

# **(Branch and Bound) and A\* Search Using Fuzzy Underestimates**

**Prepared by  
Bilal M. Bader Eddeen Rababa'a**

**Supervisors  
Prof. Naim Ajlouni  
Prof. Ranjit Biswas**

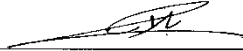
**A dissertation submitted in partial fulfillment of  
the requirements for the Degree of Doctor of  
Philosophy in Computer Science.**

**Graduate College of Computer Studies  
Amman Arab University for Graduate Studies  
May, 2007**

## AUTHORIZATION OF DISSERTATION

I, the undersigned "Bilal M. Bader Eddeen Rababa'a" authorize hereby "Amman Arab University for Graduate Studies" to provide copies of this dissertation to libraries, institutions, agencies, and other parties upon their request.

Signature: \_\_\_\_\_



© Copyright by

AMMAN ARAB UNIVERSITY FOR GRADUATE STUDIES (AAU)

## APPROVAL

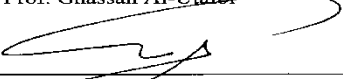
Name: **Bilal M. Bader Eddeen Rababa'a**

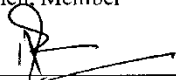
Degree: Doctor of Philosophy in Computer Science

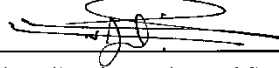
Title of dissertation: **(Branch and Bound) and A\* Search  
Using Fuzzy Underestimates**

### **Examining Committee:**

  
Chair: Prof. Ghassan Al-Utaibi

  
Dr. Khalid Kaabneh, Member

  
Dr. Ashit Kumar Dutta, Member

  
Prof. Naim Ajlouni, Member and Supervisor

Date Approved: June 24, 2007

## Acknowledgment

*"All praises and thanks to ALLAH"*

I would like to thank my supervisors Dr. Naim Ajlouni, and Dr. Ranjit Biswas who encouraged and helped me very much to produce this work; they were and are still my brothers.

Great thanks from my heart to my parents for their prayers, my wife for her patience and support, my sons and daughters for their patience too, and to my brothers, sisters, friends, and colleagues for their encouragements.

Finally, I would like to thank all lecturers, administration, and staff of Amman Arab University for Graduate Studies for their help and support.

## Dedication

I dedicate this work to:

My Parents, Wife, Children, Brothers, Sisters, and Friends

# Table of Contents

<b>Table of Contents.....</b>	<b>.VI</b>
<b>List of Tables.....</b>	<b>IX</b>
<b>List of Figures.....</b>	<b>X</b>
<b>List of Appendices.....</b>	<b>XV</b>
<b>Acronyms and abbreviations.....</b>	<b>XVI</b>
<b>Abstract.....</b>	<b>XVIII</b>
<b>Arabic Summary.....</b>	<b>XX</b>
<b>Chapter One Introduction.....</b>	<b>1</b>
1.1.Overview .....	1
1.1.1 Search Definition according to different fields.....	1
1.1.2 Artificial Intelligence Definition, and Activities .....	2
1.1.3 Fuzzy Logic, and Fuzzy Logic Activities .....	4
1.1.4 Expert System, and Fuzzy Expert System .....	5
1.1.5 Search as a Problem-Solving Process .....	6
1.1.6 Heuristic Search Definition .....	7
1.1.7.1 Blind Artificial Intelligence Searching techniques .....	9
1.1.7.3 Optimal Artificial Intelligence Searching techniques... ..	12
1.1.8 Previous Works of Fuzzy Shortest Path Problem in Networks.....	13
1.2 Statement of the Problem .....	15
1.3 Goals of this Dissertation .....	16
1.4 Dissertation Contributions.....	17
1.5 Dissertation Structure .....	17
<b>Chapter Two Preliminaries.....</b>	<b>19</b>
2.1 Fuzzy Set Theory.....	19
2.1.1 Crisp Sets and Fuzzy Sets .....	20
2.2.Illustrative example .....	26
2.1.2 Various Operations on Fuzzy Sets .....	26
2.1.3 Fuzzy Numbers.....	32
2.1.4 Hedge Definition .....	35
2.1.5 Fuzzy rule Definition.....	36
2.1.6 Fuzzy inference Definition .....	36
<b>2.2 Search Techniques.....</b>	<b>37</b>

2-2-1 Blind Methods.....	39
2.2.2 . Heuristically Informed Methods .....	51
2.2.3 Optimal Search .....	53
2.2.4 Properties of Search Methods .....	54
2.2.5 The effect of heuristic accuracy on performance .....	57
<b>Chapter three (branch and bound) search .....</b>	<b>59</b>
<b>Using fuzzy underestimates .....</b>	<b>59</b>
3.1 Introduction .....	59
3.2 Branch and Bound Search : Crisp Method.....	61
3.2.1 Adding Underestimates to (Branch and Bound) Search ..	67
3.3. Branch and Bound Search : Fuzzy Method .....	75
3.3.1 Fuzzy underestimate .....	76
3.4 Applications.....	83
3.4.1 Random net Application.....	83
3.4.2 Roads between Two Jordanian Cities Application .....	87
3.5 Conclusion .....	97
<b>Chapter four searching technique using .....</b>	<b>100</b>
<b>Fuzzy underestimates .....</b>	<b>100</b>
4.1 Introduction .....	100
4.2 Dynamic Programming .....	101
4.3 Search : Fuzzy Method.....	113
4.4 Applications.....	117
4.4.1 Random net Application.....	117
4.4.2 Roads between Two Jordanian Cities Application .....	122
4.5 Conclusion .....	132
<b>chapter five.....</b>	<b>135</b>
<b>performance evaluation and simulation.....</b>	<b>135</b>
5.1 Introduction .....	135
5.2 Simulation.....	135
5.2.1 Simulator Description.....	136
5.2.2 Examples .....	137
5.3. Simulation Survey: .....	170
5.4 Conclusion. ....	183
<b>Chapter six Conclusions and future work .....</b>	<b>185</b>
6.1 Introduction .....	185
6.2 Conclusions .....	187
6.2.1 Comparison between the Crisp Algorithms. ....	187

6.2.2 Effect of Using the Underestimated Value. ....	188
6.2.3 Effect of Using the Fuzzy Underestimated Value ( $\alpha_1$ ) .	188
6.2.4 Comparison between the Proposed Algorithms and Previous Works. ....	189
6.2.5 Comparison between the two Proposed Algorithms. ....	190
6.2.6 Effects of Time complexity, Space complexity, and Effective Branching Factor on Efficiency.....	191
6.2.7 Expected drawbacks for the two Proposed Algorithms.	192
6.3 Future Work.....	193
<b>References.....</b>	<b>194</b>
<b>Appendices.....</b>	<b>204</b>



## List of Tables

Table Number	Table Address	Page number
3.1	Evaluation of Heuristic B&B search strategies in terms of effective branching factor ( $b^*$ ).	96
4.1	Evaluation of Heuristic A* search strategies in terms of effective branching factor ( $b^*$ ).	128
5.1	Comparison of Number of Iterations for various search techniques.	165
5.2	Comparison of Time Complexity for various search techniques.	167
5.3	Comparison of Space Complexity for various search techniques.	170
5.4	Comparison of Effective Branching Factor for various search techniques.	173

## List of Figures

Figure Number	Figure Caption	Page number
2.1	Set A and elements x, y and z of U.	22
2.2	Membership function $\mu_A(t)$ of fuzzy set A.	26
2.3	Membership functions of the fuzzy sets A = "long travel time" and "very long travel time".	28
2.4	Membership functions of fuzzy sets A, B and $A \cup B$ .	29
2.5	Membership functions of the fuzzy sets of the cause of delay.	30
2.6	Membership functions of fuzzy sets A, B and $A \cap B$ .	31
2.7	Trapezoidal fuzzy number {3, 4, 6, 7}.	33
2.8	LR-type fuzzy number.	35
2.9	A basic search problem.	39
2.10	A search tree made from a net.	40
2.11	Basic search techniques classification.	41
2.12	An example of depth-first search.	44
2.13	Depth-first search algorithm explanation.	45
2.14	An example of breadth-first search.	46
2.15	Breadth-first search algorithm explanation.	48
3.1	Branch-and-Bound Search.	61
3.2	Branch and Bound Search Example.	62
3.3	Branch and Bound Search algorithm explanation.	66
3.4	Example of straight- line distances between each city and the goal.	68
3.5	Example of already traveled distances at each city.	69
3.6	B&B Search augmented by underestimates Example.	69

3.7	B&B Search augmented by underestimate/algorithm explanation.	71
3.8	Fuzzy data processing steps.	73
3.9	TFN model for “approximately a” or “approx. a”	74
3.10	TFN model for “approx. 27”.	76
3.11	TFN model for “approx. 27” with $\alpha = 0.9$ .	77
3.12	Flow chart procedure of B&B search with a fuzzy lower-bound estimate.	79
<b>Figure Number</b>	<b>Figure Caption</b>	<b>Page number</b>
3.13	Net graph with actual distance of each route.	81
3.14	Example of fuzzy estimates of remaining distances between each city and the goal.	82
3.15	The explored part of the tree.	82
3.16	Example of B&B Search augmented by fuzzy underestimates.	82
3.17	B&B Search augmented by fuzzy underestimate/ algorithm explanation.	84
3.18	Net graph from Karak to Irbid.	85
3.19	Fuzzy estimates of remaining distances.	86
3.20	A search tree of the net shown in figure (3.18).	87
3.21	The explored part of the tree.	88
3.22	Example of B&B Search augmented by fuzzy underestimates.	89
3.23	B&B Search augmented by fuzzy underestimate/ algorithm explanation.	93
3.24	Evaluation of Heuristic B&B search strategies in terms of effective branching factor.	96
4.1	An illustration of the dynamic programming principle.	100
4.2	A more precisely illustration of the DPP.	101
4.3	B&B search with DPP Search Example.	102

4.4	B&B search with DP algorithm explanation.	104
4.5	Example of straight- line distances between each city and the goal.	106
4.6	Example of already traveled distances at each city.	106
4.7	A* Search Example.	106
4.8	A* Search / algorithm explanation.	108
4.9	Flow chart procedure of B&B search with a fuzzy lower-bound estimate.	111
4.10	Net graph with actual distance of each route.	113
4.11	Example of fuzzy underestimates of distances remaining between each city and the goal.	113
4.12	The explored Part of the tree.	114
4.13	Example of A* Search augmented by fuzzy underestimates.	114
4.14	A* Search augmented by fuzzy underestimate/ algorithm explanation.	116
4.15	Net graph from Karak to Irbid with actual distance of each route.	117

Figure Number	Figure Caption	Page number
4.16	Fuzzy estimates of remaining distances from each city to Irbid.	118
4.17	A search tree of the net shown in figure (4.15).	119
4.18	The explored part of the tree.	120
4.19	Example of A* Search augmented by fuzzy underestimates.	121
4.20	A* Search augmented by fuzzy underestimate/ algorithm explanation.	125
4.21	Evaluation of Heuristic A* search strategies in terms of effective branching factor.	128
5.1	Jordan Map.	133

5.2	Net graph with actual distance of each route.	134
5.3	Splash Screen.	134
5.4	Nodes Screen without Nodes Information.	135
5.5	Nodes Screen after choosing (Sample 1).	136
5.6	Edges Screen without node's cost information.	137
5.7	Edges Screen after choosing (Sample 1).	138
5.8	Tree Screen after choosing (Sample 1).	140
5.9	Tree Screen using DB&B (Sample 1), step 1.	141
5.10	Tree Screen using DB&B (Sample 1), step 2.	142
5.11	Tree Screen using DB&B (Sample 1), step 3.	143
5.12	Tree Screen using DB&B (Sample 1), step 4.	144
5.13	Tree Screen using DB&B (Sample 1), step 5.	145
5.14	Tree Screen using DB&B (Sample 1), step 6.	146
5.15	Tree Screen using DB&B (Sample 1), step 7.	147
5.16	Tree Screen using DB&B (Sample 1), step 8.	148
5.17	Tree Screen using DB&B (Sample 1), step 9.	149
5.18	Tree Screen using DB&B (Sample 1), step 10.	150
5.19	Tree Screen using DB&B (Sample 1), step 11.	151
5.20	Results Screen using DB&B (Sample 1).	152
5.21	Net graph from Ras an Nakab to Amman .	154
5.22	Results Screen using FB&B (Sample 2).	155
<b>Figure Number</b>	<b>Figure Caption</b>	<b>Page number</b>
5.23	Tafila to Azraq.	156
5.24	Results Screen using FA* (Sample 5).	157
5.25	Net graph from Ras an Nakab to Azrak.	158
5.26	Results Screen using FB&B (Sample 3).	159
5.27	Net graph from Qatrana to Irbid.	160

5.28	Results Screen using A* Search (Sample 4).	161
5.29	Net graph from Karak to Azraq.	162
5.30	Results Screen using CB&B (Sample 15).	163
5.31	Comparison of Number of Iterations for four crisp search techniques.	166
5.32	(Line chart): Comparison of Time Complexity for three crisp search techniques.	168
5.33	(Line chart): Comparison of Time Complexity for two crisp and two fuzzy search techniques.	168
5.34	(Line chart): Comparison of Time Complexity for two search techniques.	169
5.35	(Line chart): Comparison of Time Complexity for two search techniques.	169
5.36	(Line chart): Comparison of Space Complexity for three crisp search techniques.	171
5.37	(Line chart): Comparison of Space Complexity for two search techniques.	171
5.38	(Line chart): Comparison of Space Complexity for two search techniques.	172
5.39	(Line chart): Comparison of Space Complexity for the proposed two fuzzy search techniques.	172
5.40	(Line chart): Comparison of Effective Branching Factor for four crisp search techniques.	174
5.41	(Line chart): Comparison of Effective Branching Factor for two search techniques.	174
5.42	(Line chart): Comparison of Effective Branching Factor for two search techniques.	175

## List of Appendices

Appendix Number	Appendix Title	Page number
1	Simulation Program pseudocodes	197
2	Simulation Program Codes	206
3	Fuzzy Logic	225

## Acronyms and abbreviations

Acronym/Abbrev.	Description
<b>A run</b>	Automatic run
<b>A*</b>	A* Searching Technique
<b>AAUGS</b>	Amman Arab University for Graduate Studies
<b>ABS</b>	Antilock Braking System
<b>AFLC</b>	Adaptive Fuzzy Logic Control
<b>AI</b>	Artificial Intelligence
<b>B</b>	Byte
<b>B&amp;B</b>	Branch and Bound Searching Technique
<b>BFS</b>	Breadth-First Search
<b>BM</b>	British Museum algorithm
<b>BMP</b>	British Museum Procedure
<b>BS</b>	Beam search
<b>CA*</b>	Crisp A* Searching Technique
<b>CB&amp;B</b>	Crisp Branch & Bound Searching Technique
<b>CDMA</b>	Code Division Multiple Access
<b>DAG</b>	Directed Acyclic Graph
<b>DB&amp;B</b>	Branch & Bound with Dynamic Programming
<b>DDP</b>	Discrete Dynamic Programming
<b>DFID</b>	Depth-First Iterative Deepening
<b>DFS</b>	Depth-First Search,
<b>DM</b>	Decision Maker
<b>DP</b>	Dynamic Programming
<b>DPP</b>	Dynamic Programming Principle
<b>EBF</b>	Effective Branching Factor
<b>FA*</b>	A* with Fuzzy Underestimation



<b>FB&amp;B</b>	Branch & Bound with Fuzzy Underestimation
<b>FL</b>	Fuzzy Logic
<b>FLC</b>	Fuzzy Logic Control
<b>G</b>	Goal Node
<b>G&amp;T</b>	Generate and Test Searching Technique
<b>GIS</b>	Geographic Information System
<b>HC</b>	Hill-Climbing
<b>HLS</b>	High-Level Synthesis
<b>HS</b>	Heuristic Search
<b>IEEE</b>	Institute of Electrical and Electronics Engineers
<b>IS</b>	Source of Information
<b>L-CPP</b>	Least-Cost Partial Path
<b>LR-FN</b>	Left Right -type fuzzy number.
<b>M run</b>	Manual run
<b>ms</b>	Milli Second
<b>NI</b>	Number of Iterations
<b>OCR</b>	Optical Character Recognition
<b>OERI</b>	Overall Existence Ranking Index
<b>OOP</b>	Object Oriented Programming
<b>S</b>	Start Node
<b>SC</b>	Space Complexity
<b>SPA</b>	Searching Performance Analyzer
<b>TC</b>	Time Complexity
<b>TFN</b>	Triangular Fuzzy Number
<b>TSP</b>	Traveling Salesman Problem
<b>UB&amp;B</b>	Branch & Bound with Underestimation
<b>VB</b>	Visual Basic

# **(Branch and Bound) and A\* Search Using Fuzzy Underestimates**

Prepared by  
**Bilal M. Bader Eddeen Rababa'a**

Supervisors  
**Prof. Naim Ajlouni**  
**Prof. Ranjit Biswas**

## **Abstract**

Search in Artificial Intelligence is a problem-solving technique that systematically explores a space of problem states. Branch and Bound (B&B) and A\* searching techniques are effective heuristic principle guided Artificial Intelligence Problem-Solving Techniques. Fuzzy Logic is related to Expert Systems, as an effective Expert systems tool which can deal with imprecise and uncertain data and permit inexact reasoning.

