded states obtained over all runs	P41, line 20
Average Obtained Distance	The average route distance obtained over all runs	P65, line 8
Average Process Time	The computation time required to return the solution for a query	P41, line 28
Average Relative Number of States expanded	The ratio of the number of states expanded by one algorithm over the other	P41, line 35
Average Relative Process Time	The ratio of the average computation time processed by one algorithm over the other	P41, line 37
Average Optimal Route Cost	The average cost of obtained optimal routes	P91, line 22
Average Shortest Distance	The average shortest route distance obtained over all runs	P41, line 19
C*	A bivariate best first search algorithm to process CSTQ in a graph	P26, line 1
C*-Dijkstra	A special case of C* when h(s)=0	P39, line 12
C*-P	A special case of C* using g-p as a global heuristics	P35, line 19
CSTQ	A query asking for a minimum-cost route staring from a given origin, traversing a set of categories of interest in a given order, to a given destination	P20, line 44
D-ODPrim	An algorithm to retrieve a semi-connected directed minimum spanning tree	P90, line 13
f(n)	The function to estimate the promise of a vertex n to be expanded	P15, line 20
f(s)	The function to estimate the promise of a state s to be expanded	P24, line 45
g-G	A special graph whose vertices may include normal vertices	P75, line 28
Global admissibility	A global heuristic is smaller than or equal to the actual cost	P25, line 32
Global consistency	The difference of global heuristics between two states is smaller than or equal to the actual cost of the path between these two states	P25, line 42
Global heuristic	The heuristic from the current state to the final state	P25, line 28
Graph	A structure composed of edges and vertices	P25, line 49
L#	A generalized best first search using g(s)+h(s) as f(s) to guide the search	P24, line 39
Local admissibility	The local heuristic is smaller than or equal to	P25, line 31

	the actual cost between a vertex and a subgoal	
Local consistency	The difference of local heuristics between two vertices is smaller than or equal to the actual cost of the path between them	P25, line 36
Local heuristic	The heuristic between a vertex and a subgoal	P25, line 27
Maximum additional cost by C*-Dijkstra	For each run, compared to C*-P, obtain the additional time cost required by C-Dijkstra, and the measure is the maximum among all runs	P41, line 29
Maximum additional number of states expanded by C*-Dijkstra	For each run, compared to C*-P, obtain the additional number of states expanded by C*-Dijkstra, and the measure is the maximum among all runs	P41, line 35
Maximum Process Time	The maximum time required to obtain a solution for each number of points of interest	P80, line 8
Maximum Relative Number of States expanded	For each run of an algorithm, obtain the ratio of its number of expanded states over the number of states expanded by another algorithm for the same problem, and the measure is the maximum ratio among all runs	P65, line 14
Maximum Relative Process Time	For each run of an algorithm, obtain the ratio of its process time over that of another algorithm, and the measure is the maximum ratio among all runs	P65, line 24
Minimum Additional Cost by C*-Dijkstra	For each run, compared to C*-P, obtain the additional time cost required by C-Dijkstra, and the measure is the maximum among all runs	P41, line 17
Minimum Additional Number of States expanded by C*-Dijkstra	For each run, compared to C*-P, obtain the additional number of states expanded by C*-Dijkstra, and the measure is the minimum among all runs	P41, line 25
Minimum Relative Number of States expanded	For each run of an algorithm, obtain the ratio of its number of expanded states over the number of states expanded by another algorithm for the same problem, and the measure is the minimum ratio among all runs	P65, line 18
Minimum Relative Process Time	For each run of an algorithm, obtain the ratio of its process time over that of another algorithm, and the measure is the minimum ratio among all runs	P65, line 27
Minimum Process Time	The minimum time required to obtain a solution for each number of points of interest	P80, line 6
Multivariate best first search	A best first search that uses multiple variables to describe its state	P25, line 18
No more informed (C*)	Algorithm A is no more informed than	P32, line 46

	algorithm B if the state sets used by B can not be excluded by A	
No more informed (O*)	Algorithm A is no more informed than algorithm B if the state sets used by B can not be excluded by A	P59, line 32
Normal vertices	Vertices that are neither points of interest, nor a given origin or a destination	P21, line 1
NP hard	A term used to describe the time complexity of a problem that could only be resolved in a non-deterministic polynomial time	P2, line 14
O*	A bivariate best first search to process OSTQ in a graph	P52, line 44
O*-Dijkstra	A special case of O* when h(s)=0	P64, line 13
O*-Greedy	A special case of O* when g(s)=0	P64, line 18
O*-MST	A special case of O* using MSTs to provide global heuristics	P61, line 37
O*-SCDMST	A special case of O* using SCDMST to provide global heuristics	P90, line 11
Object	A vertex in a category of interest	P26, line 10
Operator Γ	An operator to generate a child state	P26, line 23
Operator Ψ(C*)	An operator to promote the state of a vertex in C*	P26, line 25
Operator Ψ(O*)	An operator to promote the state of a vertex in O*	P53, line 18
OSTQ	A query that asks for a minimum-cost path that starts from a given origin, traverses a set of vertices of interest, and ends at a given destination	P50, line 22
Partial heuristic	The minimum heuristic from a category of interest to a goal state	P26, line 45
Percent Average Relative Increased Distance	The ratio of the average route distance obtained by O*-Greedy over the average optimal route distance obtained by O*-MST minus 1, multiplied by 100	P126, line 3
SCDST	A semi-connected spanning tree in a directed graph	P89, line 5
SCDMST	A semi-connected minimum spanning tree in a directed graph	P89, line 8
Space complexity	A term used in computer science to describe the complexity level in memory usage	P16, line 16
State	The status or condition of a vertex	P24, line 42
State (C*)	The status or condition of a vertex in C*	P26, line 22
State (O*)	The status or condition of a vertex in O*	P53, line 14
State graph (C*)	The graph composed of vertices of the same VisitOrder in C*	P26, line 26
State graph (O*)	The graph composed of vertices of the same traversed vertices of interest in O*	P53, line 19

State graph space (C*)	A space containing all the state graphs in C*	P26, line 30
State graph space (O*)	A space containing all the state graphs in O*	P53, line 22
Subgoal	A vertex in a category of interest	P25, line 14
Subgoal based global heuristic	The global heuristics for a vertex in a category of interest	P26, line 45
Sub state graph space (C*)	The implicitly generated sub state graphs by C*	P26, line 32
Sub state graph space (O*)	The implicitly generated sub state graphs by O*	P53, line 25
Time complexity	A term in computer science to describe the complexity level in computation time	P2, line 12
Vertex	A variable that presents a node in a graph	P4, line 1
VisitList/VL	A sorted list containing the traversed vertices of interest	P53, line 13
VisitOrder	An integer indicating the next category to be visited	P26, line 17

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

With the advancement of technologies in geospatial databases [1] [2] [3] [4], location based services [5] [6] [7], geographic information systems (GIS) [8] [9] [10] [11], and artificial intelligence [12] [13] [14], increasingly complex queries and their variants are being developed. As a set of route finding problems in transportation, computer science, and operations research, multi-category based queries have begun to attract attention over recent years. These queries include Trip Planning Query (TPQ) [1], Multi-Type Nearest Neighbor (MTNN) query [15], Optimal Sequenced Route query (OSR) [16] [17], Multi-Rule Partial Sequenced Route (MRPSR) query [18], and In-Route Nearest Neighbor (IRNN) [19] query. All these multi-category based queries require that the route traverse multiple points from one or more distinct categories.

For trip planning in transportation, multi-category based queries are usually performed by those who are not familiar with the region they are visiting, especially in a metropolitan area with a large floating population. Since they are not familiar with the region, they may not be able to readily locate consumer destinations, but they do know what kind of consumer destinations they want to visit. Furthermore, since nationwide enterprises, like Citibank, Giant supermarket, Shopper warehouse, Costco, Target, WalMart, and so on, may have multiple branches within a region, even local residents can use these types of queries to find a route to these business locations. Under this condition, each enterprise may be considered as one category. Generally, users may perform either ordered or non-ordered queries, i.e., Trip Planning Queries or Optimal Sequenced Route queries. Ordered visits to consumer destinations are common in the real world. For instance, people may have a set order for accomplishing their trips.

In operations research, multi-category based queries are fundamental to resolve logistical issues like multi-category goods distribution or collection. For example, different stores may have diverse types of goods demands, and a goods distributor wants an optimal route to reduce delivery costs. During this process, ordered or non-ordered visits may be

performed, and multiple deliveries may be conducted, although many such applications include additional constraints such as time windows and quantity limitations. Even though the quantity of goods is usually considered in the distribution process, cases exist where these factors do not affect the query process. For example, the quantity of each category of goods to be distributed in one route is below the demand of any store. In this case, a multi-category based query may provide fundamental support to find an optimized goods distribution solution.

In computer science, a category based path finding algorithm may be performed by an agent or a robot to collect data from categories of interest or areas of interest.

In general, the *time complexity*, a term used in computer science to describe the level of computation time required to resolve a problem [20], for algorithms to process category based queries is a substantial research challenge. A TPQ problem is NP hard (Non-deterministic Polynomial-time Hard) [1], i.e., it can only be resolved in non-deterministic polynomial time with regard to computation complexity. The time complexity for either an OSR query or a MTNN query is not mentioned in literature. However, for the naïve method that considers all possible orders to visit a set of categories with a number of choices in each category, a MTNN query is processed in exponential time in terms of the number of categories of interest and in polynomial time in terms of the maximum number of objects within a category. The time complexity for an OSR is simpler because the category visit order is predefined. It is polynomial in terms of the maximum number of objects within a category. Furthermore, it is time consuming to compute the network distance within a large network for each route search. Whenever a fast response time is required to process these queries on a large number of categories of interest within a (large) network, efficient algorithms are required to process these queries in a timely manner.

In artificial intelligence, heuristics from the problem domain have been used to expedite the automatic search process for optimal solutions in a graph [12] [13] [14]. A set of theorems exists for heuristic-based best first searches such as A* to demonstrate sub-

optimality, optimality, and optimal efficiency [12]. Consequently, this research focuses on heuristic searches. At the same time, since the category based queries are studied across different domains, and a graph can be used to represent different networks, this research focuses on developing heuristic-based efficient solutions to process these queries in a graph, and uses transportation networks and/or publically accessible graph data sources for case studies.

1.2 RESEARCH GOAL AND TASKS

The goal of the research is to efficiently process category based queries to obtain optimal or suboptimal solutions in a graph with best first searches and investigate their applications and performances for transportation trip planning.

The tasks of this research are to:

- Develop efficient algorithms to enable timely and effective response to multi-category based queries.
- Investigate how to incorporate heuristic information from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate both optimality and optimal efficiency properties in a sub state graph space.
- Evaluate the proposed algorithms' statistical performances in terms of average process time, defined as the average period required to receive a solution after a query is submitted, and average cost of obtained routes, defined as the average cost of the retrieved route, compared to the average optimal cost.
- Apply the developed algorithms to process category based trip planning in a transportation network.

1.3 RESEARCH WORK

In this research, two novel types of category based queries in a graph are proposed: Category Sequence Traversal Query (CSTQ) and Optimal Sequence Traversal Query (OSTQ). Due to the deficiency in the state specification in single-variate best first searches such as A* [14] where the promise to be expanded is evaluated upon a single

variable, *vertex*, that represents a node in a graph instead of a state of multiple variables, neither query may be appropriately processed with existing best first searches.

In this research, the traditional single-variate best first search is extended in terms of the number of variables used to specify a state to a multivariate best first search. For a multivariate state *s*, a multivariate best first search adopts the estimate to the promise of *s* to be expanded by a heuristic evaluation function *f(s)* which may depend on the description of *s*, the description of the goal state, the information gathered by the search up to that state, and any extra knowledge about the problem domain. A set of new concepts--state graph, sub state graph, sub state graph space, local heuristic, local admissibility, local consistency, global heuristic, global admissibility, and global consistency--is introduced into best first searches. As a special case of multivariate best first searches, L# using *g(s)+h(s)* as *f(s)* is developed, where *g(s)* is the cost from the origin state to s while *h(s)* is the estimated cost, or heuristic, from s to the final goal state.

Two novel bivariate best first search algorithms, C* and O*, are provided to process CSTQ and OSTQ, respectively. Both algorithms extend the traditional single-variate best first searches to bivariate best first searches. For each best first search algorithm, theoretical analyses are presented to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and to obtain a family of search strategies that demonstrate optimal and optimally efficient properties in a sub state graph space. Once global heuristics are shown to be globally consistent, C* is shown to be optimally efficient in a discrete sub state graph space where each sub state graph includes the expanded vertices to each of which every path from the origin has traversed the first *m*, a discrete integer variable, categories of interest in the given order, and O* is shown to be optimally efficient in an item-enumeration sub state graph space, where each sub state graph includes the expanded vertices to each of which every path from the origin has traversed the same set of enumerated vertices of interest. Thereafter, a family of algorithms including C*-P, C-Dijkstra, O*-MST, O*-SCDMST, O*-Dijkstra, and O*-Greedy is identified, and case studies are performed on these algorithms in transportation networks, and/or fully connected graphs, either directed or undirected.

Lastly, O*-SCDMST is adopted to efficiently retrieve optimal solutions for OSTQ using the network distance metric in a large transportation network.

Table 1 summarizes the work performed in this research.

Table 1: Summary of research work

Research work		**Description**
Category based query	**Category sequence traversal query (CSTQ)**	**Asks for a minimum-cost route starting from a given origin, traversing the categories *in a given order*, with at least one point from each category, and ending at the destination**
	Optimal sequence traversal query (OSTQ)	**Asks for a minimum-cost route starting from a given origin, traversing a set of *non-ordered* points of interest, and ending at a given destination**
Generalized best first search		1) **For a state *s* defined for a vertex and describing the status of the vertex along a path, a generalized best first search adopts estimates to the promise of *s* to be expanded by a heuristic evaluation function *f(s)* which may depend on the description of *s*, the description of the goal state, the information gathered by the search up to that state, and any extra knowledge about the problem domain.** 2) **M-variate if using m variables to describe *s***
L#		**A family of generalized best first searches using *g(s)+h(s)* as *f(s)*** ***(g(s): cost from origin state to s, and h(s): estimated cost from s to a goal state)***
C*		**A bivariate best first search to process CSTQ**
O*		**A bivariate best first search to process OSTQ**

1.4 DISSERTATION OUTLINE

The dissertation includes the introduction, literature review, independent technical papers, and conclusions and recommendations. The last section in each chapter is a list of literature references cited in that chapter.

Chapter 1: Introduction. This chapter provides the introduction to the research identified in the dissertation, including the motivation, research goals and objectives, and research approaches.

Chapter 2: Literature Review. This chapter reviews literature related to category based query processing in a graph in diverse domains including artificial intelligence, geographic information sciences, geodatabases, operations research, and transportation engineering, and how this work expands on and differentiates from the state of the practice.

Chapter 3: C*: A Bivariate Best First Search Algorithm to Process Category Sequence Traversal Queries in a Graph. This paper 1) generalizes existing single-variate best first searches to multivariate best first searches; 2) proposes a multivariate best first search algorithm, L#, and develops a set of new concepts including global heuristic, global admissibility, global consistency, local heuristic, local admissibility, local consistency, state graph, sub state graph, and sub state graph space, for best first searches; 3) provides C*, a bivariate best-first-search instance of L#, to process CSTQ in a graph, which extends the existing single-variate best first search to the bivariate best first search in the sense that the evaluation function used in C* is defined upon a bivariate state instead of the single variable to describe a vertex's identification; 4) discusses how to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate an optimal efficiency property in a discrete sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the first *m* categories of interest in the given order; and 5) adopts a bottom-up approach to provide a global heuristic, g-p, whose properties of optimality and optimal efficiency are

analyzed based on local heuristics. Since a minimum-cost path query for a given origin and destination pair is a special case of CSTQ, A* [12] [14], the single-variate best first graph search algorithm, is a special case of C*. Two special cases of C*, C*-P that uses g-p as the global heuristic, and C*-Dijkstra, are identified, and their performances are studied in two transportation networks of different scales.

Chapter 4: O*: A Bivariate Best First Search Algorithm to Process Optimal Sequence Traversal Queries in a Graph. It proposes a bivariate best first search, O*, to process OSTQ within a graph, and discusses how to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate both optimality and optimal efficiency properties in an item-enumeration sub state graph space where a sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of enumerated points of interest. Three special cases of O* are identified and their performances are studied in a transportation network. Since a minimum-cost path query for a given origin and destination pair is a special case of OSTQ, A*, the best-first graph search algorithm, is a special case of the more general O*. Since TSP is a special case of OSTQ when the origin and the destination is the same, the set of theorems provided for OSTQ is applicable to TSP.

Chapter 5: MST: to Providing a Globally Consistent Heuristic to Process Optimal Sequence Traversal Queries in a Euclidean Graph. This paper proves that O*-MST is optimally efficient in a Euclidean graph, and Kruskal's algorithm [21] with Delaunay Triangulation [22] are adopted to efficiently calculate MST heuristics. The performance of O*-MST is studied with a set of samples from a 130-city TSP problem [23].

Chapter 6: SCDMST: to Providing a Globally Consistent Heuristic to Process Optimal Sequence Traversal Queries in a Fully Connected Directed Graph. In this paper, O*-SCDMST is proposed to process OSTQ in a fully connected directed graph whose edge cost obeys the triangle inequality. O*-SCDMST is an O* algorithm using a semi-connected directed minimum spanning tree (SCDMST) to provide the global heuristic,

which is proved globally consistent. A set of experiments was performed on O*-SCDMST and two other algorithms, O*-Dijkstra and the exhaustive search that computes all possible traversal combinations, with sample problems generated from an online TSP data set [24].

Chapter 7: Optimal Sequence Traversal Query Processing in a Large Transportation Network. This paper introduces an O* algorithm, O*-SCDMST, that uses Semi-Connected Directed Minimum Spanning Trees (SCDMSTs) to obtain heuristics to efficiently process OSTQ using the network distance metric in a large transportation network. It first constructs a fully connected graph using Dijkstra algorithm [25] to obtain shortest distances between any two points from the given origin-destination pair and the points of interest. Thereafter, it uses the O*-SCDMST to retrieve the shortest route on the constructed fully connected graph. The performance of O*-SCDMST is studied in terms of average process time over different points of interest in a large dense urban transportation network.

Chapter 8: Conclusions and Recommendations. This chapter provides a summary of research results, possible applications of the developed algorithms in transportation, logistics, and computer science, and limitations of these algorithms. Recommendations for future research in transportation, logistics, and computer science have also been included.

REFERENCE:

1. F. Li, D. Cheng, M. Hadjieleftherious, G. Kollios and S. Teng. On Trip Planning Queries in Spatial Databases. Proceedings of 9th International Symposium on Spatial and Temporal Databases, 2005.
2. M. Sharifzadeh and C. Shahabi. Additively Weighted Voronoi Diagrams for the Optimal Sequenced Route Query. The 3rd Workshop on Spatio-Temporal Database Management, 2006.
3. M. O. Scholl, P. Rigaux, A. Voisard. Spatial Databases: With Application to GIS. Morgan Kaufmann, 2001.
4. S. Shekhar and S. Chawla. Spatial Databases: a Tour. Prentice Hall, 2003.
5. A. N. Papadopoulos, Y. Manolopoulos. Nearest Neighbor Search: a Database Perspective. Springer, 2005.
6. B. McQueen, J. McQueen. Intelligent Transportation Systems Architectures. Artech House Publishers, 2006.
7. A. Jagoe. Mobile Location Services: the Definitive Guide. Pearson Education, 2002.
8. Y. Zhao. Vehicle Location and Navigation Systems. Artech House Publishers, 1997.

9. G. Korte. The GIS Book. OnWord Press, 2000.
10. R. Burke, E. Napoleon, T. Ormsby. Getting to Know ArcGIS Desktop: The Basics of ArcView, ArcEditor, and ArcInfo Updated for ArcGIS 9. ESRI Press, 2004.
11. P. A. Longley, M. F. GoodChild, D. J. Maquire, D. W. Rhind. Geographic Information Systems and Science. Wiley, 2005.
12. P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 (2) (1968) 100–107,
13. R. E. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier Search. Journal of the Association for Computing Machinery. 52(5) (2005) 715-748.
14. S. Russell and P. Norvig. Artificial Intelligence: a Modern Approach (2nd edition). Prentice Hall, 2002
15. X. Ma, S. Shekhar, H. Xiong, P. Zhang. Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries. Proceedings of 14th International Symposium on Advances in Geographic Information Systems, 2006.
16. M. Kolahdouzan, M. Sharifzadeh and C. Shahabi. The Optimal Sequenced Route Query. University of Southern California, Computer Science Department, Technical Report 05-840, 2005
17. M. Sharifzadeh, C. Shahabi, Additively Weighted Voronoi Diagrams for the Optimal Sequenced Route Query, 3rd Workshop on Spatio-Temporal Database Management, pages 33-40, 2006.
18. H. Chen, W.S. Ku, M.T. Sun, R. Zimmermonn. The Multi-Rule Partial Sequenced Route Query. 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Irvine, CA, USA, 2008.
19. S. Shekhar and J. S. Yoo. Processing In-Route Nearest Neighbor Queries: a Comparison of Alternative Approaches. In GIS '03: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems, pages 9-16, New Orleans, LA, USA, 2003
20. D. Z. Du, K. I. Ko. Theory of Computational Complexity. John Wiley & Sons, 2000.
21. J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In: Proceedings of the American Mathematical Society, Vol 7, No. 1, pp. 48–50, 1956.
22. F. P. Preparata, M. I. Shamos. Computational Geometry: An Introduction. Springer-Verlag, New York. 1985.
23. 130 City Problem (Churritz). ftp://ftp.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/ch130.tsp. Last retrieved on October 30, 2008.
24. Matteo Fischetti, Paolo Toth. A polyhedral approach to the asymmetric traveling salesman problem.Management Science, Vol 43, No. 11, pp. 1520-1536.
25. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press, 1997.

CHAPTER 2: LITERATURE REVIEW

This chapter provides a review of the state-of-the-art progress related to the work presented along with limitations to existing approaches that are addressed in this dissertation. This research focuses on efficiently processing multi-category based queries to obtain optimal or suboptimal solutions in a graph with best first searches and on investigating their applications and performances in transportation trip planning. The underlying concepts that were necessary to the development of the research are multi-category based queries from computer science, operations research, and transportation engineering and best first searches primarily from artificial intelligence.

2.1 MULTI-CATEGORY BASED QUERIES

This section provides a review of the existing multi-category based queries and the corresponding search algorithms. Each query is discussed and its limitations are identified.

2.1.1 Trip Planning Query (TPQ) and Traveling Salesman Problem (TSP)

TPQ is a query conducted to find the route of minimum length starting from a given source location, passing through a set of categories, with one selection from each category, and ending at a destination. From the literature, the only reference that addressed TPQ was in a geospatial database structure [1]. No optimal solutions were identified for TPQ processing in a graph.

Within geospatial databases, F. Li *et al.* proposed four approximate algorithms, of which only two are appropriate for large data sets that typically represent large dense urban transportation networks and that must be stored on the hard disk instead of in main memory [1]. Of these two algorithms, one is nearest neighbor based (NNB). Starting from the origin, it iteratively visits the nearest neighbor of the last vertex added to the trip from all vertices in the categories not visited yet. The second algorithm is a minimum distance algorithm (MDA). Basically, it picks up a set of vertices, one vertex per category. Each vertex includes the smallest sum of the distance to the origin and destination among

all vertices from the same category. After these vertices are identified, the nearest neighbor algorithm is applied to form the trip. This minimum distance algorithm created a much better approximation bound than the nearest neighbor algorithm. However, these algorithms are only evaluated in a spatial database, not generalized to obtain sub-optimal solutions for TPQ in a graph.

As a special case of TPQ, the TSP has been a focus of research over the last thirty years. It searches for a shortest round-trip route to traverse each point of interest exactly once. Many solutions have been proposed [2] [3] [4] [5] [6]. The earliest research in TSP is in Euclidean space. A set of solutions were proposed to resolve this problem, either exactly or approximately, and the result is a Hamiltonian cycle that visits each vertex exactly once and returns to the starting vertex [2] [3] [4].

Among exact solutions, the most direct approach is to calculate all permutations and select the shortest. If the number of points of interest is n, then this solution rapidly becomes impractical because the number of permutations is n!. Using the techniques of dynamic programming, one can resolve the problem in time $O(n^2 2^n)$ [7], where O is the symbol used to capture the time complexity in computer science. Another approach is branch-and-bound consisting of a systematic enumeration of all possible solutions, where a large portion are discarded on the fly, by using the upper bound and lower estimated bounds [8].

Since TSP is NP hard, a set of sub-optimal solutions with high probability of obtaining good solutions were proposed. A constructive approximate algorithm using a minimum spanning tree (MST) [9], which has the minimum total edge cost among all spanning trees that connect all the vertices together in a graph, was proposed to find, at most, 1.5 times the shortest tour for the problem obeying the triangle inequality [4]. The minimum spanning tree was obtained through Prim's algorithm [10]. The nearest neighbor (NN) algorithm lets the salesman start from any point and choose the nearest remaining point of interest as the next to visit [11]. A set of algorithms, such as 2-Opt, 3-Opt, and n-Opt, iteratively improve the performance by removing two or more edges and replacing these

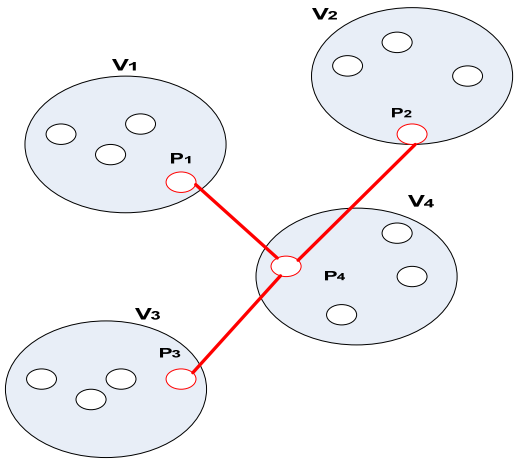
with the same number of different edges to reconnect the fragments generated by the deleted edges to obtain a reduced cost tour [12] [13]. An Ant Colony System (ACS) was presented to simulate an ant colony to generate good solutions to a TSP [14]. Simulated annealing replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter that is gradually decreased during the process. Its major concept is to reduce the possibility of staying at local minima [15]. Tabu search further prevents the search from revisiting a local minimum several times through a tabu list that remembers the visited local minima [16].

Since the solution to a TSP is a cycle, many algorithms discussed above use that characteristic to expedite the search process, which means they may not be applicable to process more general category-based queries where origin and destination are not the same.

2.1.2 Generalized Minimum Spanning Tree (GMST) problem

The GMST [17] [18] is a generalized version of the MST problem where the vertices are in different categories. In GMST, all objects in each category are clustered. Figure 1 shows an example of GMST and a solution to it, i.e., a minimum spanning tree connecting all nodes in red, where there are four clusters and each cluster represents one category. V_i is the ith category and p_i is the ith point in a category. From the fields of operations research and economics, methods including integer linear programming formulations [18] and variants of Tabu Searches (TS) [17], which uses a local or neighborhood search process to iteratively move from a solution to its neighborhood solution until some terminator criterion is achieved, were provided to resolve this problem. However, no detailed analysis was provided on the approximation bounds. The difference between GMST and TPQ is that GMST assumes every category of objects is clustered, while TPQ does not presume the distribution of points within a category of interest.

No best first algorithms were identified to process GMST in a graph.



Where

V_i is the ith category,

p_i is the ith point in a category, and

$i \in \{1,2,3,4\}$

Figure 1: An example of GMST and one solution

2.1.3 Multi-Type Nearest Neighbor (MTNN) query

MTNN query was first identified by M. Ma, etc. in 2006 [19]. The PLUB algorithm was proposed to provide an efficient computation in clustered data sets within a 2D space using Euclidean distance and to find optimal query results. No MTNN queries have been studied in a network or a graph.

2.1.4 Optimal Sequenced Route (OSR) query

OSR query was first introduced in 2005 [20]. The RLORD algorithm was provided to find an optimal solution within a 2D space using Euclidean distance [20]. It is a threshold-based iterative algorithm accompanied by an indexing technique to prune the candidate locations with various thresholds. Another use of OSR from the same authors focused on OSR query processing in a Manhattan distance based network where all links are straight and directions are either orthogonal or parallel [21]. This algorithm can not be practically applied to a general graph that conceptually represents diverse networks including transportation networks, computer networks, and so on. No other optimal solutions were identified to process OSTQ in a graph.

2.1.5 Multi-Rule Partial Sequenced Route (MRPSR) query

A MRPSR query finds a minimum-cost path that is based on a number of traveling rules (or constraints) that are expressed as sub-sequences of locations [22]. It subsumes TPQ and OSR in the sense that the order of the category can either be predefined or not in the traveling rules. It may not be a TPQ or an OSR when visit orders are only defined on a portion of the categories of interest. An algorithm that combines A* (see Section 2.2) and nearest neighbor search was proposed [22] to obtain sub-optimal solutions in a graph obeying triangle inequality. Other sub-optimal solutions including nearest neighbor search, a simple A*-variation search, were also provided. However, no optimal solution has ever been identified.

2.1.6 In-Route Nearest Neighbor (IRNN) query

An IRNN query [23] asks for a minimum cost path that starts from an in-route location, traverses a point of interest from a specified category, and ends at a given destination. It includes only one category, and is a special case of dynamic multi-category traversal problem in the spatial database domain where the category number is one. It uses nearest neighbor searches, indexing, and/or Euclidean distances to approach the optimal solution in network distance. It does not consider the traversing of multiple categories of interest.

2.1.7 Summary of Multi-Category Based Query

The research presented in this dissertation is multi-category based queries. None of the existing queries provided the necessary formulations to support this type of query as summarized in Table 1.

Table 1: Summary of the state-of-the-art category-based query processing and the research presented in the dissertation

Related work	**Differences from the proposed research**
Traveling Salesman's Problem (TSP)	1) The origin and the destination is the same; 2) A special case of OSTQ
Generalized Minimum Spanning Tree (GMST)	1) Clustered vertices are within one category; 2) No best first searches are available
Optimal Sequenced Route (OSR)	1) Algorithms are designed for queries in a spatial database; 2) No best first searches are available
Multi-Type Nearest Neighbor (MTNN) Queries	1) Existing algorithm was proposed to process queries with Euclidean distance instead of network distance; 2) No best first searches are available
Trip Planning Query (TPQ)	1) Only approximate solutions are provided; 2) No best first searches are available
Multi-Rule Partial Sequence Route (MRPSR) query	1) Only approximate solutions are provided; 2) No optimal solutions with best first searches
In-Route Nearest Neighbor (IRNN) query	1) Only one category of interest; 2) No consideration of multiple categories

2.2 BEST FIRST SEARCHES

A best first search is a type of informed search that searches a graph by expanding the most promising vertex chosen according to some rule. Existing best first searches adopt estimates to the promise of vertex n by a heuristic evaluation function f(n) that incorporates the properties of n and additional constraints including the description of the goal, the information gathered by the search up to that point, and any extra knowledge about the problem domain [24], which is also used by many researchers, including Russell & Norvig [25].

There are a number of best-first algorithms including A* [25][26], Dijkstra search [27], Greedy search [25], frontier search [28], and so on, that extract the path of minimum cost between a predefined origin-destination vertex pair in a graph. A* uses a distance-plus-cost heuristic function as *f(n)* to determine the order in which the search visits vertices in the graph [24]. Equation 1 provides one way to calculate f(n).

$$f(n)=g(n)+h(n) \quad (1)$$

where

g(n) is the path cost function of the path from the origin to the current vertex n, and

h(n) is the heuristic estimate of the distance from the current vertex n to the goal.

For *h(n)*, two important concepts exist. The first is admissibility. A heuristic is admissible if its value is less than or equal to the actual cost [25]. The other is consistency. A heuristic is consistent when the real cost of the path from any vertex A to any vertex B is always larger than or equal to the reduction in the heuristic [26]. Once a heuristic is consistent, it is always admissible [25]. A* has been shown to obtain the optimal solution, i.e., the minimum-cost solution, whenever the heuristic is admissible [25], and is optimally efficient among all best-first search algorithms guided by path-dependent evaluation functions when the heuristic is consistent [29]. Additionally, A* is complete in the sense that it will always find a solution if one exists. The space complexity, a term used in computer science to describe the complexity in terms of main memory usage [30], and time complexity of A* depend on the heuristic and, in general, is exponential. However, A* is very fast in practice.

Both Dijkstra and the Greedy algorithm can be considered as special cases of A*. The Dijkstra algorithm only considers *g(n)* as *f(n)*. The Greedy algorithm only uses *h(n)* as *f(n)*. The frontier search is similar to A* except that the frontier search only works on data sets with a consistent heuristic and does not require a closed-list to implement the search algorithm, which consequently saves space at the cost of increased computation [28].

Several A* variations exist such as anytime A* [31], hierarchical A* [32], MA* [33], and SMA*[34], which take the same form f(n) as A* but adapts A* to different scenarios to reduce time or space complexity of A*. However, none of these algorithms allows a vertex to occur multiple times along an optimal or suboptimal path, which is not necessarily true for a path that traverses multiple vertices of interest in a graph. Consequently, these algorithms can not be directly applied to resolve multi-category

based queries in a graph, which inspired the research to develop advanced best first searches to process multi-category based queries.

REFERENCE:

1. F. Li, D. Cheng, M. Hadjieleftherious, G. Kollios and S. Teng. On Trip Planning Queries in Spatial Databases. Proceedings of 9th International Symposium on Spatial and Temporal Databases, 2005.
2. S. Arora. Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems. Proceedings of 37th IEEE Symposium on Foundations of Computer Science, page 2, 1996.
3. S. Arora. Approximation Schemes for NP-hard Geometric Optimization Problems: A survey. Mathematical Programming, 2003.
4. N. Christofides. Worst-case Analysis of a New Heuristic for the Traveling Salesman Problem. Carnegie Mellon University, Computer Science Dept, Tech Technical report, 1976.
5. D. L. Applegate, R. M. Bixby, V. Chvátal. The Traveling Salesman Problem. Cook, 2006.
6. A. Dumitrescu and J. S. B. Mitchell. Approximation Algorithms for TSP with Neighborhoods in the Plane. Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms, pages 38–46, 2001.
7. M. Held and R. M. Karp. A Dynamic Programming Approach to Sequencing Problems, Journal of the Society for Industrial and Applied Mathematics 10(1) (1962): 196–210.
8. D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. The Traveling Salesman Problem: A Computational Study. Springer, 2007.
9. T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press, 1997.
10. R. C. Prim. Shortest connection networks and some generalisations. In: Bell System Technical Journal, 36 (1957), pp. 1389–1401.
11. D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II. An Analysis of Several Heuristics for the Traveling Salesman Problem. SIAM Journal on Computing. 6 (1977): 563–581.
12. D. S. Johnson. Local Optimization and the Traveling Salesman Problem. In: G. Goos & J. Hartmanis (eds) Automata, Languages and Programming, Lecture Notes in Computer Science 442, Springer, Heidelberg, 1990,446-461.
13. J. L. Bentley. Fast Algorithms for Geometric Traveling Salesman Problems. ORSA Journal on Computing 4, (1992),387-411.
14. Ma. Dorigo. Ant Colonies for the Traveling Salesman Problem. IRIDIA, Université Libre de Bruxelles. IEEE Transactions on Evolutionary Computation, 1(1) (1997):53–66.
15. E.H.L. Aarts, and J. Korst. Simulated Annealing and Boltzmann Machines: A stochastic Approach to Combinatorial Optimization and Neural Computing. John Wiley & Sons, Chichester, 1989.
16. F. Glover. Tabu Search. ORSA Journal on Computing 1, 190-206 (Part I), ORSA Journal on Computing 2, (1989), 4-32 (Part II).
17. B. Hu, M. Leitner, and G. R. Raidl. Computing Generalized Minimum Spanning Trees with Variable Neighborhood Search. In P. Hansen, N. Mladenovic, J. A. M. Pérez, B. M. Batista, and J. M. Moreno-Vega, editors, Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, Tenerife, Spain, 2005.
18. Y. S. Myung, C. H. Lee, and D. W. Tcha. On the Generalized Minimum Spanning Tree Problem. Networks, 26:231–241, 1995.
19. X. Ma, S. Shekhar, H. Xiong, P. Zhang. Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries. Proceedings of 14th International Symposium on Advances in Geographic Information Systems, 2006.
20. M. Kolahdouzan, M. Sharifzadeh and C. Shahabi. The Optimal Sequenced Route Query. University of Southern California, Computer Science Department, Technical Report 05-840, 2005

21. M. Sharifzadeh, C. Shahabi, Additively Weighted Voronoi Diagrams for the Optimal Sequenced Route Query, 3rd Workshop on Spatio-Temporal Database Management, pages 33-40, 2006.
22. H. Chen, W.S. Ku, M.T. Sun, R. Zimmermonn. The Multi-Rule Partial Sequenced Route Query. 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Irvine, CA, USA, 2008.
23. S. Shekhar and J. S. Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In GIS '03: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems, pages 9-16, New Orleans, LA, USA, 2003.
24. Judea Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.
25. S. Russell and P. Norvig. Artificial Intelligence: a Modern Approach (2nd edition). Prentice Hall, 2002.
26. P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 (2) (1968) 100–107
27. E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. In Numerische Mathematik, 1 (1959), S. 269–271
28. R. E. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier Search. Journal of the Association for Computing Machinery. 52(5) (2005) 715-748
29. R. Dechter and J. Pearl. Generalized Best-first Search Strategies and the Optimality of A*. Journal of the Association for Computing Machinery. 32(3) (1985) 505-536
30. D. Z. Du, K. I. Ko. Theory of Computational Complexity. John Wiley & Sons, 2000.
31. M. Likhachev, G. J. Gordon, S. Thrun. Ara*: Anytime A* with provable bounds on sub-optimality. In. Advances in Neural Information Processing Systems 16. MIT. Press, Cambridge, MA, 2004.
32. A. Botea, M. Muller, J. Schaeffer. Near Optimal Hierarchical Path-Finding. In Journal of Game Development, volume 1, issue 1, (2004), 7-28.
33. S. Russell. Efficient Memory-bounded Search Methods. In Proceedings of Tenth European Conference on Artificial Intelligence (ECAI-92), pages 1--5. Chichester, England: Wiley, 1992.
34. R. Zhou, Eric A. Hansen. Memory-Bounded A* Graph Search. Proceedings of the Fifteenth International Florida Artificial Intelligence Research. May 2002.

CHAPTER 3: C*: A BIVARIATE BEST FIRST SEARCH ALGORITHM TO PROCESS CATEGORY SEQUENCE TRAVERSAL QUERIES IN A GRAPH

by

Qifeng Lu

Kathleen Hancock

C*: A Bivariate Best First Search to Process Category Sequence Traversal Queries in a General Graph

Qifeng Lu, Kathleen Hancock

Civil and Environmental Engineering Department, Virginia Polytechnic Institute and State University, Alexandria, VA 22314, USA

Abstract

A Category Sequence Traversal Query (CSTQ) is a new type of category based query that determines the minimum-cost path with a predefined origin-destination pair, traversing at least a single selection from each of a set of categories in a specified order. This paper 1) generalizes existing single-variate best first searches to multivariate best first searches; 2) proposes a multivariate best first search algorithm, L#; 3) introduces a set of new concepts including global heuristic, global admissibility, global consistency, local heuristic, local admissibility, local consistency, and state graph space into multivariate best first searches; 4) provides C*, a bivariate best-first-search instance of L#, to process CSTQ in a graph, which extends the existing single-variate best first search to the bivariate best first search in the sense that the evaluation function used in C* is defined upon a bivariate state instead of a single-variate vertex; 5) discusses how to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate an optimal efficiency property in a discrete sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the first *m*, a discrete integer variable, categories of interest in the given order; and 6) adopts a bottom-up approach to provide a global heuristic, *g-p*, whose properties of optimality and optimal efficiency are analyzed based on local heuristics. Since a minimum-cost path query for a given origin and destination pair is a special case of CSTQ, A*, the single-variate best first graph search algorithm, is a special case of C*. CSTQ has distinct applications in transportation trip planning and other potential applications whenever a workflow exists to traverse a set of categories of interest. Two special cases of C*, C*-P that uses *g-p* as the global heuristic, and C*-Dijkstra, are identified, and their performances are studied for trip planning in two transportation networks of different scales.

Keywords: Bivariate, Best First Search, State, Heuristic, Category Sequence Traversal, OSR, L#, C*, A*

1. Introduction and background

As a set of minimum-cost route queries in computer science, geographic information systems, and transportation, multi-category based queries have begun to attract research attention over recent years. These queries include Trip Planning Query (TPQ) [1], Multi-Type Nearest Neighbor (MTNN) query [2], and Optimal Sequenced Route (OSR) query [3] [4]. All these multi-category based queries require that the route traverse multiple points from diverse categories.

A graph conceptually represents a complex network, such as a transportation network or a computer network, as well as other types of networks. In this context, a graph G is defined as a set of vertices, $\{V_i\}$, and a set of directed line segments, $\{E_{i,j}\}$, called arcs. $E_{i,j}$ is defined such that an arc is from vertex V_i to vertex V_j, and V_j is a successor of V_i. Each $E_{i,j}$ has an associated cost $C_{i,j}$. Only graphs with $C_{i,j}>=0$ are considered in this paper. These graphs are named as δ graphs.

In this paper, we propose a new type of category based query defined in such a δ graph: Category Sequence Traversal Query (CSTQ). Given a δ graph G with its vertices $\{V_i\}$ and edges $\{E_{ij}\}$, a set of vertices of interest VI from $\{V_i\}$ in graph G, where each vertex of interest belongs to

a specific category *C*, a starting vertex *S,* a destination vertex *D*, and a set of *normal vertices* in *G* are neither vertices of interest to traverse, nor the given origin or destination, a CSTQ retrieves the path of minimum-cost, traversing at least a single selection of each of a set of categories in a given order. CSTQ has distinct applications in transportation trip planning. For a traveler that is unfamiliar to a metropolitan area, along a route from a given origin to a given destination, he/she may not be able to specify the locations of a set of consumer destinations of interest, but certainly he/she knows what the categories of these locations are. Whenever traversal priorities are imposed upon these given categories, the trip planning problem in a transportation network corresponds to a CSTQ processing in a graph. For example, a traveler may start from a conference hall, pass a gas station, a supermarket, and a restaurant in order, and end at a hotel.

CSTQ may have potential applications in other domains whenever a set of categories of interest is required to be traversed in order.

This paper 1) generalizes existing single-variate best first searches to multivariate best first searches; 2) proposes a multivariate best first search algorithm, L#; 3) introduces a set of new concepts including global heuristic, global admissibility, global consistency, local heuristic, local admissibility, local consistency, and state graph space into best first searches; 4) provides C*, a bivariate best-first-search instance of L#, to process CSTQ in a graph, which extends the existing single-variate best first search to the bivariate best first search in the sense that the evaluation function used in C* is defined upon a bivariate state instead of a single-variate vertex; 5) discusses how to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate an optimal efficiency property in a discrete sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the first *m*, a discrete integer variable, categories of interest in the given order ; 6) adopts a bottom-up approach to provide a global heuristic, *g-p*, whose properties of optimality and optimal efficiency are analyzed based on local heuristics; and 7) identifies two special cases of C*, C*-P that uses *g-p* as the global heuristic, and C*-Dijkstra, and studies their performances in two transportation networks of different scales.

The paper is organized as follows. First, related work is discussed in Section 2. In section 3, the paper generalizes the existing best first searches and proposes L#, a multivariate best first search. This is followed in Section 4 by a discussion of C*, the bivariate best-first search algorithm to process CSTQ in a graph. Section 5 provides C*-P, a special case of C*, that uses *g-p* as the global heuristic. Section 6 presents six sets of experiments and result analyses. Section 7 describes the global heuristic used in C*. Section 8 discusses the relationship between C* and A*. Finally, the conclusion is presented.

2. Related work

Traveling salesman problem (TSP) is a query conducted to find the route of minimum impedance starting from a given source location, passing through a set of points of interest, and ending at a destination. The TSP has been a focus of research over the last thirty years. Many solutions have been proposed [5] [6] [7] [8] [9], either in a spatial database or a graph, but these solutions can not be applied directly to CSTQ because the problems are fundamentally different [1], i.e., in TSP, locations that must be traversed are given and no concept of category is involved.

This section provides a review of the state-of-the-art research on 1) multi-category based queries, and 2) best first searches.

2.1. Multi-category based queries

This section provides review of TPQ, MTNN query, OSR query, the generalized minimum spanning tree (GMST), and In-Route Nearest Neighbor (IRNN) query [10]. Their relationships to CSTQ are discussed, respectively.