A search problem and its solution by the existing crisp methods of “Branch and Bound” and A\* search have been considered in this work. We have observed that these methods can be improved by using fuzzy theory.

In this dissertation, new methods of B&B and A\* searching techniques with fuzzy underestimation to the available Fuzzy information have been proposed by applying the Triangular Fuzzy Number Model in order to add Fuzzy Underestimation to the existing algorithms. A new improved version of searching techniques under uncertainty has been suggested. The corresponding algorithms have been given, and each of the two algorithms have been explained in details using two applications.

A simulation program has been introduced to evaluate the performance of the proposed algorithms. Six Searching techniques were analyzed and compared by computing Number of Iterations, Time Complexity, Space Complexity, and Effective Branching Factor for each algorithm. The proposed algorithms have been implemented and tested; The analysis and simulation results showed that Fuzzy Underestimated A\* and Fuzzy Underestimated "Branch and Bound" search techniques have achieved better efficiency, time complexity, and effective branching factor than all other compared searching techniques. Fuzzy underestimation increases the efficiency of A\* and B&B Augmented by Crisp Underestimation, enabling it to be more informed. Space complexity for Fuzzy A\* and Fuzzy Underestimated B&B is always less than that for crisp algorithms. The analysis also proved that Time complexity for Fuzzy A\* is better than that for Fuzzy Underestimated B&B, but with more Space complexity. This is because Fuzzy A\* has more memory requirements compared to Fuzzy B&B due to A\* maintaining all the generated nodes in memory.

# Fuzzy Underestimates Branch and Bound and A\* Search Using

إستعمال أقل تقديري ضبابي في تقنيتي البحث؛ التفرع والحدّ و(A\*)

إعداد

بلال محمد بدر الدين احمد الربابعة

إشراف الاستاذ الدكتور نعيم العجلوني

الاستاذ الدكتور رانجيت بسواس

## Arabic Summary

يعتبر البحث في مجال الذكاء الصناعي (Artificial Intelligence) تقنية لحلّ المشكلات تقوم على استكشاف فضاء حالات المشكلة بشكل منظم. وتعتبر تقنيتي (A\*) و(التفرع والحدّ) (B&B) من تقنيات الذكاء الصناعي الموجهة بمبدأ إرشادي (Heuristic)، بينما يُعتبر المنطق الضبابي (Fuzzy Logic) من أدوات الأنظمة الخبيرة (Expert Systems) الفعّالة التي تتعامل مع البيانات غير الدقيقة أو غير المؤكدة والتي تُسمح بالاستنتاج غير الدقيق.

تم اعتماد تقنيتي البحث (A\*) و B&B التقليديتين في هذا العمل وقد لاحظنا إمكانية تحسينهما باستخدام النظرية الضبابية.

في هذه الأطروحة تم اقتراح طريقة جديدة لتقنيتي (A\*) و B&B باستخدام (أقل تقديري ضبابي) للمعلومات المتوفرة اعتماداً على (نموذج العد الضبابي المثلثي) وذلك بإضافة (أقل تقديري ضبابي) إلى الخوارزميات الحالية. وهكذا تم اقتراح خوارزمية جديدة مُحسّنة لتقنيات البحث تحت ظروف عدم التأكد، وتم تقديم الخوارزميتين وتوضيح كل منهما بالتفصيل بوساطة تطبيقين .

كذلك تم تقديم برنامج محاكاة لتقييم أداء الخوارزميتين المقترحتين. و تم تحليلهما ومقارنتهما بأربع تقنيات بحث اخرى وذلك بحساب عدد التكرارات والزمن المستغرق والذاكرة المحجوزة و(معامل التفرع الفعال) لكُل خوارزمية.

بعد اختبار الخوارزميتين المقترحتين؛ أظهرت نتائج المحاكاة والتحليل أن تقنيتي البحث  $A^*$  و  $B\&B$  مع (أقل تقديرضبابي) قد حققتا كفاءة أفضل، وزمن تنفيذ أقل، ومعامل تفرع أقل من كافة تقنيات البحث الأخرى، حيث يعمل (أقل تقديرضبابي) على زيادة كفاءة التقنيتين بجعلهما (أكثر إطلاعا)، أما الذاكرة المحجوزة لتقنيتي البحث  $A^*$  و  $B\&B$  مع (أقل تقديرضبابي) فإنها تكون دائما أقل من الذاكرة المحجوزة للخوارزميتين  $A^*$  و  $B\&B$  العاديتين المعروفتين. وكذلك يظهر التحليل أن الزمن المستغرق لخوارزمية ( $A^*$  الضبابية) افضل منه لخوارزمية ( $B\&B$  الضبابية) ، ولكنها تستخدم حيز ذاكرة أكثر لأنها تحتفظ بكافة النقاط المتولدة في الذاكرة.

# Chapter One

## Introduction

### 1.1.Overview

Since the beginning of creation, humans have always searched for something or someone, starting by Adam when he searched for Eve. Till now humans are still searching for things using everything they can afford.

This dissertation deals with applying **Fuzzy Logic** to enhance **Branch and Bound** and **A\* Searching Techniques**, which are two of the best **Heuristic** problem solving techniques in **Artificial Intelligence**, while **Fuzzy Logic** is used as an effective **Expert System** tool. All of these techniques will be discussed.

#### 1.1.1 Search Definition according to different fields.

Search or searching in general is the act of trying to find something or someone. It is possible to distinguish between two forms of search. One may search for an item that is known to exist, with the intent to locate it. On the other hand one may search for an item whose existence is uncertain, in order to ascertain whether it exists or not. Searching can also be a metaphorical act, most frequently in reference to intangibles such as memories and emotions (Wikipedia Org., 2007).

Search is common, and can be used in many different areas of life, where there are different meanings for search according to type or field of searching, where search may mean the act and process of locating information in various

sources. For example, looking for a book in a library catalog (buffalo, Feb. 27, 2007), locating files scattered across the Internet, when you enter the engine name or key words, popular ones include Yahoo, Lycos, Google and Alta Vista (ncsu, Feb. 27, 2007), or locating information contained in a database by entering words or numbers in a search box (siue, Feb. 27, 2007) , while search in CDMA (Code Division Multiple Access) is a process where the phone scans the phase space of the short code looking for valid signals (san, Feb. 27, 2007) .There are a lot of different searching techniques, which are related to different fields like: networks searching techniques (Li & Wu,2002), database search techniques (KU Lib.2007), and web search techniques.

Artificial Intelligence (AI) is one of the most important fields in which search is an important topic; where search in Artificial Intelligence is a problem-solving technique that systematically explores a space of problem states, i.e., successive and alternative stages in the problem-solving process. Examples of problem states might include the different board configurations in a game or intermediate steps in a reasoning process. This space of alternative solutions is then searched to find an answer (Luger, 2005). Newell and Simon have argued that this is the essential basis of human problem solving (Newell & Simon, 1976). Indeed, when a chess player examines the effects of different moves or a doctor considers a number of alternative diagnoses, they are searching among alternatives (Luger, 2005).

### **1.1.2 Artificial Intelligence Definition, and Activities**

Intelligence is the ability to learn and understand, to solve problems and to make decisions (Negnevitsky, 2002). While AI is a branch of

computer science dealing with computer systems implementing restricted but definite part of human intelligence, particularly in knowledge acquisition, perception, learning, reasoning (Bhatkar, 1994).

A machine is thought to be intelligent if it can achieve human-level performance in some cognitive task. To build an intelligent machine, we have to capture, organize and use human expert knowledge in some problem area. (Negnevitsky, 2002). Intelligent Systems can help Experts to solve difficult analysis problems. Artificial intelligence can help us to solve difficult, real-world problems, creating new opportunities in business, engineering, and many other application areas. The engineering goal of artificial intelligence is to solve real-world problems; the scientific goal of Artificial Intelligence is to explain various sorts of intelligence (Winston, 2000).

From an engineering perspective, the **description of artificial intelligence** may be summarized as the study of representation and search through which intelligent activity can be enacted on a mechanical device. This perspective has dominated the origins and growth of AI (Negnevitsky, 2002).

The definition obviously lists the core activities of modern AI science, but it is necessary that it be kept open for additional aspect of intelligent behavior (Bhatkar, 1994).

Form the above definition it is evident that, while dealing with artificial intelligence, not only the achievements of contemporary mathematics & computer science are relevant, but also the results of disciplines from the humanities such as: linguistics, cognitive science, psychology, etc., as well as psychology, neurology,



prosthetic, etc.. Apart from the character the AI might be given, the applied artificial intelligence should be viewed as part of engineering, implementing the intelligent systems for: natural language processing, speech understanding, computer vision, autonomous robots, and domain expertizing (Including the earlier systems for game-plying, theorem proving, general problems solving, etc.) (Bhatkar, 1994).

### **1.1.3 Fuzzy Logic, and Fuzzy Logic Activities**

Fuzzy sets were initiated by Zadeh (Zadeh, 1965). In (Zadeh,1973) Zadeh made an extension of the concept of a fuzzy set by an interval-valued fuzzy set (i.e. a fuzzy set with an interval-valued membership function) (Bustince & Burillo, 1995). Since then, researchers have found numerous ways to utilize this theory to develop new mathematical methods of fuzzy inference and approximate reasoning (Baldwin, 1981; Gaines, 1976;Hellendoorn ,1992; Yager, 1980; Zadeh, 1975; Sun & Hadipriono, 1995) . In 1991 there were (1400) paper dealing with fuzzy systems (Rajagopalan, Washington, Rizzoni & Guezenec, 2003).

Later many authors have used fuzzy systems in different fields of Science; for example R. Sambuc (Sambuc, 1975) in Medical diagnosis in thyroidian pathology, M. B. Gorzalczany (Gorzalczany, 1987) in approximate reasoning, I. B. Turksen in Interval-valued logic, etc ..., these works and others show the importance of these sets (Bustince & Burillo, 1995).

Fuzzy logic can be used in Automobile and other vehicle subsystems, such as ABS, Air conditioning, Cameras, Digital image processing, Rice cookers,

Dishwashers , Elevators, Refrigerators,Washing machines (which sense load size and detergent concentration and adjust their wash cycles accordingly) and other home appliances, (Wikipedia, Ret. 22 February 2007), Agriculture, GIS (Foley & Petry, 2000), Image Processing, Machine Learning, Machine Vision, Medicine, Model Validation, OCR, Fuzzy Control and Fuzzy Robots ( Fujisawa *et al*, 1993; Saffiotti, Ruspini & Kurt,1993; Silva, 1995; Magdalena & Monasterio,1993), Fuzzy Logic Control and Adaptive Fuzzy Logic Control (AFLC) (e.g., BOLLOJU, 1995; Postlethwaite, 1994), Industrial Dryer (Bremner & Postlethwaite, 1995), Coke Oven (Tobi & Hanatusa, 1992), Steam Generator (Kuan, Lin & Hsu, 1992), Sludge Plant (Yu &Kandel, 1990), Oil Processing (Aliyev &Tserkovnyy, 1990), Liquid Level (Graham & Newell,1988), Nuclear Engineering (Ruan, 1995), Shape Recognition, Telecommunications, etc ...

Unlike two-valued Boolean logic, fuzzy logic is multi-valued. It deals with degrees of membership and degrees of truth. Fuzzy logic uses the continuum of logical values between 0 (completely false) and 1 (completely true). Instead of just black and white, it employs the spectrum of colours, accepting that things can be partly true and partly false at the same time. Classical binary logic now can be considered as a special case of multi-valued fuzzy logic (Negnevitsky, 2002).

#### **1.1.4 Expert System, and Fuzzy Expert System**

Fuzzy Logic is related to Expert system as an effective Expert systems tool, where Expert system can be defined as: a computer program capable of performing at the level of a human expert in a narrow domain.Unlike conventional programs, expert systems can deal with incomplete and

uncertain data and permit inexact reasoning. However, like their human counterparts, expert systems can make mistakes when information is incomplete or fuzzy. Fuzzy Expert System is a collection of fuzzy rules and membership functions that are used to reason about data, by using fuzzy logic instead of Boolean logic.(Negnevitsky, 2002).

Fuzzy logic is an excellent heuristic method to translate Experts' knowledge and rules into a computer program (Zhou & Mouftah, 2004). Fuzzy logic reflects how people think. It attempts to model our sense of words, our decision making and our common sense (Negnevitsky, 2002).

Fuzzy systems have the attribute of expressing knowledge in the form of linguistic rules. They offer a possibility to implement expert human knowledge and experience (Godjevac, 1995).

### **1.1.5 Search as a Problem-Solving Process**

Search is a necessity to determine solutions to an enormous range of problems (Coppin, 2004) P.72, and the process of search is fundamental to the problem-solving process, where the problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found (Rich & Knight, 2000).

In order to solve many hard problems efficiently; it is often necessary to compromise the requirements of mobility and systematicity and to construct a control structure that is no longer guaranteed to find the best answer but that will almost always find a very good answer. Thus we introduce the idea of a heuristic (Rich & Knight, 2000).

### 1.1.6 Heuristic Search Definition

The word heuristic comes from the Greek word *heuriskein*, meaning "to discover," which is also the origin of *eureka*, derived from Archimedes' reputed exclamation, *heurika* (for "I have found"), uttered when he had discovered a method for determining the purity of gold (Rich & Knight, 2000).

A *heuristic* is a technique that improves the efficiency of a search process, possibly by sacrificing claims of completeness. Heuristics are like tour guides. They are good to the extent that they point in generally interesting directions; they are bad to the extent that they may miss points of interest to particular individuals. But, on the average, they improve the quality of the paths that are explored. Using good heuristics, we can hope to get good (though possibly no optimal) solutions to hard problems, such as the traveling salesman, in less than exponential time (Rich & Knight, 2000).

The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one path is available: The more accurately the heuristic function estimates the true merits of each node in the search tree (or graph), the more direct the solution process. In the extreme, the heuristic function would be so good that essentially no search would be required. The system would move directly to a solution (Rich & Knight, 2000).

Humans use intelligent search: a chess player considers a number of possible moves, a doctor examines several possible diagnoses, and a computer scientist entertains different designs before beginning to write code. Humans do not use exhaustive search: the chess player examines only moves

that experience has shown to be effective; the doctor does not require tests that are not somehow indicated by the symptoms at hand. Human problem solving seems to be based on judgmental rules that guide search to those portions of the state space that seem most "promising" (Luger, 2005).

### **1.1.7 Types of Artificial Intelligence Searching Techniques**

Search techniques (also called Search strategies, Control strategies, Search procedures, Search methodologies, or Search methods) are used to solve problems in Artificial Intelligence according to the state space representation of a problem (Bigus & Bigus, 2001), where search algorithms must keep track of the paths from a start to a goal node, because these paths contain the series of operations that lead to the problem solution (Luger, 2005).

There are many Search Methodologies in Artificial Intelligence. Search Strategies may be classified mainly as Blind, or Heuristics search (Coppin, 2004).

Blind Search technique (also called Exhaustive, Brute - Force, or Uninformed Search) includes search concepts like: Generate and Test, Depth-First Search, Breadth-First Search, and Depth-First Iterative Deepening (Coppin, 2004).

Heuristics search technique (also called informed, more informed, and more intelligent search) includes search concepts like: Hill climbing, Best-First Search, Beam Search, and Optimal Paths (Branch-&-bound, A\*, and Greedy Search) (Coppin, 2004).