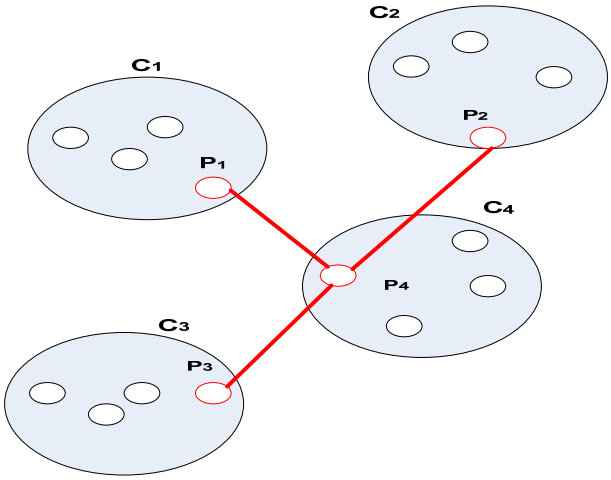
Only Li et al. recently addressed TPQ [1]. It proposed four approximate algorithms, of which two are based on integer linear programming and the others on nearest neighbor search and minimum distance search designed for large data sets that must be stored on the hard disk instead of in main memory. The two algorithms using integer linear programming transform the problem into an optimization problem with constraints to achieve approximation. Of the two algorithms proposed for large data sets stored in a network database, one algorithm is nearest neighbor based (NNB). Starting from the origin, it iteratively visits the nearest neighbor of the last vertex added to the trip from all vertices in the categories not visited yet. The other algorithm is called a minimum distance algorithm (MDA). Basically, it picks up a set of vertices, one vertex per category. Each vertex includes the smallest sum of the distance to the origin and destination among all vertices from the same category. After these vertices are identified, the nearest neighbor algorithm is applied to form the trip. This minimum distance algorithm created a much better approximation bound than the nearest neighbor algorithm. TPQ is much more computationally complex than CSTQ since the former requires consideration of all possible combinations of category visit orders.

MTNN query was first identified by M. Ma, etc. in 2006 [2]. The Page-Level Upper Bound (PLUB) algorithm was proposed to provide an efficient computation in clustered data sets within a 2D Euclidean space using Euclidean distance and find optimal query results. The PLUB algorithm works on data sets where data from the same category cluster. No MTNN queries have been studied in a network or graph.

OSR query was first introduced in 2005 within the spatial database domain [3]. The RLORD algorithm was proposed to find an optimal solution within both a metric space and a vector space [3][4]. It follows a filtering/refinement process. It is a threshold-based iterative algorithm accompanied with an indexing technique to prune the candidate locations with various thresholds. Another use of OSR from the same authors focused on OSR query processing in a Manhattan-distance based network where all road links are straight and their directions are either orthogonal or parallel [11]. A CSTQ can be regarded as a special case of OSR queries because the destination itself in a CSTQ can be regarded as an additional category to visit. However, in nature, objects in visit categories or the destination can be either dynamic or static and they are not necessarily consistent. From this perspective, a CSTQ should not be considered as a special case of OSR queries. Furthermore, the proposed solution to an OSR query is limited to spatial databases and is not generalized in a graph.

The GMST is a generalized version of the Minimum Spanning Tree (MST) problem [8] where the vertices are in different categories [12] [13]. In GMST, all objects in each category are clustered [11] [13]. Figure 1 shows an example of GMST with a corresponding solution, i.e., a minimum spanning tree connecting all vertices in red, where there are four clusters and each cluster represents one category. C_i is the ith category and p_i is the ith point in a category. From the fields of operations research and economics, approximation methods including integer linear programming formulations [13] and variants of Tabu Searches (TS) [12], which uses a local or neighborhood search process to iteratively move from a solution to its neighborhood solution until some stopping criterion is achieved, were provided to resolve this problem. However, no detailed analysis was provided on the approximation bounds, nor were optimal solutions provided.

The differences between GMST and CSTQ are: 1) GMST assumes every category of objects is clustered, while CSTQ does not; and 2) CSTQ defines the visit order while GMST does not.



Where C_i is the ith category, p_i is the ith point in a category, and $i \in \{1,2,3,4\}$

Figure 1: An example of GMST and one solution

An IRNN query [10] asks for a minimum cost path that starts from an in-route location, traverses a point of interest from a specified category, and ends at a given destination. It only provides one category. It is kind of a special case of dynamic multi-category traversal problem where the category number is only one.

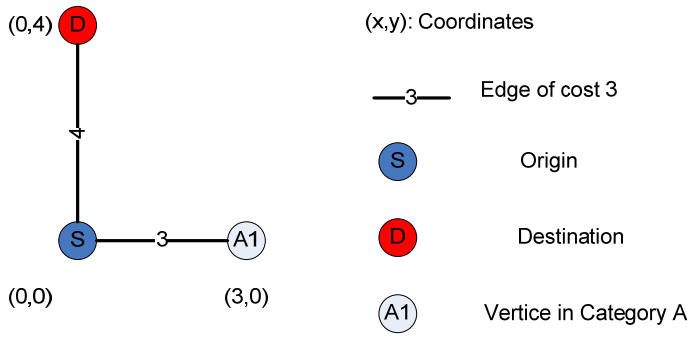
2.2. Best first searches

A best first search is a kind of informed search which searches a graph by expanding the most promising vertex according to some rule. The existing best first searches adopt estimates to the promise of vertex n by a "heuristic evaluation function f(n) which, in general, may depend on the description of n, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." [14], which is prevalently used by researchers in Artificial Intelligence (AI), including Russell & Norvig [15].

There are a number of best-first algorithms including A* [15][16], Dijkstra search [17], Greedy search [15], frontier search [18], and so on, that extract the path of minimum cost between a predefined origin-destination vertex pair in a graph. A* uses a distance-plus-cost heuristic function as *f(n)* to determine the order in which the search visits vertices in the graph [14]. *f(n)* is the sum of two functions: *g(n)*, the path cost function of the path from the origin to the current vertex *n*, and *h(n)*, the heuristic estimate of the distance from the current vertex *n* to the goal. For *h(n)*, two important concepts exist. The first is *admissibility*. A heuristic is *admissible* if its value is less than or equal to the actual cost [15]. The other is *consistency*. A heuristic is *consistent* when the real cost of the path from any vertex *A* to any vertex *B* is always larger than or equal to the reduction in heuristic [16]. Once a heuristic is consistent, it is always admissible [15]. A* has been shown to obtain the optimal solution, i.e., the minimum-cost solution, whenever the heuristic is admissible [15], and, indeed, is optimally efficient among all best-first algorithms guided by path-dependent evaluation functions when the heuristic is consistent [19]. Additionally, A* is complete in the sense that it will always find a solution if one exists. The spatial and time complexity of A* depends on the heuristic and, in general, is exponential. However, A* is very fast in practice. Both Dijkstra and the Greedy algorithm can be considered as a special case of A*. Dijkstra algorithm only considers *g(n)* as *f(n)*. The Greedy algorithm only uses *h(n)* as *f(n)*. Frontier search is similar to A* except that Frontier search only works on data sets with consistent heuristic and does not require a closed-list to implement the search algorithm, which consequently saves space at the cost of increased computation [18].

There are also a set of A* variations such as anytime A* [20], hierarchical A* [21], MA* [22], and SMA*[23], which takes the same form *f(n)* as of A* but adapts A* to different scenarios to reduce time or space complexity of A*.

All the *f(n)*s used in these best first searches are defined upon one variable n that describes a vertex and may represent a vector or a scalar, to estimate its promise. For CSTQ processing in a graph, however, a vertex may have multiple states along an optimal path, and thus these single-variate best first search algorithms may not be appropriate to process CSTQ since only one estimate, *f(n),* is kept for a vertex *n*. As an example, Figure 2 shows that due to this deficiency in state specification, the single-variate A* can not retrieve a solution for a CSTQ even though the optimal solution exists.



The CSTQ asks for a minimum-cost path starting from S, traversing a vertex in category A, and ending at D. Assume the Euclidean distance, ED, is used to calculate h(n) in A*. According to A*, at the beginning, A* with f(A)=ED(S,A1)+ED(A1,D)=8 is expanded, and A1 and D are generated. f(A1)=g(S->A1)+ED(A1,D)=8, and f(D)=g(S->D)+ED(D,A1)+ED(A1,D)=14. So A1 is expanded next, and S as its only child is generated. Now f(S)=g(S->A1->S)+ED(S,D)=10. Since vertex S with f(S)=8 is on the closed list, according to A*,this new state of f(S)=10 for vertex S is discarded. Next, only D is in the open list. However, after D is expanded, due to the same reason, the new generated state for S is discarded. Now, the open list is empty, so the search stops and A* reports no solution is found. However, it is clear that there is an optimal path: S->A1->S->D. This example shows that due to the limitation of the state specification in A*, it incorrectly discarded the state generated from A1.

Figure 2: An example of the deficiency of the state specification in A*

3. L#: A generalized best first search

A vertex in a graph may have multiple states. In this paper, we propose a generalized best first search. Instead of evaluating the promise of a vertex to be expanded, the proposed generalized best first search evaluates the promise of a state of a vertex to be expanded. A state *s* is defined upon a vertex *v* and specified in each individual problem. It defines the condition or the status of the vertex *v*, and also describes the condition of the partial path from the given origin to *v*.

For a state *s*, a best first search adopts the estimate to the promise of *s* to be expanded by a heuristic evaluation function *f(s)* which may depend on the description of *s*, the description of the goal state, the information gathered by the search up to that state, and any extra knowledge about the problem domain. A best first search is N-variate if it uses N variables to describe its state. To be consistent with the existing definition of best first searches, one variable that represents a vertex must be included to describe *s*. Furthermore, only this variable may be a vector that

represents a multi-dimensional vertex, and any of the other variables in L# is one dimensional, only representing a scalar.

The lower the *f(s)*, the higher the priority is for a state to be expanded.

A best first search, named as L#, is proposed to evaluate the promise of *s* according to the following function:

$$f(s)=g(s)+h(s) \quad (1)$$

where

f(s) is the estimate to the promise of a state *s* to be expanded,

g(s) is the cost from the origin state to the state *s*, and

h(s) is the estimate to the actual cost from the state *s* to the final state.

The way to calculate *g(s)* entirely depends on the specification of *s*, which can be defined in any meaningful possible way to resolve a specific problem. For category based query processing, if a variable, *Obj*, is used to represent a set of traversed objects of interest, and the origin vertex is V_0, then the origin value for *Obj* is null since no object has been traversed at the beginning, and at a state *S (Obj,V)* with *Obj*=[O_1,O_2,…,O_i,…,O_k] where O_i ($1=<i<=k$) is the *i*th sequentially traversed object. In this case, *g(s)* is the total cost of the path *P*

(V_0->O_1->O_2->…,->O_i->,…,->O_k->*V*).

When an object represents a category, it uses the traversed vertex in that category to represent the category and to calculate *g(s)* of a path from the origin state to *s.*

A state may be specified through more than one variable. L# is said to be *multivariate* if it uses multiple variables to describe its state.

If only a vertex identification is used to specify a state, then the proposed best first search becomes the existing single-variate best first search, and L# becomes A*.

For L#, we introduce a new set of concepts including *local heuristic*, *global heuristic*, *local admissibility, global admissibility, local consistency, global consistency,* and *state graph*.

A *local heuristic* refers to the one between a vertex and a *subgoal*, a vertex of interest in the problem domain. A *global heuristic* is the estimate to the actual cost from the current state to the goal state. In equation (1), *h(s)* is the global heuristic for state *s*.

For local heuristics and global heuristics, we define their corresponding admissible concepts: *local admissibility* and *global admissibility*. *Local admissibility* means the local heuristic is admissible. In other words, the local heuristic never overestimates the actual cost of a path between a vertex and a subgoal. *Global admissibility* means the global heuristic is admissible, i.e., it never overestimates the actual cost of a path from the current state to the goal state.

Corresponding to local heuristics and global heuristics, two consistency concepts are defined: *local consistency* and *global consistency.* *Local consistency* means the local heuristic is consistent, and its definition and characteristics are the same as in A* except that its local target, the subgoal, must be specified. Based on the consistency definition in A* [16], for a given subgoal *sg*, for any vertex *n* and *n'*, the following formulation defines local consistency:

$$h(n,sg)<=g^*(n,n')+h(n',sg) \quad (2)$$

where *g*(n,n')* is the actual cost from *n* to *n'*.

Global consistency means the global heuristic is consistent. Suppose an optimal path is from state *s* to state *s',* a global heuristic is *globally consistent* once it satisfies the following triangular inequality:

$$h(s)<=g^*(s,s')+h(s') \quad (3)$$

Where *g** is the actual (minimum) cost from *s* to *s*'.

Since states are defined upon vertices, one variable, *V*, must be used to describe the vertex. Assume all the other variables compose a variable set, *SV*. A *state graph* consists of vertices of the same values of *SV* and their incoming and outgoing edges. The number of state graphs for a problem equals the number of all possible states for a vertex.

Different from A*, L# can not be directly applied to each individual problem. Instead, the variables except the variable representing a vertex must be specified or instantiated in each individual problem.

4. C*: a bivariate best-first-search approach to process CSTQ in a graph

This section describes the details of C*, the algorithm applying best-first strategies to process CSTQs. First, for a state *s*, C* uses the distance-plus-cost function *f(s)* defined upon a state *s* as shown in equation (1) to determine the order in which the search expands states in the graph. Therefore, C* is a special case of L#. Second, C* assures that its solution satisfies the category constraint (CC): the obtained path must traverse a single selection of each of a set of categories in a given order. Consequently, the heuristic function *h(s)* has to be estimated per *object* in the next visit category. In this scenario, an *object* refers to a vertex that is a member of a category in the graph.

In the following subsections, first, a set of background definitions in C* is provided. Then the C* algorithm is presented. Next, how C* satisfies the CC and the visit order requirement is discussed, followed by the analysis of its completeness. Thereafter, the optimality and optimal efficiency for the bivariate best first search are examined and evaluated. At last, the relationship between different states of a vertex is discussed.

4.1. Definition

In C*, since a candidate path traverses multiple categories, any vertex in the graph could occur in the path multiple times. One additional parameter, *VisitOrder*, indicating the order of the next visit category along the path, is used to help determine the state of the vertex. Consequently, a state $S_{i,j}$ is defined as (V_i,VO_j), where VO_j is the order of the next category to be visited for the partial path obtained so far at vertex V_i. A state of a vertex *n* describes the traversed categories along the partial path from the origin to the vertex *n*. A successor *operator* Γ is defined on $\{S_{i,j}\}$. Its value for each $S_{i,j}$ is a set of pairs $\{S_{k(i),j},C_{i,k(i)}\}$, where *k(i)* represents the *k*th child of vertex V_i. Whenever a $S_{i,j}$ is of a vertex in the next category to visit, a Ψ *operator* will transform the state $S_{i,j}$ to $S_{i,j+1}$, at no cost by incrementing VO_j by 1, i.e., $VO_{j+1}=VO_j+1$. A state graph *SG* defined on a graph *G* is the graph *G* whose vertices are of the same *VO*. The number of state graphs for *G* is the number of *VO*s. Since *VO* is the visit order changing from 1 to the number of total categories to visit plus 1, for a problem with n categories to visit, the number of state graphs is *n+1*. These state graphs compose the state graph space $\mathcal{G}$, a discrete state graph space. A sub state graph *SSG* is a portion of a state graph *SG*. It is said to be *implicitly* generated by Γ operations in *SG*. A sub state graph space G_s is a set of sub state graphs, i.e., *SSG*s. In G_s, each point represents a sub state graph which contains the expanded vertices to each of which the path from the origin has traversed the same set of categories of interest in the given order. Since the order to visit each category is given, a discrete integer variable that identifies the order of the next visit category of interest is enough to represent the traversed set of categories of interest, which is the reason that G_s is a discrete sub state graph space. It is said to be *implicitly* generated if it is initiated with a single source state $S_{0,0}$ and a set of Γ operations and some possible Ψ operations applied to it, to its successors, and so forth. A δ sub space graph space is a special G_S with edge cost always not smaller than 0. A path from a source *S* to a goal *D*, traversing a set of categories in a predefined order, is an ordered set of states (V_i,VO_j). In C*, all sub state graphs are generated implicitly.

In addition to the concepts as discussed in section 3, two new heuristics are proposed: *partial heuristic*, and *subgoal based global heuristic*. A *partial heuristic* refers to the smallest heuristic for a path starting from an object in the next category to visit, traversing the remaining categories, and ending at the goal. A *subgoal based global heuristic* refers to the heuristic for a path starting from a vertex, traversing a specified subgoal in the next category to visit and the other remaining categories, and ending at the goal.

Furthermore, in C*, the concepts proposed in L# are instantiated as follows.

Local heuristic is the estimate to the actual cost from the current vertex to a subgoal, *sg*. In C*, a subgoal is a vertex that is in a category of interest.

Local admissibility means the local heuristic is not larger than the actual cost from the current state to a subgoal of the same *VisitOrder*.

For vertex n and vertex n', *local consistency* means the following inequality exists:

$$hl(n,sg) <= g^*(n, n') + hl(n',sg) \quad (4)$$

where

hl is the local heuristic,

sg is the subgoal, and

g(n, ,n')* is the actual cost from *n* to *n'* in the graph.

Global heuristic is the estimate to the actual cost of the path from the current state to estimate to the final goal, traversing the remaining ordered categories of interest.

Global admissibility means the global heuristic is not larger than the actual cost from the current state to evaluate to the final goal state, traversing the remaining ordered categories of interest.

Suppose an optimal path is from state *n* with *VisitOrder vo* to state *n'* with *VisitOrder vo'*. By default, *vo'>=vo*. *Global consistency* satisfies the following triangular inequality:

$$hg(n,vo) <= g^*((n,vo),(n', vo')) + hg(n',vo') \quad (vo'>=vo) \quad (5)$$

Where *hg* is the global heuristic, and

*g** is the actual (minimum) cost from *(n,vo)* to *(n',vo')*.

4.2. Algorithm

In C*, a state is specified through a vertex identification and its *VisitOrder*, indicating the order of the next visit category along the path. It uses equation (1) to evaluate *f(s)*. Since its state is specified with *(n,vo)*,which is bivariate, C* is a bivariate best first search, and a special case of L#.

C* incrementally searches all paths leading from the starting vertex, traversing categories in a predefined-order, until it finds the path of minimum cost to the goal. It first takes the paths most likely to lead towards the goal.

Like A*, C* also maintains a set of partial solutions, unexpanded leaf states of expanded states, stored in an *open list*, also called a *priority queue*, a modified queue that outputs the one with the highest priority.

Whenever an equal *f(s)* occurs, the state with a larger *VisitOrder* will be the next to expand. Otherwise, one is randomly selected to expand.

For an N-category traversal problem, C* first generates the source state that contains the given vertex and *VisitOrder* equal to 1. Next, for all the states in the open list, the algorithm expands the state with the lowest *f(s)* value, and its children states are generated. Whenever a child state to be generated contains an object in the next category to visit, its *VisitOrder* is its parent's *VisitOrder* plus 1; otherwise, its *VisitOrder* inherits its parent's. The process continues until a goal state is reached or no solution is found. Once a goal state is reached, the algorithm will retrieve the obtained path using a data structure called *backpointer*, the combination of vertex identification and *VisitOrder*, to recursively obtain the parent until the origin state is reached.

The value of *VisitOrder* is larger for a category to be visited later. Since the order is predefined, the next category to visit is recognized once the order is known. This search process guarantees that the solution satisfies the CC.

4.2.1. C pseudo code*

Given the starting state *S*, the goal *D*, and the categories to visit and their visit order, the pseudo code for C* is provided in Figure 3. The pseudo code on global consistency is discussed in section 4.7.

```
Starting vertex S, Goal vertex D, which itself is considered as a category to visit
Priority queue PQ(=set of generated (s(vertex, VisitOrder), f(s), g(s))) begins empty.
 Back pointer backpointer(vertex, VisitOrder).
PathVertexList: the vertex list along the obtained path, begins as empty
Closed list CL (= set of previously expanded (s(vertex,VisitOrder),backpointer,g(s), f(s))) begins empty.
 NC2V //the next category to visit
Calculate f for (S,null)
Put S, f,g=0, and VisitOrder =1 into PQ
While PQ is not empty
{
    remove the state with the  lowest f(s) having the largest VisitOrder from PQ. Name it n. Obtain its NC2V
   //resolve the case where the subgoal is in multiple categories
    While(n is a subgoal) Increase VisitOrder by 1, and put (n.vertex VisitOrder-1),f, and backpointer into CL
    If n is a goal, then  //a goal must be the predefined destination after all categories are visited
        Succeed; Backtracking(n)
   put n with its f, g, backpointer in CL
   For each v' in successors(n.vertex)
   {
      if v' is a sub goal, then  // v is a member of NC2V
          Increament VisitOrder by 1;
      Calculate f for (v',VisitOrder);
      Process((v',VisitOrder),f,g) //decide to place the state in PQ and/or to remove other states from CL/PQ
   }
}
Process((v',VisitOrder), f,g):
        bAdd2PQ=true;
        If v' not seen before, or (v',VisitOrder) currently in PQ with f(v',VisitOrder)>f Then
            Place/promote (v', VisitOrder) on priority queue with f, and g; END;
        If(v',VisitOrder') is in PQ Then
            If(VisitOrder'>VisitOrder && g(v',VisitOrder')<g(v',VisitOrder)) Then
                bAdd2PQ=false;
            If(VisitOrder'<VisitOrder && g(v',VisitOrder')>g(v',VisitOrder)) Then
                Delete (v',VisitOrder') from PQ
        If(v',VisitOrder) previously expanded Then
            If(global consistency) Then
                bAdd2PQ=false;
            Else
                If f(v',VisitOrder)<=f Then
                    bAdd2PQ=false;
                Else
                    Delete (v',VisitOrder') from the closed list
        Else
            If(v',VisitOrder') is in CL Then
                If(VisitOrder'>VisitOrder && g(v',VisitOrder')<g(v',VisitOrder)) Then
                    bAdd2PQ=false;
                If(VisitOrder'<VisitOrder && g(v',VisitOrder')>g(v',VisitOrder)) Then
                    Delete (v',VisitOrder') from CL
        If(bAdd2PQ) Then Place (v', VisitOrder) on priority queue with f, g
Backtracking(n):
   BackPointer parent=n.BackPointer;
   PathVertexList.addHead(n.vertex);
   while(parent.state!=null)
       parent=backPointer(parent); PathVertexList.addHead(parent.vertex);
   Output PathVertexList;
```


Figure 3: The pseudo code for C*

4.2.2. Time and space complexity

In C*, the complexity between a vertex and a subgoal is the same as for A*, whose time complexity and space complexity are dependent on the heuristic. For A*, in general, the time and space complexities are both exponential. Assume b is the branching factor, d is the largest depth to obtain the shortest path, and then both the time complexity and the space complexity are $O(b^d)$ [15]. A* is sub-exponential only when its heuristic $h(n)$ and the actual cost $h^*(n)$ satisfies the following condition [15]:

$$|h(n)-h^*(n)|<=O(\log h^*(n)) \quad (6)$$

Where

$h(n)$: the heuristic in A*, i.e., the local heuristic in C*; and

$h^*(n)$: the corresponding actual cost

For C*, assume b is the branching factor, d is the largest depth to obtain the shortest path between any two objects in the categories next to each other and between the starting vertex and any object in the first category, which is named as a *section path* in C*, n is the number of categories to visit, and m is the maximum number of objects in a category, and F_h is the worst time complexity to obtain a global heuristic. In the worst case, C* requires traversal of all these section paths, and each section path is exponential in time and space complexity. Since d is the largest depth, the time complexity between two consecutive visit categories is $O(mF_hb^d)$ and the space complexity is $O(mb^d)$. Consequently, for the corresponding CSTQ, the time complexity is $O(mnF_hb^d)$ and space complexity is $O(mnb^d)$.

If equation (6) exists for each section path, then both time and space complexity become sub-exponential.

4.3. CC satisfaction

The solution from C* must satisfy the CC to be a candidate solution to a CSTQ.

Lemma 1: The solution from C* satisfies the category constraint.

Proof:

Use contradiction.

Assume the solution obtained from C* does not traverse at least one category, C.

Since the solution is obtained, then the search stops.

According to C*, if a subgoal in category C is not reached, then the visit order can not be increased, thus there is no way to indicate that the final state is expanded. Consequently, the search does not stop. This is contradicted with the fact that the search stops. So C* does provide a solution that satisfies the category constraint.

4.4. Order

The order must be kept in C* to guarantee the solution is a candidate path to a CSTQ.

Lemma 2: Category visit order is kept in C*.

This is true because the order is explicitly imposed during the search.

4.5. Completeness

Completeness means that an algorithm finds a solution if one exists.

Theorem 1: C* is complete.

The algorithm will not stop until either the goal is reached or there is no solution when the open list is empty while the final goal is not reached.

For the single-variate A*, a set of theorems on optimality and optimal efficiency was proposed [16]. However, these theorems are provided through state specification, which is single-variate in A*. Consequently, whether the results from these theorems are still valid in the bivariate best-first-search C* is questionable. For example, A* is proved to be optimally efficient when the

heuristic is consistent in an implicitly generated sub graph. However, in C*, the search process implicitly generates a set of discrete sub state graphs. Is that C* is optimally efficient when the global heuristic is globally consistent valid in a discrete sub state graph space instead of in a sub graph? The following three sections re-examine and re-evaluate the properties on optimality and optimal efficiency in the bivariate best-first-search C*.

4.6. Admissibility and optimality

Admissibility is important in A* since it guarantees the solution is optimal. This is also true in C*.

Lemma 3: If any global heuristic for any vertex is globally admissible, then the solution is optimal.

Proof:

Use contradiction.

Given v as the number of visit categories, suppose C* finds a suboptimal path, ending in goal state G_1 with $v+1$ as the visit order since the search guarantees the solution traverses all the targeted categories in order, i.e., $f(G_1,v+1) > f^*$ where $f^* = hg^*$ *(origin state)* = cost of the optimal path. Let $hg^*(n,vo)$ as the actual cost from (n,vo) to the goal state.

There must exist a vertex n which is

- Unexpanded
- The path from start to n with a *VisitOrder vo* (stored in the *BackPointers* (n,vo) values) is the start of a true optimal path

$f(n,vo) >= f(G_1,v+1)$ (else search would not have ended)

Also $f(n,vo) = g(n,vo) + hg(n,vo) = g^*(n,vo) + hg(n,vo)$ //on optimal path

$<= g^*(n,vo) + hg^*(n,vo)$ //admissible heuristic

$= f^*$ //n is on the optimal path

So $f^* >= f(n,vo) >= f(G_1,v+1)$

Contradicting the assumption. So the solution is optimal.

An algorithm is defined as *admissible* if it is guaranteed to find an optimal path from s to the goal state for any δ graph, traversing a single selection from each of a set of categories.

Theorem 2: Once any global heuristic is globally admissible, then C* provides the optimal solution to the corresponding CSTQ, i.e., C* is admissible.

Proof:

First, based on Lemma 1 and Lemma 2, C* guarantees that the visit order is kept and CC is satisfied. Then based on Lemma 3, global admissibility guarantees solution optimality. Consequently, an optimal solution that satisfies the category visit order and CC is the optimal solution to the corresponding CSTQ, completing the proof.

4.7. Consistency

In C*, a local state refers to the combination of a vertex and its local target, the subgoal, while a global state is determined by the vertex, and its visit order. The local consistency does not care about *VisitOrder*, while global consistency does.

In best-first searches, consistency guarantees that a state will not be visited again once it is expanded [15][16]. Compared to the result from the consistency assumption identified in A* [16], in C*, the consistent global heuristic guarantees that a vertex with the same *VisitOrder* does not require consideration during the search once it is expanded. Before we prove this conclusion provided in Theorem 3, we prove another lemma first.

Lemma 4: Given the starting vertex s, for any nonclosed state (n,vo) and for any optimal path P from $(s,1)$ to (n,vo), there exists an open state (n', vo') on P with $g(n', vo')=g^*(n',vo')$, where g is the cost of the path from $(s,1)$ to (n',vo') obtained so far through C*, and g^* is the actual cost.

Proof:

Let $P = ((s,1),\ldots, (n_i,VO_{ni})\ldots,(n,vo))$, a collection of pairs of a vertex and its *VisitOrder*. If *(s,1)* is open (that is, C* has not completed even one iteration), let *n'=s*, and the lemma is trivially true since *g(s,1) = g*(s,1)*=0.

Suppose (*s,1)* is closed. Let *CL* be the set of all closed vertices n_i with *VisitOrder* VO_{ni} in *P* for which $g(n_i, VO_{ni}) = g^*(n_i, VO_{ni})$. *CL* is not empty, since by assumption (*s,1)* is within *CL*. Let *n** with *VisitOrder vo** be the element of *CL* with the highest index. Clearly, *n** is not *n*, as (*n,vo)* is nonclosed. Let *n'* with *VisitOrder vo'* be the successor of (*n*,vo*)* on *P*. (Possibly (*n',vo')* = *(n,vo)*.) Now by definition of *g*,

g(n',vo')<=g(n,vo*)+C(n*,n',Categories(vo*,vo'))*

where *C(n*, n', Categories(vo*,vo'))* is the actual cost from *n** to *n'* traversing necessary categories between them, which can also be formulated as *C((n*,vo*),(n',vo'))*;

g(n,vo*) = g*(n*, vo*)* because *n** is in *CL*, and

g(n',vo')=g*(n*,vo*)+C(n*,n',Categories(vo*,vo'))* because *P* is an optimal path. Therefore,

g(n',vo') <= g(n',vo')*. But in general, *g(n',vo')>=g*(n',vo')*, since the lowest cost *g(n',vo')* from (*s,1)* to (*n',vo')* explored at any time is never lower than the optimal cost *g*(n',vo')*. Thus *g(n',vo') = g*(n',vo')*, and moreover, (*n',vo')* must be open by the definition of CL.

Based on the lemma, the following corollary can be derived.

Corollary

Suppose *hg(n,vo) < hg*(n,vo)* for any state of vertex *n* and *visitOrder vo*, and suppose C* has not terminated. Then, for any optimal path *P* from the origin state of the starting vertex *s* to the goal state of the ending vertex *D*, there exists an open state (n',vo') on P with

f(n',vo') <=f(s,1).

Proof:

By Lemma 4, there exists an open state *n'* on *P* with *g(n',vo') = g*(n',vo')*, so by definition of *f*,

f(n',vo') = g(n',vo') + hg(n',vo')=g(n',vo') + hg(n',vo')<=g*(n',vo')+hg*(n',vo')=f*(n',vo')*.

But *P* is an optimal path, so *f*(n',vo') = f(s,1)* for all *n'* in *P*, completing the proof.

Now, based on Lemma 4, the following theorem states that global consistency guarantees a closed state will not be expanded.

Theorem 3: Assume any global heuristic is consistent, and a vertex *n* with *VisitOrder vo* is closed. Then *g(n,vo)=g*(n,vo)*, where *g(n,vo)* is the cost of the path, from the starting vertex *s*, traversing *(vo-1)* categories of interest, to *n*, obtained so far through C*, and *g*(n,vo)* is the actual cost.

Proof:

Use contradiction.

Consider the sub state graph space *Gs* in graph space in a graph space *G* just before closing *n*, and suppose that *g(n,vo)>g*(n,vo)*. Then there must be some optimal path *P* from (*s,1)* to (*n,vo)*. Since *g(n,vo)>g*(n,vo)*, C* does not find *P*.

According to Lemma 4, there exists an open state (*n', vo')* on *P* with *g(n',vo') = g*(n',vo')*. If (*n',vo') = (n,vo)*, we have proved the theorem. Otherwise,

g(n,vo)=g*(n',vo')+h*(n',n,Categories(vo',vo))*

where *h** is the actual cost of the shortest path starting from *n'*, traversing the remaining categories between *n'* and *n* when *vo>vo'*, and ending at *n*. It can also be formulated as *h*((n',vo'),(n,vo))*.

=g(n',vo')+h(n',n,Categories(vo',vo))*.

Thus,

g(n,vo)>g(n',vo')+h(n',n,Categories(vo',vo))*.

Adding the global heuristic, *hg(n,vo),* to both sides yields

g(n,vo) + hg(n,vo) > g(n',vo')+h(n',n,Categories(vo',vo))+hg(n,vo)*.

Since global consistency is satisfied:

h(n',n,Categories(vo',vo))+hg(n,vo) >=hg(n',vo'),*

then

g(n,vo)+hg(n,vo)>g(n',vo')+hg(n',vo')

or

f(n,vo)>f(n',vo')

Contradicting the fact that C* chose *(n,vo)* for expansion while *(n',vo')* was on the open list. Completing the proof.

Theorem 3 tells us that once a state (*v,vo)* is closed, whenever a (*v, vo)* is generated from another path, C* can just ignore it, which corresponds to the pseudo code on global consistency in Figure 3.

The next theorem states that f(s) is monotonically non-decreasing on the sequence of states closed by C*.

Theorem 4: Let *((s,1),…, (n_i,VO_{ni})…,(n,vo))* ((*x,y*), *x* represents the vertex, and *y* represents *VisitOrder*) be the sequence of states closed by C*. Then, if the global consistency assumption is satisfied, $p<=q$ implies $f(n_p,VO_{np})<= f(n_q,VO_{nq})$.

Proof:

Let (n,VO_n) be the next state closed by C* after closing (m,VO_m). Suppose first that the optimal path to (n,VO_n) does not traverse (m,VO_m). Then (n,VO_n) was available when (m,VO_m) was selected, and the theorem is trivially true. Now suppose that the optimal path to (n,VO_n) does traverse (m,VO_m), then

$g^*(n,VO_n))=g^*(m,VO_m)+h^*(m,n,Categories(VO_m,VO_n))$.

According to Theorem 3,

$g(n,VO_n))=g^*(n,VO_n)$ *and* $g(m,VO_m))=g^*(m,VO_m)$,

Since the global heuristic is consistent, and both *m* and *n* are on the optimal path,

$f(n, VO_n)=g(n, VO_n)+hg(n,VO_n) = g^*(n, VO_n)+hg(n,VO_n)$

$= g^*(m, VO_m)+h^*(m,n,Categories(VO_m , VO_n))+hg(n,VO_n)$

$>= g^*(m, VO_m)+hg(m, VO_m)= g (m, VO_m)+hg(m,VO_m)=f(m,VO_m)$

So $f(m,VO_m)<= f(n, VO_n)$,

completing the proof.

The following corollary is immediately available from the theorem.

Corollary

Under the premises of theorem 4, if *st* is the origin state and the state *nt* is closed, then $f(nt)<=f^*(st)$.

Proof:

Let gs be the goal state found by C*, then

$f(nt)<=f(gs)=f^*(gs)=f^*(st)$.

4.8. Consistency and optimal efficiency

The single-variate A* is optimally efficient in a sub graph *SG* when the local heuristic is consistent. Is the assumption that C* is optimally efficient when global consistency is guaranteed still valid in a discrete sub state graph space? During this section, we prove that the bivariate best-first-search C* is also optimally efficient among all best-first search admissible algorithms for CSTQ in a discrete sub state graph space once some preconditions are satisfied.

A set of procedures was identified to prove A* is optimally efficient in terms of the number of nodes to expand in a sub graph [16]. We adopt these procedures to prove C* also demonstrates the optimal efficiency property in terms of the number of states to expand in a discrete state graph space $\mathcal{G}$.

Next we prove a theorem on the optimal efficiency of C* as compared with any other admissible algorithm C that uses *no more information* about the problem than does C*.

Let θ_s^C be the state set used by algorithm C at state *s*. Then if $\theta_s^{C^*} \subset \theta_s^C$ for all states in G_s, it can be stated that algorithm C is *no more informed* than algorithm C*. The following theorem indicates that if an admissible algorithm C is no more informed than C*, then any state (vertex *v*, *VisitOrder vo*) expanded by C* must also be expanded by C. Two special cases can occur: ties occur in the value of *f used by C*, or ties never occur.* We prove the theorem for both cases.

Theorem 5: Let C be any admissible algorithm no more informed than C*. Let *Gs* be any δ sub state graph space such that given any two states, *(n,vo)* and *(n',vo')*, *(n,vo)≠(n',vo')*

$\Rightarrow$f(n,vo) $\neq$ f(n',vo')

Let the consistency assumption be satisfied by the *hg* used in C*. Then if state *(n,vo)* was expanded by C*, it was also expanded by C.

Proof:

Use contradiction.

Suppose there exists some state *sst*, which represents *(n,vo)*, expanded by C* but not by C. Let *gst** and *gst* be the goal states of origin state *ost* found by C* and C, respectively.

Since C* and C are both admissible,

f(gst)=g(gst*)+h(gst*)=g*(gst*)+0=f*(gst*)=f*(gst)=f*(ost)*.

Since C* must have expanded *sst* before closing *gst**, by theorem 4, and since no ties are allowed, we have

f(sst) <f(gst) = f*(gst)*

There exists some sub state graph space $G_{s,\theta}$, $\theta \in \Theta_s^C$, for which *hg(sst)=hg*(sst)* by the definition of hg. By theorem 3, *g(sst)=g*(sst)*. Then in the sub state graph space $G_{s,\theta}$, *f(sst)=f*(sst).*

Since C is no more informed than C*, C can not exclude the existence of $G_{s,\theta}$; but it does not expand *sst* before termination and thus is inadmissible, contradicting the assumption and completing the proof.

Define *N(C,Gs)* as the total number of states in *Gs* expanded by the algorithm C, based on the theorem, we have the following corollary.

Corollary

Under the premises of Theorem 5, the following inequality exists:

N(C*, Gs) < N(C, Gs)

with equality if and only if C expands the identical set of states as C*.

From this perspective, C* is claimed to be an optimal algorithm. C* expands the least possible states necessary to guarantee an optimal solution compared with any other no more informed admissible algorithms.

When ties exist, C* randomly expands one of the tied states whenever they have the least *f*. Now, assume the set $\mathcal{C}$* includes all algorithms that performs identically to C* but process ties differently. In other words, a member of $\mathcal{C}$* is the original C* with a random tie-breaking rule.

The next theorem extends theorem 5 to the situation where ties occur. The statement points out that for any admissible algorithm C, a member C* of $\mathcal{C}$* can always be found so that any state expanded by C* is always expanded by C.

Theorem 6: Let C be any admissible algorithm no more informed than the algorithms in $\mathcal{C}$*, and suppose the local consistency assumption is satisfied by the hg used in the algorithms in $\mathcal{C}$*. Then for any δ sub state graph space *Gs*, there exists a C*$\in$ $\mathcal{C}$*, such that every state expanded by C* is also expanded by C.

Proof:

Let C_1* be any algorithm in $\mathcal{C}$*. If every state of *Gs* expanded by C_1* is also expanded by C, then let C_1* be the C* of the theorem. Otherwise, the C* of the theorem can be constructed through the change of the tie-breaking rule of C_1*.

Let *S* as the set of the states expanded by C, and let *P=((s,1),…, (ni,*VO_{ni}*)…,(d,vo))((x,y), x* is a vertex, and *y* is the corresponding *VisitOrder, s* is the start vertex and *d* is the goal*)* be the optimal path found by C.

As long as states chosen for expansion are elements of *S*, they are expanded as prescribed by C_1*. Let *st* be the first state chosen for expansion by C_1* and *gs* as the goal state of D, which is not in *S*, then *f (st) <= f*((s,1))* according to the corollary to Theorem 4. Since *f(st)<f*((s,1))=f*(gs)* would imply that C is inadmissible (based on the argument of Theorem 5), it is true that *f(st)=f*((s,1))*. At the time C_1* selected *st*, goal state *gs* was not closed (or C_1* would have been terminated). Then by the corollary to Lemma 4, there is an open state *st'* on *P* such that *f(st')<=f*((s,1))=f(st)*. But since *st* was selected for expansion by C_1* instead of *st'*, *f(st)<=f(st')*.Consequently, *f(st')=f(st)*. Let C_2* be identical to C_1* except that the tie-breaking rule is modified just enough to choose *st'* instead of *st*. By repeating the above argument, we

obtain for some *i* a $C_i^* \in \mathcal{C}^*$ that expands only states that are also expanded by C, completing the proof of the theorem.

Based on the theorem, Corollary 1 exists.

Corollary 1

Suppose the premises of Theorem 6 are satisfied. Then for any δ sub state graph space *Gs*, there exists a $C^* \in \mathcal{C}^*$ such that N(C*,*Gs*)<=N(C,*Gs*), with equality if and only if C expands the identical set of states as C*.

It is also meaningful to know how all members of $\mathcal{C}^*$ perform against any C in the number of states expanded. Let's define a critical tie between state *st* and *st'* as one for which *f(st) =f(st')=f*((s,1))*. Then the second corollary exists.

Corollary 2

Suppose the premises of Theorem 6 are satisfied. Let R(C*, Gs) as the number of critical ties which occurred in the course of applying C* to *Gs*, then for any δ sub state graph space *Gs* and any $C^* \in \mathcal{C}^*$,

N(C*, *Gs*)<=N(C,*Gs*)+R(C*,*Gs*).

Proof:

For any noncritical tie, all alternative states must be expanded by both C and C*, or C could not be admissible. Consequently, it only requires observe that each state expanded by C* but not by C absolutely corresponds to a different critical tie where C*'s tie-breaking rule results in an inappropriate decision.

According to the theorems in subsection 4.7 and subsection 4.8, C*, the bivariate best first search, is optimally efficient in a discrete sub state graph space once its global heuristic is globally consistent.

4.9. Relationship between different states of a vertex

Since in a bivariate best first search, a vertex may have multiple states, the relationship between any two of its states can be used to prune unnecessary states.

Theorem 7: For a vertex *v*,

If(g(v,VO')>g(v,VO) and VO>=VO'), then the state *(v,VO')* is not on the optimal path.

Proof:

Since *VO>=VO'*, which means the path from the source state to *(v,VO)* traverses at least the same set of categories of interest as that from the course state to *(v,VO')*, it is clear that *g*(v,VO)>=g*(v,VO')*. According to the given condition, *g(v,VO')>g(v,VO)*, then *g(v,VO')>g*(v,VO)*, so *g(v,VO')>g*(v,VO')*. Consequently, *(v,VO')* is not on the optimal path. Completing the proof.

5. C*-P: A special case of C*

Several types of global heuristics may exist to process CSTQ with C*. For example, one possible global heuristic is the local heuristic that considers the goal vertex as the subgoal. However, this global heuristic may not take full advantage of the information that could be obtained from a CSTQ problem. In this section, a global heuristic, *g-p*, and C*-P, a special case of C* using *g-p* as the global heuristic, are proposed, a set of properties of C*-P is identified, and an example of the C*-P search process is presented.

5.1 C-P: A special case of C**

The proposed global heuristic *g-p* is calculated through local heuristics and partial heuristics. The following equation is provided to calculate *g-p(s)* for a state *s*:

$$g\text{-}p(s)=hg(v,j)= \min_{j=1}^{K} (h(v,O_{i,j})+h'(i,j)) \qquad (7)$$

where

s: a state,

v: the vertex in *s*,
K: the number of objects in the *i*th category of interest, the next category to be visited by *s*,
$O_{i,j}$: the *j*th object in the *i*th category of interest
g-p(s): the global heuristic,
hg(v,j): the global heuristic for vertex *v* with *VisitOrder j*;
$h(v,O_{i,j})$: the local heuristic between the vertex *v* and the subgoal $O_{i,j}$, and
h'(i,j): the partial heuristic starting from the *jth* object in the *ith* category of interest

To calculate *h'(i,j)*, a bottom up approach is adopted. The algorithm starts from the destination and calculates the partial heuristic between an object in the last visit category and the goal, then calculates the partial heuristic between an object in the second to the last category to visit and the goal using the results from the previous step, and continues this process until the partial heuristics for the objects in the first category to visit are calculated. Figure 4 shows the pseudo code for the algorithm.

```
Input:
    n: Number of visit categories
    O_{i,j} : the jth object in the ith category to visit
    Category(i): the ith category
Output: h'(i,j)
for (i=n;i--;i>0)
    for (j=0; j<Num(Category(i));j++)
        If(i==n)
            h'(i,j)=h(O_{i,j},D) ; // D is the destination
        Else
            //go over all possible paths to obtain the smallest heuristic
            h'(i,j)=h(O_{i,j}, O_{i+1,0})+h'(i+1,0);
            for(k=1;k<Num(Category(i+1));k++)
                if(h'(i,j)>h(O_{i,j}, O_{i+1,k})+h'(i+1,k))
                    h'(i,j)=h(O_{i,j}, O_{i+1,k})+h'(i+1,k))
```


Figure 4: a bottom up approach to calculate h's

The corresponding C* that uses *g-p* as the global heuristics is named as C*-P.

5.2 Properties of C-P*

Based on the theorems and corollaries identified in section 3, a set of properties of C*-P is identified as follows.

Theorem 8: Consider the final goal itself as a category. Assume any local heuristic between two vertices, each of which is from one of the consecutive visit categories respectively, is admissible (local admissible), then any partial heuristic *h'(i,j)* calculated with the bottom up approach is admissible.

Proof:

Use induction.

Given *l* as the reverse order a category is to visit.

1). When *l=1*, *i=n+1*, which means the destination is the final goal *D*, then $h'(i,j)=h(O_{i,j}, D)$. Since $h(O_{i,j}, D)$ is locally admissible, *h'(i,j)* is admissible

2). Suppose when *l=m,* the statement is true, i.e., *h'(n-m+2,j)* is locally admissible, which means

h'(n-m+2,j)<=h(n-m+2,j)*, where *h*(n-m+2,j)* is the actual minimum cost of the path from $O_{n-m+2,j}$, traversing the remaining categories to visit, to the final goal *D*

3). Then when *l=m+1*:

Assume *K* is the Number Of Objects in Category with VisitOrder (n-m+2).

Based on the definition of the partial heuristic,

$h'(n\text{-}m+1,j) = \min_{k=1}^{K} (h(O_{n-m+1,j}, O_{n-m+2,k}) + h'(n-m+2,k))$

$<= \min_{k=1}^{K} (h^*(O_{n-m+1,j}, O_{n-m+2,k}) + h^*(n-m+2,k)) = h^*(n-m+1,j)$

$(h^*(O_{n-m+1,j}, O_{n-m+2,k})$ is the actual cost between object $O_{n-m+1,j}$, and *object* $O_{n-m+2,k}$) $(1<=k<=K)$

So $h'(n-m+1,j)$ is admissible

So any partial heuristic is admissible

Theorem 9: If any local heuristic is locally admissible, then any global heuristic, calculated through equation (7), is globally admissible. In other words, local admissibility guarantees global admissibility.

Proof:

For any global heuristic $hg(v,vo)$ of vertex v with *VisitOrder* vo, since any local heuristic with a subgoal sg_i, the ith object in the next visit category,is locally admissible, and any partial heuristic is admissible based on Theorem 8, for all K subgoals in the next visit category,

$hg(v,vo) = \min_{i=1}^{K} (h(v,sg_i) + h'(vo,i)) <= \min_{i=1}^{K} (h^*(v,sg_i) + h'^*(vo,i)) = hg^*(v,vo)$

So $hg(v,vo)$ is admissible.

Based on Theorem 2 and Theorem 9, the following corollary is true.

Corollary

If any local heuristic is locally admissible and any global heuristic is calculated through equation (7), then C* is admissible.

The following theorem proves that *g-p* is globally consistent if local heuristics are locally consistent.

Theorem 10: If any local heuristic for any vertex towards any subgoal is consistent, then any global heuristic, calculated through equation (7), is consistent.

Proof:

Assume during the C* search process, a state (n,vo) is changed to another state(n',vo'). If we can prove that

$hg(n,vo) <= g^*(n,n',Categories(vo,vo')) + hg(n',vo')$,

where $g^*(n,n',Categories(vo,vo'))$ is the actual cost from (n,vo) to (n',vo') traversing the necessary categories between them when $vo'>vo$, then we prove the theorem.

Assume the number of objects in the next visit category is K.

1) If $vo'=vo$, since local consistency is guaranteed, then for the jth subgoal, sg_j, in the next visit category,

$h(n,sg_j) <= g^*(n,n') + h(n',sg_j)$,

so

$hg(n,vo) = \min_{j=1}^{K} (h(n,sg_j) + h'(vo,j)) <= \min_{j=1}^{K} (g^*(n,n') + h(n',sg_j) + h'(vo,j)) = g^*(n,n') + hg(n',vo)$

So the theorem is true.

2) If $vo'>vo$, Suppose the optimal path between (n,vo) and (n',vo') traverses category objects $O_{vo,j(vo)}, \ldots, O_{vo'-1,j(vo'-1)}$,and assume subgoal sg' as the one that satisfies $hg(n',vo') = hg\_sub(n',vo',sg')$, then according to the definition of h',

$h'(vo,j(vo)) <= h(O_{vo,j(vo)}, O_{vo+1,j(vo+1)}) + \ldots + h(O_{vo'-1,j(vo'-1)}, sg') + h'(vo', Index(sg'))$

Since local consistency is guaranteed and local consistency guarantees local admissibility, then

$h(O_{vo,j(vo)}, O_{vo+1,j(vo+1)}) + \ldots + h(O_{vo'-1,j(vo'-1)}, sg') <= g^*(O_{vo,j(vo)}, O_{vo+1,j(vo+1)}) + \ldots$ //local admissible

$+ g^*(O_{vo'-1,j(vo'-1)}, n') + h(n', sg')$ //local consistency

and

$h(n, O_{vo,j(vo)}) <= g^*(n, O_{vo,j(vo)})$ //local admissible

So based on the definition of global heuristic and equation (7),

$hg(n,vo) <= h(n,O_{vo,j(vo)}) + h'(vo,j(vo))$

$<= h(n,O_{vo,j(vo)}) + g^*(O_{vo,j(vo)}, O_{vo+1,j(vo+1)}) + \ldots + g^*(O_{vo'-1,j(vo'-1)}, n') + h(n',sg') + h'(vo',Index(sg'))$

$<= g^*(n, O_{vo,j(vo)}) + g^*(O_{vo,j(vo)}, O_{vo+1,j(vo+1)}) + \ldots$

$+ g^*(O_{vo'-1,j(vo'-1)}, n') + hg\_sub(n',vo',sg') + h'(vo',Index(sg'))$

$= g^*(n,n',Categories(vo,vo')) + hg(n',vo')$

Consequently,

$hg(n,vo) <= g^*(n,n',Categories(vo,vo')) + hg(n',vo')$

Based on 1) and 2), the theorem is proved.

Once g-p is globally consistent, then C*-P is optimally efficient.

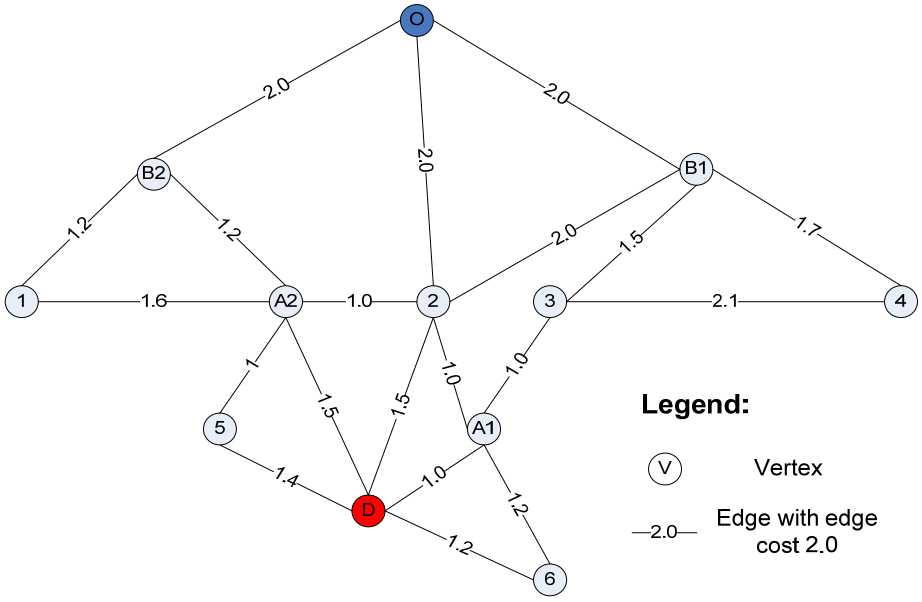
5.3 An example

To illustrate how the C*-P algorithm works, the following example is provided. Figure 5 shows an undirected graph composed of the starting vertex *O*, the goal *D*, six ordinary vertices numbered *1-6*, *A1* and *A2* in category *A*, and *B1* and *B2* in category *B*. An undirected graph is a special case of the graph definition where the edge between a pair of connected vertices is bidirectional and their costs are the same. A CSTQ demands an optimal path starting from *O*, traversing one single object in each of Category *A* and Category *B*, and ending at *D*.

This example uses equation (7) to calculate the global heuristic. The corresponding C* algorithm is named as C*-P. The local heuristic uses Euclidean distance calculated based on the spatial locations of the vertices. The pre-calculated admissible partial heuristics for objects in both categories are provided in Table 1. Since the query requires that the path traverse two categories, the goal state is *D* with *VisitOrder* 3.

Table 1: The partial heuristics for the objects in the visit categories

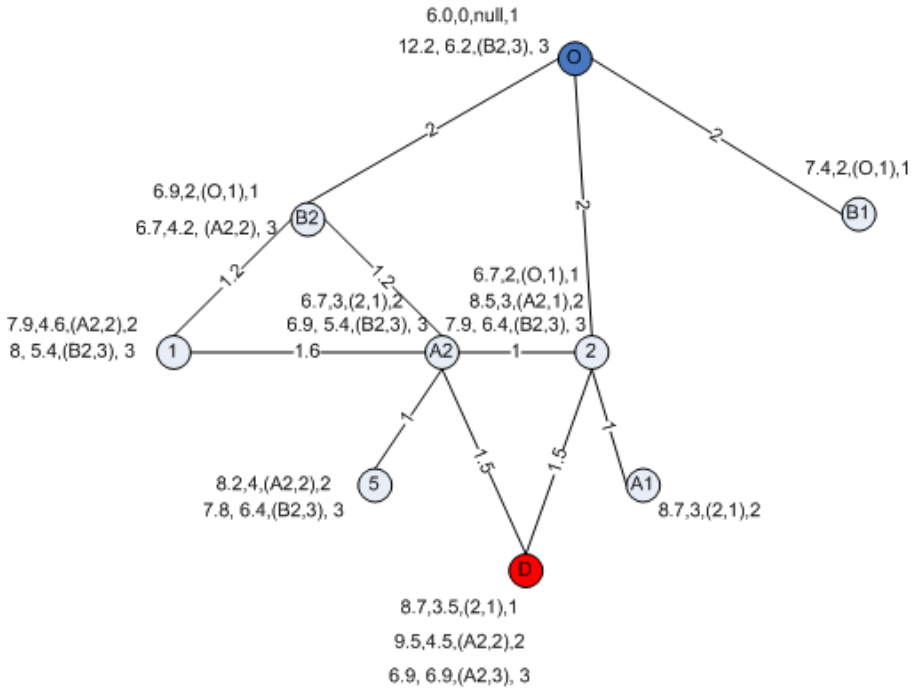
Vertex	h'
A1	5.4
A2	3.7
B1	3.2
B2	2.5



A CSTQ asks for a minimum cost path that starts from O, traverses category A and B in order, and ends at D.

Figure 5: An example: The graph and the CSTQ

The processing results of the C*-P is shown in Figure 6.



Legend:

f(s),g(s),(Vertex,VisitOrder),VisitOrder where f(s): estimate to the promise of state s; g(s): the cost from (O,1) to s; (Vertex, VisitOrder): parent; and VisitOrder: the order of the next category to visit

Before the goal state is expanded, the upper right shows the states in the open list and the lower right shows the states in the closed list.

Figure 6: The C* search process

The algorithm begins with the starting point *O* and its promise value *f(O)*. The subgoal based global heuristic for the path traversing *A1* is calculated with its local heuristic to *A1* and *A1*'s partial heuristic, and the subgoal based global heuristic for the path traversing *A2* is calculated with its local heuristic to *A2* and *A2*'s partial heuristic. The global heuristic *g-p(O,1)* is the minimum of them. The current *VisitOrder* for *O* is 1. The state is put into the priority queue. Then, the state of the lowest *f* value, vertex *O* with *VisitOrder* 1, is expanded first and put into the closed list with a null *backpointer*, and its three children, *2*, *B1* and *B2*, are generated. Neither *B1* nor *B2* is a subgoal, so they are normal vertices in this case. Again, these children's *f* and *g* values are calculated, and their *VisitOrder* remains unchanged.

Next, since vertex *2* has the lowest *f*, it is expanded and put into the closed list with a backpointer pointing to *(O,1)*, and its children *A1, A2, D, O,* and *B1* are generated. *A1* and *A2* are subgoals, so their *VisitOrder* is increased by 1, and they are put into the open list. The generated *O* and *B1* have a larger *f* than that with the same *VisitOrder* in the open list, so they are not changed on the open list. *D* is neither a subgoal nor the final goal since its *VisitOrder* is 1 instead of 3.

Again, based on the lowest *f*, C* puts *A2* with the subgoal *B2* in the closed list, expands it, and generates its 5 children, *B2, 1, 5, 2,* and *D*. Since *B2* is a subgoal in the next visit category *B*, its *VisitOrder* is increased by 1 and thus changed to 3 with the subgoal *D*. Since the *VisitOrder* for both *2* and *D* is 2 now, vertices *2* and *D* are put into the open list. Thereafter, *B2* with *VisitOrder* 3 has the lowest *f*, so *B2* is put into the closed list and expanded, and its children, *O, 1,* and *A2*, all with *VisitOrder* 3, are generated. In this case, *O* and *A2* are ordinary vertices.

Now, both *A2* with *VisitOrder* 3 and *B2* with *VisitOrder* 1 and *A2* as the subgoal have the same lowest *f*, but since the former has a larger *VisitOrder*, it is expanded first. Again, *A2*'s 5 children

are generated. All have a larger *VisitOrder* except the child *B2*. Finally, as before, *D* with *VisitOrder* 3, the goal state, is expanded. The search ends and the backtracking process to retrieve the obtained minimum-cost path in the closed list begins. The result of the backtracking process is

(D,3)->(A2,3)->(B2,3)->(A2,2)->(2,1)->(O,1)->null

Consequently, the obtained best path is *O->2->A2->B2->A2->D*, which is optimal.

6. Experiments

The purpose of the experiments is to test the performance of C* over different combinations of category numbers, object numbers in each category, and object spatial distributions in two scales of graphs. To see how the heuristic works, the special case of C* when hg=0 is used as the baseline. We name this special case as C*-Dijkstra because it is a consecutive network expansion algorithm. Whenever C*-Dijkstra reaches a subgoal, it will start network expansion based on the subgoal until either the final goal or another subgoal is reached.

One typical application of CSTQ is Ordered Trip Planning Query (OTPQ) in transportation, which asks for a shortest distance between a given origin-destination pair, traversing at least one location from each of a set of consumer destination categories in a specified order. In this scenario, a location or a consumer destination represent a vertex of interest in CSTQ. This kind of trip planning is especially useful to travelers and visitors that can not specify the detailed locations for a set of consumer destinations in an area to which they are unfamiliar. In this paper, the experiments were performed with a set of OTPQ.

The heuristic used in the experiments is calculated through equation (7), so C* is C*-P. Euclidean distance is used to calculate the local heuristics in equation (7), so the local heuristics are locally consistent. According to the theorems identified in section 4.2, *g-p* is always globally consistent. Therefore, C*-P is optimally efficient in this experiment.

In general, 30 runs for each traversal category number or each traversal consumer destination number were performed. For data set V and VI as mentioned below, only 10 runs were performed due to the expensive time cost for each run.

C* was implemented with Visual C#. The experiments were performed on a Dell desktop with 3.5GB memory, and 2.8 GHZ Pentium(R) 4 CPU.

6.1. Data set

Two scales of transportation networks are adopted. As shown in Figure 7, the first graph represents the transportation network of Fairfax City, Virginia, USA. The graph has 235 vertices and 740 directed edges. Second, a graph that represents the transportation network of Fairfax City and Fairfax County, Virginia, USA is used. As shown in Figure 8, the graph contains 35,435 vertices, and 82,926 directed edges.

To illustrate how C* works in different scenarios, three sets of consumer destinations were used. The first set is real and was collected from seven categories: post office (2 locations), Bank of America (4 locations), restaurant (1 location), department store (2 locations), hotel (1 location), gas station (20 locations), and auto repair (18 locations). Figure 7 shows the spatial distribution of these consumer destinations. Note that some consumer destinations overlap each other. The origin used in data set I discussed later is marked by the red flag symbol at left. The second set was uniformly randomly generated for the first graph, with different combinations of number of categories and object number in each category. The third set was uniformly randomly generated for the second graph, still with different combinations of number of categories and object number in each category.

To study how the location distribution of origins affects the performance, three different origins were adopted. The first is the only hotel collected from the field data collection, which is meaningful since visitors unfamiliar to a specific area may perform category based queries. The second and the third are randomly generated for the first graph and the second graph respectively.

The destinations were uniformly randomly generated.

Of the six data sets, the first four are for the first graph, and the remaining are for the second graph. In data sets I and II, the traversal order was specified based on assumed user behavior for each category sequence. For example, users may want to go to a gas station first to avoid running out of gas during the trip, or to get car services at the last stop of the trip. In the other four data sets, the order was randomly selected.

Table 2 summarizes the characteristics of the six data sets.

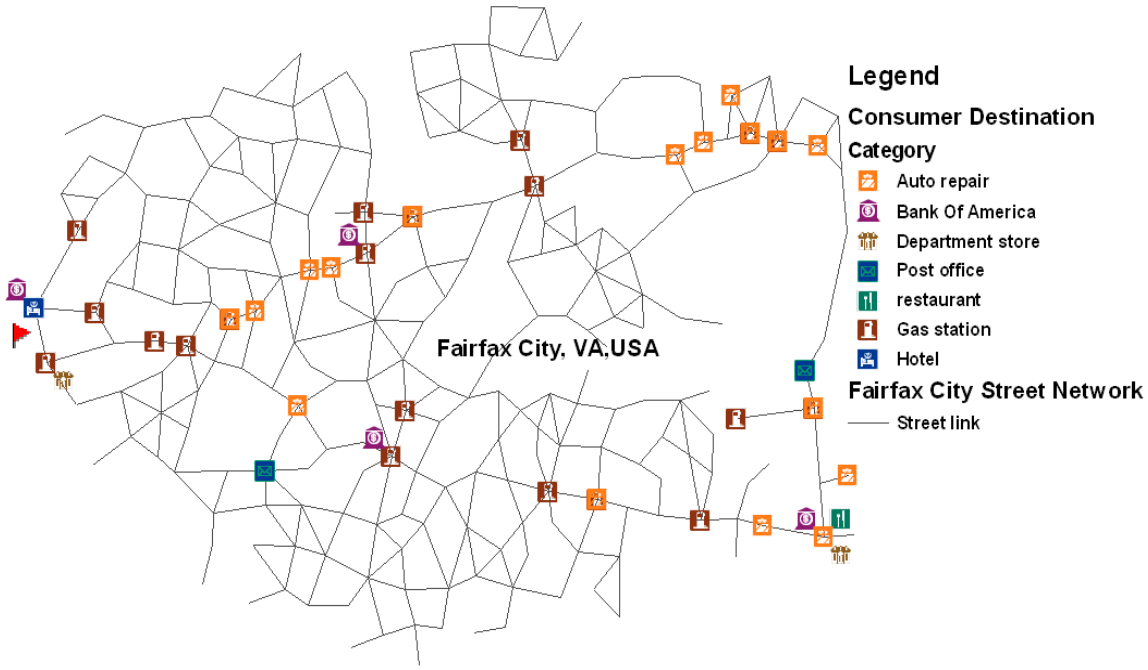


Figure 7: The Fairfax City street network and category objects in data set I

Figure 8: The Fairfax City and Fairfax County street network

Table 2: Characteristics of six data sets

Data set No.	**Origin**	**Categories and their objects**	**Traversal order**
I	Collected	Collected	Determined based on assumed user behavior
II	Randomly generated	Collected	Determined based on assumed user behavior
III	Randomly generated	Randomly generated, same number of objects, equal to 2, in each category but different category number	Randomly selected
IV	Randomly generated	Randomly generated, same number of categories, equal to 10, but different number of objects in each category	Randomly selected
V	Randomly generated	Randomly generated, same number of objects, equal to 3000, in each category but different category number	Randomly selected
VI	Randomly generated	Randomly generated, same number of categories, equal to 3, but different number of objects in each category	Randomly selected

6.2. Performance measures

The following measures were used to analyze the performance of C*-P and C*-Dijkstra.

Average Shortest Distance (ASD): the average route distance obtained over all runs, in mile*;*

Average Number of States Expanded (ANSE): the average number of expanded states obtained over all runs;

Maximum Additional Number of States expanded by C-Dijkstra (MaxANS-CD):* For each run, compared to C*-P, obtain the additional number of states expanded by C*-Dijkstra, and the measure is the maximum among all runs;

Minimum Additional Number of States expanded by C-Dijkstra (MinANS-CD):* For each run, compared to C*-P, obtain the additional number of states expanded by C*-Dijkstra, and the measure is the minimum among all runs;

Average Process Time (APT): the time required to return the solution for a query, in second;

Maximum Additional Cost by C-Dijkstra (MaxAC-CD):* For each run, compared to C*-P, obtain the additional time cost required by C-Dijkstra, and the measure is the maximum among all runs;

Minimum Additional Cost by C-Dijkstra (MinAC-CD):* For each run, compared to C*-P, obtain the additional time cost required by C-Dijkstra, and the measure is the maximum among all runs;

Average Relative Number of States expanded (ARNS): the ratio of the number of states expanded by C*-Dijkstra over by C*-P; and

Average Relative Process Time (ARPT): the ratio of the time processed by C*-Dijkstra over by C*-P.

In the following tables, NOC represents number of categories, NOEC represents number of objects in each category, and CD represents C*-Dijkstra.

6.3. Results

Since the global heuristic of C*-Dijkstra is always 0 and thus is always consistent, C*-Dijkstra is admissible. In other words, C*-Dijkstra always retrieves the optimal solution. Furthermore, since Euclidean distance between two vertices in a transportation network is always not larger than the network distance between them, local heuristic is admissible and consistent, then based

on the corollary of theorem 2, C*-P is also admissible. Consequently, both algorithms should return the same shortest distance for the same query, which was observed in the experiments.

The results for the four data sets are provided in Table 3 through Table 8 and Figure 9 through Figure 20. The minimum and the maximum are highlighted in bold for ARNS and ARPT in all six tables.

Table 3: Performance results on data set I

NOC	ASD	ANSE		MaxANS-CD	MinANS-CD	APT		MaxAC-CD	MinAC-CD	ARNS	ARPT
		C*-P	CD			C*-P	CD				
1	2.8	157	586	1089	32	0.01	0.08	0.22	0.00	3.7	5.4
2	2.6	259	1450	2638	113	0.01	0.13	0.27	0.02	5.6	**16.4**
3	3.2	319	2248	3983	204	0.03	0.32	0.77	0.03	**7.0**	11.9
4	4.9	737	4250	7965	457	0.05	0.79	1.50	0.19	5.8	16.4
5	6.5	1512	3652	4272	181	0.13	0.56	0.78	0.19	**2.4**	**4.4**
6	6.3	3463	9474	11408	450	0.10	0.46	0.56	0.16	2.7	4.4

Table 4: Performance results on data set II

NOC	ASD	ANSE		MaxANS-CD	MinANS-CD	APT		MaxAC-CD	MinAC-CD	ARNS	ARPT
		C*-P	CD			C*-P	CD				
1	2.9	167	883	1320	108	0.01	0.14	0.27	0.00	5.3	14.2
2	2.8	314	2609	4468	385	0.01	0.30	0.58	0.05	**8.3**	24.2
3	3.5	584	3863	6237	562	0.02	0.59	1.05	0.08	6.6	**26.4**
4	4.7	918	4861	7586	587	0.08	1.02	1.34	0.08	5.3	13.3
5	5.8	1374	4894	7891	347	0.12	0.98	1.55	0.17	3.6	8.2
6	6.3	4830	13290	16960	652	0.17	0.93	1.03	0.33	**2.8**	**5.3**

Table 5: Performance results on data set III

NOC	ASD	ANSE		MaxANS-CD	MinANS-CD	APT		MaxAC-CD	MinAC-CD	ARNS	ARPT
		C*-P	CD			C*-P	CD				
10	15.9	1056	1984	1377	623	0.86	2.69	2.66	1.05	**1.9**	**3.1**
20	26.1	2507	3832	1876	741	4.39	9.78	7.92	3.16	1.5	2.2
30	37.8	4224	5621	1909	1018	12.18	21.65	12.94	6.55	1.3	1.8
40	48.2	5850	7406	2532	693	24.42	39.85	22.66	7.13	**1.3**	**1.6**

Table 6: Performance results on data set IV

NOEC	ASD	ANSE		MaxANS-CD	MinANS-CD	APT		MaxAC-CD	MinAC-CD	ARNS	ARPT
		C*-P	CD			C*-P	CD				
1	23.3	1778	2501	1139	291	4.14	7.79	5.55	1.63	**1.4**	**1.9**
2	15.9	1056	1984	1377	623	1.53	4.91	5.67	2.17	1.9	3.2
3	12.0	637	1536	1263	583	0.58	2.94	3.67	1.28	2.4	5.0
4	9.8	450	1227	1252	392	0.31	1.93	3.22	0.56	2.7	6.3
5	7.3	305	1134	1117	423	0.15	1.65	2.30	0.48	**3.7**	**11.2**
6	5.8	218	750	965	152	0.08	0.74	1.52	0.09	3.4	9.3

Table 7: Performance results on data set V

NOC	*ANSE*		*MaxANS-CD*	*MinANS-CD*	*APT*		*MaxAC-CD*	*MinAC-CD*	*ARNS*	*ARPT*
	C*-P	CD			C*-P	CD				
2	13685	58309	55890	22804	362.70	2161.91	3081.56	559.90	**4.3**	**6.0**
3	11653	51883	52460	22804	236.89	1563.40	3131.53	37.29	**4.5**	6.6
4	12456	55144	63937	22804	277.21	1996.03	3127.68	567.09	4.4	7.2
5	11655	49571	52460	22804	239.34	2067.42	3123.22	561.34	4.3	**8.6**
6	11656	49572	52460	22804	244.52	2090.28	3137.53	558.22	4.3	8.5
7	11657	49573	52460	22804	246.88	2076.23	3121.87	571.04	4.3	8.4

Table 8: Performance results on data set VI

NOEC	*ANSE*		*MaxANS-CD*	*MinANS-CD*	*APT*		*MaxAC-CD*	*MinAC-CD*	*ARNS*	*ARPT*
	C*-P	CD			C*-P	CD				
3000	11653	51883	52460	22804	236.89	1563.40	3131.53	37.29	4.5	6.6
3200	11990	58329	71859	22804	265.84	1556.27	3123.04	31.24	4.9	**5.9**
3400	12587	60453	68659	22804	282.04	1693.83	3157.32	421.42	4.8	6.0
3600	11698	57973	66591	22804	248.04	2173.79	3125.49	562.26	**5.0**	**8.8**
3800	14042	60167	60240	22804	356.19	2369.16	3170.46	561.39	**4.3**	6.7

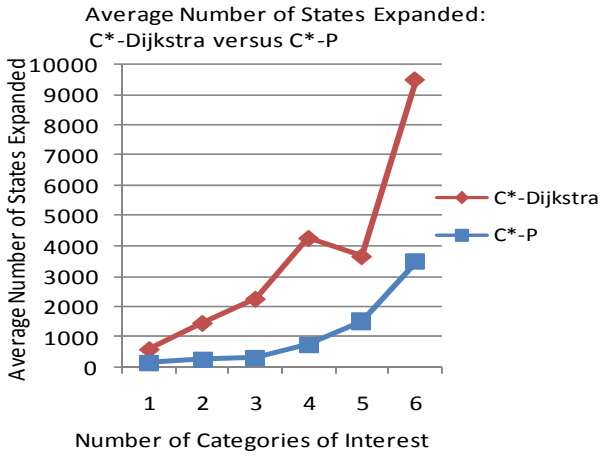


Figure 9: Average number of states expanded for data set I

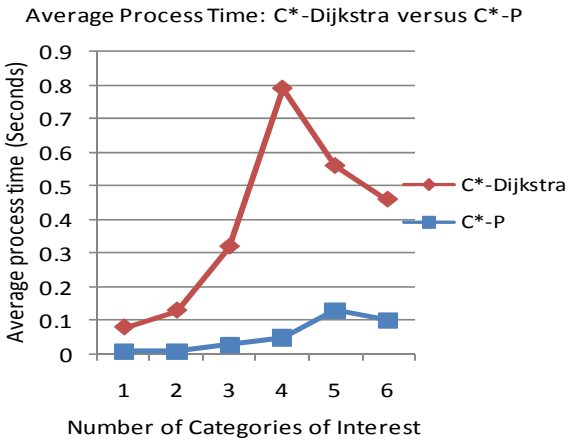


Figure 10: Average process time for data set I

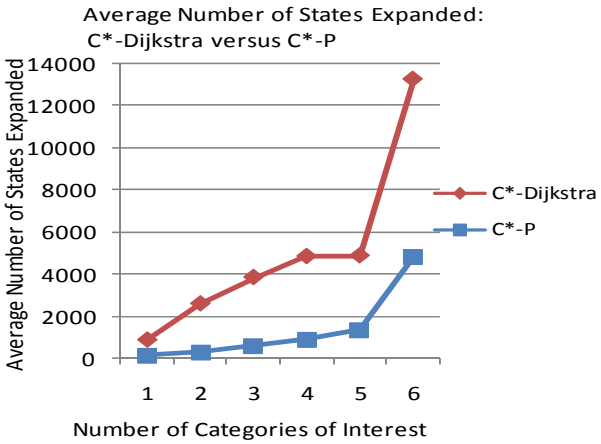


Figure 11: Average number of states expanded for data set II

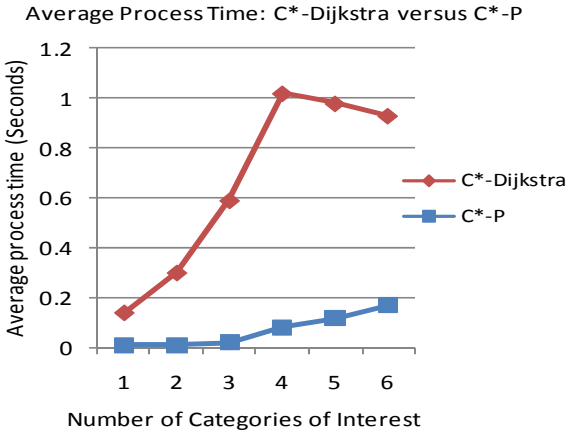


Figure 12: Average process time for data set II

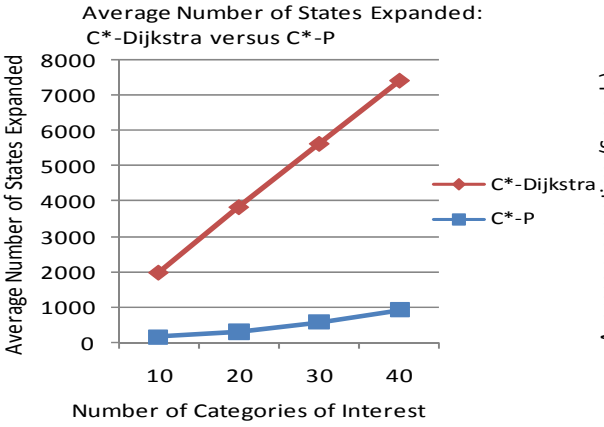


Figure 13: Average number of states expanded for data set III

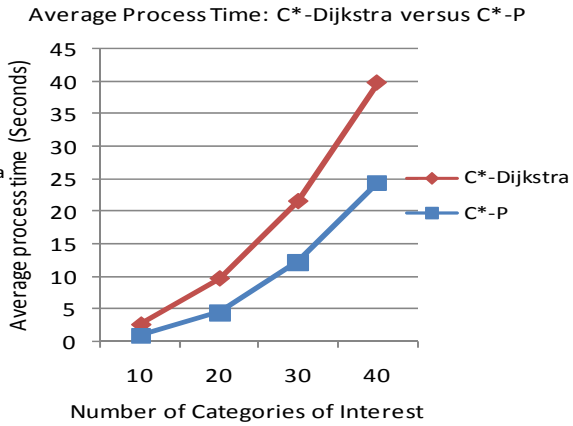


Figure 14: Average process time for data set III

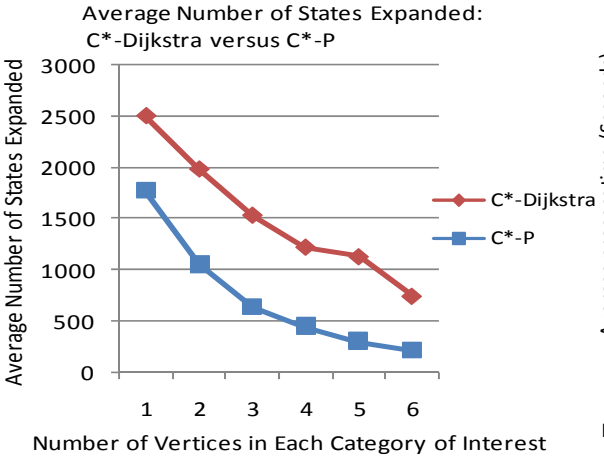


Figure 15: Average number of states expanded for data set IV

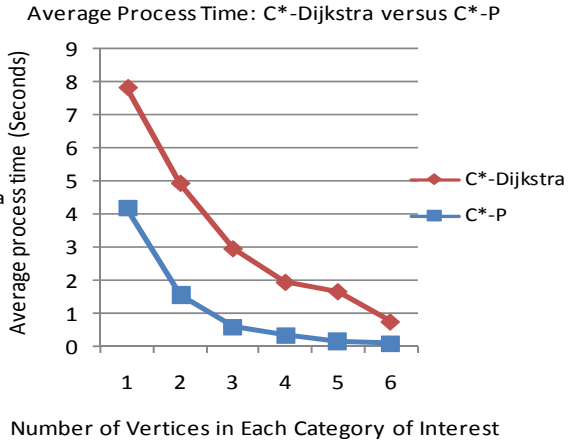


Figure 16: Average process time for data set IV

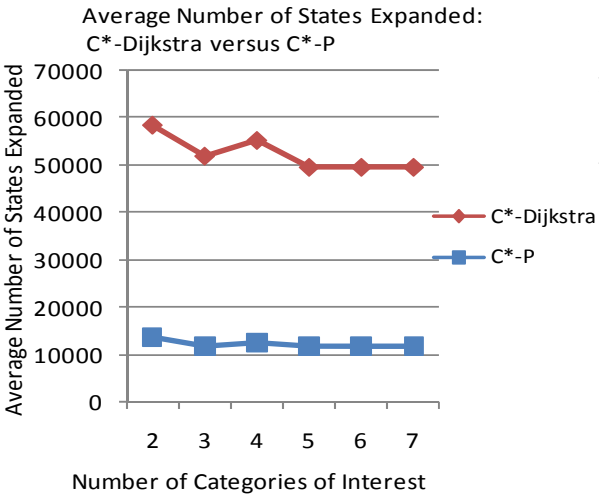


Figure 17: Average number of states expanded for data set V

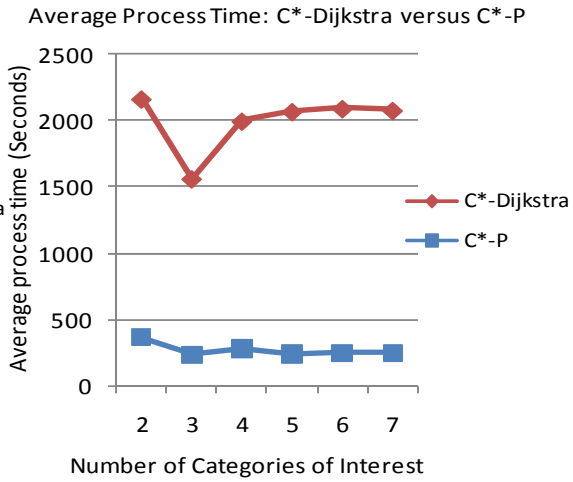


Figure 18: Average process time for data set V

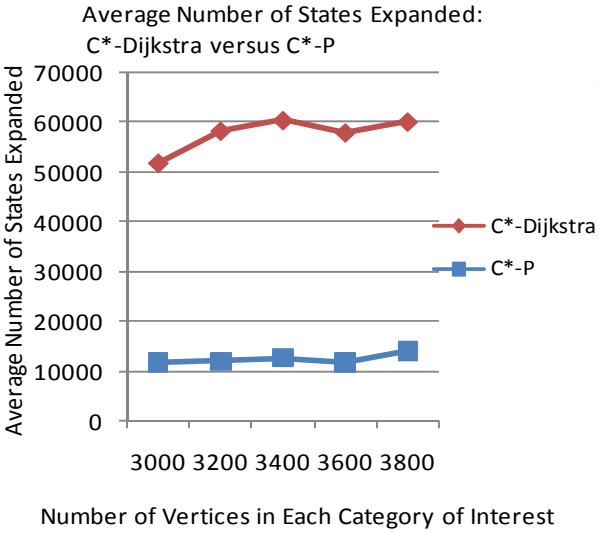


Figure 19: Average number of states expanded for data set VI

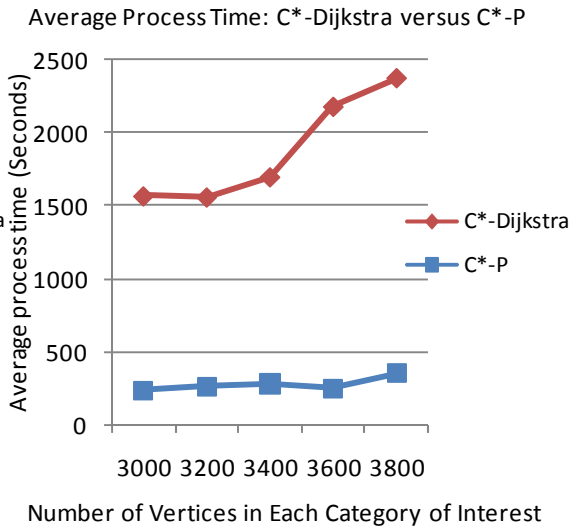


Figure 20: Average process time for data set VI

From the results presented in the tables and figures, conclusions are drawn as follows.

MinANS-CD is always larger than 0, which means that no matter how a) the number of categories of interest, b) the number of vertices in each category, and c) the spatial locations of the origins change, the number of states expanded by C*-P is always less than by C*-Dijkstra, since C*-P is more informed than C*-Dijkstra, i.e., C*-P uses more information from the problem domain. According to the highlights in bold for ARNS, C*-Dijkstra expanded between 0.3 and 7.3 times more states than C*-P on average.

From Figure 9 through Figure 20, it is clear that on average C*-P is always faster and expands less states than C*-Dijkstra no matter how a) the number of categories of interest, b) the number of vertices in each category, and c) the spatial locations of the origins change.

C*-P processes much faster than C*-Dijkstra in nearly all cases. According to the highlights in bold for ARPT, C*-Dijkstra processes a query between 0.6 and 25.4 times slower than C*-P on average. As highlighted for MinAC-CD in Table 3, one worse case exists that C*-P is only slightly slower than C*-Dijkstra when the category number to traverse is only 1 and the route distance is about 1 mile from the first data set. This is reasonable since the start vertex is near the

boundary of the street network, which forces C*-Dijkstra to perform network expansion towards only a portion of all directions, and thus the time to obtain heuristics makes C*-P slower than C*-Dijkstra.

In general, the larger the number of categories of interest, the longer the optimal route is. The larger the number of vertices in each category, the shorter the optimal route is.

An interesting phenomenon is worth noting. Neither ARNS nor ARTP has a linear relation to the number of traversal categories or the number of objects in each traversal category. In other words, the additional states expanded or additional process time required by C*-Dijkstra is not necessary proportional to the number of traversal categories or the number of objects in each traversal category.

When the number of categories of interest or the number of vertices in each category of interest varies, the performances of C*-P and C*-Dijkstra in terms of ANSE and APT vary over different data sets. According to Figure 15 and Figure 19, when the number of vertices in each category increases, in general, the average number of states expanded decreases for data set IV while increases for data set VI. According to Figure 16 and Figure 20, when the number of vertices in each category increases, in general, the average process time decreases for data set IV while increases for data set VI.

Through the comparison of the values for MaxAC-CD and MinAC-CD between Table 5 and Table 7, and between Table 6 and Table 8, we can see that in a much larger transportation network, the performance differences in terms of MaxAC-CD and MinAC-CD are substantially larger between C*-P and C*-Dijkstra. Consequently, it is preferable to use C*-P to obtain an optimal path for a CSTQ in a large transportation network.

7. The power of the estimate *hg*

Similar to A*, C* is also a family of algorithms. Its choice of a particular function *hg* corresponds to a particular algorithm from C*. *hg* can be used to tune C* for a specific application.

If *hg* is 0, then C* knows, or takes advantage of, absolutely no information from the problem domain.

In general, however, the problem domain can provide some useful information that C* can use. For example, in a CSTQ problem to retrieve a shortest route from home, traversing a set of consumer destinations in different categories, to a work place, the Euclidean distance between any two consumer destinations or between the starting point and a consumer destination or between a consumer destination and the work place is known and no larger than the corresponding actual distance. In this case, any local heuristic is admissible, thus *hg* is admissible, which guarantees an optimal solution. In this case, C* can reduce some vertices to expand compared to those algorithms knowing or using nothing from the problem domain.

To further reduce the computation, *hg* can be larger than the actual cost, and thus *hg* is not admissible. From a heuristic point of view, this might be more desirable because sometimes an admissible heuristic is not available or is hard to obtain. The result may still be optimal but is more likely to be suboptimal. Also, fewer vertices may be expanded.

To conclude, like A*, the formulation provided takes advantage of one function, *hg*, to consider all possible knowledge from the problem domain in formal theory. Different selections of function *hg* provide the flexibility to allow one to compromise between admissibility, consistency, and computational efficiency.

8. C* and A*

As a single-variate instance of L#, A* is a special case of C*, a bivariate best-first-search instance of L#. When there is no category to visit along the path from a starting point to the goal, a CSTQ becomes a query for the minimum cost path between the start and the goal, which is

exactly what A* tries to resolve. Since C* resolves CSTQs, it resolves this special case as well, and C* becomes A*.

9. Conclusion

The contribution of this paper includes five components. First, this paper generalizes the existing best first searches from single-variate best first searches to multivariate best first searches in terms of the number of variables to specify a state of a vertex in a best first search. The estimates to the promises are defined upon states instead of vertices. As such a generalized best first search, L# is proposed to process complex queries in a graph. A set of new concepts, including local heuristic, global heuristic, local admissibility, global admissibility, local consistency, global consistency, state graph, etc., are identified and introduced into multivariate best first searches. Second, this paper proposes a novel multi-category based query in a graph, CSTQ, which determines the minimum-cost path with a predefined origin-destination pair, traversing at least a single selection from each of a set of categories in a specified order. Third, this paper provides a bivariate best-first search algorithm, C*, an instance of L#, to process the query. From the literature, C* is the first to extend the existing single-variate best first search to the bivariate best first search. Fourth, this paper extends the theorems on optimality and optimal efficiency in a sub graph identified in the single-variate A* to the bivariate best-first-search C* in a discrete sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of categories of interest in the given order. Fifth, two C* algorithms, C*-P and C*-Dijkstra, are identified, and their performances are analyzed for CSTQ processing in a transportation network.

In a graph, since a minimum-cost path query for a given origin and destination pair is a special case of CSTQ, C* generalizes A*.

Due to the variability of the choice of the heuristic function, C* is a family of algorithms. Based on the characteristics, including admissibility and consistency, of any heuristic in a graph, C* demonstrates optimality and optimal efficiency correspondingly.

C*-Dijkstra is a special case of C*, and is always consistent in a δ graph. To study the performance of C* when its global heuristics are consistent, experiments were performed on C*-P and C*-Dijkstra with six data sets provided in a transportation network, and the results show that C*-P performs much better than C*-Dijkstra in terms of number of expanded nodes and the cost of process time. On average, C*-Dijkstra is 0.6~25.4 times slower and expands 0.3~7.3 times more states than C*-P.

References

[1] F. Li, D. Cheng, M. Hadjieleftherious, G. Kollios and S. Teng. On Trip Planning Queries in Spatial Databases. In: Proceedings of 9th International Symposium on Spatial and Temporal Databases, Angra dos Reis, Brazil, 2005, pp. 273-290.

[2] X. Ma, S. Shekhar, H. Xiong, P. Zhang. Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries. In: Proceedings of 14th International Symposium on Advances in Geographic Information Systems, College Park, USA, 2006, pp. 179-186.

[3] M. Kolahdouzan, M. Sharifzadeh and C. Shahabi. The Optimal Sequenced Route Query. University of Southern California, Computer Science Department, Technical Report 05-840, 2005

[4] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi. The Optimal Sequenced Route Query, The VLDB Journal: The International Journal on Very Large Data Bases, 17 (4) (2008) 765-787.

[5] S. Arora. Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems. In: Proceedings of 37th IEEE Symposium on Foundations of Computer Science, Burlington, 1996, pp. 2-11.

[6] S. Arora. Approximation Schemes for NP-hard Geometric Optimization Problems: A survey. Mathematical Programming, Springer, 97 (2003) 43-69.

[7] N. Christofides. Worst-case Analysis of a New Heuristic for the Traveling Salesman Problem. Carnegie Mellon University, Computer Science Dept, Tech Technical report, 1976.

[8] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press, 1997.

[9] A. Dumitrescu and J. S. B. Mitchell. Approximation Algorithms for TSP with Neighborhoods in the Plane. In: Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms, Washington DC, USA, 2001, pp. 38–46.
[10] S. Shekhar and J. S. Yoo. Processing in-route nearest neighbor queries: a comparison of alternative approaches. In GIS '03: Proceedings of the 11th ACM International Symposium on Advances in Geographic Information Systems, pages 9{16, New Orleans, LA, USA, 2003.
[11] M. Sharifzadeh, C. Shahabi. Additively Weighted Voronoi Diagrams for the Optimal Sequenced Route Query, In: 3rd Workshop on Spatio-Temporal Database Management, Seul. Korea, 2006, pp. 33-40.
[12] B. Hu, M. Leitner, and G. R. Raidl. Computing Generalized Minimum Spanning Trees with Variable Neighborhood Search. In P. Hansen, N. Mladenovic, J. A. M. Pérez, B. M. Batista, and J. M. Moreno-Vega,(EDs), Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search, Tenerife, Spain, 2005.
[13] Y. S. Myung, C. H. Lee, and D. W. Tcha. On the Generalized Minimum Spanning Tree Problem. Networks, 26 (1995) 231–241.
[14] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesley, 1984.
[15] S. Russell and P. Norvig. Artificial Intelligence: a Modern Approach (2nd edition). Prentice Hall, 2002.
[16] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 (2) (1968) 100–107.
[17] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. In Numerische Mathematik, 1 (1959), S. 269–271
[18] R. E. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier Search. Journal of the Association for Computing Machinery. 52(5) (2005) 715-748.
[19] R. Dechter and J. Pearl. Generalized Best-first Search Strategies and the Optimality of A*. Journal of the Association for Computing Machinery. 32(3) (1985) 505-536.
[20] M. Likhachev, G. J. Gordon, S. Thrun. Ara*: Anytime A* with provable bounds on sub-optimality. In. Advances in Neural Information Processing Systems 16. MIT. Press, Cambridge, MA, 2004.
[21] A. Botea, M. Muller, J. Schaeffer. Near Optimal Hierarchical Path-Finding. In Journal of Game Development, volume 1, issue 1, (2004), 7-28.
[22] S. Russell. Efficient Memory-bounded Search Methods. In Proceedings of Tenth European Conference on Artificial Intelligence (ECAI-92), pages 1--5. Chichester, England: Wiley, 1992.
[23] R. Zhou, Eric A. Hansen. Memory-Bounded A* Graph Search. Proceedings of the Fifteenth International Florida Artificial Intelligence Research. May 2002.