### 1.1.7.1 Blind Artificial Intelligence Searching techniques

An exhaustive search looks at objective function values at every point in the search space, one at a time. They are usually discounted due to lack of efficiency (Clay, Crispin & Crossley, 2000).

The simplest approach to search is called Generate and Test. This simply involves generating each node in the search space and testing it to see if it is a goal node. If it is, the search has succeeded and need not carry on. Otherwise, the procedure moves on to the next node (Coppin, 2004). It is also known as the British Museum algorithm, a reference to a method for finding an object in the British Museum by wandering randomly. Or, as another story goes if sufficient number of monkeys were placed in front of a set of typewriters and left alone long enough, then they would eventually produce all of the works of Shakespeare (Rich & Knight, 2000).

Three basic search strategies that systematically explore a state space are depth first, breadth first, and iterative deepening (Bratko, 1998).

Depth-first search and breadth-first search are the best-known and widest-used search methods (Coppin, 2004).

**Depth-first search (DFS)** is so called because it follows each path to its greatest depth before moving on to the next path. Depth-first search is often used by computers for search problems such as locating files on a disk, or by search engines for spidering the Internet (Coppin, 2004), and in all sorts of programs, ranging from those that do robot path planning to those that provide

natural-language access to database information (Winston, 2000).

**Breadth-first search (BFS)**, in contrast, explores the space in a level-by-level fashion. Only when there are no more states to be explored at a given level does the algorithm move on to the next level (Luger, 2005).

The **Depth-First Iterative Deepening (DFID)** algorithm involves repeatedly carrying out depth-first searches on the tree, starting with a depth-first search limited to a depth of one, then a depth-first search of depth two, and so on, until a goal node is found (Coppin, 2004), so it combines the desirable properties of depth-first and breadth-first search (Bratko, 2001).

#### 1.1.7.2 Heuristic Artificial Intelligence Searching techniques

Heuristic search (HS) is one of the older fields in artificial intelligence. Nilsson & Pearl (Nilsson, 1971; Pearl, 1984) wrote the classic introductions to the field (Schaeffer & Plant, 2000). George Polya defines heuristic as "the study of the methods and rules of discovery and invention" (Polya 1945).

Brute-force search algorithms blindly search the state space, while heuristic search algorithms use feedback or information about the problem to direct the search (Bigus & Bigus, 2001).

The simplest way to implement heuristic search is through a procedure called Hill-climbing (Pearl, 1984). **Hill-climbing (HC)** strategies expand the current state of the search and evaluate its children. The best child is selected for further expansion: neither its siblings nor its parent are retained. Hill-climbing is named for the strategy that might be used by an eager, but blind mountain climber: go uphill

along the steepest possible path until you can go no farther up. Because it keeps no history, the algorithm cannot recover from failures of its strategy (Luger, 2005).

**Steepest ascent hill climbing** is similar to hill climbing, except that rather than moving to the first position you find that is higher than the current position, you always check around you in all four directions and choose the position that is highest. (Coppin, 2004) P.99.

**Simulated annealing** is a variation of hill climbing in which, at the beginning of the process, some downhill moves may be made. The idea is to do enough exploration of the whole space early on so that the final solution is relatively insensitive to the starting state. Simulated annealing (Kirkpatrick *et al*, 1983) as a computational process is patterned after the physical process of annealing, in which physical substances such as metals are melted (i.e., raised to high energy levels) and then gradually cooled until some solid state is reached (Rich & Knight,2000) .

**Best-first search (BFS)** is a systematic control strategy combining the strengths of breadth-first and depth-first search into one algorithm. The main difference between best-first search and the brute-force search techniques is that we make use of an evaluation or heuristic function to order the Search Node objects on the queue. In this way, we choose the Search Node that appears to be best before any others, regardless of their position in the tree or graph (Bigus & Bigus, 2001).



**Beam Search (BS)** is a form of breadth-first search that employs a heuristic, as seen with hill climbing and best-first search. Beam search works using a threshold so that only the best few paths are followed downward at each level. It has the disadvantage of not exhaustively searching the entire tree and so may fail to ever find a goal node (Coppin, 2004) P.105.

### 1.1.7.3 Optimal Artificial Intelligence Searching techniques

Several methods exist that do identify the optimal path through a search tree. The optimal path is the one that has the lowest cost or involves traveling the shortest distance from start to goal node. The techniques described previously may find the optimal path by accident, but none of them are guaranteed to find it. The simplest method for identifying the optimal path is called the **British Museum procedure** (Coppin, 2004).

The following more sophisticated techniques for identifying optimal paths are outlined in this introduction: **Branch and Bound, A\***, and **Greedy search**.

Several techniques can reduce the search complexity. One is called **Branch and Bound (B&B)** (Uniform cost search) (Horowitz and Sahni 1978). Branch and bound generates paths one at a time, keeping track of the best circuit found so far. This value is used as a bound on future candidates. As paths are constructed one node at a time, the algorithm examines each partially completed path. If the algorithm determines that the best possible extension to a path, the branch, will have greater cost than the bound, it eliminates that partial path and all of its possible extensions. This reduces search considerably but still leaves an exponential number of paths (Luger, 2005; Kruse *et al*, 2000).

Branch and Bound strategy is applied in order to find optimal or near optimal solutions for most problems in high-level synthesis (HLS) (Black, 2005).

One of the most famous search algorithms used in AI is the **A\* search algorithm**, which combines the greedy search algorithm for efficiency with the uniform-cost search for optimality and completeness (Bigus & Bigus, 2001).

**Greedy search** is a variation of the A\* algorithm, where  $g(\text{node})$  is set to zero, so that only  $h(\text{node})$  is used to evaluate suitable paths. In this way, the algorithm always selects the path that has the lowest heuristic value or estimated distance (or cost) to the goal. Greedy search is an example of a best-first strategy (Coppin, 2004).

There are many **other** different **Search techniques** used to solve problems in AI like: Exchanging Heuristics, Iterated local Search, Tabu Search, Using Ant Colony Optimization, Using Genetic Algorithms for Search, Real-time A\*, Iterative-Deepening A\*, Agendas, Parallel Search, Bidirectional Search, and Nondeterministic Search (Coppin, 2004; Rich & Knight, 2000).

### **1.1.8 Previous Works of Fuzzy Shortest Path Problem in Networks**

Many literatures about the fuzzy shortest path in networks can be found, e.g., (Okada & Soper, 2000), (Dubois & Prade, 1980). Most of them are related to the fuzzy shortest path problem in networks. Dubois and Prade (Dubois & Prade, 1980) first introduced the fuzzy shortest-path problem in 1980. The major drawback of this fuzzy shortest-path problem is the lack of interpretation, where they did not develop approach to determine the shortest path (Yao & Lin, 2003).

Klein (Klein, 1991) proposed a hybrid multi-criteria algorithm based on fuzzy dynamic programming (DP) that specified each arc length within an integer value from one to a fixed number, nevertheless, the proposed approach did not provide an extension to the crisp counterpart.. In (Okada & Soper, 2000), Okada and Soper proposed a fuzzy algorithm, which was based on multiple labeling methods, to offer non-dominated paths to a decision maker(Lin & Chern, 1994; Hansen, Beckmann & Kunzi 1980). Ma and Chen (Ma & Chen, 2005) proposed the on-line fuzzy shortest path problem when the decision-making is under the condition of without knowing the on-line releasing congestion points, employing the Overall Existence Ranking Index (OERI) to rank the paths.

Yao and Lin (Yao & Lin, 2003) presents two new types of fuzzy shortest-path network problems combining statistics with fuzzy sets and a signed distance ranking. Mares and Horak (Mares & Horak, 2003) proposed that the uncertainty connected with the input data of a network can be described and investigated by means of fuzzy sets and fuzzy quantities theory. Takahashi and Yamakami (Takahashi & Yamakami, 2005) proposed a modified Okada's algorithm (Okada, 2001), using some properties observed by other authors, they also proposed a genetic algorithm technique to seek an approximated solution for large scale problems.

Okada (Okada, 2001) proposed his new algorithm by taking interaction among paths into consideration. The degree of possibility for each arc on a network "ill posed" is obtained by this algorithm. Moazeni (Moazeni, 2005) proposed a different algorithm which takes advantage of the multiple labeling

method and Dijkstra's shortest path algorithm.

Kruse *et al* (Kruse *et al*, 2000) proposed a fast estimation technique, which can be applied in high-level synthesis (HLS) to reduce the power consumption in data path components, so they applied branch and bound strategy to find optimal or near optimal solutions for this problem.

All of the previous mentioned works are about the fuzzy shortest path problem in networks and related to the use of Dijkstra's and Okada's algorithm in networks as a network's searching techniques, also most of them did not develop an exact approach to determine the shortest path.

To the best of our knowledge, no other research has been carried out where fuzzy underestimates is applied with "Branch and Bound" algorithm, or A\* algorithm which are an Artificial Intelligence Search Techniques or an Artificial Intelligence Problem-Solving Techniques.

## **1.2 Statement of the Problem**

The existing Branch & Bound augmented by underestimate and A\* searching techniques are techniques that work well on precise data, but not on imprecise data, whereas data available are not always crisp in real life.

Fuzzy sets in knowledge representation have advantages over the traditional logic, since it is, basically, a theory of graded concepts in which values of variables are a matter of degree. In describing human behavior, we generally use words which are called linguistic values rather than numbers to characterize the value of variables as well as the relations among

them. In general, most fuzzy variables can be characterized by a rating attribute and the governing fuzzy sets which appear in its constituents (Sun & Hadipriono, 1995).

The purpose of this study is to develop new techniques which are able to deal with imprecise real life data type and its hidden uncertainty during a search.

The objective of such work is to deal with the imprecise data involved in different kinds of existing searching techniques in a more efficient way.

### **1.3 Goals of this Dissertation**

Branch and Bound search augmented by underestimate and A\* searching technique are an effective heuristic principle guided Artificial Intelligence Problem-Solving Techniques.

Heuristic principle guides the search process so as to always expand the node that is currently the most promising according to the heuristic estimates.

The aim of this research is to deal with uncertainty or imprecise data in a more efficient way by applying fuzzy logic on both searching techniques, where fuzzy logic is an appropriate tool to deal with such problems (Zadeh, 1965; Klir & Yuan; 1995; Guangwu, 1993; Moore, 1966).

Fuzzy systems are able to treat uncertain and imprecise information, where they have a capability to express knowledge in the form of linguistic rules.

On the average, they improve the quality of the paths that are explored, where, using good fuzzy heuristics, we can hope to get good solutions to hard

problems in less than exponential time, by using Branch and Bound search and A\* searching techniques.

## 1.4 Dissertation Contributions

A search problem and its solution by the existing crisp methods of Branch and Bound and A\* search have been considered in this dissertation. It is observed that these methods can be improved by using fuzzy theory. Consequently a new method of Branch & Bound and A\* searching techniques with fuzzy underestimation to the available Fuzzy information (using Triangular Fuzzy Number model) has been proposed to add Fuzzy Underestimation to the existing Algorithms, thus a new improved version of searching techniques under uncertainty has been suggested to be helpful in many real life problems of computer science, specially in AI field, the corresponding algorithms have been given, and the algorithms have been explained by examples.

The proposed algorithms have been implemented and tested. The analysis and simulation results showed that Fuzzy Underestimated A\* and Fuzzy Underestimated (Branch and Bound) search techniques have achieved better efficiency, time complexity, and effective branching factor than all other compared searching techniques.

## 1.5 Dissertation Structure

This dissertation is organized as follows:

- **Chapter One:** Presents introduction to dissertation subject, artificial intelligence searching techniques, related works, fuzzy logic, expert

- system, problem statement, goals of the dissertation and dissertation contributions.
- **Chapter Two:** Provides Preliminaries for Fuzzy Set Theory, and Artificial Intelligence Search Techniques.
- **Chapter Three:** Contains Introduction for Crisp B & B, and Branch & Bound Search with Fuzzy Underestimation which represents the first dissertation work, with two applications for the proposed algorithm
- **Chapter Four:** Contains Introduction for B & B with dynamic programming, Crisp A\*, and A\* Search with Fuzzy Underestimation which represents the second dissertation work, with two applications for the proposed algorithm
- **Chapter Five:** Deals with the analysis of the research algorithms. It presents simulation for the proposed algorithms.
- **Chapter Six:** Presents conclusions, results discussion, and future work.
- **Appendices:** Three appendices are added: the first appendix contains the simulation psuedocode, the second contains simulation code, and the third appendix presents fuzzy logic information.

# Chapter Two

## Preliminaries

In this chapter we are to give some preliminaries on two areas :

- (i) fuzzy set theory., and
- (ii) Search Techniques.

### 2.1 Fuzzy Set Theory

Fuzzy sets (Atanassov, 1999; Baldwin, 1981; Bratko, 2001; Blue *et al*, 2002; Bustince *et al*, 1995; David, 1989; Dubois & Prade, 1980; 1990; Elaine, 1983; Klir& Yuan, 1995; Guangwu, 1993; Gaines, 1976; Gorzalczany, 1987; Negnevitsky, 2002; Rajagopalan *et al*, 2003; Rich & Knight, 2000; Zadeh, 1965; 1968; 1973; 1975; 1978) are of importance to us in our work in this thesis. Actually without proper understanding of fuzzy set theory, it may not be possible to understand this dissertation.

Fuzzy or multi-valued logic was introduced in the 1930s by Jan Lukasiewicz, a Polish logician and philosopher (Lukasiewicz, 1930). Later, in 1937, Max Black, a philosopher, published a paper called Vagueness: An exercise in logical analysis (Black, 1937). Then in 1965 Lotfi Zadeh, Professor and Head of the Electrical Engineering Department at the University of California at Berkeley, published his famous paper "Fuzzy sets". In fact, Zadeh rediscovered fuzziness, identified and explored it, and promoted and fought for it (Negnevitsky, 2002).

Zadeh (Zadeh, 1965), initiated the notion of fuzzy set theory as a modification of the ordinary set theory, which turned out to be of far reaching



implications. Vague notions can be modeled using this theory. A fuzzy set is a class of objects in which the transition from membership to non-membership is gradual rather than abrupt. Such a class is characterized by a membership function which assigns to an element a grade or degree of membership between 0 and 1. Fuzzy logic is the same as "imprecise logic". This new logic for representing and manipulating fuzzy terms was called fuzzy logic, and Zadeh became the Master of fuzzy logic (Negnevitsky, 2002).

A fuzzy set is a set with fuzzy boundaries, such as short, average or tall for men's height. To represent a fuzzy set in a computer, we express it as a function and then map the elements of the set to their degree of membership. Typical membership functions used in fuzzy expert systems are triangles and trapezoids (Negnevitsky, 2002).

Fuzzy logic is popular. The number of papers dealing with fuzzy logic and its application is immense, and the success in applications is evident. In 1991 there were (1400) papers dealing with fuzzy systems (Rajagopalan, Washington, Rizzoni & Guezennec, 2003).

Fuzzy logic is derived from fuzzy set theory dealing with reasoning that is approximate rather than precisely deduced from classical predicate logic. It can be thought of the application side of fuzzy set theory dealing with well thought out real world expert values for a complex problem (Klir, 1997).

### **2.1.1 Crisp Sets and Fuzzy Sets**

A set can be described either by the list method or by the rule method. We know that the process by which individuals from the universal set  $X$  are

determined to be either members or nonmembers of a set can be defined by a characteristic function or discrimination function.

For a given set  $A$ , this function assigns a value  $\mu_A(x)$  to every  $x \in X$  such that

$$\begin{aligned}\mu_A(x) &= 1 && \text{if } x \in X \\ &= 0 && \text{if } x \notin X\end{aligned}$$

Thus in the classical theory of sets, very precise bounds separate the elements that belong to a certain set from the elements outside the set. In other words, it is quite easy to determine whether an element belongs to a set or not. For example, if we denote the set of signalized intersections in a city by  $A$ , we conclude that every intersection under observation belongs to set  $A$  if it has a signal. Element  $x$ 's membership in set  $A$  is described in the classic set theory by the membership function  $\mu_A(x)$ , as follows:

$$\mu_A(x) = \begin{cases} 1, & \text{if and only if } x \text{ is member of } A \\ 0, & \text{if and only if } x \text{ is not member of } A \end{cases}$$

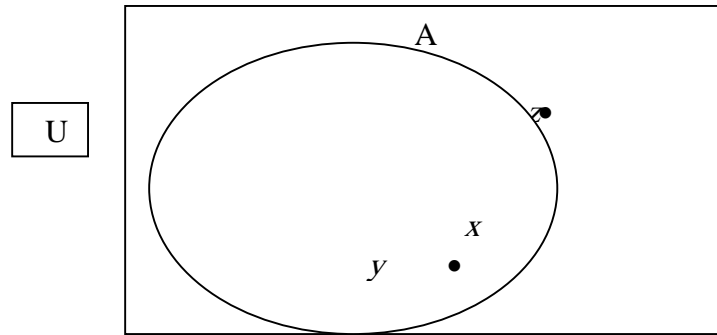


Figure 2.1: Set A and elements x, y and z of U

It is clear from Figure (2.1) that  $\mu_A(x) = 1$ ,  $\mu_A(y) = 1$ , and  $\mu_A(z) = 0$ .

Many sets encountered in reality do not have precisely defined bounds that separate the elements in the set from those outside the set. Thus, it might be said that waiting time of a vehicle at a certain signal is “long”. If we denote by **A** the set of “long waiting time at a signal,” the question logically arises as to the bounds of such a defined set. In other words, we must establish which elements belong to this set. Does a waiting time of 25 seconds belong to this set? What about 15 seconds or 90 seconds?

The air traffic between two cities can be described as having “high flight frequency.” Do flight frequencies of five flights a day, eight flights a day, three flights a day belongs to “high flight frequency” category?

Travel time between origin and destination is usually subjectively estimated as “short,” “not too long,” “long,” “medium,” “about twenty minutes”, “around half an hour”, and so on. Now, does a travel time of 40 minutes, 25 minutes, or 8 minutes belong to the set called “travel time of around half an hour”? We intuitively know that a travel time of 25 minutes belongs to the set called “travel time of around half an hour” “more” or “stronger” than a travel time of 8 minutes. In other words, there is more truth in the statement that travel time of 25 minutes is “travel time of around half an hour” than in the statement that travels time of 8 minutes is “travel time of around half an hour”.

This characteristic function can be generalized such that the values assigned to the elements of the universal set fall within a specified range and indicate the membership grade of these elements in the set in question. Such a function is called membership function and the set defined by it a fuzzy set.

The membership function for fuzzy sets can take any value form the closed interval  $[0,1]$ . Fuzzy set **A** is defined as the set of ordered pairs

$$\mathbf{A} = \{x, \mu_A(x)\}, \quad (1)$$

The following holds for the functional values of the membership function  $\mu_A(x)$

$$\begin{aligned} \mu_A(x) &\geq 0 & \forall x \in X \\ \sup_{x \in X} [\mu_A(x)] &= 1 \end{aligned} \quad (2)$$

Where  $\mu_A(x)$  is the grade of membership of element  $x$  in set **A**. The greater  $\mu_A(x)$ , the greater the truth of the statement that element  $x$  belongs to set **A**.

Let us denote the crisp set by  $X = \{x_1, x_2, \dots, x_n\}$  the finite discrete set of elements  $x_i, i = 1, 2, \dots, n$ . Set  $X$  can also be shown in the form:

$$X = x_1 + x_2 + \dots + x_n = \sum_{i=1}^n x_i, \quad (3)$$

where the sign  $+$  denotes the union of the elements. Set  $X$  is referred to as the “universe of discourse”, and it may contain either discrete or continuous values (elements).

Similarly, fuzzy set  $A$  defined over a set  $X$  is most often shown in the form:

$$A = \frac{\mu_A(x_1)}{x_1} + \frac{\mu_A(x_2)}{x_2} + \dots + \frac{\mu_A(x_n)}{x_n} = \sum_{i=1}^n \frac{\mu_A(x_i)}{x_i} \quad (4)$$

A fuzzy set  $A$  can also be defined over a set  $X$  in any of the following forms :-

(i)  $A = \{ \mu_A(x_1)/x_1, \mu_A(x_2)/x_2, \mu_A(x_3)/x_3, \dots, \mu_A(x_n)/x_n \}.$  (5)

(ii)  $A = \{ (x_1, \mu_A(x_1)), (x_2, \mu_A(x_2)), (x_3, \mu_A(x_3)), \dots, (x_n, \mu_A(x_n)) \}.$  (6)

(iii)  $A = \{ (x, \mu_A(x)) : x \in X \}.$  (7)

(iv) or, sometimes just by the membership function  $\mu_A : X \rightarrow [0,1].$  (8)

When  $X$  is continuous and not a finite set, fuzzy set  $A$  defined over set  $X$  is expressed as:

$$A = \int_X \frac{\mu_A(x)}{x}, \quad (9)$$

where the integration sign represents the union of the elements.

**Illustrative example 2-1:**

Let us consider a set  $X = \{2, 5, 9, 18, 21, 25\}$ , whose elements denote the number of vehicles waiting in line at a traffic signal.

Set **B** consists of the fuzzy set “small number of vehicles in line.” Fuzzy set **B** can be shown as:

$$B = \frac{0.95}{2} + \frac{0.55}{5} + \frac{0.20}{9} + \frac{0.10}{18} + \frac{0.05}{21} + \frac{0.01}{25} \quad (10),$$

or in other form like:

$$B = \{ (2,0.95), (5,0.55), (9,0.20), (18,0.10), (21,.05), (25,.01) \}. \quad (11)$$

The grades of membership 0.95, 0.55, ..., 0.01 are subjectively determined and indicate the “strength” of membership of individual elements in fuzzy set B. For example, 2 belongs to fuzzy set B with a grade of membership of 0.95, which comprises a “small number of vehicles in line” at the signal.

## 2.2. Illustrative example

Let us consider a fuzzy set A, which is defined as “travel time is approximately 30 minutes.” Membership function  $\mu_A(t)$ , which can subjectively be determined as shown below in figure (2.2):-

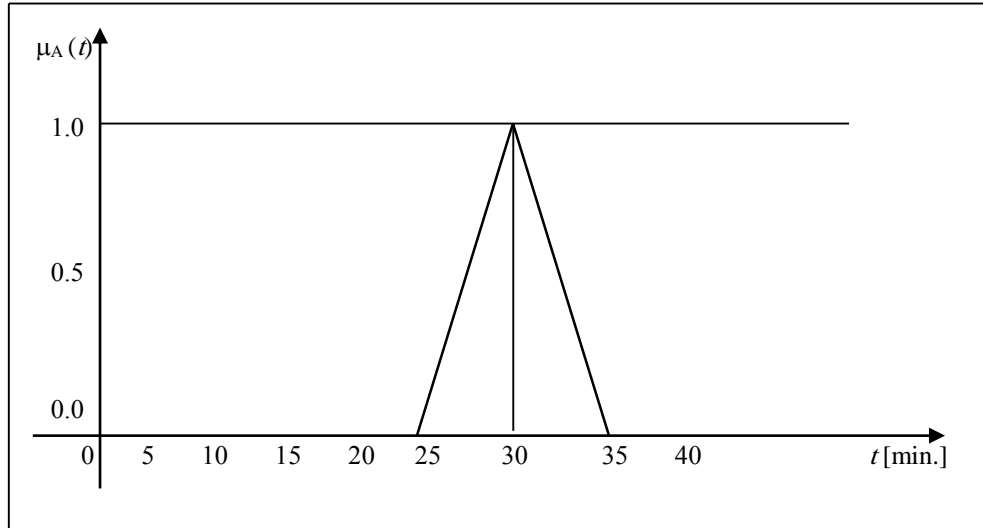


Figure 2.2: Membership function  $\mu_A(t)$  of fuzzy set A.

In this case, we have subjectively estimated that travel time between the two points can be within the limits of 25 to 35 minutes. A travel time of 30 minutes has a grade of membership of 1 and belongs to the set “travel time is approximately 30 minutes.” All travel times within the interval of 25 to 35 minutes are also members of this set because their grades of membership are greater than zero. Travel times outside this interval have grades of membership equal to zero.

### 2.1.2 Various Operations on Fuzzy Sets

Fuzzy sets can interact. These relations are called operations. The main operations of fuzzy sets are: complement, containment, intersection, and union (Negnevitsky, 2002).

In this section we recollect some basic operations on fuzzy sets. Let A and B be two fuzzy sets of X having membership functions  $\mu_A$  and  $\mu_B$  respectively.

### 2.1.2.1 Equality of two Fuzzy Sets

Two fuzzy sets A and B are equal, and denoted as  $A = B$ , if and only if

$$\mu_A(x) = \mu_B(x) \quad \text{for all elements of set X.}$$

If at least one  $x \in X$  such that  $\mu_A(x) \neq \mu_B(x)$ , then A and B are said to be 'not equal' and it is denoted as  $A \neq B$ .

### 2.1.2.2 Subsets of Fuzzy Sets

Fuzzy set A is a subset of the fuzzy set B, and denoted as  $A \subset B$ , if and only if:

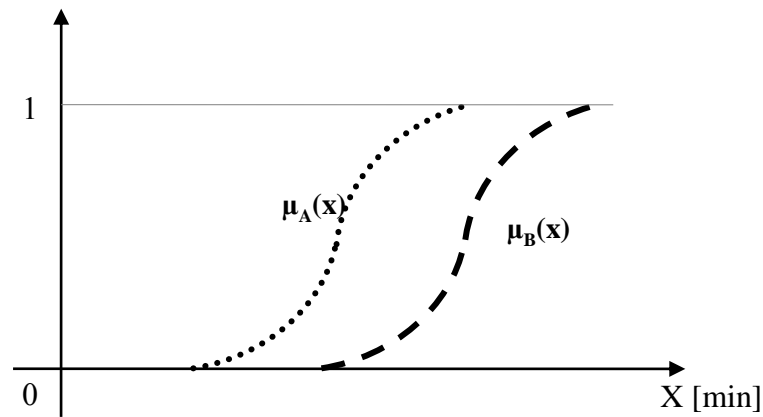
$$\mu_A(x) \leq \mu_B(x) \quad \text{for all elements x of the set X.}$$

In other words,  $A \subset B$  if for every x of X, the grade of membership in fuzzy set A is less than or equal to the grade of membership in fuzzy set B.

Consider the fuzzy sets A and B, respectively, the fuzzy sets "long travel time" and "very long travel time". The fuzzy set "very long travel time" is a subset of the fuzzy set "long travel time" since the following relation is obviously to be satisfied for every x:

$$\mu_B(x) \leq \mu_A(x)$$





**Figure 2.3:** Membership functions of the fuzzy sets  $A = \text{“long travel time”}$  and  $B = \text{“very long travel time”}$ .

**Example 2-3:**

Let  $X = \{a, b, c, d\}$  be a universe of which  $A = \{a/.9, b/.2, c/.1, d/.5\}$  and

$B = \{a/.6, b/.1, c/0, d/.1\}$  are two fuzzy sets.

Then, clearly  $B \subseteq A$ .

### 2.1.2.3 Complementation

Let  $A$  be a fuzzy set of the universe  $X$ . Then the complement of  $A$  is a fuzzy set denoted by  $A^c$  of the same universe  $X$  with the membership function  $\mu_{A^c}$  given by

$$\mu_{A^c}(x) = 1 - \mu_A(x), \quad \forall x \in X.$$

Clearly, we have  $(A^c)^c = A$ .

**Example 2-4:**

Let  $X = \{a, b, c, d\}$  be a universe, and  $A = \{a/.2, b/.6, c/.3, d/.3\}$  is a fuzzy set of  $X$ .

Then the complement of this fuzzy set A is the fuzzy set  $A^c$  in X given by :

$$A^c = \{a/.8, b/.4, c/.7, d/.7\}.$$

Clearly  $(A^c)^c = \{a/.2, b/.6, c/.3, d/.3\}$  which is the fuzzy set A.

### 2.1.2.4 The Union of Fuzzy Sets

Let A and B be two fuzzy sets of X having membership functions  $\mu_A$  and  $\mu_B$  respectively. The union of A and B, denoted by  $A \cup B$ , is the fuzzy set in X defined as the smallest fuzzy set that containing both fuzzy set A and fuzzy set B.

The membership function  $\mu_{A \cup B}$  of the union  $A \cup B$  of fuzzy sets A and B is defined as follows :

$$\mu_{A \cup B}(x) = \max \{ \mu_A(x), \mu_B(x) \} \quad \text{for every } x \text{ of } X.$$

The symbol  $\vee$  is often used instead of the symbol 'max'. The union corresponds to the operation "or". Thus we can write  $\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_B(x)$ .

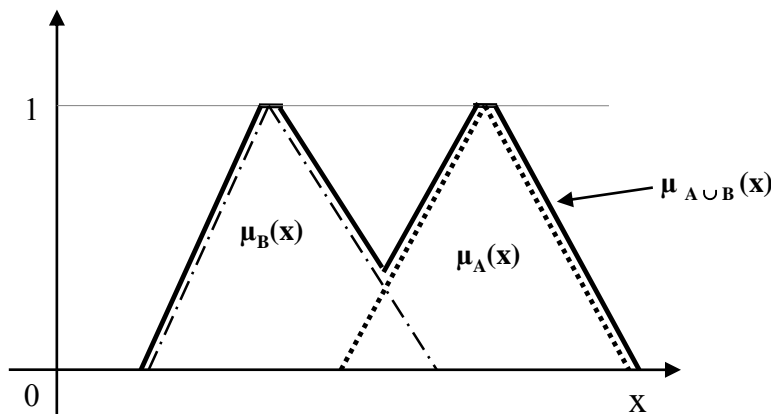
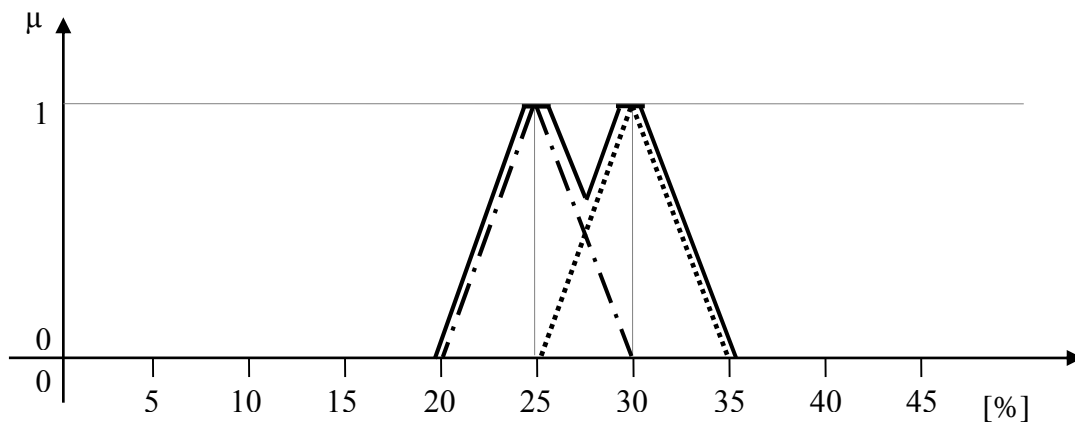


Figure 2.4: Membership functions of fuzzy sets A, B and  $A \cup B$ .

.Example

Delays in air transportation can be due to technical reasons, meteorological conditions, late or non appearance of flight crews, and so on. We assume that out of total aircraft delays, technical reasons make “approximately 30%” of the reasons for delays. We also assume that meteorological conditions cause delays in “approximately 25%” of the cases.

Now we will define fuzzy sets A and B as follows:- A defines “approximately 30% of the cause of delay”, and B defines “approximately 25% of the cause of delay”. The fuzzy set  $A \cup B$  denotes “approximately 25% or approximately 30% of the cause of delay and refers to aircraft delays due to technical reasons or meteorological conditions. The membership functions of the fuzzy set  $A \cup B$  is shown below:-



**Figure 2.5:** Membership functions of the fuzzy sets “approximately 25%”, “approximately 30%” and “approximately 25% or approximately 30%” of the cause of delay.

### 2.1.2.5 The Intersection of Fuzzy Sets

The intersection of fuzzy sets A and B is denoted by  $A \cap B$  and is defined as the largest fuzzy set of X contained in both fuzzy sets A and B. The intersection

corresponds to the operation “and”.