CHAPTER 4: O*: A BIVARIATE BEST FIRST SEARCH ALGORITHM TO PROCESS OPTIMAL SEQUENCE TRAVERSAL QUERIES IN A GRAPH

by

Qifeng Lu

Kathleen Hancock

O*: A Bivariate Best First Search Algorithm to Process Optimal Sequence Traversal Queries in a Graph

Qifeng Lu, Kathleen Hancock

Civil and Environmental Engineering Department, Virginia Polytechnic Institute and State University, Alexandria, VA 22314, USA

Abstract

An Optimal Sequence Traversal Query (OSTQ) is a new query in a graph that determines the minimum-cost path with a predefined origin-destination pair, traversing a set of non-ordered points of interest, at least once for each point. This paper proposes a bivariate best first search, O*, to process OSTQ within a graph, and discusses how to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate both optimality and optimal efficiency properties in an item-enumeration sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of enumerated points of interest. Three special cases of O* are identified and their performances are studied in a transportation network. Since a minimum-cost path query for a given origin and destination pair is a special case of OSTQ, A*, the best-first graph search algorithm, is a special case of the more general O*.

Keywords: Best First Search, Heuristic, Optimal Sequence Traversal Query, A*, C*, O*, L#, MST

1. Introduction and background

The Optimal Sequence Traversal Query (OSTQ) is a query conducted to find the minimum-cost path that starts from any given origin, passes through a set of points of interest, and terminates at a given destination. It has distinctive applications in transportation where a user may query a shortest route to start from his/her office, traverse a set of consumer destinations, and then go home. It may have potential applications in artificial intelligence where a robot is sent to collect data from multiple sensors and delivers the data to a given destination for downloading and analysis. The traveling salesman problem (TSP) is a special case of OSTQ, where the given origin and destination are the same [1] [2] [3] [4] [5] [6]. Very little published information is available for the more generalized OSTQ processed in a graph.

A graph can conceptually represent a complex network, such as a transportation network, as well as other types of networks. In this context, a graph G is defined as a set of vertices, $\{V_i\}$, and a set of directed line segments, $\{E_{i,j}\}$, called arcs. $E_{i,j}$ is defined such that an arc is from vertex V_i to vertex V_j, and V_j is a successor of V_i. Each $E_{i,j}$ has an associated cost $C_{i,j}$. Only graphs with $C_{i,j}>=0$ are considered in this paper. These graphs are named as δ graphs.

In this paper, we propose a new type of query defined in such a δ graph: Optimal Sequence Traversal Query (OSTQ). Given a δ graph G with its vertices $\{V_i\}$ and edges $\{E_{ij}\}$, a set of vertices of interest VI from $\{V_i\}$ in graph G, a starting vertex S, and a destination vertex D, an OSTQ retrieves the minimum-cost path, traversing all vertices of interest, at least once for each vertex. In such a graph, there may exist *normal vertices* that are neither vertices of interest to traverse, nor the given origin or destination. Within this context, all necessary sub state graphs are generated implicitly. An application of this query is that a consumer drives from his/her office, traverses a gas station, a coffee shop, and a post office, and gets home. Another application is that

a food delivery person starts from the restaurant he works at, traverses a set of delivery points, and then returns to the restaurant, a typical example of TSP.

This paper proposes a bivariate best-first search algorithm, O*, to process OSTQ in a graph. O* is a bivariate best first search in the sense that it uses two variables to specify its state. Both theoretical and experimental analyses of the proposed algorithm are presented. Furthermore, a formal basis is established to evaluate O*'s performance including optimality and optimal efficiency based on the choice of the heuristic function. The optimal efficiency property is established based on the comparison to any other possible admissible best first algorithm targeted to resolve OSTQ in an item-enumeration sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of enumerated vertices of interest.

The paper is organized as follows. First, related work is discussed in Section 2. In Section 3, O* is presented, together with the formal basis to evaluate O*'s performance including optimality and optimal efficiency. Section 4 provides the MST heuristic, a globally admissible heuristic for O*, and presents an example. Section 5 presents a set of experiments and an analysis of the results. Section 6 describes the heuristic used in O*. Section 7 discusses the relationship between O* and A*. Section 8 discusses the relationship between the identified theorems in this paper and the existing best first search algorithm for TSP. Finally, the conclusion is presented.

2. Related work

2.1. Travelling salesman problem (TSP)

The earliest research in TSP is in Euclidean space which searches for a shortest round-trip route to traverse each city exactly once with all cities directly connected to each other, forming a fully connected graph. A set of solutions, including dynamic programming [7], nearest neighbor [8], iterative algorithms like 2-OPT, 3-OPT, and n-OPT [9], best first search [10], ant colony simulation [11], simulated annealing [12], Tabu search [13], and so on, were proposed to resolve this problem, either exactly or approximately, and the result is a Hamiltonian cycle that visits each vertex exactly once and returns to the starting vertex. Since the solution is a cycle, which may not be true for OSTQ, many algorithms used it to expedite the search process, which means they can not be adopted to resolve OSTQ directly.

2.2. Best first searches

A best first search is a kind of informed searches. The following two subsections provide a review of single-variate best first searches and bivariate best first searches, respectively. A best first search is n-variate if it uses n variables to specify its state.

2.2.1. Single-variate best first searches

Single-variate best first search is the existing best first search that searches a graph by expanding the most promising vertex chosen according to some rule. It adopts estimates to the promise of vertex n by a "heuristic evaluation function *f(n)* which, in general, may depend on the description of *n*, the description of the goal, the information gathered by the search up to that point, and most important, on any extra knowledge about the problem domain." [14], which is prevalent used by researchers in Artificial Intelligence (AI), including Russell & Norvig [15].

Several algorithms, including A* [15][16], Dijkstra search [17], Greedy search [15], frontier search [18], and so on, extract the path of minimum cost between a predefined origin-destination vertex pair in a graph. A* uses a distance-plus-cost heuristic function as *f(n)* to determine the order in which the search visits vertices in the graph [14]. *f(n)* is the sum of two functions: *g(n)*, the path cost function of the path from the origin to the current vertex *n*, and *h(n)*, the heuristic estimate of the distance from the current vertex *n* to the goal. For *h(n)*, two important concepts exist. The first is *admissibility*. A heuristic is *admissible* if its value is less than or equal to the

actual cost [15]. The other is *consistency*. A heuristic is *consistent* when the real cost of the path from any vertex *A* to any vertex *B* is always larger than or equal to the reduction in heuristic [16]. Once a heuristic is consistent, it is always admissible [15]. A* has been shown to obtain the optimal solution, i.e., the minimum-cost solution, whenever the heuristic is admissible [15], and, indeed, is optimally efficient among all best-first algorithms guided by path-dependent evaluation functions when the heuristic is consistent [19]. Additionally, A* is complete in the sense that it will always find a solution if one exists. The spatial and time complexity of A* depends on the heuristic and, in general, is exponential. However, A* is very fast in practice. Both Dijkstra and the Greedy algorithm can be considered as a special case of A*. Dijkstra algorithm only considers g(n) as f(n). The Greedy algorithm only uses h(n) as f(n). Frontier search is similar to A* except that Frontier search only works on data sets with consistent heuristic and does not require a closed-list to implement the search algorithm, which consequently saves space at the cost of increased computation [18].

There are also a set of A* variations such as anytime A* [20], hierarchical A* [21], MA* [22], and SMA*[23], which takes the same form *f(n)* as A* but adapts A* to different scenarios to reduce time or space complexity of A*.

All the *f(n)*s used in these identified best first searches are defined upon a single variable, vertex n, to estimate its promise.

2.2.2. Bivariate best first searches

The concept of multivariate best first searches was first proposed in [24]. A multivariate best first search uses multiple variables to specify a state to be evaluated and expanded. L#, a generalized best first search that evaluates the promise upon a state in a similar form as A*, was proposed, together with a set of novel concepts in best first searches, including local heuristic, global heuristic, local admissibility, global admissibility, local consistency, global consistency, and state graph [24]. As an instance of L#, the bivariate best-first-search C* was provided to processes Category Sequence Traversal Query (CSTQ) in a graph, which asks for a minimum cost route that starts from a given origin, traverses a set of ordered categories of interest that includes multiple objects in each category, with one selection from each category, and ends at a given destination [24]. In C*, a bivariate state instead of a single-variate state is evaluated and expanded. The state in C* is defined as the combination of a vertex and its *VisitOrder*, a discrete integer variable to indicate the order of a visiting category. A vertex may have multiple states in a graph, and not all the states of a vertex may be generated and/or expanded. Through its state specification, C* extends the theorems on optimality and optimal efficiency identified in single-variate A* to the bivariate best first search in a discrete sub state graph space.

As a bivariate best first search, C* substantially improves the ability of best first searches to process more complex queries. However, C*'s state specification is still not adequate to process OSTQ because it does not consider different visit orders of a vertex of interest. Therefore, C* can not be used to process OSTQ.

In conclusion, none of the identified searches is appropriate for acquiring optimal solutions for OSTQ in a graph. However, the characteristics of optimality and optimal efficiency from these best-first algorithms are valuable and will be identified and discussed in the proposed algorithm for OSTQ in a graph.

3. O*: a bivariate best-first-search approach to process OSTQ in a graph

This section describes the details of O*, the bivariate best first search algorithm to process OSTQs in a graph. First, for a state *s*, O* uses the same-form distance-plus-cost function *f(s)* as C*, shown in equation (1), to determine the order in which the search visits vertices in the graph [24]. Therefore, similar to C*, O* is another instance of L#. Second, O* assures that its solution traverse all points of interest.

$$f(s)=g(s)+h(s) \qquad (1)$$

where

f(s) is the estimate to the promise of a state *s* to be expanded,

g(s) is the cost from the origin state to the state *s*, and

h(s) is the estimation to the actual cost from the state *s* to the final state.

The lower the *f(s)*, the higher is the priority for a state to be expanded.

In the following subsections, first, a set of best first search concepts identified for O* is discussed, followed by the presentation of the O* algorithm. Next, how the algorithm assures the traversal of all vertices of interest is discussed, followed by the analysis of its completeness. Thereafter, the optimality and optimal efficiency for the bivariate best first search are examined and evaluated in an item-enumeration sub state graph space. Finally, the relationship between different states of a vertex is discussed.

3.1. Definition

In O*, a state $S_{i,j}$ is defined as (V_i,VL_j), where VL_j is an ordered list of j vertices of interest that are traversed along the path obtained so far at vertex V_i. A state of a vertex n describes the traversed points of interest along the partial path from the origin to the vertex n. A successor operator Γ is defined on $\{S_{i,j}\}$. Its value for each $S_{i,j}$ is a set of pairs $\{S_{k(i),j},C_{i,k(i)}\}$, where $k(i)$ represents the kth child of vertex V_i. Whenever a $S_{i,j}$ is of a vertex of interest, a Ψ operator will transform the state $S_{i,j}$ to $S_{i,j+1}$, (V_i,VL_{j+1}), at no cost by adding the vertex of interest to VL_j. A state graph *SG* defined on a graph *G* is the graph *G* whose vertices are of the same *VL*. The number of state graphs for *G* is the number of *VL*s. Since a *VL* is a sorted list storing traversed vertices of interest, in a problem with n vertices of interest, the number of state graphs is 2^n. These state graphs compose the state graph space G, an item-enumeration graph space. A sub state graph *SSG* is a portion of a state graph *SG*. It is said to be *implicitly* generated by Γ operations in *SG*. A sub state graph space G_s is a set of sub state graphs, i.e., *SSG*s. In G_s, each point represents a sub state graph which contains the expanded vertices to each of which the path from the origin has traversed the same set of vertices of interest described by an item enumeration variable. It is said to be *implicitly* generated if it is initiated with a single source state $S_{0,0}$ and a set of Γ operations and some possible Ψ operations applied to it, to its successors, and so forth. A δ sub space graph space is a special G_S with edge cost always not smaller than 0. A path from a source *S* to a goal *D*, traversing a set of vertices of interest, is an ordered set of states (V_i,VL_j). In O*, all sub state graphs are generated implicitly.

Since OSTQ traverses multiple vertices of interest between the origin and the destination, its heuristic estimation is similar to in C* but different from that in A* which is directly based on the currently generated vertex and the destination. Since O* is an instance of L#, the following concepts identified for L# [24] are also applicable to O*: *local heuristic, global heuristic, local admissibility, global admissibility, local consistency, and global consistency.* For these concepts, the only difference between in C* and O* is the state used in their corresponding definitions. For example, in O*, a global heuristic, *hg*, is estimated based on the currently generated vertex, the remaining vertices of interest to traverse, and the final goal, while C*'s global heuristic is estimated based on the remaining categories of interest to traverse instead of the remaining vertices of interest to traverse [24]. The following describes these concepts in O*.

Local heuristic is the estimate to the actual cost from the current vertex to a *subgoal, sg*. A subgoal is a vertex that is a point of interest.

Local admissibility means the local heuristic is not larger than the actual cost from the current state to a subgoal, *sg*, of the same *VisitList*.

For two vertices n and n', *local consistency* means the following inequality exists:

$$hl(n,sg) <= g^*(n, n') + hl(n',sg) \qquad (2)$$

where

hl is a local heuristic, and

$g^*(n, ,n')$ is the actual cost from *n* to *n'* in the graph.

Global heuristic is the estimate to the actual cost from the current state to estimate to the final goal.

Global admissibility means the global heuristic is not larger than the actual cost from the current state to evaluate to the final goal state.

Global consistency means the global heuristic is consistent. Suppose an optimal path is from state *n* with *VisitList vl* to state *n'* with *VisitList vl'*. By default, *vl'>=vl*. Global consistency has the following triangular inequality:

$$hg(n,vl) <= g^*((n,vl),(n', vl')) + hg(n',vl') \quad (vl' >= vl) \qquad (3)$$

Where *hg* is the global heuristic, and
g^* is the actual (minimum) cost from *(n,vl)* to *(n',vl')*.

3.2. The O Algorithm*

O* incrementally searches all paths leading from the starting vertex, traversing the vertices of interest, until it finds a path of minimum cost to the goal. It first takes the paths most likely to lead towards the goal.

Like C*, O* also maintains a set of partial solutions, unexpanded leaf states of expanded vertices. These solutions are stored in an *open list*, also called a *priority queue*, which is a sorted queue based on each element's priority. Same as in C*, the priority is assigned to a state *s* based on the function

Even though C* and O* use the same-form bivariate distance-plus-heuristic function *f(s)*, they use different state definitions.

The lower the *f(s)*, the higher is the priority for a vertex to be expanded. Whenever an equal f(s) occurs, the state with a larger *VisitList* will be the next to expand. Otherwise, one is randomly selected. State *A*'s *VisitList, vl_A*, is *larger* than State *B*'s *VisitList*, vl_B, i.e., $vl_A > vl_B$, if the length of vl_A is longer. In other words, the path from the start state to *A* traverses more vertices of interest than that to *B*.

For an N-point traversal problem, O* first generates the source state that contains the given vertex and an empty *VisitList*. For all the states in the open list, the algorithm expands the state with the lowest *f(s)* value, and its children states are generated. A child state always inherits the *VisitList* of its parent whenever the child is not a vertex of interest; otherwise, the child's *VisitList* will be incremented by adding the vertex to it. The process continues until a goal state whose vertex is the final goal and *VisitList* contains all the vertices of interest or no solution is found. Once a goal state is reached, the algorithm will retrieve the obtained path using a data structure called *backpointer*, the combination of vertex identification and *VisitList,* to recursively obtain the parent until the origin state is reached.

3.2.1. O pseudo code*

Given an estimation function for *hg(s)*, the starting vertex *S*, the goal *D*, and the vertices of interest to visit, the pseudo code for O* is provided in Figure 1. The pseudo code on global consistency will be discussed in section 3.6.

3.2.2. Time and space complexity

In O*, the complexity between a vertex and a subgoal is the same as in A*, whose time complexity and space complexity are dependent on the heuristic. For A*, in general, the time and space complexities are both exponential. Assume *b* is the branching factor, *d* is the largest depth to obtain the shortest path, and then both the time complexity and the space complexity are $O(b^d)$ [9]. A* is sub-exponential only when its heuristic *h(x)* and the actual cost *h*(x)* satisfies the following condition [9]:

$$|h(x) - h^*(x)| <= O(\log h^*(x)) \qquad (4)$$

Where
h(x): the heuristic in A*, i.e., the local heuristic in O*; and
h(x)*: the corresponding actual cost

For O*, assume b is the branching factor, d is the largest depth to obtain the shortest path between two objects including the origin, the destination, and the specified vertices of interest to traverse, which is named as a *section path* in O*, m is the number of vertices of interest specified to traverse, and F_h is the worst time complexity to obtain a global heuristic, then the state length is $m+1$, the number of state graphs is 2^m. In the worst case, O* requires traverse paths resulting from all possible order combinations of vertex of interest and store all state graphs, and each section path is exponential in time and space complexity, then the corresponding time complexity is $O(m!F_h b^d)$, and space complexity is $O(m2^m b^d)$.

To reduce the time complexity to sub-exponential, the heuristics used in O* must be sufficiently close to the actual cost to reduce the number of candidate order combinations from of permutation level to of sub-exponential level. It is beyond the scope of this research to explore in what situations this reduction in time complexity can be achieved.

3.3. Traversal constraints of vertices of interest

The solution from O* must traverse all the vertices of interest to be a candidate solution to an OSTQ.

Lemma 1: The solution from O* satisfies the traversal constraint of vertices of interest.

Proof:

Use contradiction.

Assume the solution obtained from O* does not traverse at least one vertex of interest.

Since the solution is obtained, then the search stops.

According to O*, if a subgoal, a specified vertex of interest, is not reached, then it can not be added to the *VisitList* of any vertex generated by O*, and thus there is no state whose *VisitList* contains the vertex of interest. Consequently, the final state will never be reached. In other words, the search will not stop to report a solution if there is one. This is contradicted with the assumption. So O* does provide a solution that satisfies the constraint to traverse the specified vertices of interest.

3.4. Completeness

Completeness means that an algorithm finds a solution if one exists.

Theorem 1: O* is complete.

The algorithm will not stop until either the goal is reached or there is no solution to the OSTQ.

For A* and C*, a set of theorems on optimality and optimal efficiency was proposed respectively [16][24]. However, these theorems are provided through state specification. A* is single-variate and C* is a bivariate best first search in terms of the number of variables to specify a state. Both state specifications are different from in O*. Consequently, whether the results from these theorems are still valid in the bivariate best-first-search O* is questionable. For example, C* is proved to be optimally efficient when the heuristic is consistent in an implicitly generated discrete sub state graph space, However, in O*, the search process implicitly generates a set of item-enumeration sub state graphs. Is the assumption that O* is optimally efficient when the global heuristic is globally consistent valid in an item-enumeration sub state graph space instead of in a discrete sub state graph space? The following three sections re-examine and re-evaluate the properties on optimality and optimal efficiency in the bivariate best-first-search O*.

3.5. Admissibility and optimality

Admissibility is important in A* since it guarantees the solution is optimal. This is also true in O*.

```
Starting vertex S, Goal vertex D, VisitList: a sorted list to store the traversed vertices of interest
Back pointer backpointer(vertex, VisitList)
PathVertexList that holds the vertex along the obtained path, begins as empty
bAdd2PQ=false: Indicator that indicates whether a generated vertex is put into PQ, not be put into PQ by default
Priority queue PQ(=set of generated (s(vertex,VisitList), f(s),g(s))) begins empty.
Closed list CL (= set of previously visited (s(vertex,VisitList), backpointer, f(s),g(s))) begins empty.
Put (S, VisitList =null), f=hg(S,null), and g(S,null)=0 into PQ
While (PQ is not empty)
{     remove the state with the lowest f having the largest VisitList from PQ. Name it n.
    If n is a goal, then  //a goal must be the predefined destination after all  vertices of interest are visited
    {   Succeed;
        PathVertexList.addHead(n.vertex);
        //back tracking to obtain the route
        parent=BestPath.BackPointer(n);
        while(parent.state!=null)
            PathVertexList.addHead(parent.Vertex);  parent=BestPath.backPointer(parent);
        output PathVertexList; //output the minimum-cost path}
    Else
    {   put n with its f,g,and backpointer in CL
        For each v' in successors(n.vertex)
        {       if v' is an unvisited sub goal, then
                  VisitList'=(n.VisitList).add(v'); // add n to the VisitList
              else
                  VisitList'=n.VisitList;
                g(v',VisitList')= g(n)+Cost(n.vertex,v');
                hg (v',VisitList')=calculateGlobalHeuristic(v',VisitList');
                 f (v',VisitList')=g(v',VisitList')+hg(v',VisitList')
                Process((v',VisitList'), f, g) //decide to place the state in PQ and/or to remove other states from
                  //CL/PQ}
    }
}
Process((v',VisitList'), f, g):
    bAdd2PQ=true;
    If v' not seen before, or (v',VisitList') currently in PQ with f(v',VisitList')>f Then
            Place/promote (v', VisitList') on priority queue with f ,g; END;
    If (v', VisitList*) is in PQ, Then
        If(VisitList* is a super set of  VisitList' && g(v',VisitList*)<g(v',VisitList')) Then
            bAdd2PQ=false;
        If(VisitList* is a sub set of VisitList' && g(v',VisitList*)>g(v',VisitList')) Then
            Delete (v',VisitList*) from PQ
    If(v',VisitList') previously expanded Then
        If(global consistency) Then
            bAdd2PQ=false;
        else
                If f(v',VisitList')<=f Then
                    bAdd2PQ=false;
                else
                    Delete (v',VisitList*) from the closed list
    Else
         If(v',VisitList*) is in CL Then
                If(VisitList* is a super set of  VisitList' && g(v',VisitList*)<g(v',VisitList')) Then
                    bAdd2PQ=false;
                If(VisitList* is a sub set of VisitList' && g(v',VisitList*)>g(v',VisitList')) Then
                    Delete (v',VisitList*) from CL

    If(bAdd2PQ) Then Place (v',VisitList') with f,  and g on priority queue
```


Figure 1: The pseudo code for O*

Lemma 2: If any global heuristic for any vertex is admissible, then the solution is optimal.
Proof:

Use contradiction.

Suppose O* finds a suboptimal path, ending in goal state (G_1,vl_g) where vl_g contains all the vertices of interest since the search guarantees the solution traverses all the vertices of interest, i.e., $f(G_1,vl_g) > f^*$ where $f^* = hg^*$ *(origin state)* = cost of the optimal path. Let $hg^*(n,vl)$ as the actual cost from a state (n,vl) (n is the vertex and vl is its *VisitList*) to the goal state.

There must exist a state (n,vl) which is

- Unexpanded
- The path from the origin state to (n,vl) is the start of a true optimal path

$f(n,vl) >= f(G_1, vl_g)$ (else search would not have ended)

Also $f(n,vl) = g(n,vl) + hg(n,vl) = g^*(n,vl) + hg(n,vl)$ //on optimal path

$<= g^*(n,vl) + hg^*(n,vl)$ //admissible heuristic

$= f^*$ //n is on the optimal path

So $f^* >= f(n,vl) >= f(G_1, vl_g)$

Contradicting the assumption. So the solution is optimal.

Once the global heuristic is admissible, then the solution is optimal.

An algorithm is defined as *admissible* if it is guaranteed to find an optimal path from s to the goal state for any δ graph, traversing at least once for each specified vertex of interest.

Theorem 2: Once any global heuristic is admissible, then O* provides the optimal solution to the corresponding OSTQ, i.e., O* is admissible.
Proof:

First, based on Lemma 1, O* guarantees that each specified vertex of interest is traversed at least once. Then based on Lemma 2, global admissibility guarantees solution optimality. Consequently, an optimal solution that satisfies the vertices of interest traversal constraint is the optimal solution to the corresponding OSTQ, completing the proof.

3.6. Consistency

In O*, a local state refers to the combination of a vertex and its local target, the subgoal, while a global state is determined by the vertex, and its *VisitList*. The local consistency does not care about *VisitList*, while global consistency does.

In best-first searches, consistency guarantees that a state will not be visited again once it is expanded [9][11]. Compared to the result from the consistency assumption identified in A* [11], in O*, the consistent global heuristic guarantees that a vertex with the same *VisitList* does not require consideration during the search once it is expanded. Before we prove this conclusion provided in Theorem 3, we prove another lemma first.

Lemma 3: Given the starting vertex *s*, for any nonclosed vertex *n* with *VisitList vl* and for any optimal path *P* from $(s,null)$ to (n,vl), there exists an open state (n', vl') on *P* with $g(n', vl')=g^*(n',vl')$, where *g* is the cost of the path from $(s,null)$ to (n',vl) obtained so far through O*, and g^* is the actual cost.
Proof:

Let $P = ((s,null),\ldots, (n_i,VL_{ni})\ldots,(n,vl))$, a collection of pairs of a vertex and its *VisitList* . If $(s,null)$ is open (that is, O* has not completed even one iteration), let $n'=s$, and the lemma is trivially true since $g(s,null) = g^*(s,null)=0$.

Suppose s is closed. Let *CL* be the set of all closed states (n_i,VL_{ni}) in *P* for which $g(n_i, VL_{ni}) = g^*(n_i, VL_{ni})$. *CL* is not empty, since by assumption $(s,null)$ is within *CL*. Let n^* with *VisitList* vl^* be the element of *CL* with the highest index. Clearly, (n^*,vl^*) is not (n,vl), as (n,vl) is nonclosed. Let n' with *VisitList* vl' be the successor of n^* on *P*. (Possibly $(n',vl') = (n,vl)$.) Now by definition of g,

$g(n',vl')<=g(n^*,vl^*)+C(n^*,n', vl^*->vl')$

where $C(n^*, n', vl^*->vl')$ is the actual cost from n^* to n' traversing a set of vertices of interest between them, and can also be formulated as $C((n^*,vl^*),(n',vl'))$;

$g(n^*,vl^*) = g^*(n^*, vl^*)$ because (n^*,vl^*) is in *CL*, and

$g^*(n',vl')=g^*(n^*,vl^*)+C(n^*,n', vl^*->vl')$ because P is an optimal path. Therefore, $g(n',vl') <= g^*(n',vl')$. But in general, $g(n',vl')>=g^*(n',vl')$, since the lowest cost $g(n',vl')$ from $(s,null)$ to (n',vl') explored at any time is never lower than the optimal cost $g^*(n',vl')$. Thus $g(n',vl') = g^*(n',vl')$, and moreover, n' must be open by the definition of CL.

Based on the lemma, the following corollary can be derived.

Corollary

Suppose $hg(n,vl) < hg^*(n,vl)$ for any state n with *VisitList* vl, and suppose O* has not terminated. Then, for any optimal path P from *orgin* s to the goal state of D, there exists an open state (n',vl') on P with

$f(n',vl') <=f(s,null)$.

Proof:

By Lemma 3, there exists an open state n' on P with $g(n',vl') = g^*(n',vl')$, so by definition of f,

$f(n',vl') = g(n',vl') + hg(n',vl')=g^*(n',vl') + hg(n',vl')<=g^*(n',vl')+hg^*(n',vl')=f^*(n',vl')$.

But P is an optimal path, so $f^*(n',vl') = f(s,null)$ for all n' in P, completing the proof.

Now, based on Lemma 3, the following theorem states that global consistency guarantees a closed state will not be expanded.

Theorem 3: Assume any global heuristic is consistent, and a state (n, vl) is closed. Then $g(n,vl)=g^*(n,vl)$, where $g(n,vl)$ is the cost of the path, from the starting vertex s, traversing the vertices of interest in vl, to n, obtained so far through O*, and $g^*(n,vl)$ is the actual cost.

Proof:

Use contradiction.

Consider the sub state graph space Gs just before closing (n,vl), and suppose that $g(n,vl)>g^*(n,vl)$. Then there must be some optimal path P from $(s,null)$ to (n,vl). Since $g(n,vl)>g^*(n,vl)$, O* does not find P.

According to Lemma 3, there exists an open state (n',vl') on P with $g(n',vl') = g^*(n',vl')$. If $(n',vl') = (n,vl)$, we have proved the theorem. Otherwise,

$g^*(n,vl)=g^*(n',vl')+h^*(n',n, vl'->vl)$

where h^* is the actual cost of the shortest path starting from n', traversing the additional vertices of interest in vl' compared to vl, and ending at n. It can also be formulated as $h^*((n',vl'),(n,vl))$.

$=g(n',vl')+h^*(n',n, vl'->vl)$.

Thus,

$g(n,vl)>g(n',vl')+h^*(n',n, vl'->vl)$.

Adding the global heuristic, $hg(n,vl)$, to both sides yields

$g(n,vl) + hg(n,vl) > g(n',vl')+h^*(n',n, vl'->vl)+hg(n,vl)$.

Since global consistency is satisfied, then

$h^*(n',n, vl'->vl)+hg(n,vl)>=hg(n',vl')$,

then

$g(n,vl)+hg(n,vl)>g(n',vl')+hg(n',vl')$

or

$f(n,vl)>f(n',vl')$

Contradicting the fact that O* chose (n,vl) for expansion while (n',vl') was on the open list. Completing the proof.

Theorem 3 tells us that once a state (v, vl) is closed, whenever a new path to (v,vl) is generated, O* can just ignore it. The code block on global consistency in Figure 1 reflects this fact.

The next theorem states that $f(s)$ is monotonically non-decreasing on the sequence of states closed by O*.

Theorem 4: Let $((s,null),\ldots, (n_i,VL_{ni})\ldots,(n,vl))$ ((x,y), x represents the vertex, and y represents *VisitList*) be the sequence of states closed by O*. Then, if the global consistency assumption is satisfied, $p<=q$ implies $f(n_p,VL_{np})<= f(n_q,VL_{nq})$.

Proof:

Let (n,VL_n) be the next state closed by O* after closing (m,VL_m). Suppose first that the optimal path to (n,VL_n) does not traverse (m,VL_m), then (n,VL_n) was available when (m,VL_m) was selected,

and the theorem is trivially true. Now suppose that the optimal path to (n,VL_n) does traverse (m,VL_m), then

$g^*(n,VL_n))=g^*(m,VL_m)+h^*(m,n, VL_m->VL_n)$.

According to Theorem 3,

$g(n,VL_n))=g^*(n,VL_n)$ *and* $g(m,VL_m))=g^*(m,VL_m)$,

Since global consistency is satisfied,

$f(n, VL_n)=g(n, VL_n)+hg(n,VL_n) = g^*(n, VL_n)+hg(n,VL_n)$

$= g^*(m, VL_m)+h^*(m,n, VL_m->VL_n)+hg(n,VL_n)$

$>= g^*(m, VL_m)+hg(m, VL_m)= g (m, VL_m)+hg(m,VL_m)=f(m,VL_m)$

So $f(m,VL_m)<= f(n, VL_n)$,

completing the proof.

The following corollary is immediately available from the theorem.

Corollary

Under the premises of theorem 4, if the state *sn* is closed, then *f(sn)<=f*((s,null))*.

Proof:

Let o be the goal state found by O*, then

$f(sn)<=f(o)=f^*(o)=f^*((s,null))$.

3.7. Consistency and optimal efficiency

A* is optimally efficient when the heuristic is consistent. Since O* uses best-first search like A*, O* holds the property of optimal efficiency, which means O* is optimally efficient among all best-first search admissible algorithms for OSTQ once some preconditions are satisfied.

A set of procedures was identified to prove A* and C* are optimal efficient in terms of the number of vertices/states to expand in a sub graph and a discrete sub state graph space respectively [10][11]. We adopt these procedures to prove O* also demonstrates the optimal efficiency property in an item-enumeration sub state graph space.

Next we prove a theorem on the optimal efficiency of O* as compared with any other admissible algorithm O that uses *no more information* about the problem than does O*.

Let θ_s^O be the state index set used by algorithm O at state *s*. Then if $\theta_s^{O^*} \subset \theta_s^O$ for all state *s* in G_s, it can be stated that algorithm O is *no more informed* than algorithm O*. The following theorem indicates that if an admissible algorithm O is no more informed than O*, then any state (*vertex n*, *VisitList vl*) expanded by O* must also be expanded by O. Two special cases can occur: ties occur in the value of f used by O*, or ties never occur. We prove the theorem for both cases.

Theorem 5: Let O be any admissible algorithm no more informed than O*. Let *Gs* be any δ sub state graph space such that given any two states, *(n,vl)* and *(n',vl')*,

(n,vl)≠(n',vl')

⇒f(n,vl) ≠ f(n',vl')

Let the consistency assumption be satisfied by the *hg* used in O*. Then if state *(n,vl)* was expanded by O*, it was also expanded by O.

Proof:

Use contradiction.

Suppose there exists some state *st*, which represents *(n,vl)*, expanded by O* but not by O. Let *o** and o be the goal states of origin *s* found by O* and O, respectively.

Since O* and O are both admissible,

$f(o^*)=g(o^*)+h(o^*)=g^*(o^*)+0=f^*(o^*)=f^*(o)=f^*((s,null))$.

Since O* must have expanded *st* before closing *o**, by theorem 4, and since no ties are allowed, we have

$f(st) <f(o^*) = f^*(o)$

There exists some sub state graph space $G_{s,\theta}$, $\theta \in \Theta_s^O$, for which $hg(st)=hg^*(st)$ by the definition of *hg*. By theorem 4, $g(st)=g^*(st)$. Then in the sub state graph space $G_{s,\theta}$, $f(st)=f^*(st)$.

Since O is no more informed than O*, O can not exclude the existence of $G_{s,\theta}$; but it does not expand *st* before termination and thus is inadmissible, contradicting the assumption and completing the proof.

Define *N(O,Gs)* as the total number of states in *Gs* expanded by the algorithm O, based on the theorem, we have the following corollary.

Corollary

Under the premises of Theorem 5, the following inequality exists:

N(O*, *Gs*) < N(O, *Gs*)

with equality if and only if O expands the identical set of states as O*.

From this perspective, O* is claimed to be an optimal algorithm. O* expands the least possible states necessary to guarantee an optimal solution compared with any other no more informed admissible algorithms.

When ties exist, O* randomly expands one of the tied states whenever they have the least *f*. Now, assume the set O^* includes all algorithms that performs identically to O* but process ties differently. In other words, a member of O^* is the original O* with a random tie-breaking rule.

The next theorem extends theorem 5 to the situation where ties occur. The statement points out that for any admissible algorithm O, a member O* of O^* can always be found so that any state expanded by O* is always expanded by O.

Theorem 6: Let O be any admissible algorithm no more informed than the algorithms in O^*, and suppose the global consistency assumption is satisfied by the *hg* used in the algorithms in O^*. Then for any δ sub state graph space *Gs*, there exists an O*∈ O^*, such that every state expanded by O* is also expanded by O.

Proof:

Let O_1* be any algorithm in O^*. If every state of *Gs* expanded by O_1* is also expanded by O, then let O_1* be the O* of the theorem. Otherwise, the O* of the theorem can be constructed through the change of the tie-breaking rule of O_1*.

Let *S* as the set of the states expanded by O, and let *P=((s,null),…, (ni,*VL_{ni}*)…,(d,vl))((x,y), x* is a vertex, and *y* is the corresponding *VisitList , s* is the start state and *d* is a goal state*)* be the optimal path found by O.

As long as states chosen for expansion are elements of *S*, they are expanded as prescribed by O_1*. Let o be the first state chosen for expansion by O_1*, which is not in *S*, then *f (o) <= f*(s,null)* according to the corollary to Theorem 4. Since *f(o)<f*((s,null))=f*(gs) (gs is the goal state)* would imply that O is inadmissible (based on the argument of Theorem 5), it is true that *f(o)=f*((s,null))*. At the time O_1* selected *o*, goal state *gs* was not closed (or O_1* would have been terminated). Then by the corollary to Lemma 3, there is an open state *o'* on *P* such that *f(o')<=f*((s,null))=f(o)*. But since o was selected for expansion by O_1* instead of *o'*, *f(o)<=f(o')*.Consequently, *f(o')=f(o)*. Let O_2* be identical to O_1* except that the tie-breaking rule is modified just enough to choose *o'* instead of o. By repeating the above argument, we obtain for some *i* an O_i*∈ O^* that expands only states that are also expanded by O, completing the proof of the theorem.

Based on the theorem, Corollary 1 exists.

Corollary 1

Suppose the premises of theorem 6 are satisfied. Then for any δ sub state graph space *Gs*, there exists an O*∈ O^* such that N(O*,*Gs*)<=N(O,*Gs*), with equality if and only if O expands the identical set of states as O*.

It is also meaningful to know how all members of O^* perform against any O in the number of states expanded. Let's define a critical tie between state *sn* and state *sn'* as one for which *f(sn) =f(sn')=f*((s,null))*. Then the second corollary exists.

Corollary 2

Suppose the premises of theorem 6 are satisfied. Let R(O*, Gs) as the number of critical ties which occurred in the course of applying O* to *Gs*, then for any δ sub state graph space *Gs* and any O*∈ O^*,

N(O*, *Gs*)<=N(O,*Gs*)+R(O*,*Gs*).

Proof:

For any noncritical tie, all alternative states must be expanded by both O and O*, or O could not be admissible. Consequently, it only requires observe that each state expanded by O* but not by O absolutely corresponds to a different critical tie where O*'s tie-breaking rule results in an inappropriate decision.

3.8. Relationship between different states of a vertex

Since in a bivariate best first search, a vertex may have multiple states, the relationship between any two of its states can be used to prune unnecessary states.

Theorem 7: For a vertex *v*,

If(g(v,VL')>g(v,VL) AND VL is a super or equal set of VL'), then the state *(v,VL')* is not on the optimal path.

Proof:

Since *VL is a super or equal set of VL'*, which means *VL* contains at least the same set of traversed vertices of interest as *VL'*, it is clear that $g^*(v,VL)>=g^*(v,VL')$. According to the given condition, $g(v,VL')>g(v,VL)$, then $g(v,VL')>g^*(v,VL)$, so $g(v,VL')>g^*(v,VL')$. Consequently, *(v,VL')* is not on the optimal path. Completing the proof.

4. MST heuristic: A globally admissible heuristic

During O* search, it is natural to calculate the global heuristic for a vertex considering all order combinations of the remaining vertices of interest. However, due to the large number of permutations, it rapidly becomes impractical. Consequently, in O*, to process OSTQ with a large number of vertices of interest, it is necessary to efficiently calculate the global heuristics, without explicit consideration of order combinations. To illustrate how O* works, a heuristic is identified as follows.

A Minimum Spanning Tree (MST) is the one with the minimum total edge cost among all spanning trees that connect all the vertices together in a graph. In a graph whose edge cost satisfies the triangle inequality, a minimum spanning tree, MST-P, can be built from an optimal path for an OSTQ, so the total edge cost of MST-P is not larger than of the optimal path. Similarly, another minimum spanning tree, MST-G, can be build upon the whole graph. Since the optimal path is within the graph, the total edge cost of MST-G is not larger than the total edge cost of MST-P, and consequently not larger than the cost of the optimal path. In other words, MST-G provides a lower cost bound to an optimal path for an OSTQ, and thus the heuristic provided by a MST, *MST global heuristic*, is globally admissible. We define the O* algorithm that uses MST to provide global heuristics as O*-MST. The Prim algorithm [4] used to calculate the MST is easy to implement and its time complexity is $O(n^2)$ (n is the number of vertices to traverse), which is efficient to calculate global heuristics in a dense graph.

MST global heuristic is calculated based on the fully connected graph that is composed of targeted vertices plus the currently generated node and the goal vertex. The pseudo code to obtain the cost of MST is shown in Figure 2.

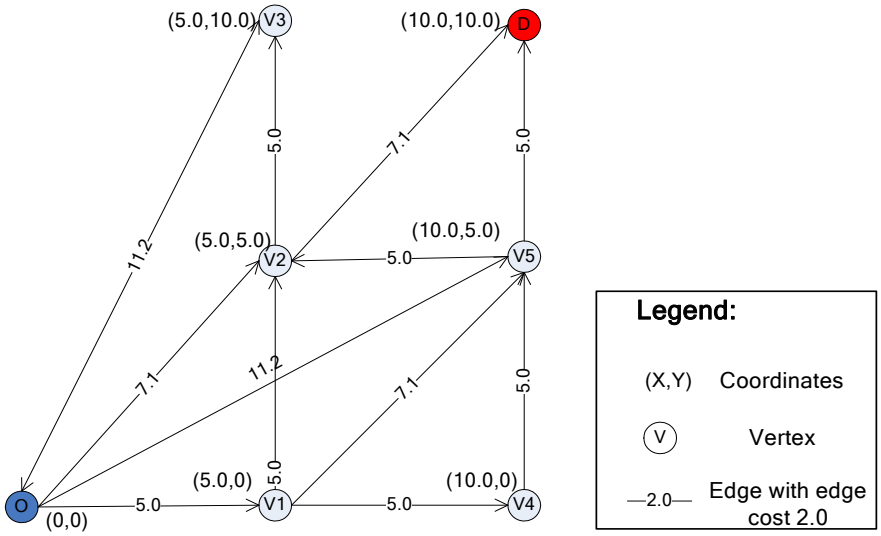
Input: A fully connected weighted graph with vertices V and edges E.
Initialize: V_{new} = {n}, where n is an arbitrary vertex (starting point) from V, E_{new}= {}
Repeat until V_{new}=V:
Choose edge (x,y) from E with minimal weight such that x is in V_{new} and y is not
(Choose arbitrarily if there are multiple edges with the same weight)
Add y to V_{new}, add (x,y) to E_{new}
Output: The sum of the weights of all Es in E_{new}

Figure 2: the pseudo code to calculate MST global heuristic

4.1 An example

To illustrate how the algorithm works, the following example is provided. Figure 3 shows a directed graph composed of the starting vertex *O*, the goal *D*, and five ordinary vertices marked as *V1*-*V5*. An OSTQ demands an optimal path starting from *O*, traversing the interested vertices *V2* and *V5*, and ending at *D*. Euclidean distance is adopted to calculate the distance between any two vertices using their coordinates, and the MST global heuristic is used in this example. Since Euclidean distance obeys triangle inequality, the MST global heuristic is admissible.

The search process of O* is shown in Figure 4, the coordinates are used to calculate the cost of the minimum spanning tree. The Euclidean length between two points is considered as the cost between them.



Origin: Vertex O; Destination: Vertex D; Vertices of interest to traverse: V2 and V5.

Figure 3: An example: The graph and the OSTQ

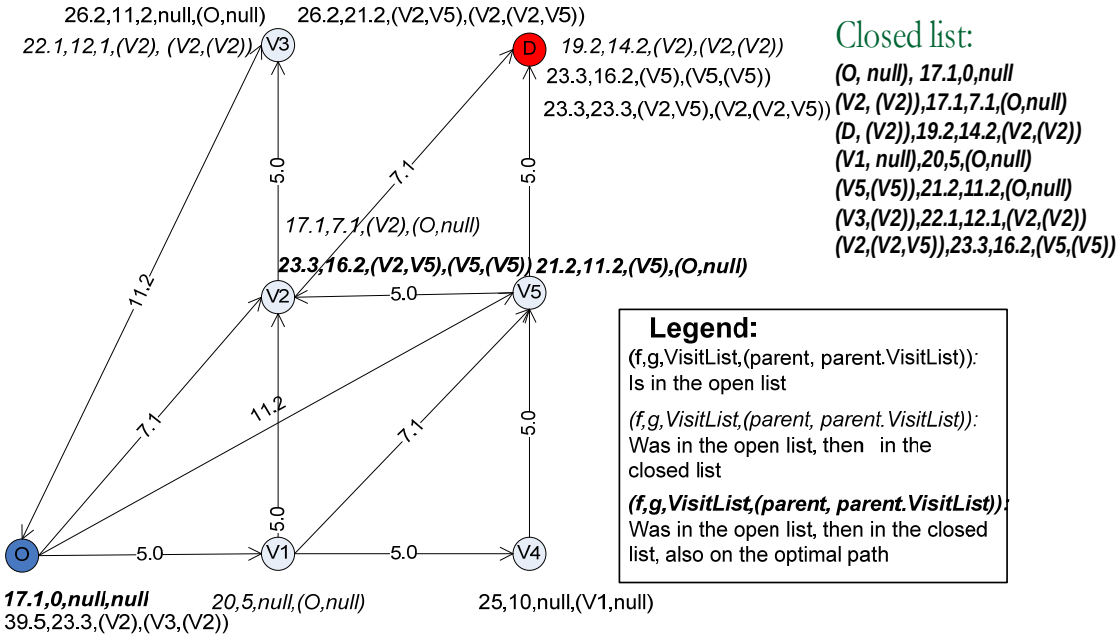


Figure 4: The O* search process

The algorithm begins with the starting point *O* with *VisitList* as null and parent (*parentpointer)* as null. Its *f* value is 17.1, the cost of the minimum spanning tree of *O*,*V2*,*V5*, and *D*. The state is put into the priority queue. Then, the state of the lowest *f* value, vertex *O* with *VisitList* null is expanded first and put into the closed list with a null *backpointer*, and its four children, *V1*, *V2*, *V3*, and *V5*, are generated. Neither *V1* nor *V3* is a subgoal, so they are normal vertices in this case, and their *VisitList*s remain null. *V2* and *V5* are subgoal, so their *VisitList*s are updated accordingly. For each of these four vertices, O* calculates its heuristic based on its corresponding minimum spanning tree, and then calculates its *f* value. All generated states are put into the priority queue.