Membership function  $\mu_{A \cap B}(x)$  of the intersection  $A \cap B$  is defined as follows:

$$\mu_{A \cap B}(x) = \min \{ \mu_A(x), \mu_B(x) \} \quad \text{for every } x \text{ of } X.$$

The symbol  $\wedge$  is often used instead of the symbol ‘min’. Thus we can write:

$$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x).$$

**Example 2-6:**

Suppose that the visibility on airport runways and the height of the cloud base are measured in metres. Visibility can be “good,” “medium,” or “poor.” The cloud base can be “low” or “high”.

Consider the fuzzy sets A and B defined as follows: A defines “poor visibility on airport runways,” and B defines “high cloud base”. Then the fuzzy set  $A \cap B$  denotes “poor visibility on airport runways and high cloud base”.

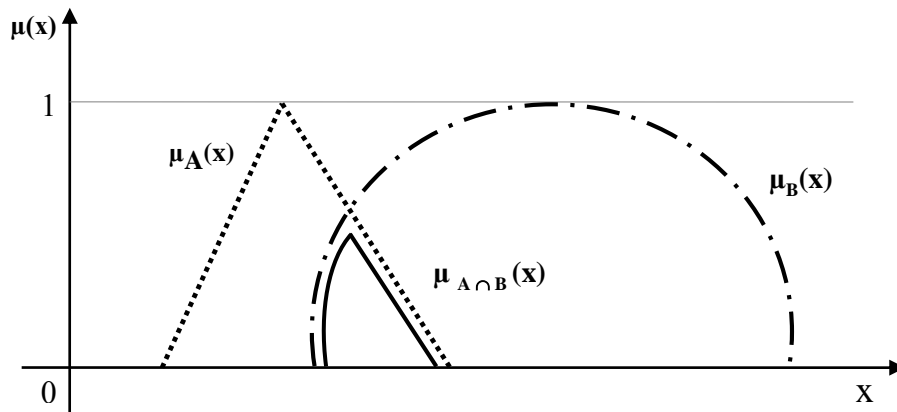


Figure 2.6 Membership functions of fuzzy sets A, B and  $A \cap B$ .

### 2.1.3 Fuzzy Numbers

The concept of fuzzy numbers have been very successfully used in many areas like fuzzy linear programming, fuzzy optimization problems, fuzzy databases, fuzzy algebras etc. to list a few only.

To study the fuzzy numbers, we need to have a prior knowledge of convex fuzzy sets and normalized fuzzy sets.

#### 2.1.3.1 Convex Fuzzy set

A fuzzy set  $A$  of the universe  $X$  is called to be convex if

$$\mu_A(\lambda x_1 + (1 - \lambda)x_2) \geq \min(\mu_A(x_1), \mu_A(x_2)),$$

where  $x_1, x_2 \in X, \lambda \in [0,1]$ .

#### 2.1.3.2 Normalized Fuzzy set

Let  $A$  be a fuzzy set of the universe  $X$ . If  $\sup \mu_A(x) = 1$ , then the fuzzy set  $A$  is called to be a normal fuzzy set.

#### 2.1.3.3 Fuzzy Number

A fuzzy number  $M$  is a convex normalized fuzzy set  $M$  of the real line  $R$  whose membership function is at least segmentally continuous and has the functional value  $\mu_M(x) = 1$  at precisely one element; such that:

- (i) It exists exactly one  $x_0 \in R$  with  $\mu_M(x_0) = 1$   
( $x_0$  is called the mean value of  $M$ ).
- (ii)  $\mu_M(x)$  is piecewise continuous.

There are two types of fuzzy numbers popularly used in different domains of applications. These are :-

- (i) Triangular fuzzy number (TFN).
- (ii) Trapezoidal fuzzy number.

An example of a triangular fuzzy number “approximately 30” can be shown in figure (2.2).

For computational efficiency, trapezoidal membership functions are often used. Figure (2.7) shows such a fuzzy set, which is called ‘approximately 5’ and which would normally be defined as quadruple {3, 4, 6, 7}. It is actually a fuzzy interval.

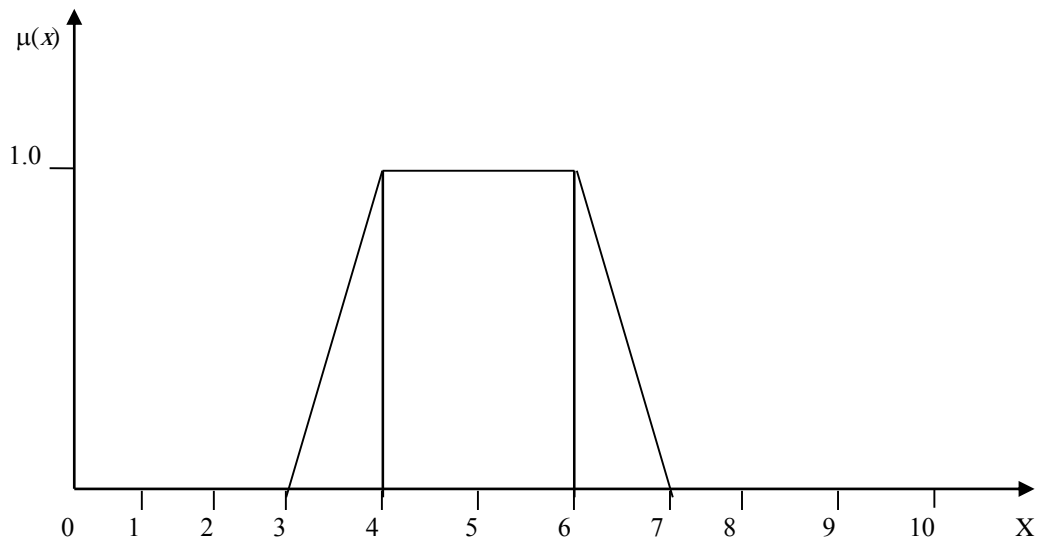


Figure 2.7: Trapezoidal fuzzy number {3, 4, 6, 7}

### 2.1.3.4 Positive Fuzzy Number

A fuzzy number M is called positive (negative) if its membership function is such that  $\mu_M(x) = 0, \quad \forall x < 0 \quad (\forall x > 0).$

**Example 2-7:**

The following fuzzy sets are fuzzy numbers:

$$(1) \text{ 'Approximately 5' } = \{ (3,.2), (4,.6) (5, 1), (6,.7), (7,.1) \}$$

$$(2) \text{ 'Approximately 12' } = \{ (10,.3), (11,.8), (12,1), (13,.4), (14,.2) \}$$

Clearly,  $\{(6,.4), (7, 1), (8,1), (9,.2)\}$ , is not a fuzzy number because;  $\mu(7)$  and  $\mu(8)$  both are equal to 1, and thus it is not a convex normalized fuzzy set.

The algebraic operations with crisp numbers are very common in practice. In order to use fuzzy sets in applications, we will have to deal with fuzzy numbers and the extension principle is one way to extend algebraic operations from crisp to fuzzy numbers.

**2.1.3.5 Definition**

A fuzzy number  $M$  is called to be of LR - type if  $\exists$  reference functions  $L$  (for left),  $R$  (for right), and scalars  $\alpha > 0$ ,  $\beta > 0$  with

$$\mu_M(x) = \begin{cases} L\left(\frac{m-x}{\alpha}\right) & \text{for } x \leq m. \\ R\left(\frac{x-m}{\beta}\right) & \text{for } x \geq m. \end{cases}$$

where 'm' is called the mean value of M. Here 'm' is a real number and  $\alpha$ ,  $\beta$  are known as the left and right spreads respectively.

We represent  $M$  as  $(m, \alpha, \beta)_{LR}$ , where Fig 2.18 shows a LR-type fuzzy number.

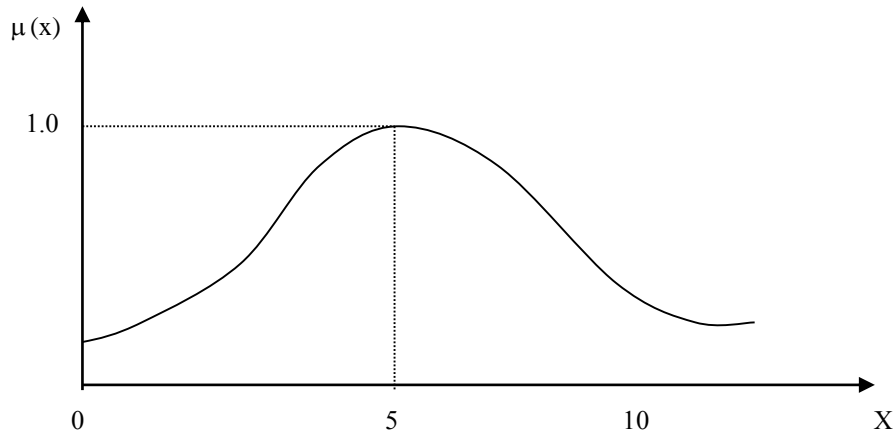


Fig 2.8: LR-type fuzzy number.

### 2.1.3.6 $\alpha$ -cut of a vague set.

The  $\alpha$ -cut of the Fuzzy set A is a crisp subset denoted by the symbol  $A_\alpha$  of the set X, where:-

$$A_\alpha = \{ x : x \in X, \mu(x) \geq \alpha \}.$$

#### **Example 2-8:**

Suppose that the Fuzzy set  $A = \{ (x_1, 0.6), (x_2, 0.3), (x_3, 0.8), (x_4, 0.2) \}$

If the decision maker provided,  $\alpha = 0.4 =$  threshold value,

Then  $A_\alpha = \{ x_1, x_3 \}$ .

But if the decision maker provided,  $\alpha = 0.25$

Then  $A_\alpha = \{ x_1, x_2, x_3 \}$ .

### 2.1.4 Hedge Definition

A qualifier of a fuzzy set used to modify its shape. Hedges include adverbs such as 'very', 'somewhat', 'quite', 'more or less' and 'slightly'. They perform mathematical operations of concentration by reducing the degree of membership of fuzzy elements (e.g. very tall men), dilation by increasing the degree of



membership (e.g. more or less tall men) and intensification by increasing the degree of membership above 0.5 and decreasing those below 0.5 (e.g. indeed tall men).

### **2.1.5 Fuzzy rule Definition**

A conditional statement in the form: IF x is A THEN y is B, where x and y are linguistic variables, and A and B are linguistic values determined by fuzzy sets.

### **2.1.6 Fuzzy inference Definition**

The process of reasoning based on fuzzy logic. Fuzzy inference includes four steps: fuzzification of the input variables, rule evaluation, aggregation of the rule outputs, and defuzzification.

#### **2.1.6.1 Fuzzification Definition**

The first step in fuzzy inference; the process of mapping crisp (numerical) inputs into degrees to which these inputs belong to the respective fuzzy sets.

#### **2.1.6.2 Defuzzification Definition**

The last step in fuzzy inference; the process of converting a combined output of fuzzy rules into a crisp (numerical) value. The input for the defuzzification process is the aggregate set and the output is a single number.

## ٢,٢ Search Techniques

The process of search is fundamental to the problem-solving process in Artificial Intelligence, where the problem can then be solved by using the rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found (Rich & Knight,2000).

Search techniques are used to solve problems in Artificial Intelligence according to the state space representation of a problem (Bigus & Bigus, 2001) , where Search algorithms must keep track of the paths from a start to a goal node, because these paths contain the series of operations that lead to the problem solution (Luger, 2005).

State space is a formalism for representing problems. State space is a directed graph whose nodes correspond to problem situations and arcs to possible moves. A particular problem is defined by a start node and a goal condition. A solution of the problem then corresponds to a path in the graph. Thus problem solving is reduced to searching for a path in a graph (Bratko, 1998).

There are two main approaches to searching a search tree, which roughly correspond to the top-down and bottom-up approaches. **Data-driven search** starts from an initial state and uses actions that are allowed to move forward until a goal is reached. This approach is also known as **forward chaining**. Alternatively, search can start at the goal and work back toward a start state, by

seeing what moves could have led to the goal state. This is **goal-driven search**, also known as **backward chaining** (Coppin, 2004).

Very briefly, if the solution found is to be applied or used on a regular basis then it is important that this solution be as efficient as possible, even if it means sacrificing search time. In other words solution cost should be minimal at the expense of a high search cost. On the other hand, if the problem is a one-off then optimizing the solution cost may no longer be a priority, instead optimizing the search cost may be the number one concern.

Search Strategies may be classified mainly as Blind, Heuristics, and Optimal Paths search (Coppin, 2004).

Blind or Exhaustive Search techniques include search concepts like: Generate and Test, Depth-First Search, Breadth-First Search, and Depth-First Iterative Deepening

Heuristics or informed search techniques include search concepts like: Hill climbing, Best-First Search , Beam Search, and Optimal Paths search techniques include search concepts like: Branch-and-bound, Discrete Dynamic Programming, and A\*.

### **Assumptions**

We will consider the following two assumptions in our work:

- 1- For our work, when we refer to "search," we are talking about data-driven search, where there are two directions in which to search: search forward through the state space, or backward from the goal (Doyle, Dec 5.,2005), but we are to use Directed Acyclic Graph (DAG) because it is impossible to "go back up" the structure once a state has been

reached (Coppin, 2004; Luger, 2005).

2- We need to be careful to remove repetitions of paths, or loops, because those should add redundancy to the graph and make searching it inefficient, where general graph search algorithms must detect and eliminate loops from potential solution paths, whereas tree searches may gain efficiency by eliminating this test and its overhead (Coppin, 2004; Luger, 2005).

In this section we are to explain how to find paths through nets, thus solving search problems. In particular, we will explain in details Depth-First Search, and Breadth-First Search (which are the best-known and widest-used search methods) (Coppin, 2004), with their algorithms and examples involving map traversal. We will also explain briefly Hill Clamping Search, and British Museum Procedure. Finally, we will explain several important properties that search methods should have in order to be most useful.

### **2-2-1 Blind Methods**

In order to explain many techniques, one can look at the problem of route planning, in particular planning a route from start node (S) to goal node (G) in the following map.

We begin with an example;

### Example 2.1:

Suppose that Mr. X is trying to find some path from one city to another city using a highway map such as the one shown in Figure (2.9). The starting point in the city, which might be called (start node), and ending point in the city, which might be called (goal node).

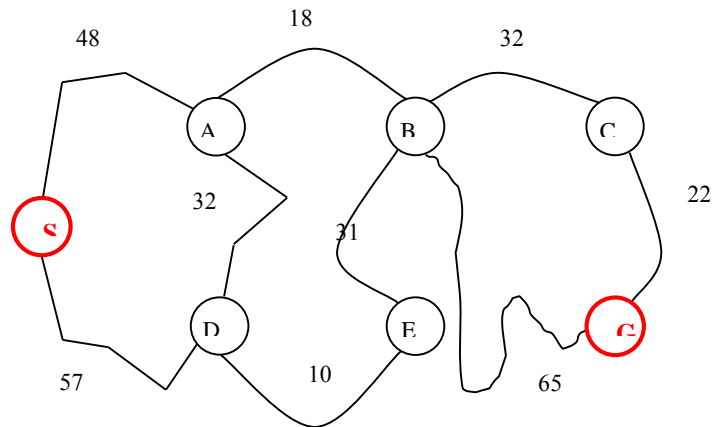


Fig 2.9: A basic search problem, where a path is to be found from the start node, S, to the goal node, G .

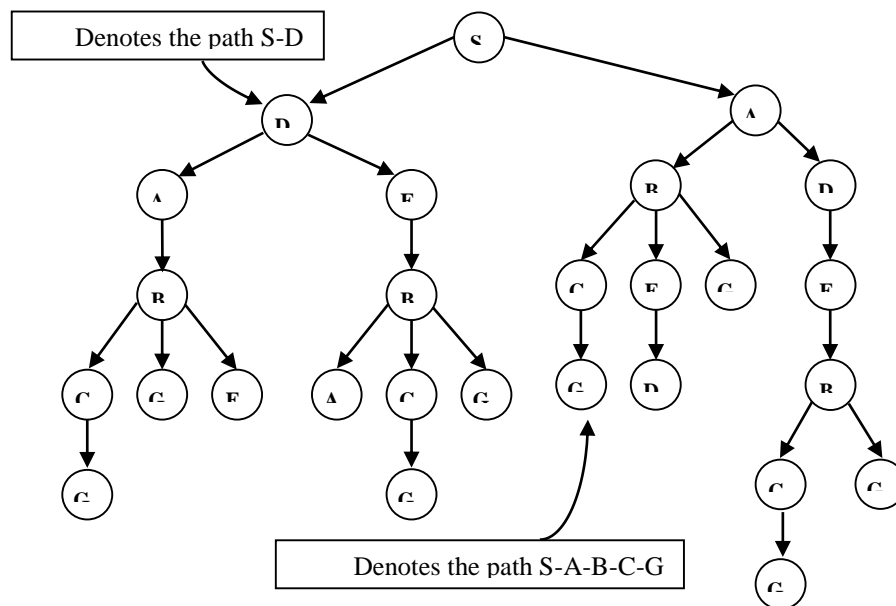
If Mr. X needs to go to the goal city often, then finding a good path is worth a lot of search time. On the other hand, if Mr. X needs to make the trip only once, and if it is hard to find any path then he may be content as soon as he finds any path, even though he could find a better path with more work.