Next, since vertex *V2* has the lowest *f*, 17.1, it is expanded and put into the closed list with a backpointer pointing to *(O,null)*, and its children *V3, and D* are generated. Neither *V3* nor *D* is a subgoal, so their *VisitList*s remain the same. The generated two states are put into the open list.

Now *D* has the lowest *f*, but it does not reach the goal state yet. So *D* is expanded. *D* has no child, so no new state is generated.

Again, based on the lowest *f*, O* puts *V1* in the closed list, expands it, and generates its 3 children, *V2, V5*, and *V4*. Since *V2* is a subgoal, *V2* is added to *V1*'s *VisitList.* Now the closed list already has a closed state of *V2* with *VisitList* as *(V2)*, and its *f* is smaller, therefore the new generated state of *V2* is discarded. *V5* is also a subgoal, so *V5* is added to *V1*'s *VisitList* to become *V5*'s *VisitList*. However, *V5* with the same *VisitList* was already generated in the open list and its *f* is smaller, and thus the new generated state is discarded. *V4* is not a subgoal, so its *VisitList* is the same as *V1*'s and the new state of *V4* is put into the open list.

Next, *V5* has the lowest *f*, so it is put into the closed list and then expanded. Its two children, *V2* and *D* are generated and their new states are put into the open list. Again, *V2* is a subgoal and thus its *VisitList* becomes *(V2,V5)* (the order is rearranged since *VisitList* is a sorted list). *D* is still not a subgoal, and thus its *VisitList* is the same as *V5*'s.

Then, *V3* has the lowest f, so it is put into the closed list and then expanded. A new state for its child, *O,* is generated and put into the priority queue.

Now, both *V2* and *D* have the lowest *f*, but *V2*'s *VisitList* contains *D*'s *VisitList*, so *V2* is put into the closed list and expanded, and two new states of *V2* and *D* are generated and put into the open list.

At last, *D* has the lowest *f*, so it is put into the closed list and expanded, and it is a final goal, so the whole search process stops. The optimal path is found. Thereafter, the backtracking module works to retrieve the best path:

(D,(V2,V5))->(V2,(V2,V5))->(V5,(V5))->(O,null) ->null

In Figure 4, the elements in normal style are in the open list; those in italic style were in the open list first, and then moved to the closed list; and those in both italic and bold styles were initially put into the open list, then moved to the closed list, and are on the optimal path. The closed list in Figure 4 is its snapshot before the final state is expanded by O*-MST.

5. Experiments

The purpose of the experiments is to test the performance of O*'s three special cases over different numbers of vertices of interest, and different origin/destination locations. To see how the heuristic works, the special case of O* when *hg(s)=0* is used as the baseline. We name this special case as O*-Dijkstra because it is a consecutive network expansion algorithm, which always expands the node having the minimum-cost path from the source [5]. Whenever O*-Dijkstra reaches a subgoal, it will start network expansion based on the subgoal until either the final goal or another subgoal is reached. To see how O* approximately resolves an OSTQ or a TSP, O*-Greedy, another special case of O* when *g(s)=0* is used to calculate *f(s)*, is also studied. O*-Greedy is greedy since it expands a state that appears to be the closet to the goal, which may generate a sub-optimal solution.

The global heuristic used in the experiments is the MST global heuristic, and we name the corresponding O* as O*-MST. To calculate MST heuristics, for any two points in a MST, we use their Euclidean distance, which is always not larger than their actual network distance. Also, Euclidean distances satisfy the triangle inequality, so O*-MST is optimal in this experiment.

For each data set, 30 runs were performed.

O* was implemented with Visual C#. The experiments were performed on a Dell desktop with 3.5 GB memory, 2.8 GHZ Pentium(R) 4 CPU, and Windows XP operating system.

5.1. Data set

A graph that represents the transportation network of Fairfax City, Virginia, USA is adopted. The street network is shown in Figure 5. The graph has 235 vertices and 740 directed edges.

To illustrate how O* works in different scenarios, two data sets were used. Data in the first set includes real origins and randomly generated destinations and vertices of interest. The data in the second set are all randomly generated. Figure 5 shows the spatial locations of the real data in the first data set. Table 1 summarizes the characteristics of the two data sets.

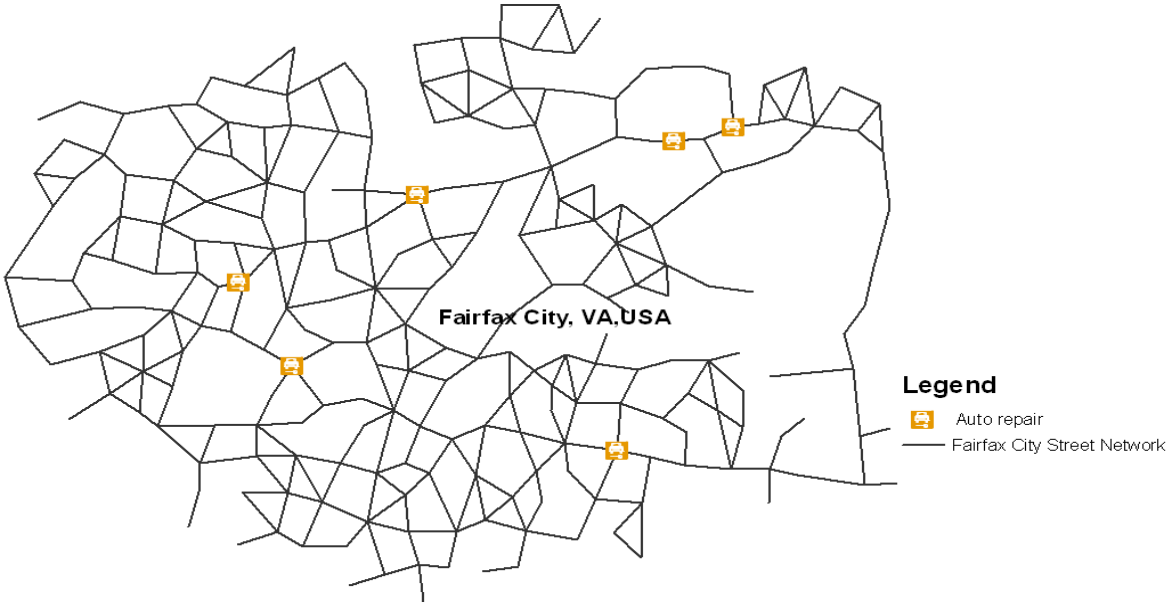


Figure 5: The street network and the collected real locations used in the first data set

Table 1: Characteristics of two data sets

Data set No.	Origin	Destination	vertices of interest
I	Six auto repair centers	Randomly generated	Randomly generated
II	Randomly generated	Randomly generated	Randomly generated

5.2. Performance measures

The following measures were used to analyze the performance of the three O* special cases.

Average Obtained Distance (AOD): the average route distance obtained over all runs, in mile;

Average Number of States Expanded (ANSE): the average number of expanded states obtained over all runs;

Average Relative Number of States expanded (ARNS): the ratio of the number of states expanded by either O*-Dijkstra or O*-Greedy over by O*-MST;

Maximum Relative Number of States expanded (MaxRNS): For each run of O*-Dijkstra or O*-Greedy, obtain the ratio of its number of expanded states over the number of states expanded by O*-MST for the same problem, and the measure is the maximum ratio among all runs;

Minimum Relative Number of States expanded (MinRNS): For each run of O*-Dijkstra or O*-Greedy, obtain the ratio of its number of expanded states over the number of states expanded by O*-MST for the same problem, and the measure is the minimum ratio among all runs;

Average Process Time (APT): the time required to return the solution for a query, in second;

Average Relative Process Time (ARPT): the ratio of the time processed by O*-Dijkstra or O*-Greedy over by O*-MST;

Maximum Relative Process Time (MaxRPT): For each run of O*-Dijkstra or O*-Greedy, obtain the ratio of its process time over that of O*-MST, and the measure is the maximum ratio among all runs; and

Minimum Relative Process Time (MinRPT): For each run of O*-Dijkstra or O*-Greedy, obtain the ratio of its process time over that of O*-MST, and the measure is the minimum ratio among all runs.

The results are shown in Table 2 and Table 3. In both tables, OD represents O*-Dijkstra, and OG represents O*-Greedy.

5.3. Results

The results are provided in Table 2 through Table 5. In Table 4 and Table 7, a dash indicates that a value for OD is not available since the time to obtain a result exceeded a reasonable expected solution time. Figure 6 through Figure 11 visualize the major results in these tables.

Table 2: Performance results on expanded states in data set I

Number of vertices of interest	***AOD***			***ANSE***			***ARNS***		***MaxRNS***		***MinRNS***	
	OD	**O***	**OG**	**OD**	**O***	**OG**	**OD**	**OG**	**OD**	**OG**	**OD**	**OG**
2	5.5	5.5	7.4	859	292	88	2.9	0.3	5.4	0.4	2.1	0.1
3	6.3	6.3	9.1	2159	483	283	4.5	0.6	6.0	0.9	3.9	0.1
4	6.7	6.7	8.7	5276	933	179	5.7	0.2	13.0	0.3	4.2	0.1
5	8.3	**8.3**	**14.1**	19184	2898	310	6.6	0.1	7.8	0.1	6.0	0.1

Table 3: Performance results on process time in data set I

Number of vertices of interest	***APT***			***ARPT***		***MaxRPT***		***MinRPT***	
	OD	**O***	**OG**	**OD**	**OG**	**OD**	**OG**	**OD**	**OG**
2	0.48	0.09	0.08	4.3	0.9	11.2	0.5	3.4	0.0
3	2.95	0.20	0.19	14.8	1.0	24.3	0.6	11.0	0.0
4	20.85	0.91	0.05	22.9	0.0	92.1	0.1	16.0	0.0
5	266.4	6.49	0.10	41.4	0.0	61.2	0.0	34.0	0.0

Table 4: Performance results on expanded states in data set II

Number of vertices of interest	***AOD***			***ANSE***			***ARNS***		***MaxRNS***		***MinRNS***	
	OD	**O***	**OG**	**OD**	**O***	**OG**	**OD**	**OG**	**OD**	**OG**	**OD**	**OG**
2	6.0	6.0	7.9	870	362	141	2.4	0.3	4.2	**1.1**	1.3	0.1
3	6.5	6.5	8.6	1860	503	220	3.7	0.4	5.7	1.0	1.5	0.1
4	6.7	6.7	8.9	5341	1012	230	5.3	0.2	8.9	1.0	2.9	0.0
5	8.2	8.2	11.6	17115	2479	343	6.9	0.1	13	**4.4**	3.4	0.0
6	9.0	9.0	12.5	51923	3921	220	13.0	0.0	20	0.3	4.3	0.0
7	-	9.6	12.7	-	5643	183	-	0.0	-	0.2	-	0.0
8	-	9.9	12.9	-	10147	172	-	0.0	-	0.1	-	0.0
9	-	10.0	13.0	-	19341	188	-	0.0	-	0.0	-	0.0
10	-	11.5	14.8	-	69095	232	-	0.0	-	0.0	-	0.0
11	-	11.2	14.5	-	36783	203	-	0.0	-	0.0	-	0.0
12	-	12.5	18.9	-	384587	339	-	0.0	-	0.0		0.0

Table 5: Performance results on process time in data set II

Number of vertices of interest	APT			ARPT		MaxRPT		MinRPT	
	OD	O*	OG	OD	OG	OD	OG	OD	OG
2	0.28	0.07	0.02	4	0.3	6.3	0.8	2.8	0.0
3	1.25	0.12	0.04	10.4	0.3	14.4	0.8	7.2	0.0
4	12.12	0.54	0.05	22.4	0.9	30.2	0.6	8.9	0.0
5	146.54	3.22	0.14	45.5	0.0	69	**5.1**	25.3	0.0
6	1323.87	8.53	0.05	155.2	0.0	275.2	0.1	59.4	0.0
7	-	14.87	0.04	-	0.0	-	0.0	-	0.0
8	-	30.34	0.04	-	0.0	-	0.0	-	0.0
9	-	10.67	0.05	-	0.0	-	0.0	-	0.0
10	-	319.95	0.03	-	0.0	-	0.0	-	0.0
11	-	105.43	0.04	-	0.0	-	0.0	-	0.0
12	-	2949.68	0.11	-	0.0	-	0.0	-	0.0

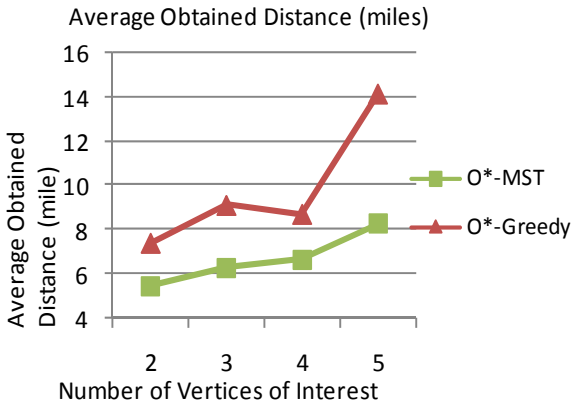


Figure 6: Average obtained distances of different algorithms in data set I

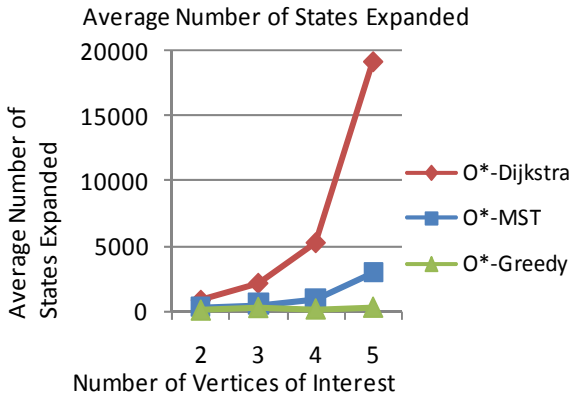


Figure 7: Average number of states expanded of different algorithms in data set I

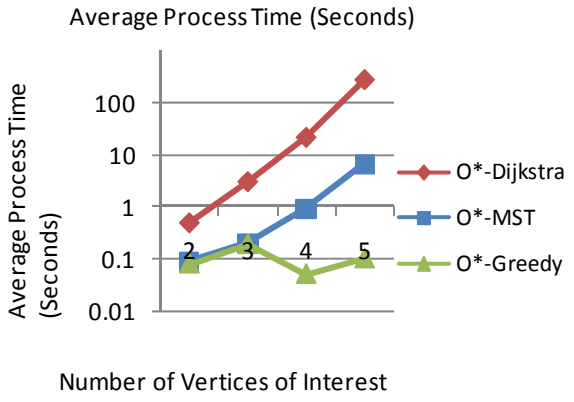


Figure 8: Average process time of different algorithms in data set I

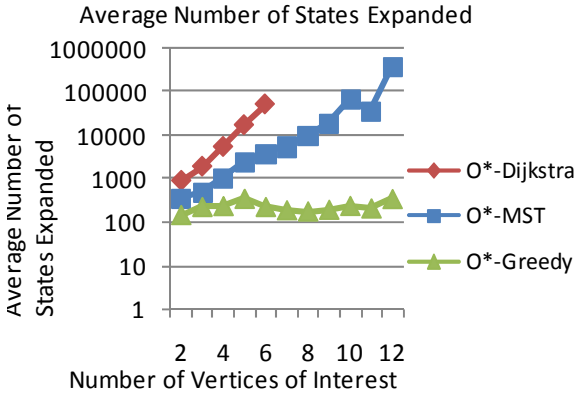


Figure 9: Average number of states expanded of different algorithms in data set II

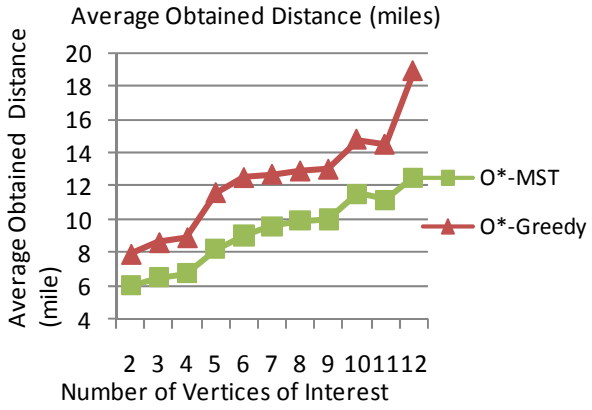


Figure 10: Average obtained distance of different algorithms in data set II

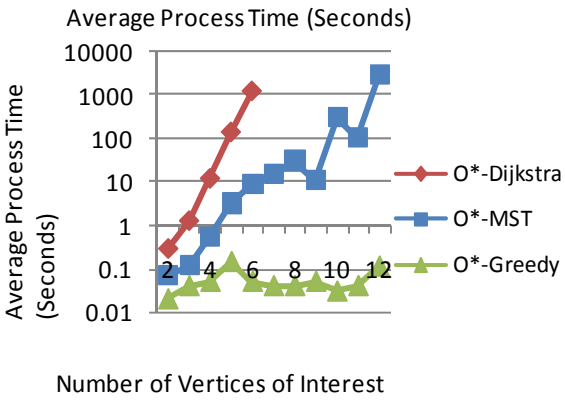


Figure 11: Average process time of different algorithms in data set II

From the results presented in the tables and figures, conclusions are drawn as follows:

Since the global heuristic of O*-Dijkstra is always 0 and thus is always consistent, O*-Dijkstra is admissible. In other words, O*-Dijkstra always retrieves the optimal solution. Furthermore, since MST heuristics in a transportation network are admissible, O*-MST always retrieves the optimal solution. O*-Greedy, however, is greedy and expands a state that appears to be the closet to the goal, which means it finds a sub-optimal solution. The AOD measure values for O*-Dijkstra and O*-MST from Table 2 and Table 4 are consistent to this rule. According to Figure 6 and Figure 10, O*-Greedy retrieves sub-optimal solutions. The worst case of O*-Greedy, as highlighted in Table 2, retrieves an average route of 70% times longer than that of the corresponding O*-MST.

As shown in Figure 7 and Figure 9, in general, for O*-Dijkstra and O*-MST, the larger the number of vertices of interest to traverse, the larger the number of states to expand, and the longer the time to process the query. It also seems the number of expanded states is at least exponential in terms of the number of vertices of interest.

The MinRNS values in Table 2and Table 4 show that O*-MST expands less states than O*-Dijkstra. This is due to the fact that O*-MST uses more information from the domain, i.e., O*-MST is more informed than O*-Dijkstra.

The ARNS and MaxRNS values in Table 2 and Table 4 show that O*-Greedy expands less than O*-MST at average, but sometimes it may expand more states than O*-MST, which is highlighted in bold for MaxRNS in Table 4. This is because O*-Greedy is greedy, only finding a suboptimal solution. Accordingly, O*-Greedy is much faster than O*-MST on average, only occasionally slower as the case highlighted in bold for MaxRPT in Table 5.

Based on the ARNS values, the larger the number of vertices of interest, the larger the difference of the number of expanded states between O*-Dijkstra and O*-MST, or, the worse the performance of O*-Dijkstra compared to O*-MST. On the contrary, compared to O*-MST, O*-Greedy performs better with the increase of the number of vertices of interest.

According to Figure 8 and Figure 11, on average, O*-Dijstra is the slowest among all, and O*-MST retrieving optimal solutions is still slower than O*-Greedy retrieving suboptimal solutions. Even through O*-MST requires additional time to compute the global heuristics, according to the MinRPT values, O*-MST still always performs faster than O*-Dijkstra.

Consistent to the observation to the ARNS values, based on the ARPT values, O*-Dijkstra performs worse while O*-Greedy performs better with the increase of the number of vertices of interest when compared to O*-MST.

Since O*-Greedy performs very fast, it is beneficial to show how it performs with more vertices of interest. So another set of experiments was performed to study the behavior of O*-Greedy on data set II. The number of vertices of interest consecutively changes from 2 to 115.

For each number of vertices of interest, the origin, the destination, and the vertices of interest to traverse were randomly chosen and the number of runs is 30.

Figure 12 shows the relationship between the average number of expanded states and the number of vertices of interest. The number of expanded states does not vary much and occasionally several peaks occur, which implies that the computation complexity of O*-Greedy still depends on the heuristic estimation, same as of O*. Generally, O*-Greedy's performance is relatively stable.

Figure 13 shows the relationship between the average process time and the number of vertices of interest. The curve shows a slow but relatively steady exponential trend. The average process time is still within 40 seconds for all cases.

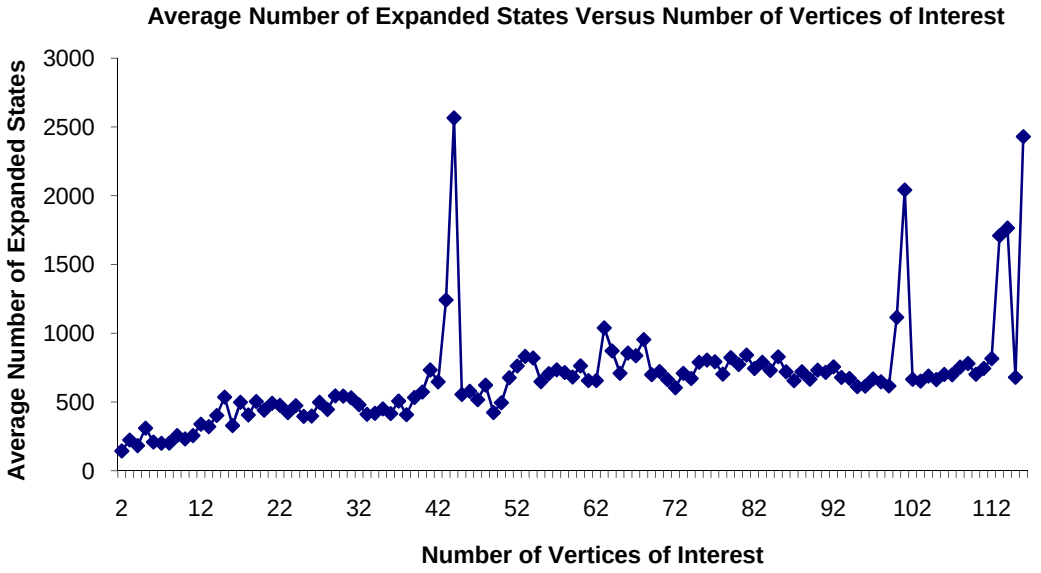


Figure 12: The average number of expanded states versus the number of vertices of interest

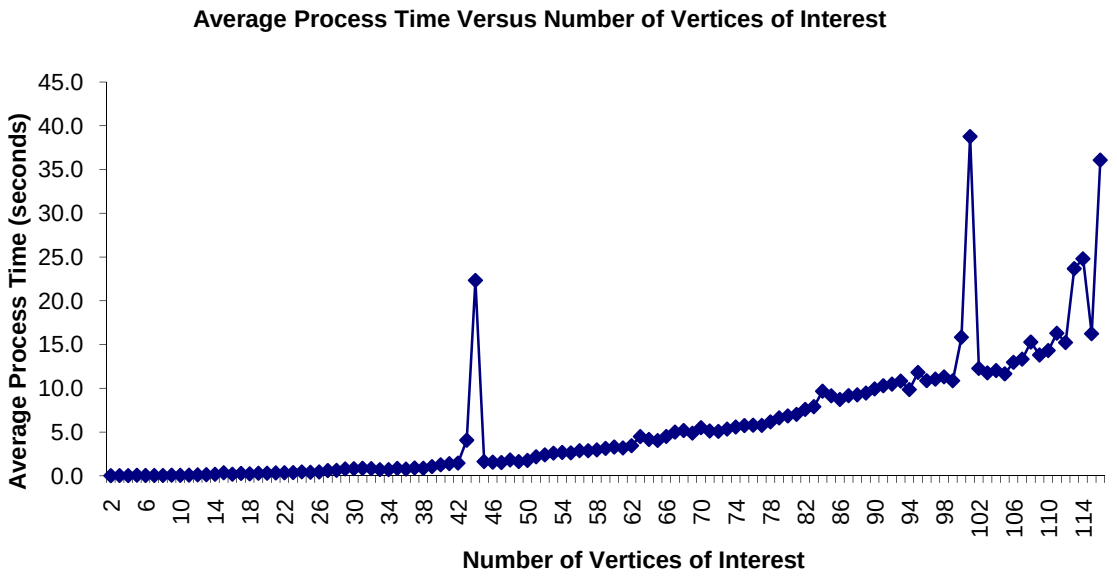


Figure 13: The average process time versus the number of vertices of interest

6. The power of the estimate *hg*

Similar to A* and C*, O* is also a family of algorithms. Its choice of a particular function *hg* corresponds to a particular algorithm from the family. *hg* can be used to tune O* for a specific application.

If *hg* is 0, then O* knows, or at least takes advantage of, absolutely no information from the problem domain.

In general, however, the problem domain can provide some useful information that O* can use. For example, in an OSTQ problem to retrieve a shortest route from home, traversing a set of consumer destinations, to a work place, the Euclidean distance between any two consumer destinations or between the starting point and a consumer destination or between a consumer destination and the work place is known and no larger than the corresponding actual network distance. In this case, any local heuristic is admissible, thus *hg* is admissible, which guarantees an optimal solution. In this case, O* can reduce some vertices to expand compared to those algorithms knowing or using nothing from the problem domain.

To further reduce the computation, *hg* can be larger than the actual cost, and thus *hg* is not admissible. From a heuristic point of view, this might be more desirable because sometimes an admissible heuristic is not available or is hard to obtain. The result may still be optimal but is more likely to be suboptimal. Also, fewer vertices may be expanded. From another perspective, a large number of approximate algorithms have been developed to obtain sub-optimal solutions to TSP in a fully connected graph, which can be directly used as the heuristic estimation in O* to obtain suboptimal solutions.

To conclude, like A* and C*, the formulation provided takes advantage of one function, *hg*, to consider all possible knowledge from the problem domain in formal theory. Different selections of function *hg* provide the flexibility to allow one to compromise between admissibility, consistency, and computational efficiency.

7. O* and A*

When no points of interest are visited along the path from a starting point to the goal, an OSTQ becomes a query for the minimum cost path between the start and the goal, which is exactly what A* resolves. Since O* resolves OSTQs, it resolves this special case as well. Consequently, A* is a special case of O*, and O* is a generalization of A*.

8. Theoretical Support for Best First Searches for TSP

As a special case of OSTQ, TSP has been studied extensively [1] [2] [3] [4] [5] [6]. A set of solutions were proposed to resolve this problem, either exactly or approximately, and the result is a Hamiltonian cycle that visits each vertex exactly once and returns to the starting vertex. One exact approach uses MST as an admissible heuristic to resolve a TSP with a single-variate best-first search [10], where the graph is solely composed of targeted vertices. Researchers may use the term *A* with MST* when discussing this method to resolve TSP, however, strictly speaking, it may not be appropriate. A* is single-variate, only using vertex identification to define its state, which may not be enough to process TSP in a general graph. It is more appropriate to state that the TSP can be resolved with a bivariate best first search. To make this clear, an example is provided as follows.

In Figure 14, the given origin and destination is *S*, and two vertices of interest, *A* and *B*, require traversal. Note that the virtual edge is used to calculate MST cost for a state. The optimal solution should be either *S*->*A*->*S*->*B*->*S* or *S*->*B*->*S*->*A*->*S*. Now examine the A* search with MST. At the beginning, the initial state *IS,* the identification of *S,* is generated and its *f(n)* is calculated using MST, and then, since *IS* is the only candidate state to expand, it is expanded, and at the same time, according to A*, it is clear that *S* is a goal state. Consequently, the A* search stops and reports that it finds the optimal solution, which is not the actual optimal solution to the TSP since the resulted route does not traverse either A or B. However, the example is still not persuasive enough since the goal state is only a special case and certainly A* can be easily modified to accommodate this phenomenon. Now assume A* is modified to be able to recognize that the state is not a final goal. Then A* continues to generate state *A* and state *B*, since both have the same *f(n)*, assume *A* is randomly chosen to expand, then *S* is a child of *A* and thus a new state, *NS*, for *S* is generated, since A* uses the identification of a vertex to define a state, *NS*=*IS*. According to A*, *NS*'s *f(n)* value must be compared to *IS*'s *f(n)*. Since *NS* has a larger *f(n)*, *NS* will be discarded. The same scenario will happen to *B*, and thus there will be no candidates to expand. The search stops and reports that no result is found. Now we can conclude that the state specification for any vertex in A* must be modified to process TSP. This conclusion is important because A* presents the properties of optimality and optimal efficiency through its single-variate state specification [16]. Consequently, the property that the admissibility guarantees an optimal solution in A* can not be directly applied to the solution with MST for TSP.

No formal mathematical theories have been provided to analyze the possible properties that can result from a bivariate best first search for TSP. More importantly, due to the deficiency of the single-variate state specification, the theorems developed for a single-variate best first search can hardly be applied to identify and prove a globally consistent heuristic for TSP. However, since TSP is a special case of OSTQ, the set of theories proposed in this paper provides theoretical support to obtain optimal and optimally efficient solutions for TSP, including the special case of the best first search with the MST heuristic to resolve TSP.

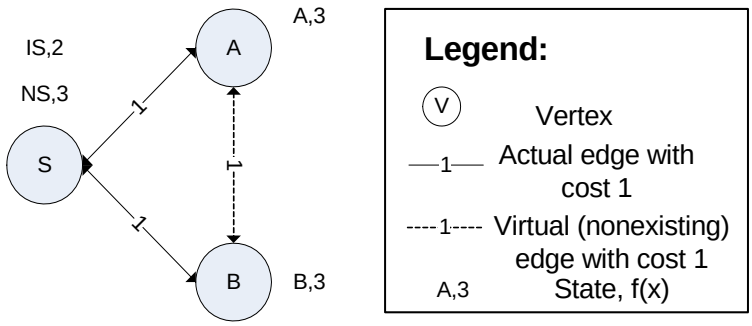


Figure 14: An example to illustrate the search process of "A* with MST" for TSP

9. Conclusion

The contribution of this paper includes four components. First, this paper proposes a novel query in a graph, OSTQ, which determines the minimum-cost path with a predefined origin-destination pair, traversing a set of vertices of interest, and provides O*, an instance of L#, to process the query in a heuristic way without explicit consideration of the order permutation. Second, this paper discusses how to incorporate heuristic information from the problem domain into a formal mathematical theory of graph searching and to provide O* a family of search strategies that demonstrate optimality and optimal efficiency properties in an item-enumeration sub state graph space where a sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of enumerated points of interest. Third, the paper identifies O*-MST, O*-Greedy, and O*-Dijkstra, three special cases of O*, and studies their performance. Finally, since TSP is a special case of OSTQ, the proposed set of theories also provides theoretical support to obtain optimal and optimally efficient solutions for TSP.

O* is a family of algorithms, allowing choices of the heuristic function. Based on the characteristics, including global admissibility and global consistency, of any heuristic in a graph, O* demonstrates optimality and optimal efficiency.

When O* is applied to resolve a minimum-cost route problem without any vertices of interest to traverse, O* functions the same as A*, and thus A* is a special case of O*, or, O* is a generalization of A*.

O*-Dijkstra is a special case of O*, and is always consistent in a δ graph. O*-Greedy is greedy and always expands a state that appears to be closest to the goal state. O*-MST is admissible when the OSTQ problem presents a triangle inequality property. To study the performance of these O* special cases, experiments were performed on O*-MST, O*-Greedy, and O*-Dijkstra with two data sets provided in a transportation network, and the results show that O* always expands a smaller set of states and performs faster than O*-Dijkstra to obtain an optimal solution. In general, the approximate O*-Greedy is the fastest to obtain a solution.

References

[1] S. Arora. Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems. In: Proceedings of 37th IEEE Symposium on Foundations of Computer Science, Burlington, 1996, pp. 2-11.

[2] S. Arora. Approximation Schemes for NP-hard Geometric Optimization Problems: A survey. Mathematical Programming, Springer, 97 (2003) 43-69.

[3] D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. The Traveling Salesman Problem: A Computational Study. Springer, 2007.

[4] N. Christofides. Worst-case Analysis of a New Heuristic for the Traveling Salesman Problem. Carnegie Mellon University, Computer Science Dept, Tech Technical report, 1976.

[5] T. Cormen, T. Leiserson, R. Rivest, and T. Stein. Introduction to Algorithms. The MIT Press, 1997.

[6] A. Dumitrescu and J. S. B. Mitchell. Approximation Algorithms for TSP with Neighborhoods in the Plane. In: Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms, Washington DC, USA, 2001, pp. 38–46.
[7] M. Held and R. M. Karp. A Dynamic Programming Approach to Sequencing Problems, Journal of the Society for Industrial and Applied Mathematics 10(1) (1962): 196–210.
[8] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II. An Analysis of Several Heuristics for the Traveling Salesman Problem. SIAM Journal on Computing. 6 (1977): 563–581.
[9] J. L. Bentley. Fast Algorithms for Geometric Traveling Salesman Problems. ORSA Journal on Computing 4, (1992),387-411.
[10] M. Held, R. M. Karp.The Traveling-Salesman Problem and Minimum Spanning Trees. Operations Research. 18 (1970),1138-1162.
[11] Ma. Dorigo. Ant Colonies for the Traveling Salesman Problem. IRIDIA, Université Libre de Bruxelles. IEEE Transactions on Evolutionary Computation, 1(1) (1997):53–66.
[12] E.H.L. Aarts, and J. Korst. Simulated Annealing and Boltzmann Machines: A stochastic Approach to Combinatorial Optimization and Neural Computing. John Wiley & Sons, Chichester, 1989.
[13] F. Glover. Tabu Search. ORSA Journal on Computing 1, 190-206 (Part I), ORSA Journal on Computing 2, (1989), 4-32 (Part II).
[14] Judea Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley, 1984.
[15] S. Russell and P. Norvig. Artificial Intelligence: a Modern Approach (2nd edition). Prentice Hall, 2002.
[16] P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 (2) (1968) 100–107
[17] E. W. Dijkstra. A Note on Two Problems in Connexion with Graphs. In Numerische Mathematik, 1 (1959), S. 269–271
[18] R. E. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier Search. Journal of the Association for Computing Machinery. 52(5) (2005) 715-748
[19] R. Dechter and J. Pearl. Generalized Best-first Search Strategies and the Optimality of A*. Journal of the Association for Computing Machinery. 32(3) (1985) 505-536
[20] M. Likhachev, G. J. Gordon, S. Thrun. Ara*: Anytime A* with provable bounds on sub-optimality. In. Advances in Neural Information Processing Systems 16. MIT. Press, Cambridge, MA, 2004.
[21] A. Botea, M. Muller, J. Schaeffer. Near Optimal Hierarchical Path-Finding. In Journal of Game Development, volume 1, issue 1, (2004), 7-28.
[22] S. Russell. Efficient Memory-bounded Search Methods. In Proceedings of Tenth European Conference on Artificial Intelligence (ECAI-92), pages 1--5. Chichester, England: Wiley, 1992.
[23] R. Zhou, Eric A. Hansen. Memory-Bounded A* Graph Search. Proceedings of the Fifteenth International Florida Artificial Intelligence Research. May 2002.
[24] Q. Lu, K. Hancock. C*: A Bivariate Best First Search to Process Category Sequence Traversal Queries in a Graph. Submitted to the journal Artificial Intelligence, an International Journal. 2008.

CHAPTER 5: MST: TO PROVIDING A GLOBALLY CONSISTENT HEURISTIC TO PROCESS OPTIMAL SEQUENCE TRAVERSAL QUERIES IN A EUCLIDEAN GRAPH

by

Qifeng Lu

Kathleen Hancock

MST: to Providing a Globally Consistent Heuristic to Process Optimal Sequence Traversal Queries in a Euclidean Graph

Qifeng Lu, Kathleen Hancock

Civil and Environmental Engineering Department, Virginia Polytechnic Institute and State University, Alexandria, VA 22314, USA

Abstract

An Optimal Sequence Traversal Query (OSTQ) is a query in a graph that asks for a minimum-cost path with a predefined origin-destination pair, traversing a set of non-ordered points of interest, at least once for each point. O*-MST, the best first search algorithm to process OSTQ in a graph, uses Minimum Spanning Tree (MST) calculated with the Prim's algorithm to provide an admissible global heuristic. In this paper, we consider the case where OSTQ is processed by O*-MST within a Euclidean graph, which is fully connected and whose edge cost between each pair of points is the Euclidean distance between the two points. This paper 1) proves that in such a Euclidean graph, the MST global heuristic is globally consistent, and consequently, O*-MST is optimally efficient; 2) adopts Delaunay Triangulation to efficiently calculate MST heuristics through the Kruskal's algorithm; and 3) studies the performance of O*-MST with Kruskal's algorithm to process OSTQ in a Euclidean graph, using two other algorithms, O*-Dijkstra and the exhaustive search that computes all possible traversal combinations, as baselines. Based on the results, O*-MST can process OSTQ of up to 16 points of interest in about 30 minutes on average and of up to 17 points of interest in only 0.8 seconds at best. O*-MST is always faster than O*-Dijkstra when the number of points of interest is larger than 2. On average, O*-MST is sub-exponential, and significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of cities to traverse is larger than 9.

Keywords: Best First Search, Heuristic, OSTQ, MST, O*-MST, O*-Dijkstra, Euclidean graph, Kruskal

1. Introduction

Optimal Sequence Traversal Query (OSTQ) is a graph based query that retrieves a minimum cost path that starts from a predefined vertex, traverses a set of given vertices, and ends at a predefined vertex (Qifeng Lu, Kathleen Hancock, 2008). The graph, g-G, may include normal vertices that are neither vertices of interest, nor the given origin or destination.

Similar to a travelling salesman problem (Thomas H. Cormen, Charles E. Leiserson, etc., 1997), OSTQ is NP hard (Michael R. Garey, Donald S. Johnson, etc., 1974) (Wikipedia, 2008). The naïve method that retrieves the optimal solution would compute all possible order combinations of the given points of interest. The time complexity is O(n!), where n is the number of points of interest.

O*, a bivariate best first algorithm that uses problem domain knowledge to guide the search for the optimal solution to an OSTQ in a g-G graph, was recently proposed in Artificial Intelligence (AI) (Qifeng Lu and Kathleen Hancock, 2008). As exact solutions, two special cases of O*, O*-MST and O*-Dijkstra, were proposed.

In this paper, we apply O* to a special case of the graph g-G, a two dimensional Euclidean graph, which is fully connected and whose edge cost between each pair of points is the Euclidean distance between those two points. To processing OSTQ in such a 2D Euclidean graph corresponds to a set of trip planning problems in transportation. An application example is that in

a 2D Euclidean space, an agent/robot may start from a given origin, traverse a set of points of interest to collect information, then ends at a given destination for information analysis.

In such a 2D Euclidean graph, 1) the globally admissible MST heuristic used in O*-MST (Qifeng Lu and Kathleen Hancock, 2008) is further proved to be globally consistent, and 2) a MST can be efficiently calculated with the Kruskal's algorithm (Joseph. B. Kruskal, 1956) instead of the Prim's algorithm adopted by the existing O* algorithm. This paper investigates the performance of O*-MST with Kruskal's algorithm to process OSTQ in a 2D Euclidean graph. O*-Dijkstra (Qifeng Lu and Kathleen Hancock, 2008) and the exhaustive search algorithm that computes all possible order combinations of the points of interest are used as two baselines.

2. Background

In a *g-G* graph, O* incrementally searches all paths leading from the starting vertex and traversing the vertices of interest, until it finds a path of minimum cost to the goal. O* uses a vertex's identification plus its *VisitList*, a sorted list that consists of a sorted sequence containing traversed vertices of interest along the path, to describe a state in the search. At each step, O* uses a distance-plus-cost heuristic function, defined as *f(s)* for a path *s*, to determine the order in which the search visits states in the graph:

$$f(s)=g(s)+h(s) \qquad (1)$$

where

g(s) is the cost function of the path from the initial state to the current state, and

h(s) is the heuristic estimate of the distance from the current state to the goal state.

O* first takes the paths most likely to lead towards the goal, which means the lower the *f(s)*, the higher is the priority for a vertex to be expanded.

Whenever an equal *f(s)* occurs, the state with a larger *VisitList* will be the next to expand. Otherwise, one is randomly selected. State A's *VisitList*, vl_A, is larger than State B's *VisitList*, vl_B, i.e., $vl_A > vl_B$, if the length of vl_A is longer. In other words, the path from the start state to A traverses more vertices of interest than that to B.

For an N-point traversal problem, O* first generates the source state that contains the given vertex and an empty *VisitList*. For all the states in the open list, the algorithm expands the state with the lowest *f(s)* value, and its children states are generated. A child state always inherits the *VisitList* of its parent whenever the child is not a vertex of interest; otherwise, the child's *VisitList* will be incremented by adding the vertex to it. The process continues until a goal state whose vertex is the final goal and *VisitList* contains all the vertices of interest or no solution is found. Once a goal state is reached, the algorithm will retrieve the obtained path using a data structure called *backpointer*, the combination of vertex identification and *VisitList,* to recursively obtain the parent until the origin state is reached.

O*-MST is a special case of O* in that it uses a Minimum Spanning Tree (MST) (Thomas H. Cormen, Charles E. Leiserson, etc., 1997) to compute *h(s)* in an undirected graph. A MST is the tree with the minimum total edge cost among all spanning trees that connect all the vertices together in a graph. A MST example is shown in Figure 1 where the MST is highlighted in dark black. O*-MST always obtains the optimal solution if an OSTQ problem obeys the triangle inequality in a δ graph whose edge cost is not smaller than 0, i.e., a shortest path from vertex *A* to vertex *B* that requires traversing any interested vertex *v* is always longer than or equal to the shortest path from *A* to *B* without that requirement.

In O*, two important definitions exist: global consistency and global admissibility. *Global admissibility* means that the heuristic *h(s)* is admissible, i.e., its value is always smaller than or equal to the actual cost of the minimum-cost path from the current state to the goal state. Suppose an optimal path is from vertex *n* with *VisitList vl* to vertex *n'* with *VisitList vl'*. By default,

vl'>=vl, which means *vl'* contains at least all the visited vertices of interest contained in *vl*. The heuristic *h(s)* is *globally consistent* if the following triangle inequality exists:

hg(n,vl)<=g(n,n', vl->vl')+hg(n',vl') (vl'>=vl)* (2)

Where *g** is the actual (minimum) cost from *(n,vl)* to *(n',vl')*.

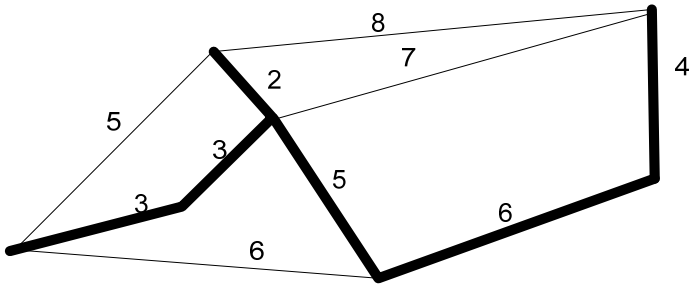


Figure 1: a MST example

Global admissibility guarantees that O* retrieves an optimal solution if one exists. The MST global heuristic used by O*-MST is globally admissible (Qifeng Lu and Kathleen Hancock, 2008). Global consistency not only guarantees that O* retrieves an optimal solution, but also guarantees that once a state is expanded by O*, it will not be expanded again by O*. Global consistency in O* results in optimal efficiency in terms of the states to be expanded, i.e., if *h(s)* is globally consistent, compared to any other no-more-informed admissible heuristic search algorithm, O* expands the minimum number of states.

No related work has been identified for optimal solutions to process OSTQ in a Euclidean graph.

3. MST: to Providing a Globally Consistent Heuristic to Process OSTQ in a Euclidean Graph

As discussed in section 2, the MST heuristic is globally admissible. However, in a special undirected δ graph, named as *SG*, which is fully connected and whose edge cost obeys the triangle inequality, MST heuristic can be globally consistent, which results in an optimally efficient best first search. The property is proved through Theorem 1.

Theorem 1: A MST heuristic is globally consistent in a fully connected undirected graph whose edge costs obey the triangle inequality.