The most obvious way to find a solution is to look at all possible paths. Of course, one would discard paths that revisit any particular city, so that he or she cannot get stuck in a loop– such as S-A-D-S-A-D-S-A-D-...

With looping paths eliminated, one can arrange all possible paths from the

start node S in a search tree (a special kind of semantic-tree in which each node denotes a path).

Figure (2.10) shows a search tree that consists of nodes denoting the possible paths that lead outward from the start node S of the net shown in figure (2.9).



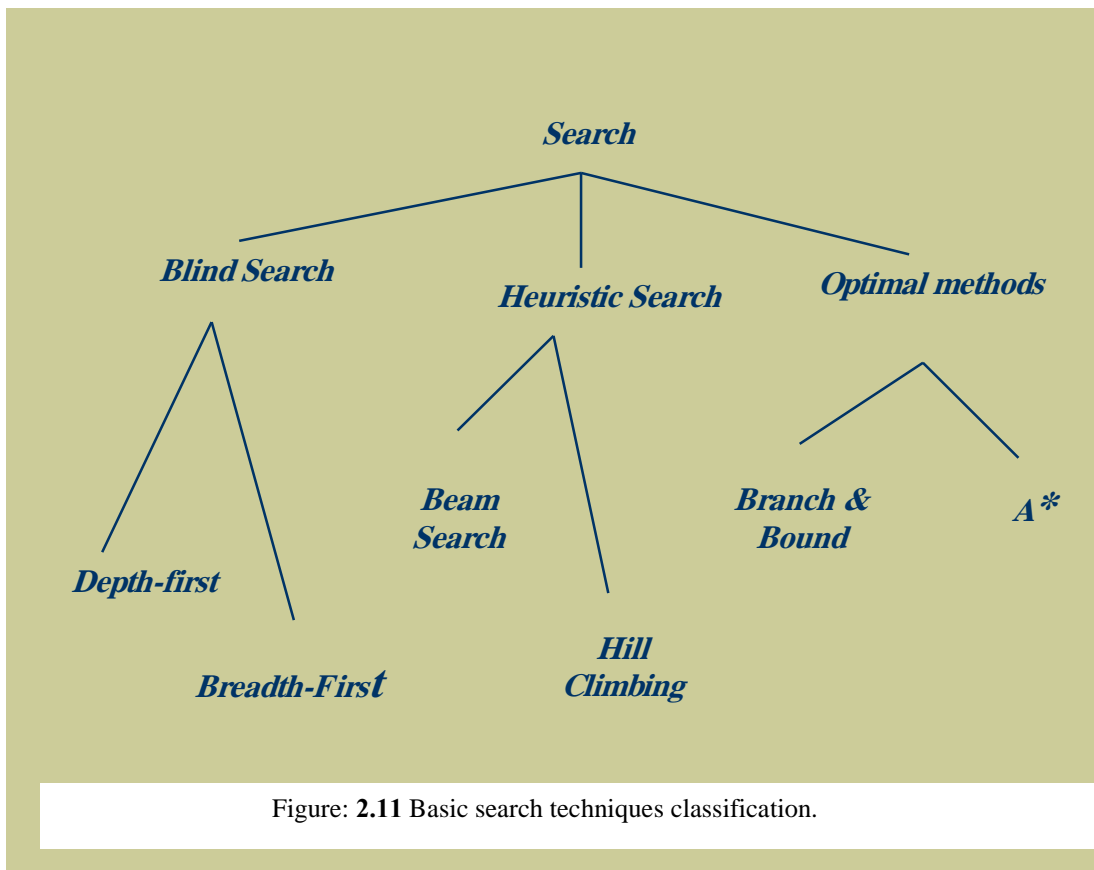
**Figure: 2.10:** A search tree made from a net. Each node denotes a path. Each child node denotes a path that is a one-step extension of the path denoted by its parent. Nets can be converted into search trees by tracing out all possible paths until searcher cannot extend any of them without creating a loop.

There are two important costs to consider with respect to search-based problem solving:

1. Search Cost – The cost of finding a solution (computation cost when **finding** a path)
2. Solution Costs – The cost of using this solution (travel cost expended when **traversing** the path).<sup>1</sup>

<sup>1</sup> This cost might be a representation of the miles necessary in car travel or cost of an air flight between the two cities

The basic idea in nearly all of the search techniques is to maintain and extend a set of partial solution sequences, different search techniques offer different guarantees with regards to both of these costs, Figure (2.11) shows the basic search techniques classification.



In general the longer one spends searching, the better resulting solution; that is, high search costs usually mean low solution costs, depending on the type of problem and the way in which the solution will be used. Search may or may not be a good idea, and more importantly one particular type of search may be preferred over another.

Very briefly, if the solution found is to be applied or used on a regular basis then it is important that this solution be as efficient as possible, even if it means higher search time. In other words solution cost should be minimal at the expense of a high search cost.

On the other hand, if the problem is a one-off then optimizing the solution cost may no longer be a priority, instead optimizing the search cost may be the number one concern.

Note that, although each node in a search tree denotes a path, there is no room in the diagram to write out each path at each node. Accordingly, each node is labeled with only the terminal node of the path it denotes. Each (**child**) denotes a path that is a one-city extension of the path denoted by its (**parent**).

The node with no parent is called the (**root node**). The nodes at the bottom, the ones with no children, are called (**leaf nodes**). One node is the (**ancestor**) of another, a (**descendant**), if there is a chain of one or more branches from the ancestor to the descendant.

If a node has  $b$  children, it is said to have a (**branching factor**) of  $b$ . If the number of children is always  $b$  for every nonleaf node, then the tree is said to have a branching factor of  $b$ .

In Figure(2.10), the root node denotes the path that begins and ends at the start node S. The child of the root node labeled A denotes the path S-A. Each path, such as S-A, that does not reach the goal is called a (**partial path**). Each path that does reach the goal is called a (**complete path**), and the corresponding node is called a (**goal node**). Determining the children of a node is called (**expanding**) the node. Nodes are said to be (**open**) until they are



expanded, whereupon they become (**closed**).

Note that search procedures start out with no knowledge of the ultimate size or shape of the complete search tree. All they know is where to start and what the goal is. Each must expand open nodes, starting with the root node until it discovers a node that corresponds to an acceptable path.

The total number of paths in a tree with branching factor  $b$  and depth  $d$  is  $b^d$ . Thus, the number of paths is said to explode exponentially as the depth of the search tree increases.

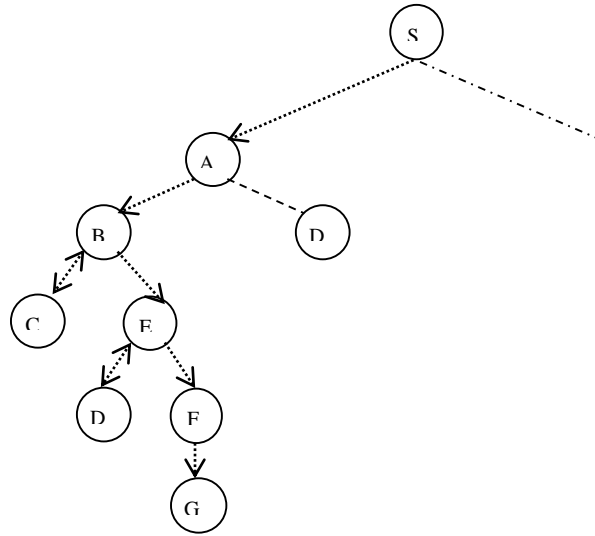
Accordingly, searcher always tries to deploy a search method that is likely to develop the smallest number of paths.

### 2.2.1.1 Depth-First Search (DFS)

Given that one path is as good as any other, one simple way to find a path is to pick one of the children at every node visited, and to work forward from that child. Other alternatives at the same level are ignored completely, as long as there is hope of reaching the goal using the original choice. This strategy is the essence of **depth-first search**.

Using a convention that the alternatives are tried in left-to-right order, the first thing to do is to dash headlong to the bottom of the tree along the leftmost branches, as shown in Figure (2.12).

\But because a headlong dash leads to leaf node C, without encountering C, the next step is to back up to the nearest ancestor node that has an unexplored alternative. The nearest such node is B. The remaining alternative at B is better, bringing eventual success through E in spite of another dead end at D. Figure (2.12) shows the nodes encountered.



**Figure 2.12:** An example of depth-first search.

If the path through E had not worked, then the procedure would move still farther back up the tree, seeking another viable decision point from which to move forward. On reaching A, the procedure would go down again, reaching the goal through D.

Having learned about depth-first search by way of an example, you can see that the procedure, is as follows:

---

To conduct a depth-first search,

- 1- Form a one-element queue consisting of a zero-length path that contains only the root node.
- 2- Until the first path in the queue terminates at the goal node or the queue is empty,
  - 2.1- Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
  - 2.2- Reject all new paths with loops.

2.3- Add the new paths, if any, to the *front* of the queue.

3- If the goal node is found, announce success; otherwise, announce failure.

Figure (2.13) will explain the algorithm of depth-first search according to the previous example shown in Figure (2.12):

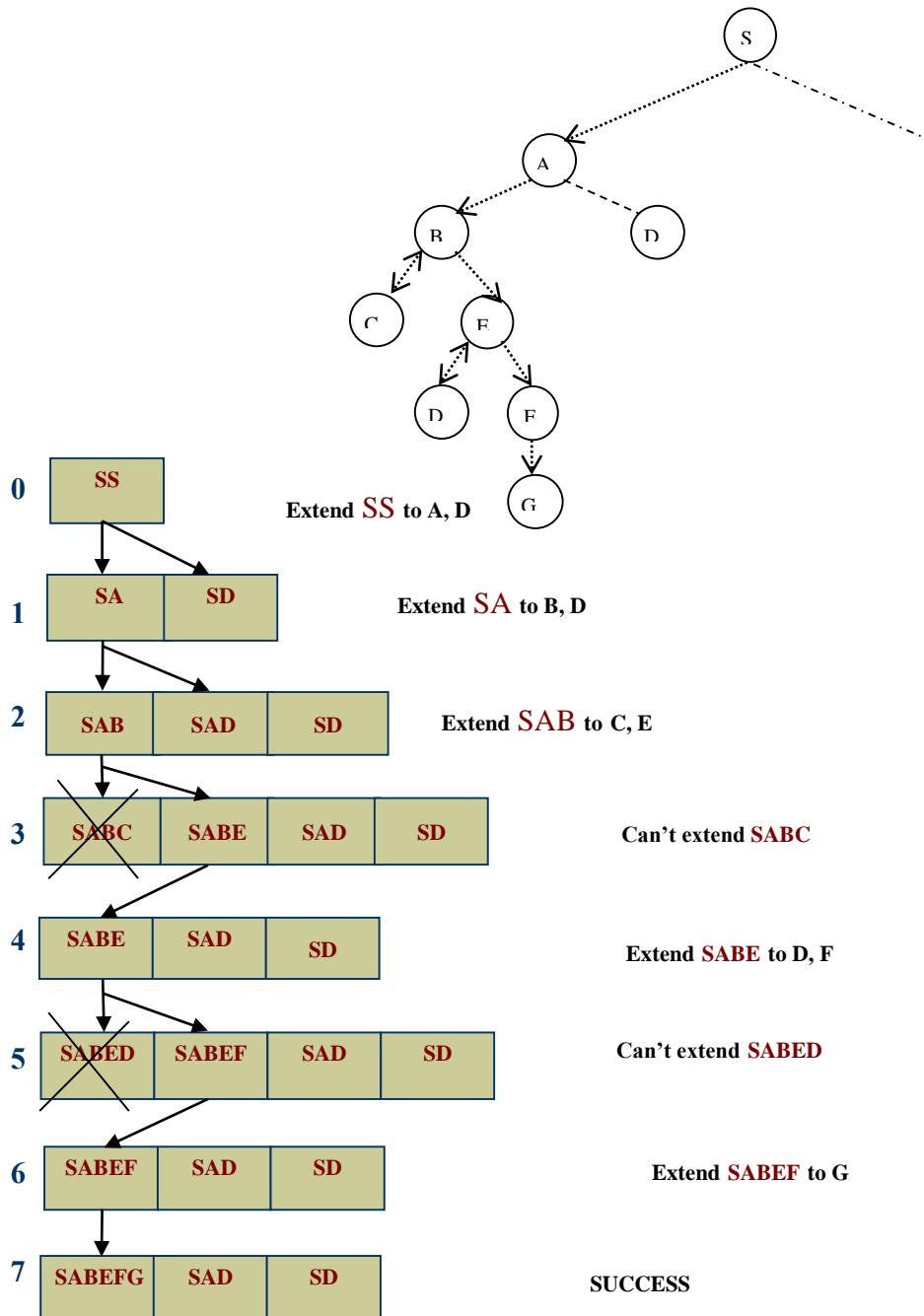


Figure: 2.13 Depth-First Search algorithm explanation

Depth-first search is usually simpler to implement than breadth-first search, and it usually requires less memory usage because it only needs to store information about the path it is currently exploring, whereas breadth-first search needs to store information about all paths that reach the current depth. This is one of the main reasons that depth-first search is often used by computers for search problems such as locating files on a disk, or by search engines for spidering the Internet (Coppin, 2004).

### 2.2.1.2 Breadth -First Search (BFS)

Breadth-first search checks all paths of a given length before moving on to any longer paths, where downward motion proceeds level by level, until the goal is reached. In Figure (2.14), breadth-first search discovers a complete path to node G on the third level down from the root level.

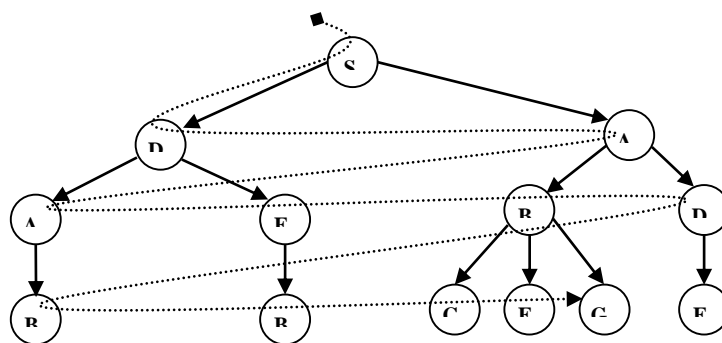


Figure 2.14: An example of breadth-first search. Downward motion proceeds level by level, until the goal is reached.

A procedure for breadth-first search resembles the one for depth-first search, differing only in where new elements are added to the queue.