Proof:

Suppose an optimal path is from vertex *n* with *VisitList vl* to vertex *n'* with *VisitList vl'*. By default, *vl'>=vl*, which means *vl'* contains not less visited vertices of interest than *vl*. Assume *hg(n,vl)* is the MST heuristic for the state *(n,vl)* and *hg(n',vl')* is the MST heuristic for the state *(n',vl')*. Let *U* be the set of all the vertices in the MST for *(n,vl)*, *V* be the set of all the vertices in the MST for *(n',vl')*, and *W* be the set of all the visited vertices of interest along the optimal path from *(n,vl)* to *(n',vl')*. It is clear that *U=V+W*. Since all the edge costs satisfy the triangle inequality, and the graph is fully connected, the only vertex within both *V* and *W* is *n'*. A minimum spanning tree, MST_o, can be obtained from the optimal path from *(n,vl)* to *(n',vl')*, whose total edge cost is not larger than the optimal path cost. Therefore, it is clear that MST_o plus the MST for *(n',vl')* forms a spanning tree for all the vertices in *U*, and MST for *(n,vl)* is the minimum spanning tree for the same set of vertices, so according to the definition of minimum spanning tree, its cost is the minimum among all spanning trees. Consequently

Cost(MST(n,vl))<=Cost(MST_o)+Cost(MST(n',vl'))

<=g(n,n',vl->vl')+Cost(MST(n',vl')) (g* is the same as defined in equation (2))*

So *hg(n,vl)<=g*(n,n',vl->vl')+hg(n',vl').* Completing the proof.

A Euclidean graph is fully connected and undirected since the Euclidean distance between any two points is the same in both directions. Also, its edge cost obeys the triangle inequality. Therefore, in a Euclidean graph, MST heuristic is globally consistent, and consequently, O*-MST is optimally efficient.

4. The Kruskal's Algorithm in a Euclidean Graph

Two prevalent algorithms exist to calculate MST in an undirected graph: Krukal's algorithm (Joseph. B. Kruskal, 1956) and Prim's algorithm (Robert C. Prim, 1957). Figure 2 and Figure 3 present the pseudo code for both algorithms respectively. Table 1 presents their time and space complexity.

```
Input: A set of trees ST, where each vertex in the graph is a separate tree, and
       a set S containing all the edges in the graph
while S is not empty
       remove an edge with a minimum weight from S
       if that edge connects two different trees, then add it to ST, and combine two
            trees into a single tree
       otherwise discard that edge
Output: ST, which is the MST
```

Figure 2: The pseudo code for Kruskal's algorithm

```
Input: A connected directed graph with vertices V and edges E.
Initialize: V_new = {n}, where n is an arbitrary node (starting vertex) from V, and E_new = {}
Repeat until V_new = V:
   Choose edge (x,y) from E with minimal cost such that x is in V_new and y is not (if there
        are multiple edges with the same weight, choose arbitrarily)
   Add y to V_new, add (x, y) to E_new
Output: V_new and E_new describe a minimal spanning tree
```

Figure 3: The pseudo code for Prim's algorithm

Table 1: Space and Time Complexity for Prim's Algorithm and Kruskal's Algorithm

Algorithm	Time complexity	Space complexity
Prim	$O(V^2)$	$O(V^2)$
Kruskal	O(E*log*E)	O(E)

Where
V is the number of vertices in a graph, and
E is the number of edges in a graph.

Prim is more efficient in a dense graph while Kruskal is more efficient in a sparse graph. For a Euclidean graph, it seems that it is more time consuming for Kruskal's algorithm to obtain a MST than for Prim's algorithm since $E=V^2$. However, in a two dimensional Euclidean graph, the time complexity of Kruskal's algorithm can be further reduced to O(V*log*V) through the Delaunay Triangulation (Franco P. Preparata, Michael Ian Shamos, 1985). Through Delaunay triangulation, the fully connected planar graph is reduced to *PG*, a planar graph with at most 3V number of edges, and it is proved that *PG* contains all MST edges (Franco P. Preparata, Michael Ian Shamos, 1985). Consequently, O(E*log*E) becomes O(V*log*V).

In this paper, the Kruskal's algorithm based on Delaunay Triangulation is adopted to expedite the OSTQ processing.

5. Experiments

The purpose of the experiments is to test the performance of O*-MST with Kruskal's algorithm after Delaunay Triangulation to calculate MST heuristics in a Euclidean graph. O*-Dijkstra (Qifeng Lu and Kathleen Hancock, 2008) and the exhaustive search that computes all possible order combinations of the points of interest that correspond to vertices of interest in O* are used as two baselines. Furthermore, the O*-MST's performance using Kruskal's algorithm with Delaunay Triangulation is compared to using Prim's algorithm without Delaunay triangulation.

The experiments were performed on a Toshiba Satellite A215 Laptop with 2.0GB memory (RAM), AMD Turion™ 64*2 Mobile Technology TL-56 1.80HZ processors, and Windows Vista™ Home Premium operating system.

5.1. Data set

A 130 city TSP problem (Churritz) (130 City Problem (Churritz), 2008) is used as the data set for this experiment. The data set contains Coordinates(X,Y) for 130 cities as shown in Figure 4. In this experiment, a set of OSTQ problems is randomly generated. First, the number of points of interest consecutively changes from 2 to 17. Second, for each number of points of interest, 30 problem samples are randomly generated, i.e., the origin, the destination, and the points of interest in each problem sample are randomly selected. Consequently, a set of 480 OSTQ problems was generated. The whole set is used for O*-MST using Kriskal's algorithm with Delaunay triangulation, the partial set with up to 11 cities is used for O*-Dijkstra, the partial set with up to 12 cities is used for the exhaustive search method, and the partial set with up to 15 cities is used for O*-MST with Prim's algorithm.

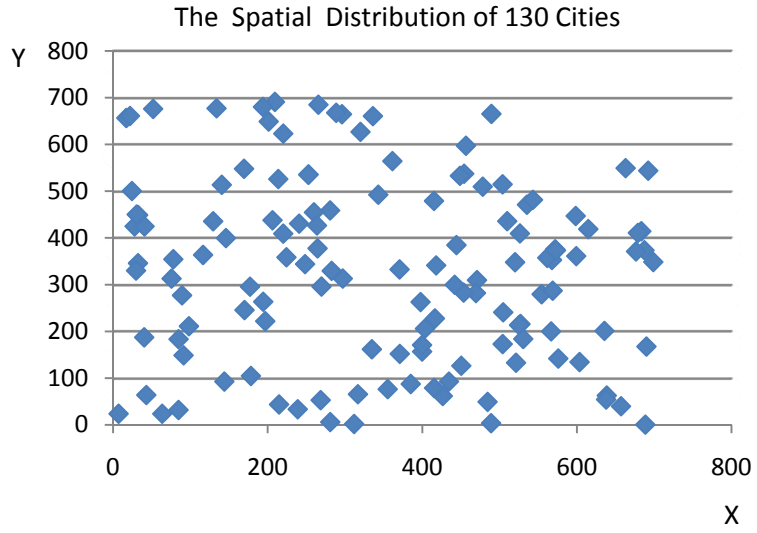


Figure 4: The spatial distribution of 130 cities (130 city problem (Churritz), 2008)

5.2. Performance measures

To study the performances of the three algorithms, the following performance measures are identified.

Average Shortest Route Distance (ASRD): the average shortest route distance obtained over all runs (miles);

Minimum Process Time (MinPT): the minimum time required to obtain a solution for each number of points of interest (seconds);

Maximum Process Time (MaxPT): the maximum time required to obtain a solution for each number of points of interest (seconds);

Average Process Time (APT): the average time required to process a query over all runs (seconds);

5.3. Results and Discussion

Results are presented in Table 2 and Table 3. *ES* represents the naïve exhaustive search approach, and *NPoI* represents the number of points of interest, i.e., the number of cities to traverse. The "-" indicates that a value is not available since the time to obtain a result exceeded a reasonable expected solution time.

Table 2: Performance results for O*-MST using Kriskal's algorithm after Delaunay Triangulation, O*-Dijkstra, and the naïve exhaustive search approach

NPoI	***ASRD***	***MinPT***		***MaxPT***		***APT***		
		O*-MST	O*-Dijkstra	O*-MST	O*-Dijkstra	O*-MST	O*-Dijkstra	ES
2	925.703	0	0	0.078	0	0.003	0.000	0.000
3	1121.818	0	0	0.015	0.016	0.001	0.002	0.000
4	1269.045	0	0	0.016	0.016	0.002	0.008	0.000
5	1399.514	0	0.046	0.016	0.063	0.005	0.051	0.000
6	1532.475	0	0.218	0.109	0.39	0.024	0.315	0.000
7	1678.878	0	1.31	0.374	2.012	0.073	1.784	0.003
8	1763.873	0.016	8.19	1.279	11.091	0.214	9.723	0.023
9	1848.778	0.015	47.003	4.462	61.985	0.525	56.664	0.245
10	1943.869	0.019	263.466	4.122	387.211	0.714	324.805	2.963
11	2034.248	0.032	1237.95	38.602	1791.435	3.785	1575.589	35.876
12	2081.955	0.078	-	73.095	-	8.626	-	512.585
13	2185.539	0.095	-	137.034	-	20.991	-	-
14	2249.412	0.246	-	916.465	-	90.224	-	-
15	2308.745	0.517	-	793.927	-	84.372	-	-
16	2376.916	0.492	-	1831.573	-	190.324	-	-
17	2444.334	0.745	-	4063.257	-	529.167	-	-

Note: Abbreviations in the table**:** *NPoI: Number of Points of Interest; ASRD: Average Shortest Route Distance; MinPT: Minimum Process Time; MaxPT: Maximum Process Time; APT: Average Process Time; ES: Exhaustive search*

Table 3: Performance results for O*-MST: Kriskal's algorithm with Delaunay Triangulation versus Prim's algorithm without Delaunay Triangulation

NPoI	***ASRD***	***MinPT***		***MaxPT***		***APT***	
		Prim's	Kriskal's	Prim's	Kriskal's	Prim's	Kriskal's
2	925.703	0	0	0.03	0.078	0.001	0.003
3	1121.818	0	0	0.001	0.015	0.000	0.001
4	1269.045	0	0	0.006	0.016	0.002	0.002
5	1399.514	0.001	0	0.039	0.016	0.009	0.005
6	1532.475	0.001	0	0.251	0.109	0.043	0.024
7	1678.878	0.002	0	0.671	0.374	0.118	0.073
8	1763.873	0.004	0.016	2.68	1.279	0.393	0.214
9	1848.778	0.008	0.015	10.326	4.462	1.026	0.525
10	1943.869	0.015	0.019	9.242	4.122	1.280	0.714
11	2034.248	0.03	0.032	80.983	38.602	7.717	3.785
12	2081.955	0.062	0.078	124.487	73.095	17.225	8.626
13	2185.539	0.088	0.095	333.731	137.034	48.515	20.991
14	2249.412	0.327	0.246	2604.289	916.465	217.470	90.224
15	2308.745	0.724	0.517	1633.235	793.927	167.191	84.372

Note: Abbreviations in the table**:** *NPoI: Number of Points of Interest; ASRD: Average Shortest Route Distance; MinPT: Minimum Process Time; MaxPT: Maximum Process Time; APT: Average Process Time*

Figure 5 through Figure 7 are provided to visualize the performance measures provided in Table 2.

First, in general, the larger the number of consumer destinations to traverse, the longer the distance of the shortest route for the corresponding OSTQ.

Based on MinPT, O*-MST can retrieve the optimal solution within 0.8 seconds for an OSTQ of 17 NPoI. However, based on MaxPT, it still may require 4064 seconds for another query with the same number of points of interest. This implies that O*-MST's performance depends on how closely the selected MST heuristic approaches to the actual cost.

Based on MinPT shown in Figure 5, O*-MST outperforms O*-Dijkstra over all runs.

Based on MaxPT shown in Figure 6, O*-MST outperforms O*-Dijkstra when NPoI is larger than 2. This is due to the fact that O*-MST requires additional time to compute the MST heuristic. However, when NPoI becomes larger, this additional time is no longer a dominant factor.

Based on APT shown in Figure 7, when NPoI is less than 10, both O*-MST and O*-Dijkstra are slower than the naïve method. This is due to the fact that both O*-MST and O*-Dijkstra require operations on their closed lists and priority queues, and O*-MST requires the computation of MST heuristics. When NPoI is larger than 9, O*-MST performs increasingly faster than the naïve method. O*-Dijkstra always performs worse on average than the naïve method as shown in Figure 7. However, O*-Dijkstra may outperform the naïve method when NPoI is larger than a certain threshold, although this has not been shown. On average, O*-MST can process OSTQ of up to 16 points of interest in about 30 minutes.

In Figure 5 through Figure 7, it is noticeable that O*-Dijkstra is mostly exponential in time complexity. According to Figure 7, O*-MST is mostly sub-exponential in average process time, which is highly desirable to efficiently process OSTQs.

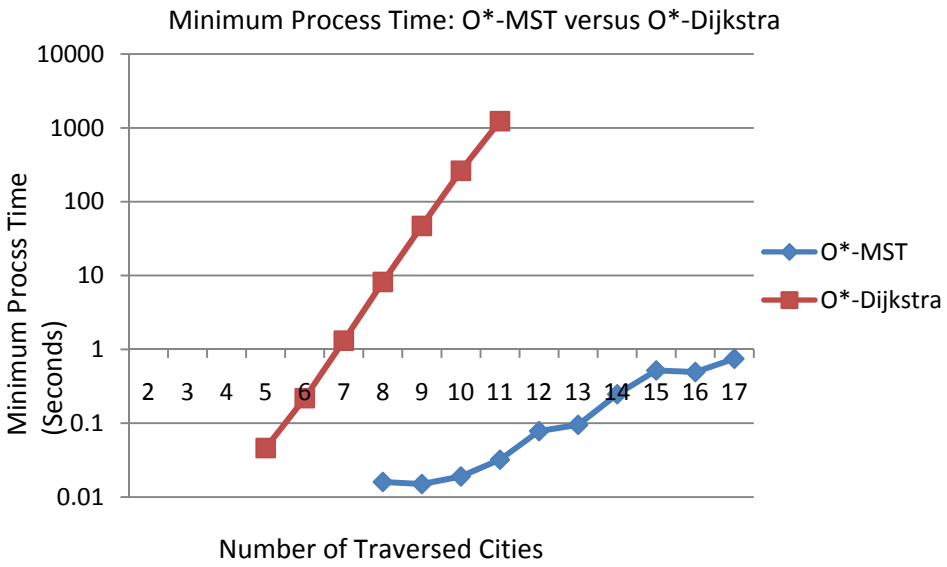


Figure 5: Minimum process time over different number of traversed consumer destinations: O*-MST versus O*-Dijkstra

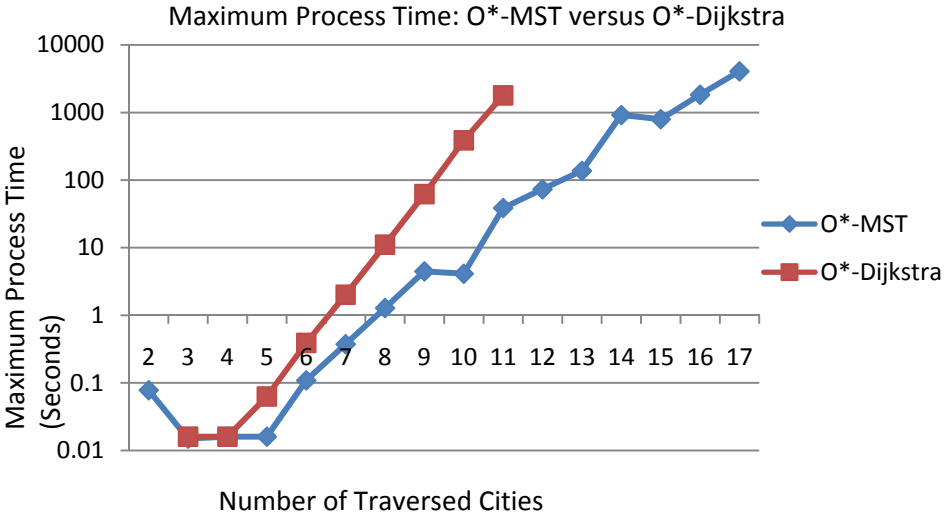


Figure 6: Maximum process time over different number of traversed consumer destinations: O*-MST versus O*-Dijkstra

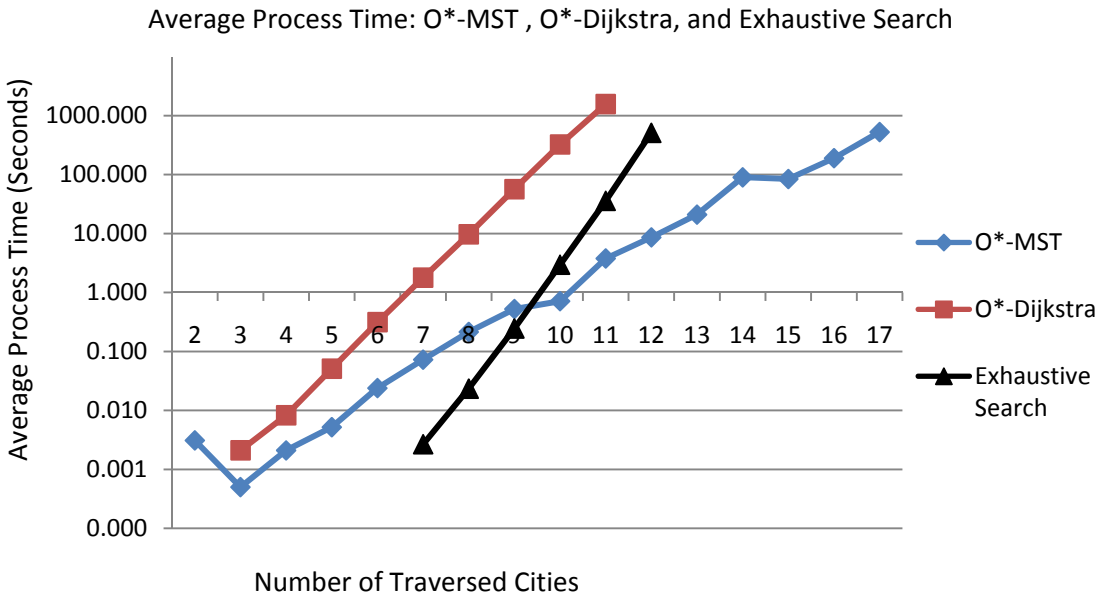


Figure 7: Average process time over different number of traversed cities: O*-MST, O*-Dijkstra, and Exhaustive Search (ES)

Figure 8 through Figure 10 are provided to visualize the performance measures provided in Table 3.

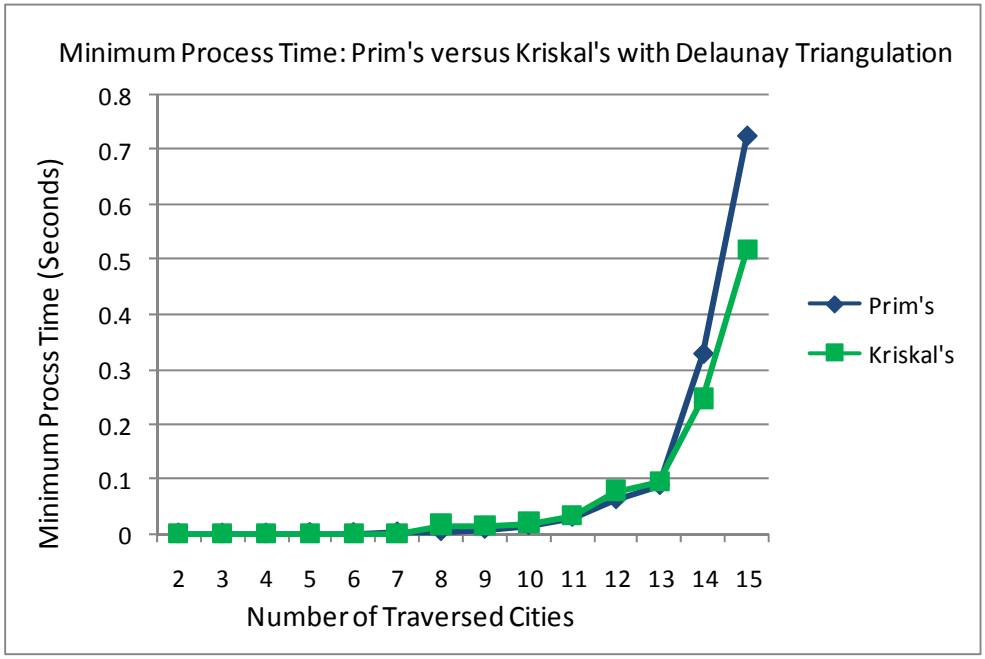


Figure 8: Minimum process time over different number of traversed consumer destinations: O*-MST versus O*-Dijkstra

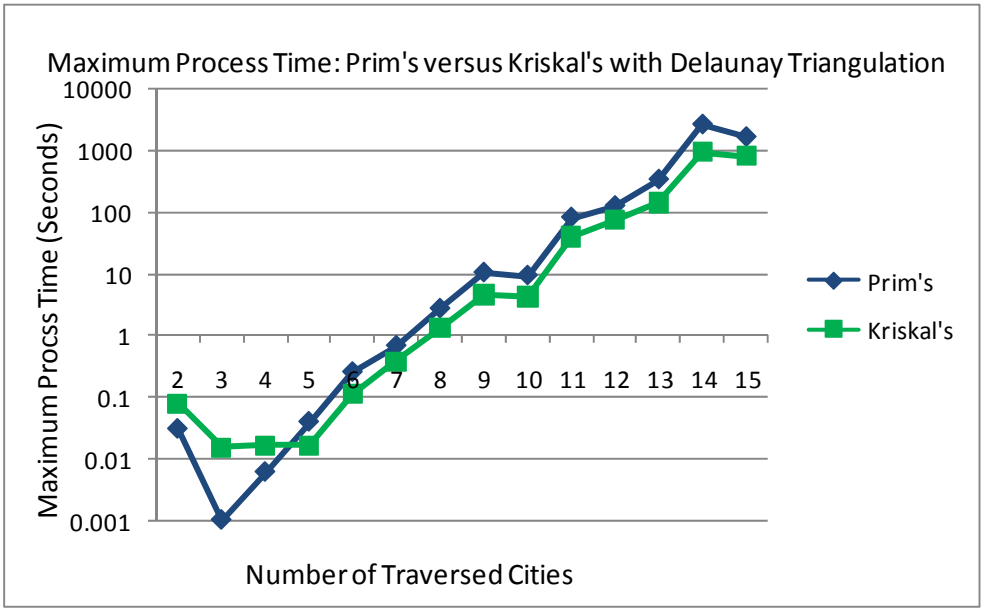


Figure 9: Maximum process time over different number of traversed consumer destinations: O*-MST versus O*-Dijkstra

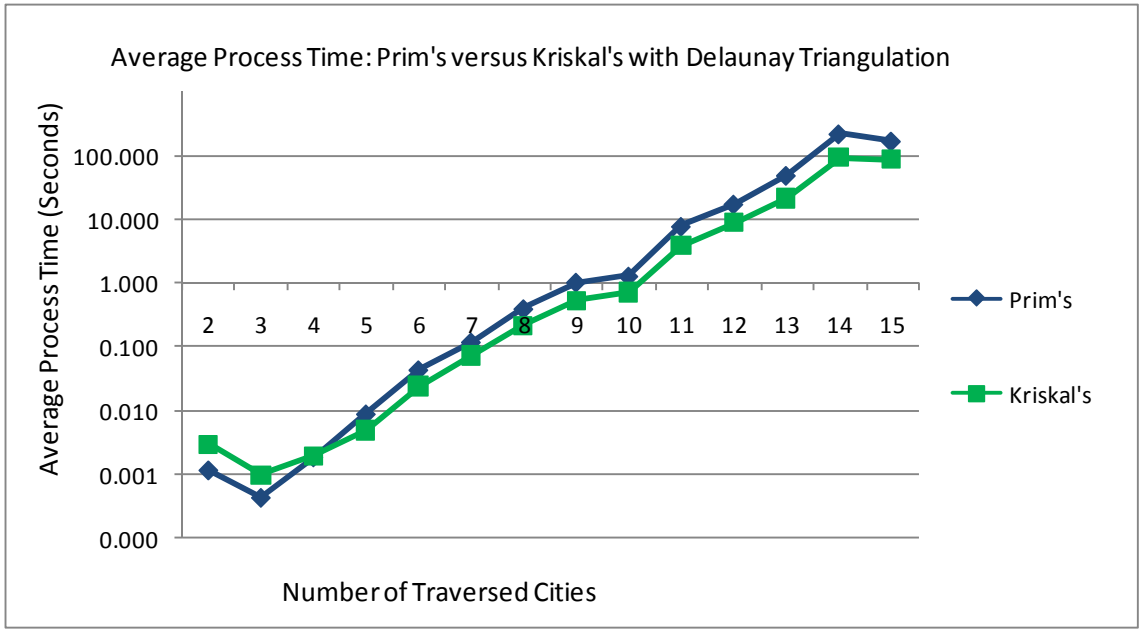


Figure 10: Average process time over different number of traversed consumer destinations: O*-MST versus O*-Dijkstra

Based on Figure 8 through Figure 10, it is clear when process time is beyond 0.1 seconds, using Kriskal's algorithm with Delaunay Triangulation is always faster than using Prim's algoirthm. Furthermore, according to the trend in each of these three figures, the larger the number of traversed cities is, the faster is using Kriskal's algorithm with Triangulation than using Prim's algorithm to compute the MSTs.

6. Conclusion

The contribution of this paper includes three components: 1) the MST global heuristic is proved to be globally consistent in a fully connected undirected graph whose edge costs obey the triangle inequality; 2) to compute the MST heuristic in O*-MST to process OSTQ in a Euclidean graph, based on the analyses on time complexity and experiment results, the Kruskal's algorithm based on Delaunay Triangulation is more efficient than the Prim's algorithm; and 3) the corresponding O*-MST's performance is statistically studied using a real data set. Based on the results, O* can retrieve an optimal solution for an OSTQ of up to 16 points of interest in about 30 minutes on average and of up to 17 points of interest within 0.8 seconds at best. O*-MST is always faster than O*-Dijkstra when the number of points of interest is larger than 2. On average, O*-MST is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of cities to traverse is larger than 9.

References

130 City Problem (Churritz). ftp://ftp.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/ch130.tsp. Last retrieved on October 30, 2008.

Franco P. Preparata, Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York. 1985.

Joseph. B. Kruskal. *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*. In: Proceedings of the American Mathematical Society, Vol 7, No. 1, pp. 48–50, 1956.

Michael R. Garey, Donald S. Johnson, and Larry Stockmeyer. *Some simplified NP-complete problems*. Proceedings of the sixth annual ACM symposium on Theory of computing, p.47-63. 1974.

Qifeng Lu, Kathleen Hancock. *O*: A Bivariate Best First Search Algorithm to Process Optimal Sequence Traversal Query in a Graph*. Submitted to the Journal: Journal of Artificial Intelligence Research, 2008

Robert C. Prim: *Shortest connection networks and some generalizations*. In: *Bell System Technical Journal*, 36, pp. 1389–1401,1957

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 1997

Wikipedia. *Travelling Salesman Problem*. http://en.wikipedia.org/wiki/Traveling_salesman_problem, Last retrieved on October 30, 2008.

CHAPTER 6: SCDMST: TO PROVIDING A GLOBALLY CONSISTENT HEURISTIC TO PROCESS OPTIMAL SEQUENCE TRAVERSAL QUERIES IN A FULLY CONNECTED DIRECTED GRAPH

by

Qifeng Lu

Kathleen Hancock

SCDMST: to Providing a Globally Consistent Heuristic to Process Optimal Sequence Traversal Queries in a Fully Connected Directed Graph

Qifeng Lu, Kathleen Hancock

Civil and Environmental Engineering Department, Virginia Polytechnic Institute and State University, Alexandria, VA 22314, USA

Abstract

An Optimal Sequence Traversal Query (OSTQ) is a query in a graph that asks for a minimum-cost path with a predefined origin-destination pair, traversing a set of non-ordered points of interest, at least once for each point. As a best first search algorithm, O* was proposed to process OSTQ in a graph. In this paper, we propose O*-SCDMST to process OSTQ in a fully connected directed graph whose edge cost obeys the triangle inequality. O*-SCDMST is an O* algorithm using a semi-connected directed minimum spanning tree (SCDMST) to provide the global heuristic, which is proved globally consistent. Two other algorithms, O*-Dijkstra and the exhaustive search that computes all possible traversal combinations, are used as baselines. Based on the results, O*-SCDMST can process OSTQ of up to 14 points of interest in 10 minutes on average, and of up to 15 points of interest within only 4 seconds at best. On average, O*-SCDMST is always faster than O*-Dijkstra, and is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of points of interest is larger than 9.

Keywords: Best First Search, Heuristic, OSTQ, O*-SCDMST, O*-Dijkstra

1. Introduction

Optimal Sequence Traversal Query (OSTQ) is a graph based query that retrieves a minimum cost path that starts from a predefined vertex, traverses a set of given vertices, and ends at a predefined vertex (Qifeng Lu, Kathleen Hancock, 2008). The graph, g-G, may include normal vertices that are neither vertices of interest, nor the given origin or destination.

Similar to a travelling salesman problem (Thomas H. Cormen, Charles E. Leiserson, etc., 1997), OSTQ is NP hard (Michael R. Garey, Donald S. Johnson, etc., 1974) (Wikipedia, 2008). The naïve method that retrieves the optimal solution would compute all possible order combinations of the given points of interest. The time complexity is O(n!), where n is the number of points of interest.

O*, a best first algorithm that uses problem domain knowledge to guide the search for the optimal solution to an OSTQ in a g-G graph, was recently proposed (Qifeng Lu and Kathleen Hancock, 2008). As exact solutions, two special cases of O*, O*-MST and O*-Dijkstra, were proposed in the paper.

In this paper, we consider a special graph FDG, a fully connected directed graph composed of vertices of interest, the predefined origin vertex, or the predefined destination vertex, and whose vertices edge costs obey the triangle inequality, i.e., a shortest path from vertex *A* to vertex *B* that requires traversing any interested vertex *v* is always longer than or equal to the shortest path from *A* to *B* without that requirement. To process OSTQ in such a FDG with O*, a semi-connected directed minimum spanning tree (SCDMST) is proposed to calculate the global heuristic, which is further proved as globally consistent. To see how well this global heuristic works, O*-Dijkstra (Qifeng Lu and Kathleen Hancock, 2008) and the naïve search method that computes all possible order combinations of the vertices of interest are used as two baselines. A set of experiments was performed on a benchmark data set. The result is provided and analyzed.

2. Background

In a *g-G* graph, O* incrementally searches all paths leading from the starting vertex and traversing the vertices of interest, until it finds a path of minimum cost to the goal. O* uses a vertex's identification plus its *VisitList*, a sorted list that consists of a sorted sequence containing traversed vertices of interest along the path, to describe a state in the search. At each step, O* uses a distance-plus-cost heuristic function, defined as *f(s)* for a state *s*, to determine the order in which the search visits states in the graph:

$$f(s)=g(s)+h(s) \qquad (1)$$

where

g(s) is the cost function of the path from the initial state to the current state, and

h(s) is the heuristic estimate of the distance from the current state to the goal state.

O* first takes the paths most likely to lead towards the goal, which means the lower the *f(s)*, the higher is the priority for a vertex to be expanded.

Whenever an equal *f(s)* occurs, the state with a larger *VisitList* will be the next to expand. Otherwise, one is randomly selected. State A's *VisitList*, vl_A, is larger than State B's *VisitList*, vl_B, i.e., $vl_A > vl_B$, if the length of vl_A is longer. In other words, the path from the start state to A traverses more vertices of interest than that to B.

For an N-point traversal problem, O* first generates the source state that contains the given vertex and an empty *VisitList*. For all the states in the open list, the algorithm expands the state with the lowest *f(s)* value, and its children states are generated. A child state always inherits the *VisitList* of its parent whenever the child is not a vertex of interest; otherwise, the child's *VisitList* will be incremented by adding the vertex to it. The process continues until a goal state whose vertex is the final goal and *VisitList* contains all the vertices of interest or no solution is found. Once a goal state is reached, the algorithm will retrieve the obtained path using a data structure called *backpointer*, the combination of vertex identification and *VisitList,* to recursively obtain the parent until the origin state is reached.

In O*, two important definitions exist: global consistency and global admissibility. *Global admissibility* means that the heuristic *h(s)* is admissible, i.e., its value is always smaller than or equal to the actual cost of the minimum-cost path from the current state to the goal state. Suppose an optimal path is from vertex *n* with *VisitList vl* to vertex *n'* with *VisitList vl'*. By default, *vl'>=vl*, which means *vl'* contains at least all the visited vertices of interest contained in *vl*. The heuristic *h(s)* is *globally consistent* if the following triangle inequality exists:

$$hg(n,vl) <= g^*(n,n', vl->vl') + hg(n',vl') \quad (vl'>=vl) \qquad (2)$$

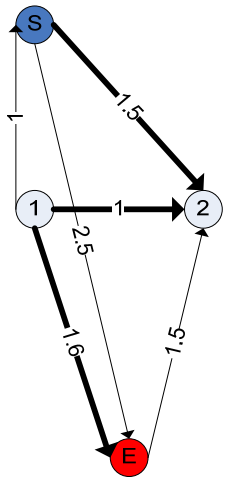
Where *g** is the actual (minimum) cost from *(n,vl)* to *(n',vl')*.

Global admissibility guarantees that O* retrieves an optimal solution if one exists. The MST global heuristic used by O*-MST to process OSTQ in a g-G graph whose edge costs obey the triangle inequality is globally admissible (Qifeng Lu and Kathleen Hancock, 2008a). Global consistency not only guarantees that O* retrieves an optimal solution, but also guarantees that once a state is expanded by O*, it will not be expanded again by O*. Global consistency in O* results in optimal efficiency in terms of the states to be expanded, i.e., if *h(s)* is globally consistent, compared to any other no-more-informed admissible heuristic search algorithm, O* expands the minimum number of states. The MST global heuristic is further proved to be globally consistent in an undirected fully connected graph that obeys the triangle inequality (Qifeng Lu and Kathleen Hancock, 2008b). In a fully connected graph, however, since a MST is defined upon an undirected graph, the MST heuristic is no longer proper to process OSTQ. In other words, neither the Prim's algorithm nor the Kriskal's algorithm used to compute MSTs (Qifeng Lu and Kathleen Hancock, 2008b) outputs a MST where any two points are guaranteed to be connected.

No related work has been identified for optimal solutions to process OSTQ in a fully connected directed graph.

3. SCDMST: to Providing a Globally Consistent Heuristic to Process OSTQ in a Fully Connected Directed Graph

Consider a directed graph, G(V,E), where V and E are the set of vertices and edges, respectively, and a starting vertex s and an ending vertex e in E. Associated with each edge (i,j) in V is a cost c(i,j). Let |V|=n and |E|=m. A semi-connected directed spanning tree, SCDST, is defined as a graph which connects, without any cycle, all vertices with n-1 arcs, while vertex s only has outgoing arcs and vertex e only has incoming arcs. It is semi-connected in the sense that it does not necessarily connect any two points together. A semi-connected directed minimum spanning tree (SCDMST) is the graph with the minimum total edge cost among all SCDSTs. In other words, the problem is to find a SCDST, G(V,S) where S is a subset of E, such that the sum of c(i,j) for all (i,j) in S is minimized. Figure 1 shows a SCDMST example.



Where

S: starting vertex, E: ending vertex.

The SCDMST tree is highlighted in dark black. The SCDMST starts from S, connects vertex 1 and vertex 2, and ends at E. No cycle exists. 3 edges are used to connect 4 vertices. Only outgoing edges exist for S and incoming edges exist for E. The SCDMST is semi-connected since not any two points are connected through the tree. For example, 2 and E are not connected together in the obtained SCDMST.

Figure 1: A SCDMST example

The SCDMST heuristic is globally consistent in a fully connected directed graph whose edge costs obey the triangle inequality. The property is proved through Theorem 1.

Theorem 1: A SCDMST heuristic is globally consistent in a fully connected directed graph whose edge costs obey the triangle inequality.

Proof:

Suppose an optimal path is from vertex *n* with *VisitList vl* to vertex *n'* with *VisitList vl'*. By default, *vl'>=vl*, which means *vl'* contains not less visited vertices of interest than *vl*. Assume *hg(n,vl)* is the SCDMST heuristic for the state *(n,vl)* and *hg(n',vl')* is the SCDMST heuristic for the state *(n',vl')*. Let *U* be the set of all the vertices in the SCDMST for *(n,vl)*, *V* be the set of all the vertices in the SCDMST for *(n',vl')*, and *W* be the set of all the visited vertices of interest along the optimal path from *(n,vl)* to *(n',vl')*. It is clear that *U=V+W*. Since all the edge costs satisfy the triangle inequality, and the graph is fully connected, the only vertex within both *V* and *W* is *n'*. A semi-connected directed minimum spanning tree, $SCDMST_o$, can be obtained from the optimal path from *(n,vl)* to *(n',vl')*, whose total edge cost is not larger than the optimal path cost.

Therefore, it is clear that $SCDMST_o$ plus the SCDMST for *(n',vl')* forms a semi-connected directed spanning tree for all the vertices in *U*, and SCDMST for *(n,vl)* is the semi-connected directed minimum spanning tree for the same set of vertices. Therefore, according to the definition of the semi-connected directed minimum spanning tree, its cost is the minimum among all semi-connected directed spanning trees. Consequently

Cost(SCDMST(n,vl))<=Cost(SCDMST$_o$)+Cost(SCDMST(n',vl'))

<=g(n,n',vl->vl')+Cost(SCDMST(n',vl')) (g* is the same as defined in equation (2))*

So *hg(n,vl)<=g*(n,n',vl->vl')+hg(n',vl').* Completing the proof.

The O* algorithm that uses SCDMSTs to provide global heuristics is named as O*-SCDMST.

4. The D-ODPrim Algorithm to Retrieve a SCDMST from a Fully Connected Directed Graph

In this paper, D-ODPrim is proposed to obtain a SCDMST from a directed graph. Its pseudo code is provided in Figure 2.

Input: A connected directed weighted graph with vertices V and edges E, the starting vertex s, and the ending vertex e.
Initialize: V_{new} = {x}, where x is the starting vertex from V, E_{new} = {}
Remove the incoming edges of s and the outgoing edges of e in E
Repeat until V_{new} = V:
  Choose edge (u,v) from E with the minimal weight such that one vertex is in V_{new} and the other is not (if there are multiple edges with the same weight, choose arbitrarily)
  Add v to V_{new}, add (u, v) to E_{new}
Output: V_{new} and E_{new} describe a semi-connected directed minimal spanning tree

Figure 2: The pseudo code for D-ODPrim algorithm

D-ODPrim is a variation of the Prim algorithm that calculates a MST for an undirected graph (Robert C. Prim, 1957). Similar to Prim's algorithm, D-ODPrim continuously increases the size of a tree starting with the given starting vertex until it spans all the vertices.

Four steps are used to prove D-ODPrim outputs a SCDMST if a solution exists.

Assume the number of vertices is *N*. Name the obtained graph as *DG*.

Step 1: Prove that *DG* contains *N*-1 edges.

Proof: Every time a new edge connecting to a new vertex is added to E_{new}, until the *N*-1 points are added to V_{new}. Consequently, for N vertices, the algorithm will try *N*-1 times, thus *N*-1 edges will be added to form *DG*.

Step 2: Prove that *DG* does not contain a cycle.

Proof: According to the algorithm, if the edge direction is neglected in *DG*, *DG* connects any two points together with *N*-1 edges, which is impossible to have a cycle. So the directional *DG* does not contain a cycle.

Step 3: Prove that no outgoing edges for the ending vertex e and no incoming edges for the starting vertex s.

Proof: Since the algorithm removes the incoming edges of s and the outgoing edges of e from the candidate edge set, it is no way to add these edges into *DG*.

Step 4: Prove that the total edge cost is the minimum.

Proof: Use contradiction. Suppose another SCDST, DG_1, has a smaller total edge cost. Then there must be at least one vertex that uses an edge of smaller cost in DG_1 than in *DG*, which conflicts the fact that every time the algorithm finds the minimum cost edge for a vertex to add it to *DG*.

Based on the four steps, it is clear that D-ODPrim retrieves a SCDMST.

For D-ODPrim, the time complexity is $O(V^2)$ and space complexity is $O(V^2)$.

5. Experiments

The purpose of the experiments is to test the performance of O*-SCDMST to calculate the global heuristics in a FDG. O*-Dijkstra (Qifeng Lu and Kathleen Hancock, 2008) and the exhaustive search that computes all possible order combinations of the points of interest are used as two baselines.

The experiments were performed on a Toshiba Satellite A215 Laptop with 2.0GB memory (RAM), AMD Turion™ 64*2 Mobile Technology TL-56 1.80HZ processors, and Windows Vista™ Home Premium operating system.

5.1. Data set

An asymmetric TSP problem (Fischetti) with 34 points of interest (Matteo Fischetti, Paolo Toth, 1997), corresponding to vertices of interest in O*,is used as the data set for this experiment. The data set contains the edge costs between any two points. In this experiment, a set of OSTQ problems are generated from this data set. First, the number of points of interest consecutively changes from 2 to 15. Second, for each number of points of interest, 30 problem samples are randomly generated, i.e., the origin, the destination, and the points of interest in each problem sample are randomly selected from the 34 points. Consequently, a set of 420 problems is generated. The whole problem set is used for O*-MST, the partial set with up to 12 points of interest are used for both O*-Dijkstra and the exhaustive search method.

5.2. Performance measures

To study the performances of the three algorithms, the following performance measures are identified.

Average Optimal Route Cost (AORC): the average optimal route cost obtained over all runs;

Minimum Process Time (MinPT): the minimum time required to obtain a solution for each number of points of interest (seconds);

Maximum Process Time (MaxPT): the maximum time required to obtain a solution for each number of points of interest (seconds);

Average Process Time (APT): the average time required to process a query over all runs (seconds);

5.3. Results and Discussion

The results are presented in Table 1. *ES* represents the naïve exhaustive search approach, and *NPoI* represents the number of points of interest, i.e., the number of cities to traverse. The "-" indicates that a value is not available since the time to obtain a result exceeded a reasonable expected solution time.

Table 1: Performance results for O*-SCDMST, O*-Dijkstra, and the naïve exhaustive search

NPoI	***AORC***	***MinPT***		***MaxPT***		***APT***		
		O*-SCDMST	O*-Dijkstra	O*-SCDMST	O*-Dijkstra	O*-SCDMST	O*-Dijkstra	ES
2	367.30	0.000	0.000	0.036	0.022	0.001	0.001	0.000
3	435.33	0.000	0.000	0.003	0.003	0.000	0.001	0.000
4	495.67	0.000	0.001	0.007	0.010	0.002	0.006	0.000
5	558.47	0.001	0.004	0.027	0.063	0.008	0.029	0.000
6	612.60	0.002	0.023	0.152	0.282	0.043	0.135	0.000
7	648.50	0.003	0.086	0.774	1.509	0.166	0.592	0.004
8	680.73	0.045	0.228	2.208	7.312	0.532	2.287	0.024
9	715.40	0.052	0.677	7.621	45.609	1.775	9.262	0.260
10	752.80	0.036	3.124	51.956	215.837	5.633	41.112	2.927
11	794.87	0.176	4.268	314.782	786.551	29.027	137.429	36.082
12	824.13	0.389	34.097	234.940	1479.480	66.821	356.693	568.001
13	847.10	3.541	-	1109.164	-	245.073	-	-
14	879.43	2.059	-	3900.673	-	548.081	-	-
15	900.00	3.370	-	20857.746	-	2204.143	-	-

Note: Abbreviations in the table**:** *NPoI: Number of Points of Interest; AORC: Average Optimal Route Cost; MinPT: Minimum Process Time; MaxPT: Maximum Process Time; APT: Average Process Time; ES: Exhaustive search*

Figure 3 through Figure 5 are provided to visualize the performance measures provided in Table 2.

First, in general, the larger the number of consumer destinations to traverse, the larger the cost of the optimal route for the corresponding OSTQ.

Based on MinPT, O*-SCDMST can retrieve the optimal solution within 4 seconds for an OSTQ of 15 NPoI. However, based on MaxPT, it still may require 20858 seconds for another query with the same number of points of interest. This implies that O*-SCDMST's performance depends on how closely the selected SCDMST heuristic approaches to the actual cost.

Based on MinPT shown in Figure 3, O*-SCDMST outperforms O*-Dijkstra over all runs.

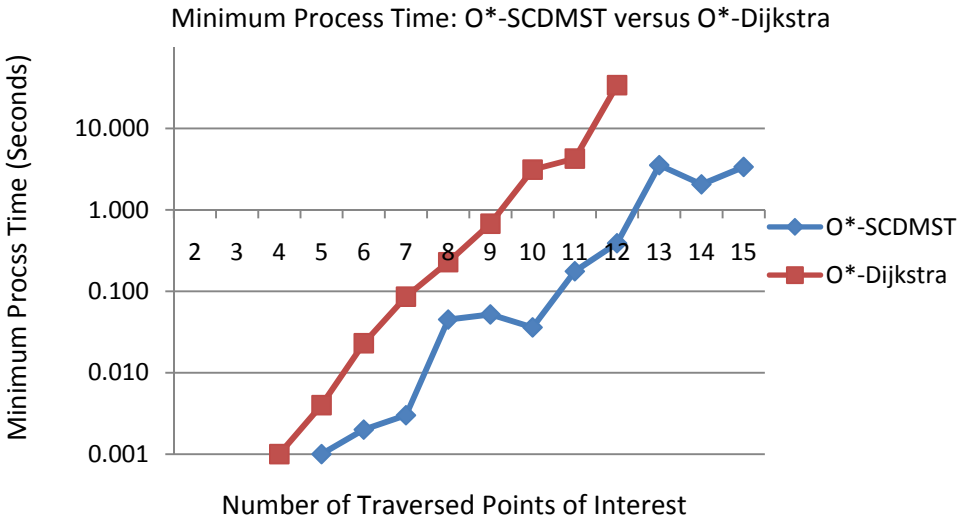


Figure 3: Minimum process time over different number of traversed consumer destinations: O*-SCDMST versus O*-Dijkstra

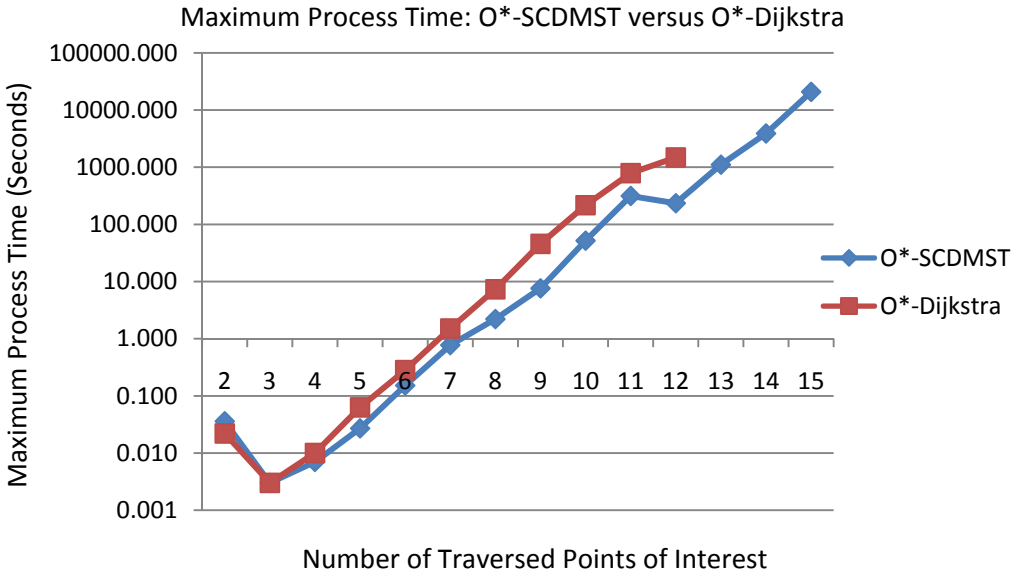


Figure 4: Maximum process time over different number of traversed consumer destinations: O*-SCDMST versus O*-Dijkstra

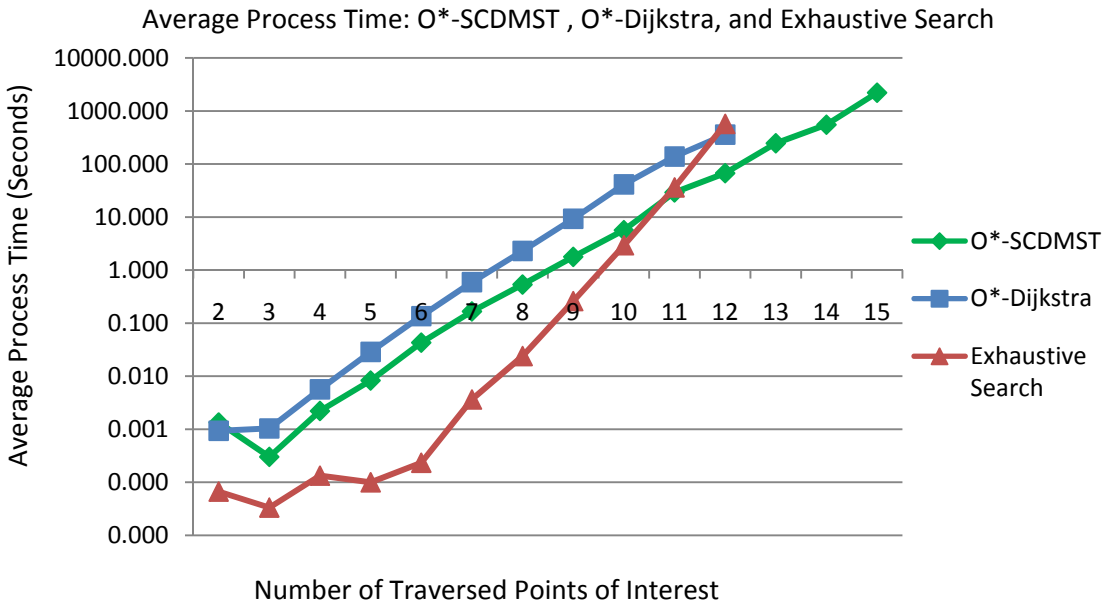


Figure 5: Average process time over different number of traversed cities: O*-SCDMST, O*-Dijkstra, and Exhaustive Search (ES)

Based on MaxPT shown in Figure 4, O*-SCDMST outperforms O*-Dijkstra when NPoI larger than 2. This is due to the fact that O*-SCDMST requires additional time to compute the SCDMST heuristic. However, when NPoI becomes larger, this additional time is no longer a dominant factor.

Based on APT shown in Figure 5, when NPoI is less than 11, both O*-SCDMST and O*-Dijkstra are slower than the naïve method. This is due to the fact that both O*-SCDMST and O*-Dijkstra require operations on their closed lists and priority queues, and O*-SCDMST requires the computation of SCDMST heuristics. When NPoI is larger than 10, O*-SCDMST performs increasingly faster than the naïve method. O*-Dijkstra always performs worse on average than the naïve method when NPoI is smaller than 12. However, when NPoI is larger than 11, O*-Dijkstra begins to outperform the naïve method. On Average, O*-SCDMST can process OSTQ of up to 14 NPoI within 10 minutes.

In Figure 3 through Figure 5, it is noticeable that O*-Dijkstra is mostly exponential in time complexity. According to Figure 6, O*-SCDMST is mostly exponential in average process time.

6. Conclusion

The contribution of this paper includes four components: 1) A SCDMST heuristic is proposed to calculate the global heuristics in O* to process OSTQ in a fully connected directed graph; 2) D-ODPrim is provided to retrieve a SCDMST for a directed connected graph; 3) The SCDMST heuristic is proved to be globally consistent if the edge costs of a fully connected directed graph obey the triangle inequality; 4) the corresponding O*-SCDMST's performance is statistically studied using a real data set. Based on the results, O*-SCDMST can retrieve an optimal solution for an OSTQ of 15 points of interest within 4 seconds at best and of 14 points of interest in 10 minutes on average. O*-SCDMST is always faster than O*-Dijkstra when the number of points of interest is larger than 2. On average, O*-SCDMST is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of points of interest is larger than 11.

References

Franco P. Preparata, Michael Ian Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York. 1985.

Joseph. B. Kruskal. *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*. In: Proceedings of the American Mathematical Society, Vol 7, No. 1, pp. 48–50, 1956.

Matteo Fischetti, Paolo Toth. A polyhedral approach to the asymmetric traveling salesman problem.Management Science, Vol 43, No. 11, pp. 1520-1536.

Michael R. Garey, Donald S. Johnson, and Larry Stockmeyer. *Some simplified NP-complete problems*. Proceedings of the sixth annual ACM symposium on Theory of computing, p.47-63. 1974.

Qifeng Lu, Kathleen Hancock. *O*: A Bivariate Best First Search Algorithm to Process Optimal Sequence Traversal Query in a Graph*. Submitted to the Journal: Journal of Artificial Intelligence Research, 2008a

Qifeng Lu, Kathleen Hancock. *MST: to Providing a Globally Consistent Heuristic to Process Optimal Sequence Traversal Query in a Euclidean Graph.* Submitted to the journal Transportation Science, 2008b

Robert C. Prim: *Shortest connection networks and some generalizations*. In: *Bell System Technical Journal*, 36, pp. 1389–1401,1957

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 1997

Wikipedia. *Travelling Salesman Problem*. http://en.wikipedia.org/wiki/Traveling_salesman_problem, last retrieved on October 30, 2008.

CHAPTER 7: OPTIMAL SEQUENCE TRAVERSAL QUERY PROCESSING IN A LARGE TRANSPORTATION NETWORK

by

Qifeng Lu

Kathleen Hancock

OPTIMAL SEQUENCE TRAVERSAL QUERY PROCESSING IN A LARGE TRANSPORTATION NETWORK

Qifeng Lu
Civil and Environmental Engineering Department, Virginia Polytechnic Institute and State University, Alexandria, VA 22314, USA
qilu1@vt.edu

Kathleen Hancock
Civil and Environmental Engineering Department, Virginia Polytechnic Institute and State University, Alexandria, VA 22314, USA
hancockk@vt.edu

Abstract

Optimal Sequence Traversal Query (OSTQ) is a graph based query that retrieves a minimum cost path that starts from a predefined vertex, traverses a set of given vertices, and ends at a predefined vertex. One of its applications is trip planning in transportation. A user may start from his/her office, visit a set of consumer destinations, and end at home. In this paper, O*-SCDMST (Semi-Connected Directed Minimum Spanning Tree), a best first heuristic based search algorithm proposed in the artificial intelligence domain, is introduced to retrieve an optimal solution for an OSTQ in network distance in a large transportation network. The network used for the case study is the street network of Fairfax County and Fairfax City in Virginia, US. A set of experiments were performed on O*-SCDMST, O*-Dijkstra, and the naïve exhaustive search method, with a collection of consumer destinations within this area. The results show that O*-SCDMST is always faster than O*-Dijkstra when the number of points of interest is larger than 6. In the best case, O*-SCDMST (Minimum Spanning Tree) can retrieve an optimal solution for an OSTQ of 15 points of interest in less than 3 seconds. On average, O*-SCDMST is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of points of interest is larger than 10.

Keywords

OSTQ, O*, O*-Dijkstra, O*-SCDMST, Routing, Trip Planning

1. Introduction

Optimal Sequence Traversal Query (OSTQ) is a graph based query that retrieves a minimum cost path that starts from a predefined vertex, traverses a set of given vertices, and ends at a predefined vertex (Qifeng Lu, Kathleen Hancock, 2008a). The graph, *g-G*, may include normal vertices that are neither vertices of interest, nor the given origin or destination. In transportation, an OSTQ corresponds to a trip planning query that asks for a shortest or quickest path that traverses a set of locations within the given origin and destination. For example, a user may start from his/her office, go to a supermarket, a book store, a restaurant, a movie theater, and end at home.

Similar to a travelling salesman problem (Thomas H. Cormen, Charles E. Leiserson, etc., 1997), OSTQ is NP hard (Michael R. Garey, Donald S. Johnson, etc., 1974) (Wikipedia, 2008). The naïve method that retrieves the optimal solution would compute all possible order combinations of the given points of interest. The time complexity is O(n!), where n is the number of locations of interest.

O*, a bivariate best first algorithm that uses problem domain knowledge to guide the search for the optimal solution to an OSTQ in a *g-G* graph, was recently proposed (Qifeng Lu and Kathleen Hancock, 2008a). As exact solutions, two special cases of O*, O*-SCDMST (Semi-Connected Directed Minimum Spanning Tree) (Qifeng Lu and Kathleen Hancock, 2008b) and O*-Dijkstra (Qifeng Lu and Kathleen Hancock, 2008a), were identified to process OSTQ in a fully connected directed graph. The O*-SCDMST

is proved to be optimally efficient in a fully connected directed graph that obeys the triangle inequality.

In this paper, O*-SCDMST algorithm is introduced to process OSTQs in a large transportation network with a network distance metric. To analyze its performance, both O*-Dijkstra and the naïve exhaustive search method that computes all possible order combinations of the points of interest are used as the baselines. Experiments were performed with a set of collected consumer destinations within a large dense urban transportation network, the Fairfax County and Fairfax City network in Virginia, USA.

2. Background

In a *g-G* graph, based on some rule, O* incrementally searches all paths from the starting vertex and traversing the vertices of interest, until it finds a path of minimum cost to the goal. O* uses a vertex's identification plus its *VisitList*, a sorted list that consists of a sorted sequence containing traversed vertices of interest along the path, to describe a state in the search. At each step, O* uses a distance-plus-cost heuristic function, defined as *f(s)* for a state *s*, to determine the order in which the search visits states in the graph:

$$f(s)=g(s)+h(s) \qquad (1)$$

where

g(s) is the cost function of the path from the initial state to the current state, and

h(s) is the global heuristic that estimates the distance from the current state to the goal state.

O* first takes the paths most likely to lead towards the goal, which means the lower the *f(s)*, the higher is the priority for a vertex to be expanded.

Whenever an equal f(s) occurs, the state with a larger *VisitList* will be the next to expand. Otherwise, one is randomly selected. State A's *VisitList*, vl_A, is larger than State B's *VisitList*, vl_B, i.e., $vl_A > vl_B$, if the length of vl_A is longer. In other words, the path from the start state to A traverses more vertices of interest than that to B.

For an N-point traversal problem, O* first generates the source state that contains the given vertex and an empty *VisitList*. For all the states in the open list, the algorithm expands the state with the lowest *f(s)* value, and its children states are generated. A child state always inherits the *VisitList* of its parent whenever the child is not a vertex of interest; otherwise, the child's *VisitList* will be incremented by adding the vertex to it. The process continues until a goal state whose vertex is the final goal and *VisitList* contains all the vertices of interest or no solution is found. Once a goal state is reached, the algorithm will retrieve the obtained path using a data structure called *backpointer*, the combination of vertex identification and *VisitList*, to recursively obtain the parent until the origin state is reached.

O*-SCDMST is a special case of O* in that it uses a Semi-Connected Directed Minimum Spanning Tree (SCDMST) (Qifeng Lu, Kathleen Hancock, 2008b) to compute *h(s)* in a directed graph.

Consider a directed graph, *G(V,E)*, where *V* and *E* are the set of vertices and edges, respectively, and a starting vertex *s* and an ending vertex *e* in *E*. Associated with each edge *(i,j)* in *V* is a cost *c(i,j)*. Let $|V|=n$ and $|E|=m$. A semi-connected directed spanning tree, SCDST, is defined as a graph which connects, without any cycle, all vertices with $n-1$ arcs, while vertex *s* only has outgoing arcs and vertex *e* only has incoming arcs. It is

semi-connected in the sense that it does not necessarily connect any two points together. A semi-connected directed minimum spanning tree (SCDMST) is the graph with the minimum total edge cost among all SCDSTs. In other words, the problem is to find a SCDST, *G(V,S)* where *S* is a subset of *E*, such that the sum of *c(i,j)* for all *(i,j)* in *S* is minimized.

A SCDMST example is shown in Figure 1 where the SCDMST is highlighted in dark black. O*-SCDMST always obtains the optimal solution if an OSTQ problem obeys the triangle inequality in a δ graph whose edge cost is not smaller than 0, i.e., a shortest path from vertex *A* to vertex *B* that requires traversing any vertex *v* of interest is always longer than or equal to the shortest path from *A* to *B* without that requirement.

In O*, two important definitions exist: global consistency and global admissibility. *Global admissibility* means that the heuristic *h(s)* is admissible, i.e., its value is always smaller than or equal to the actual cost of the minimum-cost path from the current state to the goal state. Suppose an optimal path is from vertex *n* with *VisitList vl* to vertex *n'* with *VisitList vl'*. By default, *vl'>=vl*, which means *vl'* contains at least all the visited vertices of interest contained in *vl*. The heuristic *h(x)* is *globally consistent* if the following triangle inequality exists:

$$hg(n,vl) <= g^*(n,n', vl->vl') + hg(n',vl') \quad (vl'>=vl) \qquad (2)$$

Where *g** is the actual (minimum) cost from *(n,vl)* to *(n',vl')*.

Global admissibility guarantees that O* retrieves an optimal solution if one exists. Global consistency not only guarantees that O* retrieves an optimal solution, but also guarantees that once a state is expanded by O*, it will not be expanded again by O*. Global consistency in O* results in optimal efficiency in terms of the states to be expanded, i.e., if *h(s)* is globally consistent, compared to any other no-more-informed admissible heuristic search algorithm, O* expands the minimum number of states.

No related work has been identified for optimal solutions to process OSTQ in a transportation network.

3. OSTQ Processing in Network Distance in a Large Transportation Network

In this paper, O*-SCDMST is introduced to process OSTQ in network distance. To take full advantage of the globally consistent heuristic in a fully connected graph whose edge costs obey the triangle inequality and gain more information for the heuristic in O*-SCDMST from the problem domain, the shortest network distances between any two points from the points of interest and the given origin-destination pair are obtained first to form a matrix *M*. Then a fully connected graph is constructed using only these points, and any edge cost is the obtained shortest network distance between the two end points of the edge. If a path between two points does not exist, then their edge is assigned an infinitely large cost value. If the cost of the optimal solution is infinitely large, then an optimal solution does not exist. Figure 2 provides a reconstruction example. It is clear that the edge cost of the graph satisfies the triangle inequality.

Calculating the shortest distance between any two points in a large transportation network is time consuming to. To efficiently calculate these distances, TransCAD, the commercial software package in transportation planning (Caliper Corporation, 2005), is adopted. TransCAD is efficient because it uses Dijkstra (Thomas H. Cormen, Charles E. Leiserson, etc., 1997) and spatial index to calculate the shortest distances between a set of origins and a set of destinations (Caliper Corporation, 2005). In other words, since

Dijkstra is efficient to process one origin multiple destination route problem, and spatial index can be used to speed up the retrieval for a large network data stored in a hard disk, their combinations can be used to efficiently obtain the shortest distances between a set of origins and a set of destinations in a large transportation network.

4. Experiments and Result Analysis

The purpose of the experiments is to show the performance of O*-SCDMST in a large transportation network.

O*-Dijkstra, O*-SCDMST, and the exhaustive search were all coded in C#.NET, and GISDK (Caliper Corporation, 2007) was used to interface with TransCAD to obtain the shortest network distances between any two points. The experiments were performed on a Toshiba Satellite A215 Laptop with 2.0GB memory (RAM), AMD Turion™ 64*2 Mobile Technology TL-56 1.80HZ processors, and Windows Vista™ Home Premium operating system.

4.1 Data Set

The underlying transportation network is of Fairfax County and Fairfax City in Virginia, USA, a large dense metropolitan area. The network contains 35,435 vertices, and 82,926 directed edges. 964 actual consumer destinations from 36 categories were collected within the study region as summarized in Table 1. To simulate a scenario where a user starts from his/her office, traverses a set of consumer destinations, and ends at home, the commercial area and the residential area within Fairfax County were identified through the zoning layer (Fairfax County, 2008) as shown in Figure 3. 30 origins were randomly generated from the business area, and 30 corresponding destinations were randomly generated from the residential area. Figure 4 shows the street network, and the locations of the randomly generated origins and destinations plus the collected consumer destinations. Each selected consumer destination corresponds to a point of interest in OSTQ.

For each of the 30 randomly generated origin-destination pairs, the number of consumer destinations of interest varies from 2 to 15. For each number of consumer destinations of interest, 4 consumer destinations from different categories were randomly selected from all the collected consumer destinations. In total, there are 1680 experimental samples. All samples are used to perform tests on O*-SCDMST, and 1320 samples with less than 13 consumer destinations are used on the naïve method and O*-Dijkstra. To simply the process, consumer destinations, given origins, and given destinations, are represented by their closest vertex in the network.

4.2 Performance Measures

To study the performances of the three algorithms, the following performance measures are identified.

Average Shortest Route Distance (ASRD): the average shortest route distance obtained over all runs (miles);

Average Pre-Process Time (APPT): the average time required to obtain shortest distances between any two points including points of interest and the given origin-destination pair (seconds);

Minimum Process Time (MinPT): the minimum time required to obtain a solution for each number of points of interest (seconds);

Maximum Process Time (MaxPT): the maximum time required to obtain a solution for each number of points of interest (seconds);

Average Process Time (APT): the average time required to process a query over all runs (seconds);

Average Relative Process Time (Ave RPT): the ratio of the average time processed by either the exhaustive search or O*-Dijkstra over by O*-SCDMST;

Maximum Relative Process Time (Max RPT): the maximum ratio of the time processed by either exhaustive search or O*-Dijkstra over by O*-SCDMST among all runs;

Minimum Relative Process Time (Min RPT): the minimum ratio of the time processed by either exhaustive search or O*-Dijkstra over by O*-SCDMST among all runs.

4.3 Results and Discussion

The results are presented in Table 2. *ES* represents the naïve exhaustive search approach, and *NPoI* represents the number of points of interest, i.e., the number of consumer destinations to traverse. O*-D represents O*-Dijkstra, and O*-S represents O*-SCDMST. The processing time refers to the computer time required to obtain the shortest distance for an OSTQ on the constructed fully connected graph.

Figure 5 through Figure 10 are provided to visualize the performance measures provided in Table 2.

First, the larger the *NPoI* is, the longer it takes to obtain the shortest distances among any two points, including points of interest and the given origin-destination pair. The maximum *APPT* is 1.5 seconds for 15 points of interest. For all three approaches, *APPT* can generally be neglected when *NPoI* is larger than 11 when compared to *APT*.

In general, the larger the number of consumer destinations to traverse, the longer the distance of the shortest route for the corresponding OSTQ.

Based on *MinPT*, O*-SCDMST can retrieve the optimal solution within 1.5 seconds for an OSTQ of 15 *NPoI* from a fully connected graph. In other words, if *APPT* is included, O*-SCDMST only requires 3 seconds to obtain the optimal solution. However, based on *MaxPT*, it still may require 15376 seconds for another query with the same number of points of interest. This implies that O*-SCDMST's performance depends on how closely the selected SCDMST heuristic approaches to the actual cost.

Based on *MinPT* shown in Figure 5 and *MaxPT* shown in Figure 6, O*-SCDMST outperforms O*-Dijkstra in both measures over all runs.

Based on *APT* shown in Figure 7, when *NPoI* is less than 11, both O*-SCDMST and O*-Dijkstra are slower than the naïve method. This is due to the fact that both O*-SCDMST and O*-Dijkstra require operations on their closed lists and priority queues, and O*-SCDMST requires the computation of SCDMST heuristics. When *NPoI* is larger than 10, O*-SCDMST performs increasingly faster than the naïve method. O*-Dijkstra always performs worse on average than the naïve method as shown in Figure 7. However, O*-Dijkstra may outperform the naïve method when *NPoI* is larger than a certain threshold, although this has not been shown.

Based on *AveRPT* shown in Figure 8, in general, O*-SCDMST is much more efficient than O*-Dijkstra, and as *NPoI* becomes larger, O*-SCDMST performs increasingly faster than O*-Dijkstra.

Based on *MinAPT* shown in Figure 9, O*-Dijkstra may be faster than O*-SCDMST when *NPoI* is less than 7. This may be due to the fact that O*-SCDMST requires

additional time to calculate SCDMST heuristics. However, when *NPoI* becomes larger, O*-SCDMST is always faster than O*-Dijkstra, which implies that the time to calculate SCDMST heuristics is not dominant any more. Also, both O*-SCDMST and the exhaustive search present an ascending trend in MinAPT with the increase of *NPoI*.

Finally, based on MaxAPT shown in Figure 10, in the worst case, O*-Dijkstra may still perform faster than the exhaustive search when *NPoI* is larger than a certain threshold, although the threshold has not been established.

5. Conclusion

In transportation, an OSTQ can be regarded as a trip planning query. The paper applies O*-SCDMST to a standard transportation engineering problem to efficiently obtain shortest paths for OSTQ in a large transportation network. A set of experiments were performed on O*-SCDMST, O*-Dijkstra, and the naïve method. Based on the results, in the best case, O* can retrieve an optimal solution for an OSTQ of 15 points of interest within 3 seconds. O*-SCDMST is always faster than O*-Dijkstra when the number of points of interest is larger than 6. On average, O*-SCDMST is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of points of interest is larger than 10.

References

Caliper Corporation. *GISDK manual*. Caliper Corporation, Boston, USA, 2007

Caliper Corporation. *TransCAD manual*. Caliper Corporation, Boston, USA, 2005

Fairfax County. *Fairfax County GIS & Mapping Department*. http://www.co.fairfax.va.us/gisapps/pdfViewer/default.htm, last retrieved on December 12, 2008.

Michael R. Garey, Donald S. Johnson, and Larry Stockmeyer. *Some simplified NP-complete problems*. Proceedings of the sixth annual ACM symposium on Theory of computing, p.47-63. 1974.

Qifeng Lu, Kathleen Hancock. *O*: A Bivariate Best First Search Algorithm to Process Optimal Sequence Traversal Query in a Graph*. Submitted to the Journal: Journal of Artificial Intelligence Research, 2008a

Qifeng Lu, Kathleen Hancock. SCDMST: to Providing a *Globally Consistent Heuristic to Process Optimal Sequence Traversal Queries in a Fully Connected Directed Graph*. Submitted to Journal of Artificial Intelligence Research, 2008b

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 1997

Wikipedia. *Travelling Salesman Problem*. http://en.wikipedia.org/wiki/Traveling_salesman_problem, last retrieved on October 30, 2008.

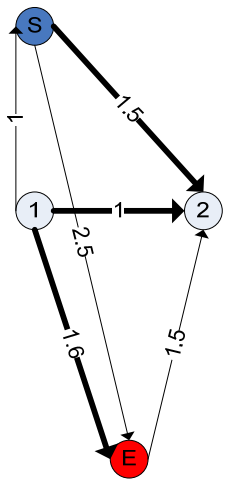
Table 1: Distribution of Number of Consumer Destinations for Each of 36 Categories

Category No.	Category Name	Number of Consumer Destinations
1	Athletic Wear	7
2	Books	14
3	Cellular Products and Services	5
4	Children's Fashions	13
5	Children's Shoes	11
6	Department Stores	27
7	Electronics	20
8	Entertainment	8
9	Flowers	4
10	Grocery & Drugs	12
11	Health & Beauty Services	33
12	Home Furnishings & Kitchen	22
13	Jewelry	17
14	Luggage & Leather Goods	10
15	Men's Fashions & Accessories	15
16	Men's Shoes	14
17	Music & Videos	10
18	Restaurants	37
19	Specialty Food & Beverage	14
20	Sports & Recreation	4
21	Supermarket	18
22	Tobacco	2
23	Toys & Games	8
24	Women's Fashions & Accessories	20
25	Women's Shoes	14
26	Hospital	4
27	Target	66
28	Kmart	4
29	Walmart	3
30	Auto repair	240
31	Gas Station	199
32	Bank of America	31
33	Citibank	6
34	Major Hotel	79
35	Costco	3
36	Post office	30

Table 2: Performance results for O*-SCDMST, O*-Dijkstra, and the naïve exhaustive search approach

NPoI	***ASRD***	**APPT**	***MinPT***		***MaxPT***		***APT***			***AveRPT***		***Min RPT***		***Max RPT***	
			O*-S	O*-D	O*-S	O*-D	O*-S	O*-D	ES	O*-D	ES	O*-D	ES	O*-D	ES
2	22.6	0.4	0.2	0.2	0.6	1.7	0.3	0.4	0.0	1.4	0.0	0.5	0.0	2.6	0.0
3	27.7	0.4	0.3	0.3	0.5	1.0	0.3	0.4	0.0	1.1	0.0	0.6	0.0	3.1	0.0
4	32.5	0.5	0.3	0.3	0.8	1.1	0.4	0.4	0.0	1.1	0.0	0.5	0.0	2.9	0.0
5	35.0	0.6	0.4	0.4	0.6	1.7	0.5	0.6	0.0	1.2	0.0	0.8	0.0	3.7	0.0
6	41.0	0.7	0.4	0.5	0.8	2.9	0.5	0.9	0.0	1.7	0.0	0.9	0.0	5.7	0.0
7	45.1	0.7	0.5	0.8	1.3	8.4	0.7	2.5	0.0	3.8	0.0	1.4	0.0	12.0	0.0
8	48.3	0.8	0.6	2.1	3.5	37.7	1.0	9.0	0.1	8.9	0.1	3.0	0.1	32.2	0.2
9	50.9	0.9	0.7	8.4	17.3	89.4	2.4	43.1	0.4	18.0	0.2	3.7	0.2	94.4	0.4
10	53.3	1.1	0.7	45.7	75.2	423.4	7.8	229.7	3.5	29.6	0.5	4.2	0.1	420.8	0.9
11	55.3	1.1	0.8	280.3	210.6	2124.5	19.0	1109.7	41.6	58.5	2.2	7.0	0.5	1917.6	60.0
12	55.3	1.2	0.9	409.3	1078.3	14845.7	78.4	3789.2	584.0	48.3	7.3	7.2	0.3	9141.4	655.2
13	58.5	1.3	1.0	-	1881.3	-	147.0	-	-	-	-	-	-	-	-
14	59.7	1.4	1.4	-	3005.6	-	190.9	-	-	-	-	-	-	-	-
15	60.7	1.5	1.3	-	15376.3	-	562.1	-	-	-	-	-	-	-	-

Abbreviations: *O*-S: O*-SCDMST; O*-D: O*-Dijkstra; ES: Exhaustive search; NPoI: Number of Points of Interest; ASRD: Average Shortest Route Distance; APPT: Average Pre-Process Time; MinPT: Minimum Process Time; MaxPT: Maximum Process Time; APT: Average Process Time; AveRPT: Average Relative Process Time; MinRPT: Minimum Relative Process Time; MaxRPT: Maximum Relative Process Time*

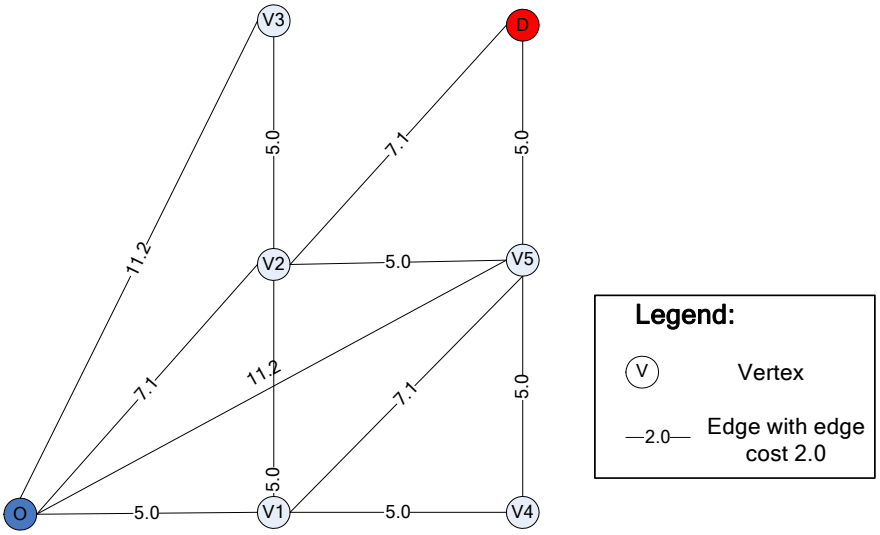


Where

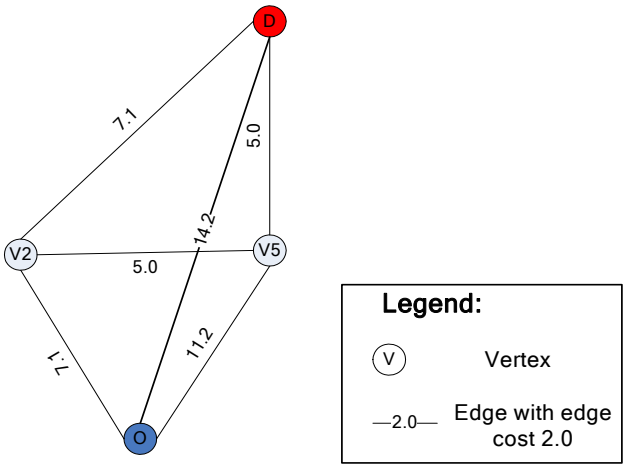
S: starting vertex, E: ending vertex.

The SCDMST tree is highlighted in dark black. The SCDMST starts from S, connects vertex 1 and vertex 2, and ends at E. No cycle exists. 3 edges are used to connect 4 vertices. Only outgoing edges exist for S and incoming edges exist for E. The SCDMST is semi-connected since not any two points are connected through the tree. For example, 2 and E are not connected together in the obtained SCDMST.

Figure 1: A SCDMST example



2(a): An undirected graph and the OSTQ problem (Origin: *O*, Destination: *D*, vertices to traverse: *V2* and *V5*)



2(b) The constructed fully connected graph

Figure 2: A graph reconstruction example

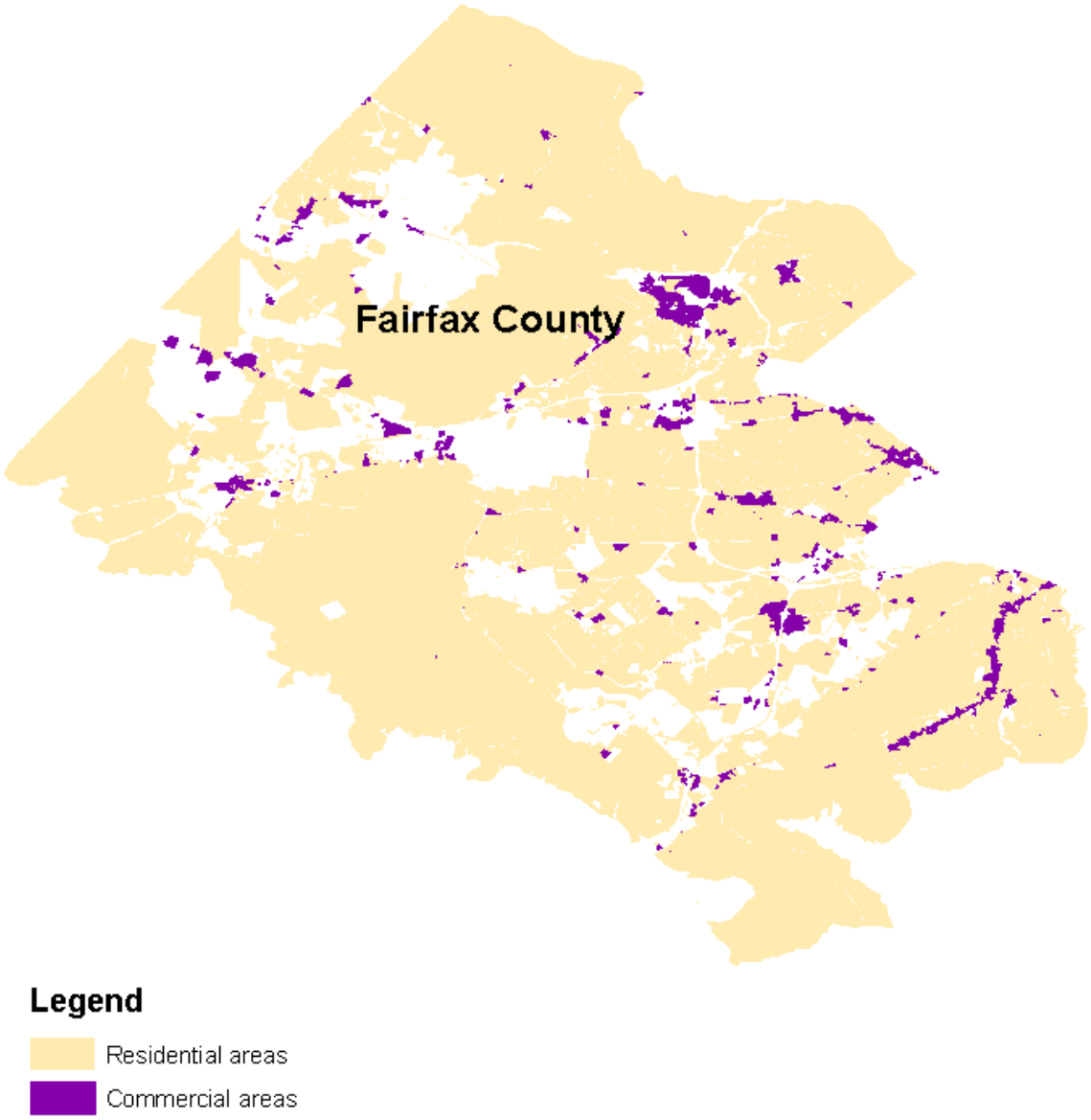


Figure 3: Distribution of residential areas and commercial areas in Fairfax County

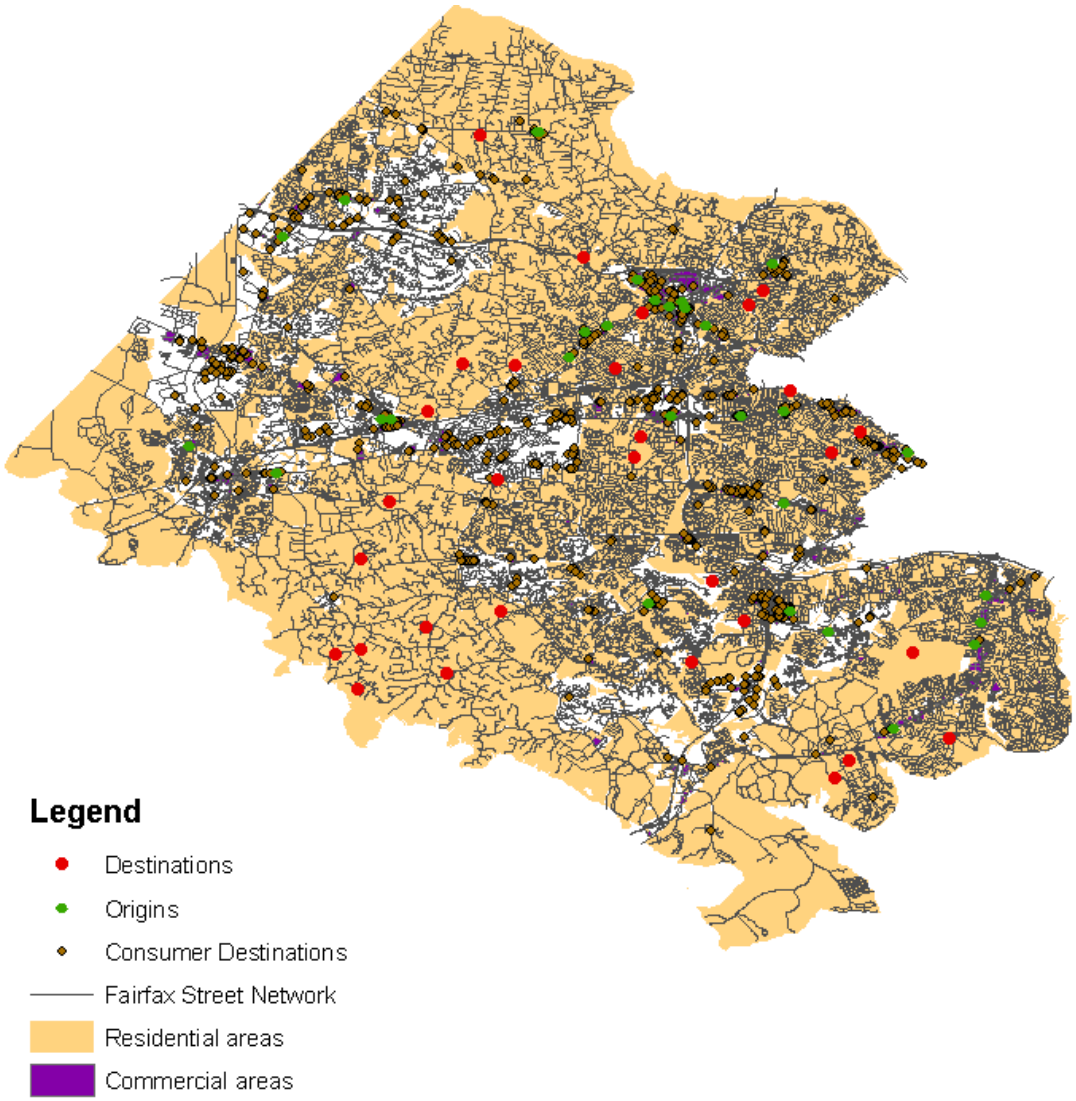


Figure 4: The Fairfax street network and the location distribution of origins, destinations, and consumer destinations

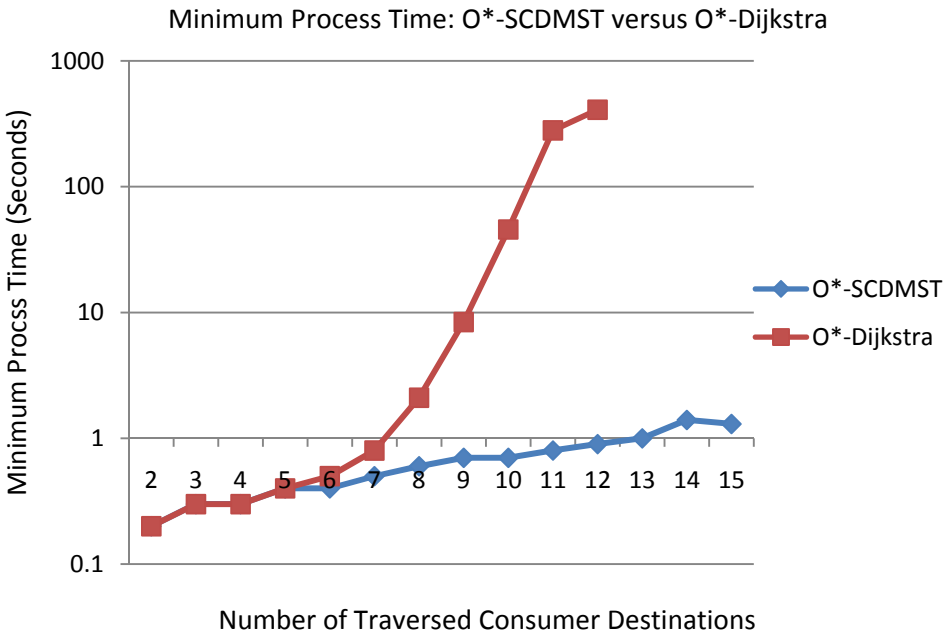


Figure 5: Minimum process time over different number of traversed consumer destinations: O*-SCDMST versus O*-Dijkstra

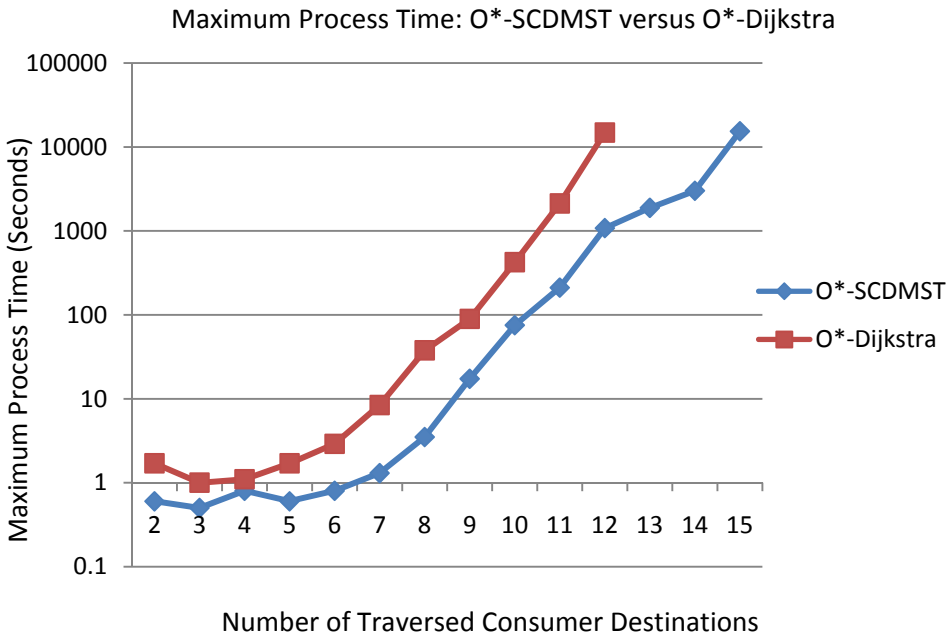


Figure 6: Maximum process time over different number of traversed consumer destinations: O*-SCDMST versus O*-Dijkstra

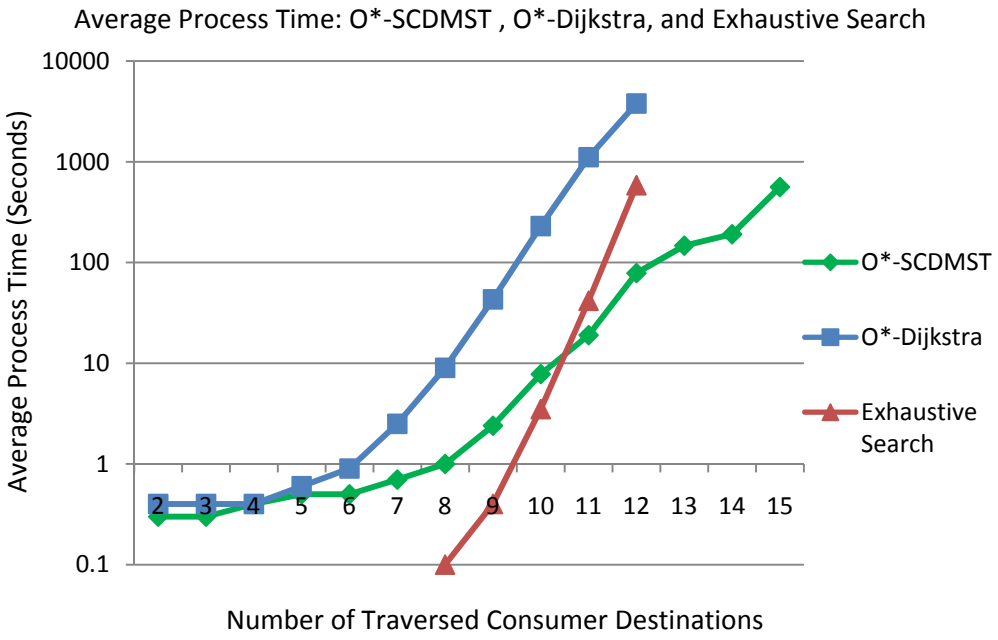


Figure 7: Average process time over different number of traversed consumer destinations: O*-SCDMST, O*-Dijkstra, and Exhaustive Search (ES)