---

To conduct a breadth-first search,

- 1- Form a one-element queue consisting of a zero-length path that contains only the root node.
  - 2- Until the first path in the queue terminates at the goal node or the queue is empty,
    - 2.1- Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
    - 2.2- Reject all new paths with loops.
    - 2.3- Add the new paths, if any, to the **back** of the queue.
  - 3- If the goal node is found, announce success; otherwise, announce failure.
- 

Figure (2.15) will explain the algorithm of breadth -first search according to the previous example shown in Figure (2.14):

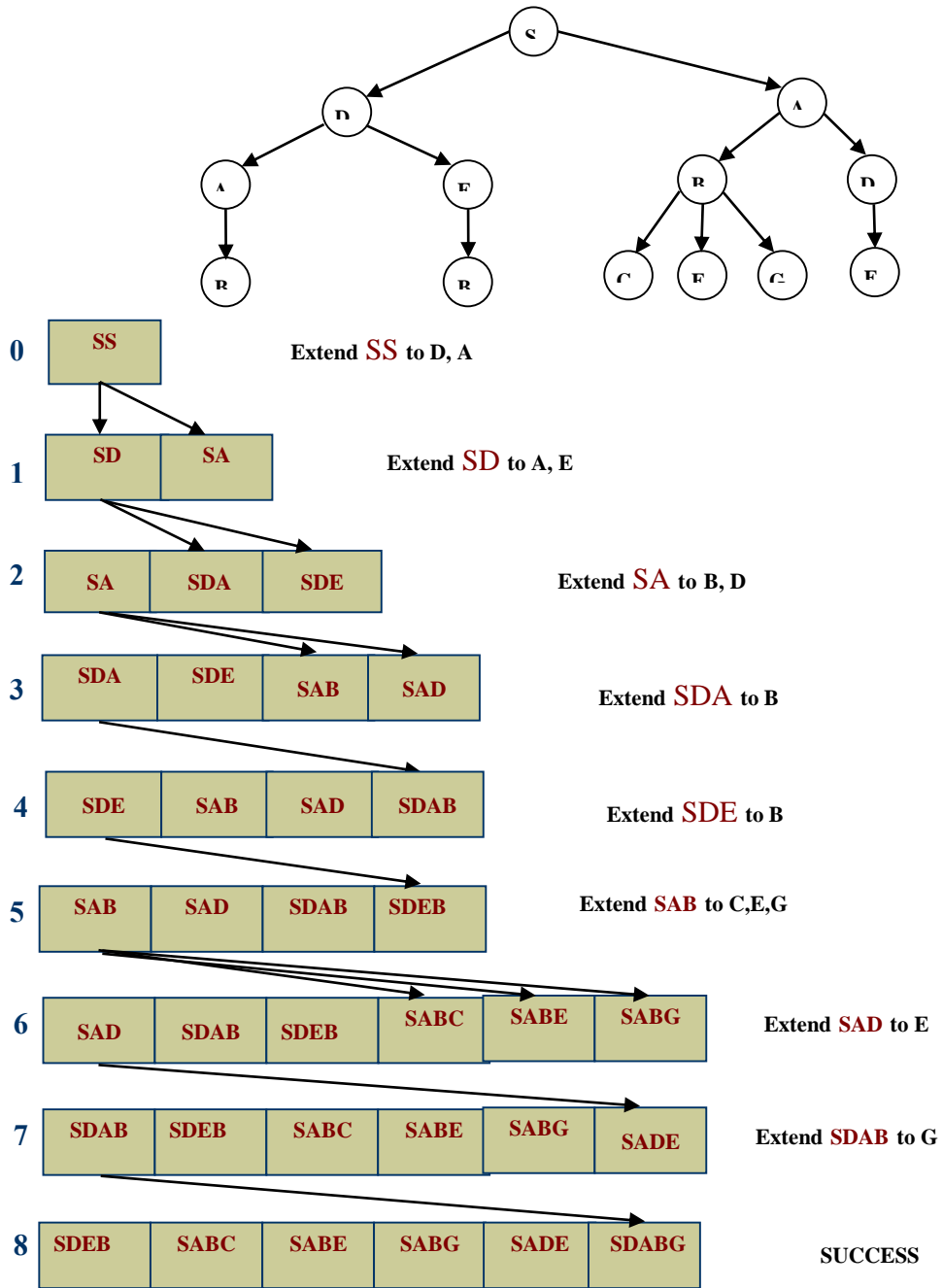


Figure 2.15: Breadth First Search algorithm explanation

The right choice for the appropriate searching technique depends on the tree, where depth-first search is a good method when one is confident that all partial paths either reach dead ends or become complete paths after a reasonable number of steps. In contrast, depth-first search is a bad method if there are long paths, even infinitely long paths, that neither reach dead ends nor become complete paths. In those situations, we need alternative search methods.

Breadth-first search works even in trees that is infinitely deep or effectively infinitely deep. On the other hand, breadth-first search is a wasteful method when all paths lead to the goal node at more or less the same depth.

Note that breath-first search is a bad method if the branching factor is large or infinite, because of exponential explosion. Breadth-first search is a good method when one is confident that the branching factor is small. One may also choose breadth-first search, instead of depth-first search, if the researcher is worried that there may be long paths, even infinitely long paths, that neither reach dead ends nor become complete paths.

In most cases the researcher is uninformed about the search problem, in such cases researcher cannot rule out either a large branching factor or long useless paths. In such situations, the researcher may want to seek a middle ground between depth-first search and breadth-first search. One way to seek such a middle ground is to choose nondeterministic search. When nondeterministic search is being used, one can expand an open node that is chosen at random. In this way, one ensures that the search algorithm will not get

stuck chasing either too many branches or too many levels.

### 2.2.2 . Heuristically Informed Methods

Search efficiency may improve spectacularly if there is a way to order the choices so that the most promising are explored earliest. In many situations, one can make measurements to determine a reasonable ordering. When taking the advantage of such measurements; those methods are called heuristically informed methods.

Heuristic search is one of the older fields in artificial intelligence. Nilsson & Pearl (Hart *et al.*, 1968; Hart *et al.*, 1972) wrote the classic introductions to the field (Schaeffer & Plant, 2000).

George Polya defines *heuristic* as "the study of the methods and rules of discovery and invention" (Polya, 1945). This meaning can be traced to the term's Greek root, the verb *eurisco*, which means "I discover." When Archimedes emerged from his famous bath clutching the golden crown, he shouted "Eureka!" meaning "I have found it!". In state space search, *heuristics* are formalized as rules for choosing those branches in a state space that are most likely to lead to an acceptable problem solution (Luger, 2005).

A heuristic is a technique that improves the efficiency of a search process, possibly by sacrificing claims of completeness. Heuristics are like tour guides. They are good to the extent that they point in generally interesting directions; they are bad to the extent that they may miss points of interest to particular individuals. But, on the average, they improve the quality of the paths that are explored. Using good heuristics, we can hope to get good (though possibly no optimal) solutions



to hard problems, such as the traveling salesman, in less than exponential time.

The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available: The more accurately the heuristic function estimates the true merits of each node in the search tree (or graph), the more direct the solution process. In the extreme, the heuristic function would be so good that essentially no search would be required. The system would move directly to a solution (Rich & Knight, 2000).

### **2.2.2.1 Hill Climbing (HC)**

The simplest way to implement heuristic search is through a procedure called *hill-climbing* (Pearl, 1984).

To move through a tree of paths using hill climbing, one proceeds as he would in depth-first search, except that searcher orders his/her choices according to some heuristic measure of the remaining distance to the goal. The better the heuristic measure is, the better hill climbing will be relative to ordinary depth-first search. One can note that Hill climbing is depth-first search with a heuristic measurement that orders choices as nodes are expanded, where quality measurements turn Depth-First Search into Hill Climbing.

Hill-climbing strategies expand the current state of the search and evaluate its children. The best child is selected for further expansion: neither its siblings nor its parent are retained. Hill-climbing is named for the strategy that might be used by an eager, but blind mountain climber: go uphill along the steepest possible path until you can go no farther up. Because it keeps no history, the

algorithm cannot recover from failures of its strategy.

From a procedural point of view, hill climbing differs from depth-first search in only one detail; there is an added step: ***Sort the new paths, if any, by the estimated distances between their terminal nodes and the goal.***

### **2.2.3 Optimal Search**

Several methods exist that do identify the optimal path through a search tree. The optimal path is the one that has the lowest cost or involves traveling the shortest distance from start to goal node. The techniques described previously may find the optimal path by accident, but none of them are guaranteed to find it (Coppin, 2004).

Optimal Search techniques deal with search situations in which the cost of traversing a path is of primary importance. In this section we are to explain the British Museum procedure, while we will explain the more sophisticated techniques for identifying optimal paths like: branch and bound, discrete dynamic programming, and A\* procedures in the next chapters.

#### **2.2.3.1 British Museum Procedure (BMP)**

One Procedure for finding the shortest path through a net is to find all possible paths and to select the best one from them. This Procedure is known as British Museum Procedure, where BMP looks every where.

British Museum procedure is the simplest method for identifying the optimal path. This process involves examining every single path through the search tree and returning via the best path that was found. Because every path is examined, the optimal path must be found. This process is implemented as an extension of one of the exhaustive search techniques, such as depth-first or breadth-first

search, but rather than stopping when a solution is found, the solution is stored and the process continues until all paths have been explored. If an alternative solution is found, its path is compared with the stored path, and if it has a lower cost, it replaces the stored path. If the breadth and depth of the tree are small, then there are no problems.

Unfortunately, the size of search trees is often large, making any procedure for finding all possible paths extremely unpalatable. Suppose that instead of the number of levels being small, it is moderately large. Suppose further that the branching is completely uniform and that the number of alternative branches at each node is  $b$ . Then, in the first level, there will be  $b$  nodes. For each of these  $b$  nodes, there will be  $b$  more nodes in the second level, or  $b^2$ . Continuing this analysis leads to the conclusion that the number of nodes at depth  $d$  must be  $b^d$ . For even modest breadth and depth, the number of paths can be large. For example;  $b = 10$  and  $d = 10$  yields 10 billion paths. Fortunately, there are strategies that enable optimal paths to be found without all possible paths being found first.

#### **2.2.4 Properties of Search Methods**

There are several important properties that search methods should have in order to be most useful. In particular, we will look at the following properties (Coppin, 2004):

- Complexity
- Completeness
- Optimality
- Admissibility

## **Complexity**

In discussing a search method, it is useful to describe how efficient that method is, over time and space.

The *time complexity* defines how fast an algorithm performs and scales (Bigus & Bigus, 2001). The **time complexity** of a method is related to the length of time that the method would take to find a goal state (Coppin, 2004).

The *space complexity* describes how much memory the algorithm requires to perform the search (Bigus & Bigus, 2001). The **space complexity** is related to the amount of memory that the method needs to use (Coppin, 2004).

It is normal to use Big-O notation to describe the complexity of a method. For example, breadth-first search has a time complexity of  $O(b^d)$ , where  $b$  is the branching factor of the tree, and  $d$  is the depth of the goal node in the tree (Coppin, 2004).

## **Completeness**

A search method is **complete** if it is guaranteed to find a solution (a goal state) if one exists. Breadth-first search is complete, but depth-first search is not because it may explore a path of infinite length and never find a goal node that exists on another path (Bigus & Bigus, 2001; Coppin, 2004).

## **Optimality**

A search algorithm is **optimal** if it is guaranteed to find the best solution from a set of possible solutions (Bigus & Bigus, 2001). In other words, it will find the path to a goal state that involves taking the least number of steps (Coppin, 2004). This does not mean that the search method itself is efficient, it might take a great deal of time for an optimal search method to identify the optimal

solution, but once it has found the solution, it is guaranteed to be the best one. This is fine if the process of searching for a solution is less time consuming than actually implementing the solution. On the other hand, in some cases implementing the solution once it has been found is very simple, in which case it would be more beneficial to run a faster search method, and not worry about whether it found the optimal solution or not.

Breadth-first search is an optimal search method, but depth-first search is not. Depth-first search returns the first solution it happens to find, which may be the worst solution that exists. Because breadth-first search examines all nodes at a given depth before moving on to the next depth, if it finds a solution, there cannot be another solution before it in the search tree.

In some cases, the word *optimal* is used to describe an algorithm that finds a solution in the quickest possible time, in which case the concept of **admissibility** is used in place of optimality. An algorithm is then defined as **admissible** if it is guaranteed to find the best solution. A\* is admissible when the heuristic function never overestimates (Doyle, Dec. 5, 2005).

**Successful search method should satisfy three properties:**

1. **Completeness (must be complete):** It eventually produces all possible solutions. In other words, it must generate every possible solution; otherwise it might miss a suitable solution.
2. **non-redundant (must be nonredundant):** It never proposes a solution more than once. In other words, it should not generate the same solution twice.

3. **informed** (must be **well informed**): It uses information to limit the possibilities and hence the number of solutions proposed. This means that it should only propose suitable solutions and should not examine possible solutions that do not match the search space. A function  $h_1$  is more informed than a function  $h_2$  if for all non-goal nodes  $n$ ,  $h_2(n) > h_1(n)$  (Doyle, Dec 5.,2005; Coppin, 2004).

### 2.2.5 The effect of heuristic accuracy on performance

One way to characterize the quality of a heuristic is the **effective branching factor**  $b^*$ . If the total number of nodes generated by A\* for a particular problem is  $N$ , and the solution depth is  $d$ , then  $b^*$  is the branching factor that a uniform tree of depth  $d$  would have in order to contain  $N + 1$  nodes. Thus,

$$N+1=1 + b^* + (b^*)^2 + \dots + (b^*)^d .$$

For example. if A\* finds a solution at depth 5 using 52 nodes:

$$52+1=1 + b^* + (b^*)^2 + \dots + (b^*)^5 . ,$$

then the effective branching factor is 1.92.

To calculate this , we can use the well known mathematical identity:

$$\frac{x^n - 1}{x - 1} = 1 + x + \dots + x^{n-1} , \text{ then } b^* \text{ can be calculated by:-}$$

$$\frac{b^{*(d+1)} - 1}{b^* - 1} = N + 1 = 1 + b^* + \dots + b^{*d}$$

This enables us to write a polynomial for which  $b^*$  is zero, and we can solve this using numerical techniques such as Newton's method.

The effective branching factor can vary across problems. instances, but

usually it is fairly constant for sufficiently hard problems. Therefore, experimental measurements of  $b^*$  on a small set of problems can provide a good guide to the heuristic's overall usefulness. A well - designed heuristic would have a value of  $b^*$  close to 1, allowing fairly large problem to be solved (Russell& Norvig, 2003; Colton, 2005).

# Chapter three

## (branch and bound) search

### Using fuzzy underestimates

#### 3.1 Introduction

The Traveling Salesman Problem (TSP), in which a salesman makes a complete tour of the cities on his route and visits each city exactly once while traveling the shortest possible distance, is an example of a problem that has a combinatorial explosion. As such, it cannot be solved using breadth-first or depth-first search for problems of any realistic size. Unfortunately, there are many problems which have this form and which are essentially intractable. In these cases, finding the best possible answer is not computationally feasible, and so we have to settle for a good answer (Bigus & Bigus, 2001).

The most common problem-solving technique for the situation depicted above is what is called heuristic search. It generally encompasses a collection of methods, principles and criteria for guiding problem-solving activities, based on rule of thumb, on discrimination between the good and bad search-step selected, or simply on repeated evaluation of the progress made toward selected, or simply on repeated evolution of the progress made toward the solution goal. In doing so we apply our heuristic knowledge, gained from hands-on experience. Such knowledge involves a mixture of facts, simplified evaluation criteria, and rules of thumb. The heuristic search technique includes search concepts like: Hill-climbing, Best-first search, Beam search, Branch & Bound Search, and A\* (Bhatkar, 1994).



**Branch and Bound** is one of several techniques which can reduce the search complexity (Horowitz and Sahni 1978; Luger, 2005). Branch and bound algorithm traces a decision tree whose leaves represent all possible solutions. Design decisions are made at each internal node while the leaves of the subtree rooted at an internal node are the solutions due to that decision. Given a best solution found during execution of the branch and bound algorithm, a subtree can be pruned if a lower bound estimate of the cost function of all solutions of the subtree is higher than the cost of the current best solution. Tight and fast computable lower bounds therefore improve the run time requirements of such algorithms (Kruse et al. , 2000).

**Branch and Bound** augmented by underestimate search is an improved version for B&B search. It uses the known cost combined with an estimate of the distance from the state to the goal in order to choose the best node to expand. **Branch and Bound augmented by underestimate** is complete and optimal, and has memory requirements comparable to depth-first search (Bigus & Bigus, 2001).

The existing Branch and Bound searching technique is a technique that works well on precise data, but not on imprecise data whereas data available are not always crisp in real life. Fuzzy logic might be an appropriate tool to enhance dealing with such problems.

In this work, a new type of Branch and Bound searching technique using fuzzy underestimates is proposed. Our objective in this dissertation, is to deal with the imprecise data involved in different kind of existing searching techniques,

in more efficient ways. Thus an improved version of searching techniques under uncertainty has been suggested, which will be helpful in many real life problems of computer science, especially in AI field.