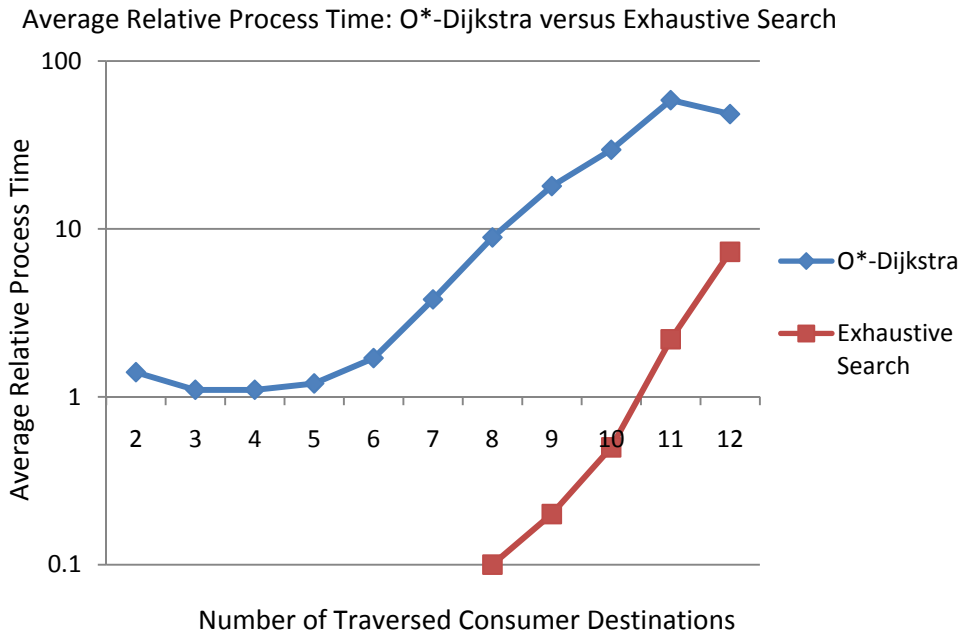


Figure 8: Average relative process time over different number of traversed consumer destinations: O*-Dijkstra versus Exhaustive Search (ES)

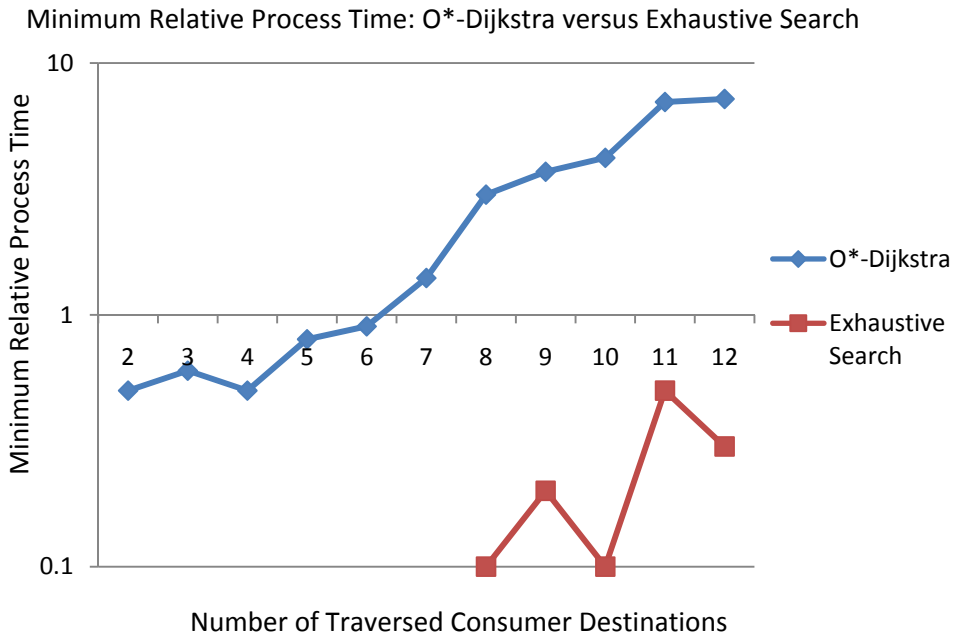


Figure 9: Minimum relative process time over different number of traversed consumer destinations: O*-Dijkstra versus Exhaustive Search (ES)

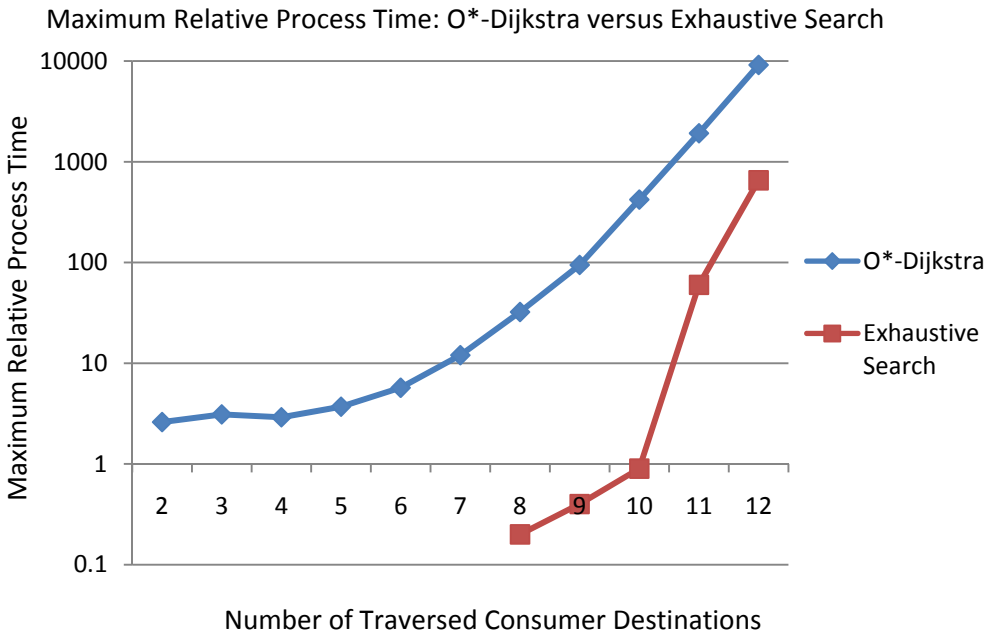


Figure 10: Maximum relative process time over different number of traversed consumer destinations: O*-Dijkstra versus Exhaustive Search (ES)

CHAPTER 8: SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

8.1 SUMMARY

Category-based queries are an important aspect of trip planning in transportation [1][2][3][4][5]. The motivation for this research was to develop advanced category-based queries corresponding to different trip planning scenarios and to provide optimal or near optimal solutions in a transportation network. This work was then generalized to a graph, and two category based queries were developed. Category Sequence Traversal Query (CSTQ) determines a minimum-cost path between a given origin and destination by traversing a set of categories of interest in a given order, with at least one selection from each category, and Optimal Sequence Traversal Query (OSTQ) determines a minimum-cost path between a given origin and destination by traversing a set of points of interest without a predefined order. Due to the extensive computing time required to process these two queries in a large scale environment such as a dense urban transportation network, efficient algorithms are necessary whenever processing time is a consideration. In this research, bivariate best first searches, which have the ability to incorporate inadmissible, admissible, and consistent global heuristics, as discussed in Chapter 3 through Chapter 7, from a problem domain to expedite the search process to retrieve sub-optimal, optimal, and optimally efficient solutions respectively, were developed to efficiently process CSTQ and OSTQ in a graph, corresponding to trip planning applications in transportation.

Figure 1 provides the overall hierarchy of the algorithms developed in this research and their relationships with the existing single-variate best first searches [6] [7] and A*[6] [7]. C*, and O* are instances of L#. When no categories of interest exist in CSTQ or no points of interest exist in OSTQ, C* or O* becomes A*, which is also an instance of L#. All these are algorithm families. For example, two different heuristics in C* will result in two algorithms that may provide solutions in different computation time. If any global heuristic used in C* or O* is inadmissible, admissible, or consistent, then the corresponding C* or O* algorithm will provide a sub-optimal, an optimal, or an optimally efficient solution.

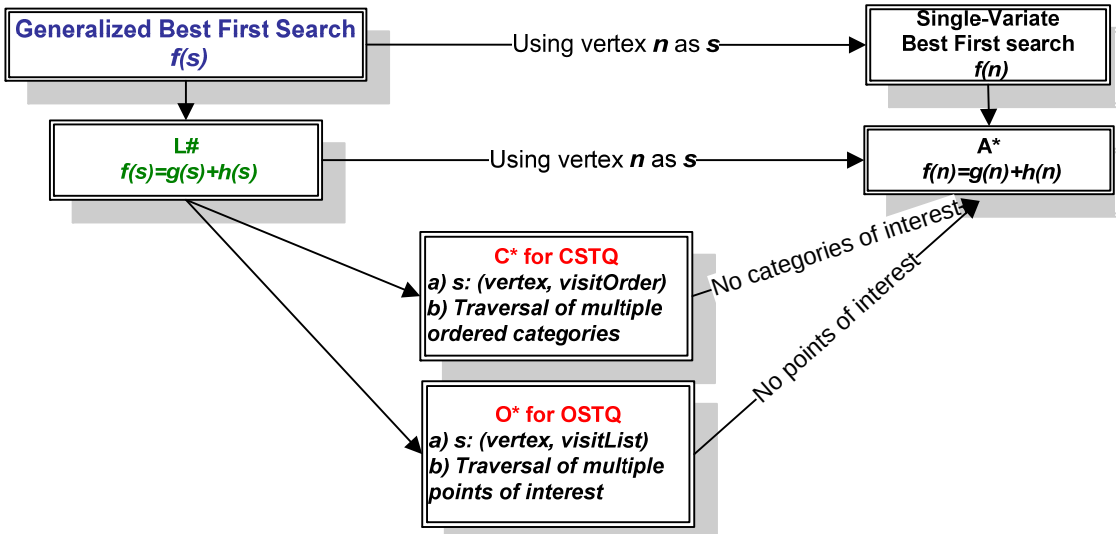


Figure 1： Algorithms developed in this research and their relationships with the existing single-variate best first searches and A*.

This research developed a set of best first searches to incorporate heuristics from a problem domain to expedite the processing for category based queries and provided the corresponding theorems to develop sub-optimal, optimal, and optimally efficient algorithms. Traditional best first searches use the evaluation function *f(n)* to estimate the promise of a vertex *n* to be expanded [6][7], which only keeps one *f(n)* for *n* and is not sufficient to process multi-category based queries in a graph because multiple states of a vertex describing the category traversal statuses is required during the search.

The bivariate best first searches, C* and O*, were developed to process category based queries including Category Sequence Traversal Query (CSTQ) and Optimal Sequence Traversal Query (OSTQ) respectively. It extends the traditional best first searches from single variate to multivariate search algorithms. Instead of evaluating the promise to expand a vertex, these generalized best first searches evaluate the promise of a state *s* to be expanded. A state *s* describes the status of a vertex and determines the condition of the route from the origin state to *s*. A vertex may have multiple states. Accordingly, one *f(s)* will be kept for each *s*, and thus multiple *f(s)*s can be recorded during the search for each vertex *n*. Consequently, the bivariate best first searches can be used to resolve the issues that are not processed by the existing single-

variate best first searches. Table 1 summarizes the differences between single-variate best first searches and bivariate best first searches to process category based queries identified in this research.

Table 1: Differences between single-variate and bivariate best first searches to process category based queries identified in this research

Best first searches	**Able to process multi-category based queries in a general graph?**	**Requires explicitly store partial routes to process category based queries?**	**What space is optimal efficiency proved in?**
Single-Variate Best First Search	No if a vertex is on a path multiple times	Yes	Sub graph of entire problem
Bivariate best first search	**Yes**	**No**	**Sub state-graph space**

The following summarizes the research in each chapter.

Chapter 3 presents the research that 1) generalizes existing single-variate best first searches to multivariate best first searches; 2) proposes a multivariate best first search algorithm, L#, and develops a set of new concepts including global heuristic, global admissibility, global consistency, local heuristic, local admissibility, local consistency, state graph, sub state graph, and sub state graph space, for best first searches; 3) provides C*, a bivariate best-first-search instance of L#, to process CSTQ in a graph, which extends the existing single-variate best first search to the bivariate best first search in the sense that the evaluation function used in C* is defined upon a bivariate state instead of the single variable to describe a vertex's identification; 4) discusses how to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate an optimal efficiency property in a discrete sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the first m categories of interest in the given order; and 5) adopts a bottom-up approach to provide a global heuristic, g-p, whose properties of optimality and optimal efficiency are analyzed based on local heuristics. Since a minimum-cost path query for a given origin and destination pair is a special case of CSTQ, A* [6] [7], the single-variate best first graph search

algorithm, is a special case of C*. Two special cases of C*, C*-P that uses g-p as the global heuristic, and C*-Dijkstra, are identified, and their performances are studied in two transportation networks of different scales.

Chapter 4 presents the research that proposes a bivariate best first search, O*, to process OSTQ within a graph, and discusses how to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and how a family of search strategies can demonstrate both optimality and optimal efficiency properties in an item-enumeration sub state graph space where a sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of enumerated points of interest. Three special cases of O* are identified and their performances are studied in a transportation network. Since a minimum-cost path query for a given origin and destination pair is a special case of OSTQ, A*, the best-first graph search algorithm, is a special case of the more general O*. Since TSP is a special case of OSTQ when the origin and the destination is the same, the set of theorems and algorithms provided for OSTQ is applicable to TSP.

Chapter 5 proves that O*-MST is optimally efficient in a Euclidean graph, and Kruskal's algorithm [8] with Delaunay Triangulation [9] are adopted to efficiently calculate MST heuristics. The performance of O*-MST is studied with a set of samples from a 130-city TSP problem [10].

In Chapter 6, O*-SCDMST is proposed to process OSTQ in a fully connected directed graph whose edge cost obeys the triangle inequality. O*-SCDMST is an O* algorithm using a semi-connected directed minimum spanning tree (SCDMST) to provide the global heuristic, which is proved globally consistent. A set of experiments was performed on O*-SCDMST and two other algorithms, O*-Dijkstra and the exhaustive search that computes all possible traversal combinations, with sample problems generated from an online TSP data set [11].

Chapter 7 introduces an O* algorithm, O*-SCDMST, that uses Semi-Connected Directed Minimum Spanning Trees (SCDMSTs) to obtain heuristics to efficiently process OSTQ using the network distance metric in a large transportation network. It first constructs a fully connected graph using Dijkstra algorithm [12] to obtain shortest distances between any two

points from the given origin-destination pair and the points of interest. Thereafter, it uses the O*-SCDMST to retrieve the shortest route on the constructed fully connected graph. The performance of O*-SCDMST is studied in terms of average process time over different points of interest in a large dense urban transportation network.

8.2 CONCLUSIONS

This section describes the contributions of the research, summarizes a set of experiment results, and discusses the possible applications and extensions, the applicability of the developed algorithms within environments of different computation power, and limitations of the research.

8.2.1 Contributions of the Research

The contribution of this research includes five components. First, two novel types of category based queries in a graph are presented: Category Sequence Traversal Query (CSTQ) and Optimal Sequence Traversal Query (OSTQ). CSTQ, the multi-category traversal query with a specified order of visit, allows a user to obtain a minimum-cost route in a graph between a given origin-destination pair, sequentially traversing a set of vertices of interest, each of which may be chosen from a set of candidates in the same category. CSTQ is different from an Optimal Sequenced Route (OSR) query [13] [14] since an OSR query does not have a predefined destination. OSTQ allows a user to obtain a minimum-cost route in a graph between a given origin-destination pair, traversing a set of non-ordered vertices of interest. OSTQ extends the functionality of a Traveling Salesman Problem (TSP) [15] [16] [17] to allow the origin to be different from the destination. Second, the traditional single-variate best first search [6] [7] is extended to multivariate best first searches in the sense that multiple variables are used to specify the state of a vertex to be evaluated and expanded. For a state *s*, a multivariate best first search adopts estimates to the promise of *s* to be expanded by a heuristic evaluation function *f(s)* which, in general, may depend on the description of *s*, the description of the goal state, the information gathered by the search up to that state, and on any extra knowledge about the problem domain. L#, the multivariate best first search that uses $g(s)+h(s)$ as $f(s)$ to evaluate the promise of a state to be expanded, is presented, and a set of corresponding new concepts--state graph, sub state graph, sub state graph space, local heuristic, local admissibility, local consistency, global heuristic, global admissibility, and global consistency--is introduced. The multivariate best-first-search L# can not be directly applied to CSTQ or OSTQ because the variables in L# must first be

specified or instantiated in C* or O*. In C*, a state *s* is specified through (*vertex, visitOrder*) where a *vertex* represents a node in a graph and *visitOrder* represents the traversed categories by the partial path from the origin state to *s* while in O*, the state *s* is specified through (*vertex, visitList*) where a *vertex* still represents a node in a graph and *visitList* contains all the traversed vertices of interest from the origin state to *s*.

Third, two bivariate best-first-search instances of L#, C* and O*, are provided to process CSTQ and OSTQ, respectively. Both algorithms extend the traditional single-variate best first searches [6] [7] to bivariate best first searches. For each best first search algorithm, theoretical analyses are presented to incorporate heuristic information obtained from the problem domain into a formal mathematical theory of graph searching and to obtain a family of search strategies that demonstrate optimal and optimally efficient properties in a sub state graph space. Once global heuristics are globally consistent, C* is optimally efficient in a discrete sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of categories of interest in the given order, and O* is optimally efficient in an item-enumeration sub state graph space where each sub state graph contains the expanded vertices to each of which every path from the origin has traversed the same set of enumerated vertices of interest. In addition, since TSP is a special case of OSTQ, the proposed set of theories for O* also provides theoretical support to obtain optimal and optimally efficient solutions for TSP in a general graph.

Fourth, a family of algorithms including C*-P, C*-Dijkstra, O*-MST, O*-SCDMST, and O*-Dijkstra which result in optimal solutions is identified. An additional approximation algorithm, O*-Greedy that provides near-optimal solutions, is also presented. Table 2 summarizes these algorithms. Based on the results of case studies performed on these algorithms in transportation networks, and/or fully connected graphs, either directed or undirected, the following general conclusions are presented, followed by more detailed discussions. C*-Dijkstra and O*-Dijkstra are always optimal. C*-P, O*-MST, and O*-SCDMST are optimal under some conditions and optimally efficient when they satisfy additional constraints. C*-P is optimal and optimally efficient in a transportation network using network distance as the metric to process CSTQ. O*-MST is optimal in a transportation network using network distance as the metric. O*-MST and

O*-SCDMST are optimally efficient in a fully connected graph whose edge costs satisfy the triangle inequality. The approximation algorithm O*-Greedy retrieves near-optimal solutions more quickly than the other O* algorithms for OSTQ processing. Since heuristics from a problem domain are adopted by C*-P, O*-MST, or O*-SCDMST, in general, C*-P is much more efficient in retrieving optimal solutions for CSTQ than C*-Dijkstra, and O*-MST or O*-SCDMST is much more efficient in retrieving optimal solutions for OSTQ than O*-Dijkstra.

Table 2: Developed algorithms for C* and O* in this research

Algorithm Family	***Algorithm***	***Heuristics***	***Solution***
C* for Category Sequence Traversal Query (Chapter 3)	**C*-Dijkstra (Chapter 3)**	**No heuristics**	**Optimal**
	C*-P (Chapter 3)	**g-p**	**1) Optimal if g-p is globally admissible 2) Optimally efficient if g-p is globally consistent**
O* for Optimal Sequence Traversal Query (Chapter 4, Chapter 5, Chapter 6, Chapter 7)	**O*-Dijkstra (Chapter 4)**	**No heuristics**	**Optimal**
	O*-MST (Chapter 4, Chapter 5)	**Using Minimum Spanning Tree (MST) to provide global heuristics in a graph obeying the triangle inequality**	**1) Optimal; 2) Optimally efficient in a fully connected graph**
	O*-SCDMST (Chapter 6, Chapter 7)	**Using Semi-Connected Directed Minimum Spanning Tree (SCDMST) to provide global heuristics in a fully connected graph obeying the triangle inequality**	**Optimally efficient**
	O*-Greedy (Chapter 4)	**Using Minimum Spanning Tree to provide the global heuristics in a graph obeying the triangle inequality *(f(s)=h(s))***	**near-optimal**

To evaluate the performance of the developed algorithms, their time complexity and space complexity were analyzed for the worst case. A set of experiments were performed and reported

in Chapter 3 through Chapter 7 to evaluate the actual minimum, average, maximum computation time and number of expanded states.

The performance of any best first search depends on the closeness of its adopted heuristics to the actual costs [6], which is the case for the developed multivariate best first searches in this research. For example, if a heuristic is always equal to the actual cost, then C* will only expand states that are on the optimal paths, and thus C* will retrieve an optimal solution very quickly.

Table 3 summarizes the space and time complexities of the developed algorithms in the worst case. The complexities are derived directly from the corresponding discussions in Section 4.2.2 in Chapter 3 and Section 3.2.2 in Chapter 4.

Table 3: Time and space complexities of the developed algorithms in the worst cases

Complexity （Worst Case）	**C* for CSTQ**		**O* for OSTQ**		
	C*-P	C*-Dijkstra	O*-MST	O*-SCDMST	O*-Dijkstra
Time	$O(NM^2b^d)$	$O(NMb^d)$	$O(N!b^dN^2)$	$O(N!N^2)$	$O(N!b^d)$
Space	$O(NMb^d)$	$O(NMb^d)$	$O(N2^Nb^d)$	$O(N2^N)$	$O(N2^Nb^d)$

Where
N: Number of categories of interest
M: Maximum number of objects in a category
b: Branching factor, the number of children at each vertex
d: Largest depth to obtain a minimum-cost path between any two points within the set {origin, destination, points of interest from different categories}, where *depth* of a vertex is the length of the path to its root of a generated tree during the search process.

From Table 3, clearly the time complexity of an O* admissible algorithm that provides optimal solutions is more severe than its space complexity.

Based on the performance of average number of expanded states and average computation time, C*-P always uses less time to expand a smaller set of states than C*-Dijkstra (Table 3 through Table 8 in Chapter 3); O*-MST always uses less time to expand a smaller set of states than O*-Dijkstra (Table 2 through Table 5 in Chapter 4). Therefore, on average, more informative

heuristics lead to better performance in terms of space usage and computation time. Furthermore, O*-Greedy always uses less time to expand a smaller set of states than O*-Dijkstra or O*-MST (Table 2 through Table 5 in Chapter 4), so it is always desirable to obtain sub-optimal solutions if either computation time or space usage is a concern during a search.

Finally, in response to the original motivation for this research, O*-SCDMST is adopted to efficiently retrieve optimal solutions for OSTQ using network distance as the metric in a large transportation network.

8.2.2 Experimental Results

The purposes of the experiments are to evaluate the performances of the developed algorithms in terms of computation time, number of expanded states, and obtained route distance. A set of measures, including average process time, minimum process time, maximum process time, average number of states expanded, minimum number of states expanded, maximum number of states expanded, average obtained distance, average shortest distance, and so on, was developed and reported in Chapter 3 through Chapter 7. Summaries of these experimental results are presented below.

A set of experiments was performed on C*-P and C*-Dijkstra with four data sets provided in a transportation network, and the results show that C*-P performs much better than C*-Dijkstra in number of expanded nodes and processing time. On average, C*-Dijkstra is between 0.6 and 25.4 times slower and expands between 0.3 and 7.3 times more states than C*-P.

Another set of experiments was performed with O*-MST, O*-Dijkstra, and O*-Greedy in a transportation network. O*-MST performs better than O*-Dijkstra in terms of the average number of states expanded and the average processing time. In general, O*-Greedy being near-optimal is the fastest of these evaluated.

A third set of experiments was performed using O*-MST where Kruskal's algorithm with Delaunay Triangulation is adopted to efficiently calculate the MST heuristic in a Euclidean graph. Based on the results, O* can retrieve an optimal solution for an OSTQ of up to 16 points of interest in about 30 minutes on average and of up to 17 points of interest within 0.8 seconds at

best. O*-MST is always faster than O*-Dijkstra when the number of points of interest is larger than 2. On average, O*-MST is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of points to traverse is larger than 9.

A fourth set of experiments was performed using O*-SCDMST with SCDMSTs to provide global heuristics to process OSTQ in a fully connected directed graph. Based on the results, O*-SCDMST can retrieve an optimal solution for an OSTQ of 15 points of interest within 4 seconds at best and of 14 points of interest in 10 minutes on average. O*-SCDMST is always faster than O*-Dijkstra when the number of points of interest is larger than 2. On average, O*-SCDMST is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of points of interest is larger than 11.

The last set of experiments was performed with O*-SCDMST to process OSTQ in a large transportation network. Based on the results, in the best case, O* can retrieve an optimal solution for an OSTQ of 15 points of interest within 3 seconds. O*-MST is always faster than O*-Dijkstra when the number of points of interest is larger than 6. On average, O*-MST is significantly faster than both O*-Dijkstra and the naïve exhaustive search when the number of points to traverse is larger than 10.

From the experiments in Chapter 4, it is clear that a greedy, or approximate, approach leads to a faster response, but a worse solution in terms of the cost, or length, of the obtained path. Based on the Average Obtained Distance (AOD) in Table 4 in Chapter 4, Figure 2 shows the performance differences in terms of the average actual travel distance between having no approximation using O*-MST and having approximation using O*-Greedy. An algorithm with no approximation always retrieves the minimum-cost path, so the obtained travel distance is always smaller.

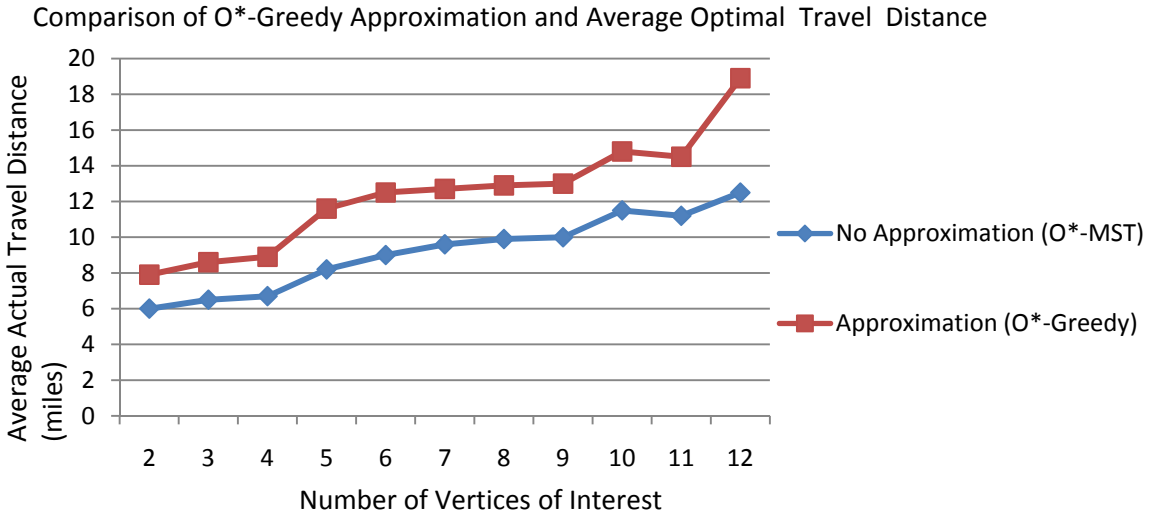


Figure 2: Comparison of O*-Greedy approximation and average optimal travel distance

Furthermore, based on the Average Process Time (APT) in Table 5 in Chapter 4, Figure 3 shows the performance differences in terms of average computation time between using no approximation with O*-MST and using approximation with O*-Greedy. Based on the results, the approximate O*-Greedy algorithm always retrieves a solution substantially faster than the exact O*-MST algorithm.

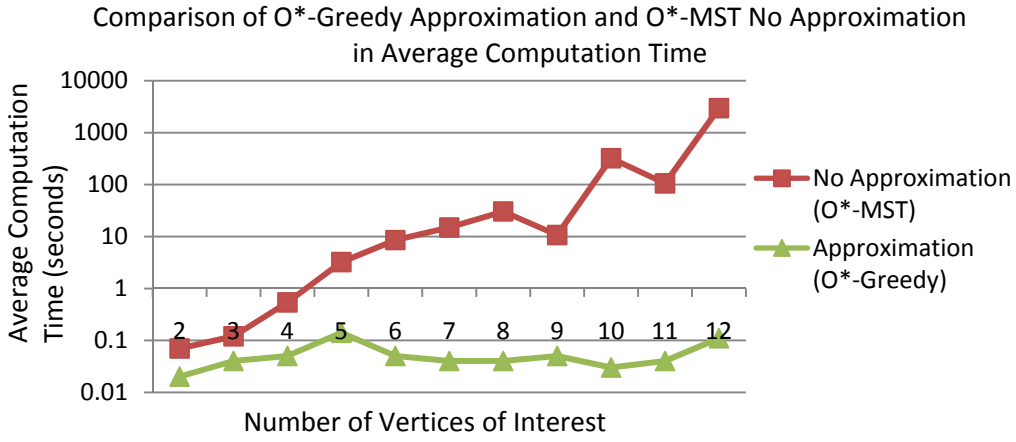


Figure 3: Comparison of O*-Greedy approximation and O*-MST no approximation in average computation time

The tradeoff between percent average relative increased distance and average reduced processing time based on this experiment is shown in Figure 4, where each point is labeled with the corresponding number of vertices of interest that were visited. The *Percent Average Relative Increased Distance* is the ratio of the average route distance obtained by O*-Greedy over the average optimal route distance obtained by O*-MST minus 1, multiplied by 100.

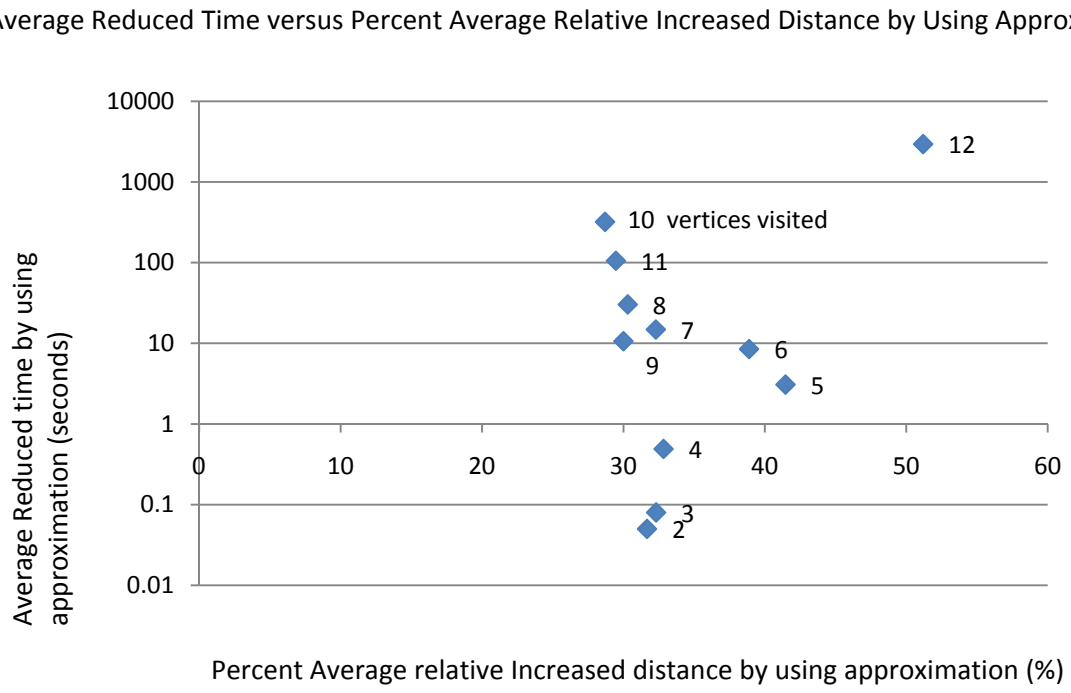


Figure 4: Average reduced time versus percent average relative increased distance by using approximation

It is clear that, in general, when the number of vertices of interest increases, the average reduced time increases, but the average relative increased distance does not necessarily increase, which is highly desirable.

Based on results of the experiments, it is beneficial to incorporate heuristics from the problem domain to accelerate the search process for an exact solution. However, the common concern of traditional best first searches is still valid for multivariate best first searches. The performance of a best first search always depends on the closeness of the adopted heuristics to the actual costs.

Consequently, it is difficult to predict the exact processing time for such a best first search and to identify the upper-bound cost of a route obtained from an approximate best first search algorithm such as O*-Greedy.

8.2.3 Applications of the Research

OSTQ and CSTQ and their corresponding algorithms can be applied directly to trip planning in transportation. CSTQ, the multi-category traversal query with a specified order of visit, allows a user to obtain a shortest route in a transportation network between his/her office and his/her home, sequentially visiting a set of points including a gas station, a supermarket, a coffee shop, and a department store, each of which may be chosen from a set of candidates. In this case, the Euclidean distance between any two points in a transportation network can be considered as a local heuristic, which is locally consistent. The global heuristic, g-p, obtained using this local heuristic is globally consistent, which means that when C*-P is adopted to obtain a solution to this CSTQ, it is optimally efficient and the result is optimal.

Users may have preferences to objects in the same category. A hierarchy of categories can be designed to fulfill this requirement. For example, in a Department Store category, there may be two sub-categories including high quality/price (such as Macy's) and mid level quality/price (such as JCPenney). Alternatively, sub-categories could be specific chains. In this case, users may choose to visit the Department Store category, or choose to visit a sub-category, either Macy's or JCPenney. In the former case, an obtained route may traverse either a Macy's or a JCPenney store. In the latter case, an obtained route will traverse the specified sub-category, either Macy's or JCPenney.

OSTQ, the multi-vertex traversal query, allows a user to obtain a shortest route in a transportation network between his/her office at Alexandria, and his/her home at Springfield, visiting the Giant supermarket at Springfield Plaza at Springfield, VA, the BJ wholesale store at Alexandria, VA, the Bank of America branch at 1717 King St Alexandria, VA in the optimized order. O*-SCDMST can optimize the order to visit these destinations and obtain the corresponding optimal route.

A rich set of problems from diverse domains corresponds to a TSP, which is a special case of OSTQ. As shown by the example in Chapter 4, the existing best first search for TSP can not resolve the issue where a point occurs multiple times along an optimal or suboptimal path, while the developed O* algorithms, including O*-MST, O*-Dijkstra, O*-SCDMST, and O*-Greedy can retrieve optimal or suboptimal solutions that satisfy this constraint. Therefore, this research extends the traditional best first search for TSP to effectively retrieve optimal or suboptimal solutions for more general TSPs. OSTQ extends TSP in the sense that the origin and the destination are not necessary the same in OSTQ, which make OSTQ more practical in the real world.

Many problems with additional constraints can be transformed to a standard CSTQ or OSTQ. Therefore, C* or O* does not need to be extended or changed. For example, O* can process a "no highway" OSTQ. The existing algorithms can be adopted to retrieve the shortest route between a given origin and a destination, excluding all edges that are classified as highway. Thereafter, a fully connected graph can be built, and O* can be applied directly to obtain the optimal order for points of interest. For another example, if a user changes points of interest during a trip, his/her route can be adjusted accordingly through the application of C* or O* to the new set of points of interest with his/her current location as the origin.

In addition, these algorithms can be further extended by the incorporation of the vertex properties. A vertex may have additional information of interest to a user. For example, in transportation, an intersection may have movement constraints such as no left turns or u-turns, and possible delays to obey the intersection operation regulations such as signal or stop/yield signs, etc. C* and O* are flexible enough to consider these properties during the state generation-expansion process and incorporate the cost, infinity in the case of an exclusion, incurred at each vertex into the total cost.

OSTQ may be applied to logistics in some limited situations when no time-window constraints or capacity constraints exist. OSTQ extends the ability of TSP to a set of pickup/delivery problems with different origins and destinations to traverse multiple locations of interest which can be processed by O* algorithms. For example, a small package delivery driver may start from

the depot and deliver packages to a set of delivery points before completing his trip at a garage. Conversely, a mail carrier can start at home and pick up mail from mail drop boxes completing the trip at the post office.

In computer science, an intelligent agent collects data from diverse locations and delivers it to a specified location for analysis. The agent can either compute the optimal path with O*-MST to arrive at the destination faster or obtain a suboptimal path faster with less computation power consumed with a greedy approach such as O*-Greedy.

In computer science/engineering or intelligent transportation systems, location based services that incorporate category based trip planning such as CSTQ and OSTQ and a collection of consumer destinations across the nation can be provided to travelers and visitors through wireless service providers such as Verizon, AT & T, and Sprint. Even without the locating service provided by a GPS device, today's technology can approximately locate a Smartphone's location through the signal strengths of the wireless signal towers nearby. Therefore, automatic location based services with O*-SCDMST, O*-Greedy or C*-P implemented in an in-vehicle navigation system can be deployed by a wireless provider.

8.2.4 Applicability of the developed algorithms within environments of different computation power

The developed algorithms are designed, as comparable algorithms are, to be implemented and incorporated in systems such as GIS, on laptops and desktops, or through handhelds, and other devices connected to the Internet.

For CSTQ processing corresponding to trip planning in a metropolitan area, C*-P is efficient enough to provide optimal solutions to traverse multiple categories of consumer destinations in a given order in a timely manner and can be implemented in currently available devices including hand-held devices, desktops, laptops, and web servers for either in-vehicle navigation with or without a Global Positioning System (GPS) support or online routing services. For OSTQ processing corresponding to trip planning in a metropolitan area, it would be desirable to incorporate inadmissible heuristics into O* to provide an inadmissible algorithm which could

retrieve sub-optimal solutions in a timely manner similar to C*-P. Whenever power consumption in a wireless device or computation time is a critical factor to obtain a solution for an OSTQ, O*-Greedy can be adopted for users having multiple destinations since O*-Greedy is significantly efficient in terms of computation time and the computational energy cost. It is also possible to implement O*-SCDMST to retrieve optimal solutions to traverse multiple locations of interest on a powerful back-end server or an online computation platform such as Google Earth, and then to return the obtained route in a map with detailed directions to a client.

8.2.5 Limitations of the Research

First of all, the performance of the developed multivariate best first algorithms in this research, including L#, C*, O*, C*-P, O*-MST, O*-SCDMST, and O*-Greedy, depends on the closeness of the adopted heuristics to the actual costs. Consequently, it is difficult to predict the exact time for these best first algorithms to process category based queries such as CSTQ and OSTQ.

The second concern of the identified algorithms in this research is that they may not be able to properly incorporate the specific constraints in logistics such as capacity. It is necessary for the identified algorithms to cooperate or interact with other algorithms to process queries with this kind of constraint. For example, to deliver a certain amount of commodities to diverse stores with a larger amount of demands, one truck may not be able to deliver all the required commodities to all stores. Consequently, it may be more meaningful to use an additional algorithm to assign the workload and adjust the set of stores of interest for each truck driver before a trip planning query is performed.

8.3 Recommendations for Future Research

The expected next phase of this research is to apply L# to resolve additional trip planning queries such as Multi-Rule Partial Sequenced Route (MRPSR) query which finds a minimum-cost path that is based on a number of traveling rules (or constraints) that are expressed as sub-sequences of locations. How to combine the order consideration and the selection of one point from multiple points within a category is necessary to define the variables in L# that are used to process a MRPSR query in a graph.

Different heuristics can be identified for an application in different situations. For CSTQ processing in a graph, if a certain global heuristic *gh* is globally admissible, then a global heuristic, *gh'*, that is always smaller than *gh* is also globally admissible. Consequently, the C* algorithm using *gh'* as its global heuristic still retrieves optimal paths. For example, for CSTQ processing in a transportation network, if C* uses the local heuristic, *LH*, the Euclidean distance between a vertex and the final destination, as its global heuristic, since *LH* is never larger than the global heuristic g-p used by C*-P, the C* algorithm retrieves optimal solutions. The average performance of this C* algorithm is better than C*-Dijkstra in terms of the computation time to retrieve an optimal solution when it adopts useful heuristics from the problem domain while C*-Dijkstra does not. Furthermore, compared to C*-P, it uses less time to obtain the global heuristics, which may outperform C*-P when the number of points in each category of interest is substantially large so that high computation time is required to obtain a g-p.

Sub-optimal solutions can be obtained through inadmissible heuristics which can substantially expedite the search process. To resolve the actual trip planning problem, sometimes sub-optimal solutions are highly desirable even when an algorithm is optimally efficient because it may still require extensive processing time to obtain the exact solutions. Furthermore, since the time complexity is more severe in an O* optimal algorithm, it is preferable to obtain sub-optimal solutions for OSTQ processing. For example, in a large transportation network with a large set of candidate points from each of a large number of categories, it still requires a large amount of time for C*-P to retrieve optimal solutions for CSTQ. Therefore, if a C* algorithm uses a global heuristic much larger than g-p, then this algorithm may retrieve acceptable suboptimal solutions much faster than C*-P.

In computer science, due to the high space and time complexity in C* and O*, similar to enormous variations of A*, a set of variations of C* and O* may be pursued. These variations include but are not limited to any time C*/O*, hierarchical C*/O*, parallel C*/O*, memory constraint C*/O*, and so on. The former three can be used to reduce the time complexity of C*/O* to obtain near-optimal or optimal solutions, while the latter may be used to reduce the memory usage while retrieving an optimal solution with the expense of increased computation time.

As indicated in the limitations section, these algorithms are not designed to capture many constraints in actual operations. Research incorporating these algorithms into logistics and/or operations research solution procedures may provide improvements to existing solutions.

A graph can conceptually represent a transportation network, a computer network, a wireless network, and so on. Processing OSTQ and CSTQ in a graph may correspond to specific path findings in these networks. Future research on path finding corresponding to OSTQ and CSTQ processing in a graph may be performed in these networks.

REFERENCE:

1. F. Li, D. Cheng, M. Hadjieleftherious, G. Kollios and S. Teng. On Trip Planning Queries in Spatial Databases. Proceedings of 9th International Symposium on Spatial and Temporal Databases, 2005.
2. X. Ma, S. Shekhar, H. Xiong, P. Zhang. Exploiting a Page-Level Upper Bound for Multi-Type Nearest Neighbor Queries. Proceedings of 14th International Symposium on Advances in Geographic Information Systems, 2006.
3. M. Kolahdouzan, M. Sharifzadeh and C. Shahabi. The Optimal Sequenced Route Query. University of Southern California, Computer Science Department, Technical Report 05-840, 2005
4. M. Sharifzadeh, C. Shahabi, Additively Weighted Voronoi Diagrams for the Optimal Sequenced Route Query, 3rd Workshop on Spatio-Temporal Database Management, pages 33-40, 2006.
5. H. Chen, W.S. Ku, M.T. Sun, R. Zimmermonn. The Multi-Rule Partial Sequenced Route Query. 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Irvine, CA, USA, 2008.
6. S. Russell and P. Norvig. Artificial Intelligence: a Modern Approach (2nd edition). Prentice Hall, 2002.
7. P. E. Hart, N. J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics SSC4 (2) (1968) 100–107
8. J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. In: Proceedings of the American Mathematical Society, Vol 7, No. 1, pp. 48–50, 1956.
9. F. P. Preparata, M. I. Shamos. Computational Geometry: An Introduction. Springer-Verlag, New York. 1985.
10. 130 City Problem (Churritz). ftp://ftp.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp/ch130.tsp. Last retrieved on October 30, 2008.
11. Matteo Fischetti, Paolo Toth. A polyhedral approach to the asymmetric traveling salesman problem.Management Science, Vol 43, No. 11, pp. 1520-1536.
12. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Introduction to Algorithms. The MIT Press, 1997.
13. M. Kolahdouzan, M. Sharifzadeh and C. Shahabi. The Optimal Sequenced Route Query. University of Southern California, Computer Science Department, Technical Report 05-840, 2005
14. M. Sharifzadeh, C. Shahabi, Additively Weighted Voronoi Diagrams for the Optimal Sequenced Route Query, 3rd Workshop on Spatio-Temporal Database Management, pages 33-40, 2006.
15. S. Arora. Polynomial Time Approximation Schemes for Euclidean TSP and Other Geometric Problems. Proceedings of 37th IEEE Symposium on Foundations of Computer Science, page 2, 1996.

16. S. Arora. Approximation Schemes for NP-hard Geometric Optimization Problems: A survey. Mathematical Programming, 2003.
17. N. Christofides. Worst-case Analysis of a New Heuristic for the Traveling Salesman Problem. Carnegie Mellon University, Computer Science Dept, Tech Technical report, 1976.