In this chapter we will consider a search problem and its solution by the existing crisp method of branch and bound search. Consequently we will propose a new method of branch and bound with fuzzy underestimates, giving the corresponding algorithm, and explaining the algorithm by two applications (examples).

We are to explain in details **Branch and Bound Search (Crisp Method)** by providing crisp Branch and Bound algorithm, and crisp Branch and Bound augmented by underestimates algorithm giving : Search Examples, Search algorithm explanations for both procedures.

Then we are to explain in details **Branch and Bound Search (Fuzzy Method)** by providing the suggested algorithm explanation giving : Branch and Bound Fuzzy Method algorithm, and flow chart explanation.

Finally we are to explain in details two applications as examples for the suggested algorithm.

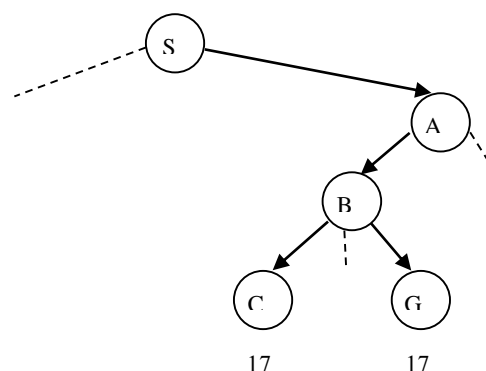
### **3.2 Branch and Bound Search : Crisp Method**

One way to find optimal paths with less work is to use branch-and-bound search, where B&B Expands the least-cost partial path. The basic idea is simple. Suppose an optimal solution is desired for the highway map shown previously. Also suppose that an other source has told you that S-A-B-G is the optimal solution. Being a scientist, however you do not trust others.

Nevertheless, knowing that the length of S-A-B-G is 17; you can eliminate some work that you might otherwise do. For example, as shown in Figure (3.1) there is no need to consider paths that start with S-A-B-C, because their length has to be at least 17, given that the length of S-A-B-C is already 17.

**Figure 3.1: Branch-and-Bound Search.**

Length of the complete path from S to G, S-A-B-G is 17. Similarly, the length of the partial path S-A-B-C also is 17 and any additional movement along a branch will make it longer than 17. Accordingly, there is no need to pursue S-A-B-C any further because any complete path starting with S-A-B-C has to be longer than a complete path already known. Only the other paths emerging from S, from S-A and from S-A-B have to be considered, as they may provide a shorter path.



More generally, the branch-and-bound scheme always keeps track of all partial paths contending for further consideration. The shortest one is extended one level, creating as many new partial paths as there are branches. Next, these new paths are considered, along with the remaining old ones: again, the shortest is extended. This process repeats until the goal is reached along some path. Because the shortest path was always the one chosen for extension, the path first reaching the goal is “likely” to be the optimal path.

To turn “likely” into “certain” the searcher has to extend all partial paths until they are as long as or longer than the complete path. The reason is that the last step in reaching the goal may be long enough to make the supposed solution longer than one or more partial paths. It might be that only a tiny step would extend one of the partial paths to the solution point. To be sure that this is not so, instead of terminating when a path is found, you terminate when the shortest partial path is longer than the shortest complete path.

The procedure differs from the basic search procedures only in the steps shown in bold italic bellow:

---

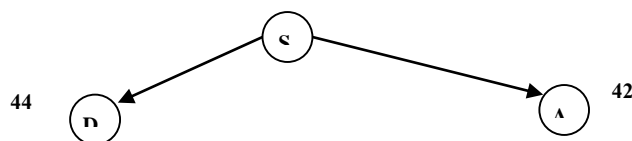
To conduct a branch-and-bound search,

- 1- Form a one-element queue consisting of a zero-length path that contains only the root node.
- 2- Until the first path in the queue terminates at the goal node or the queue is empty,
  - 2.1- Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
  - 2.2- Reject all new paths with loops.
  - 2.3- **Add the remaining new paths, if any, to the queue.**
  - 2.4- **Sort the entire queue by path length with least-cost paths in front.**
- 3- If the goal node is found, announce success; otherwise, announce failure.

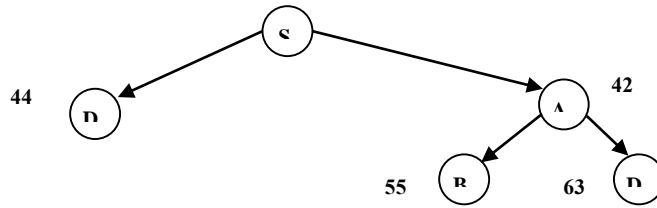
---

Now look again at the map-traversal problem, and note how branch-and-bound works when started with no partial paths, Figure (3.2) illustrates the exploration sequence, where the numbers beside the nodes denotes the length of each path (cost) .

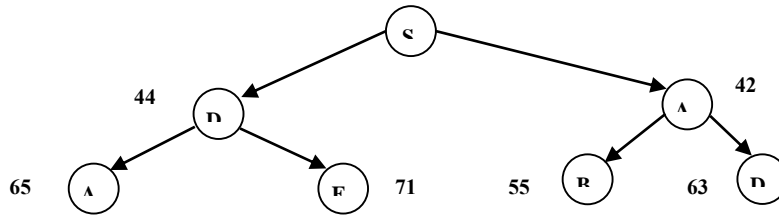
**Figure 3.2:** Branch and Bound Search Example



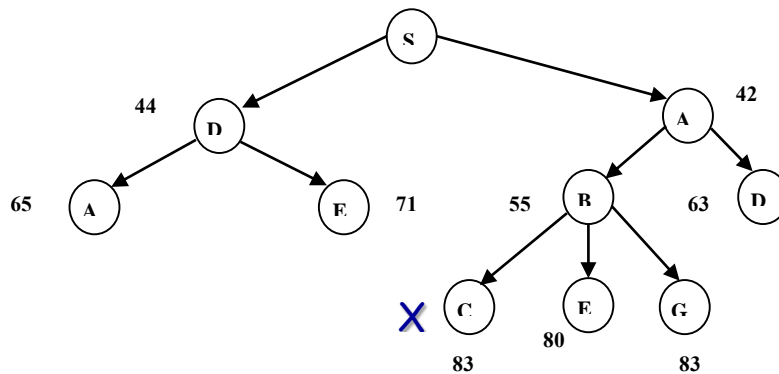
- 1- In the first step, the partial-path distance of S-A is found to be 42, and that of S-D is found to be 44; partial path S-A is therefore selected for expansion.



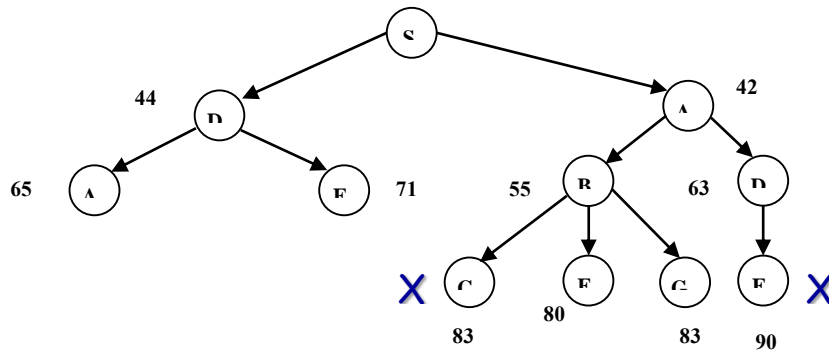
2- Next, S-A-B and S-A-D are generated from S-A with partial path distances of 55 and 63.



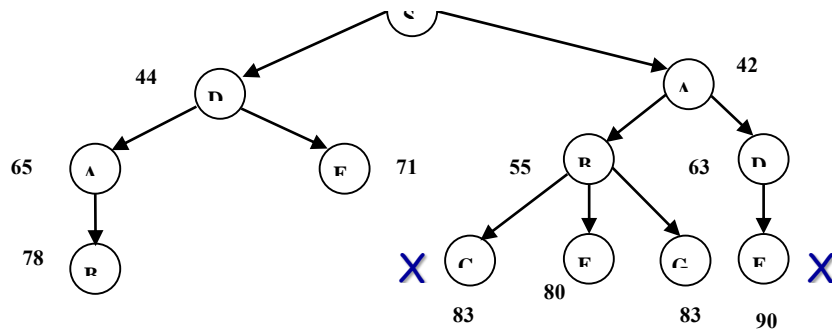
3- Now S-D, with a partial path distance of 44, is expanded, leading to partial paths to S-D-A and S-D-E. At this point, there are four partial paths, with the path S-A-B being the shortest with a partial path distance of 55.



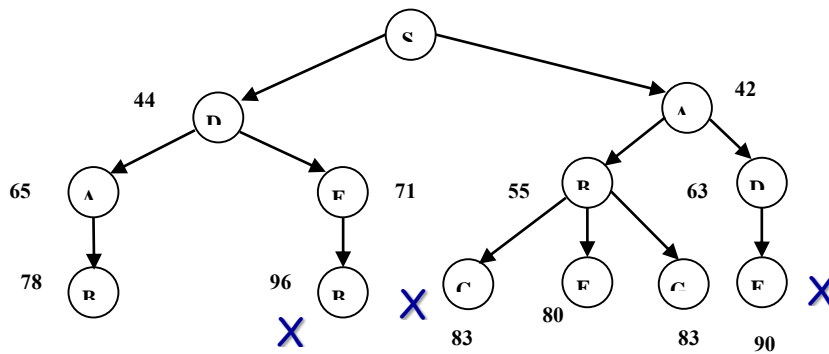
4- Then expanding S-A-B, leads to S-A-B-C, S-A-B-E, and S-A-B-G with partial path distances of 83, 80, and 83, where S-A-B-G is the shortest complete path, but to be absolutely sure, all partial paths with partial path distances less than 83 must be expanded. There is no need to extend the partial path S-A-B-C, because its partial-path distance of 83 is equal to that of the complete path.



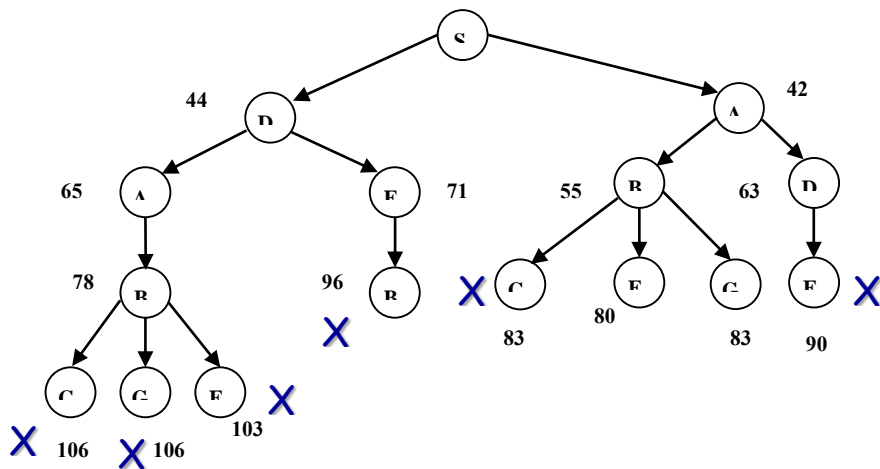
5-Now S-A-D, with a partial path distance of 63, is expanded, leading to partial path S-A-D-E. At this point, there are six partial paths, with the path S-D-A being the shortest with a partial path distance of 65.



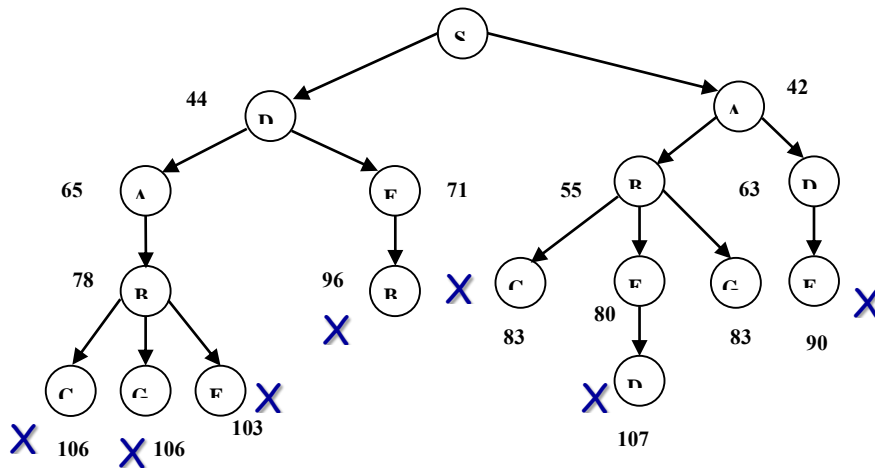
6-Then S-D-A-B is generated from S-D-A with partial path distances 78.



7-Now S-D-E-B is generated from S-D-E with partial path distances 96. After the seventh step, partial path S-D-A-B is the shortest partial path.



8-Expanding S-D-A-B leads to partial paths terminating at C, G, and E.

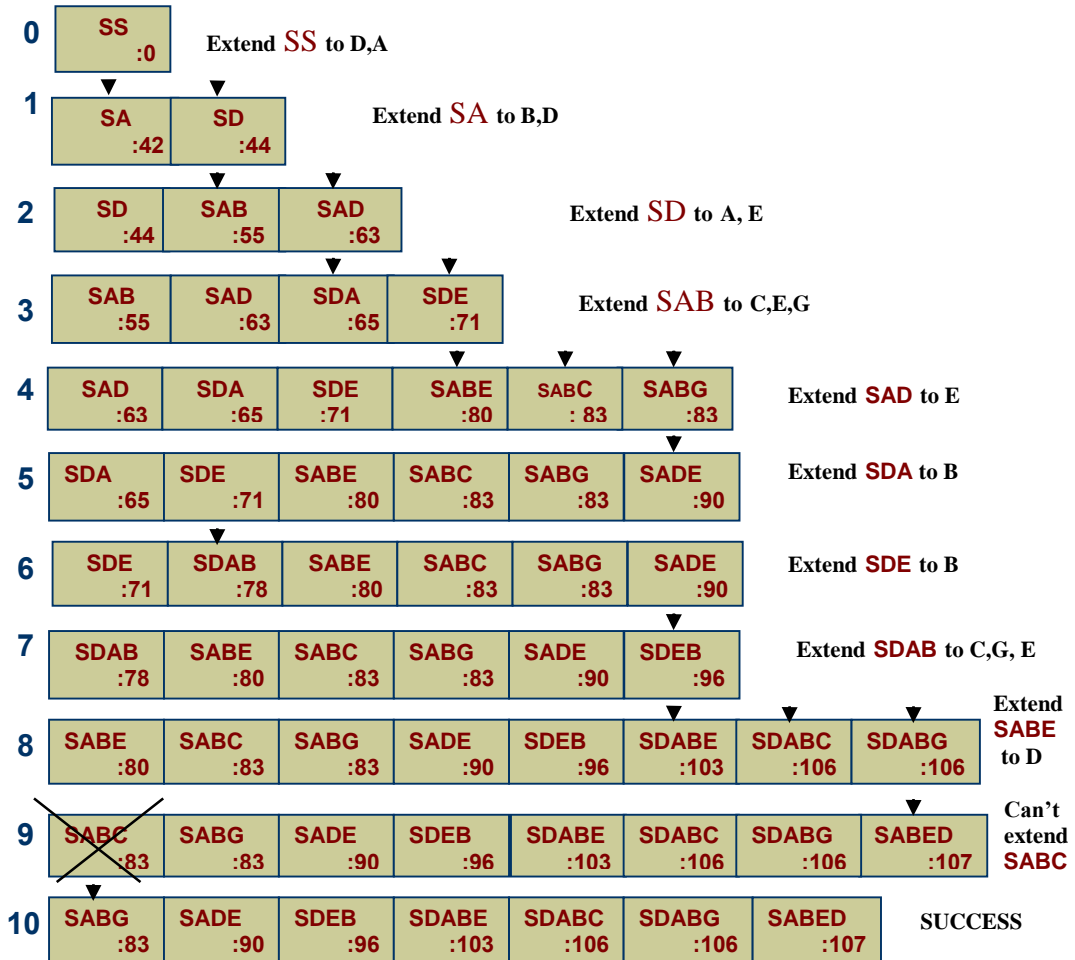
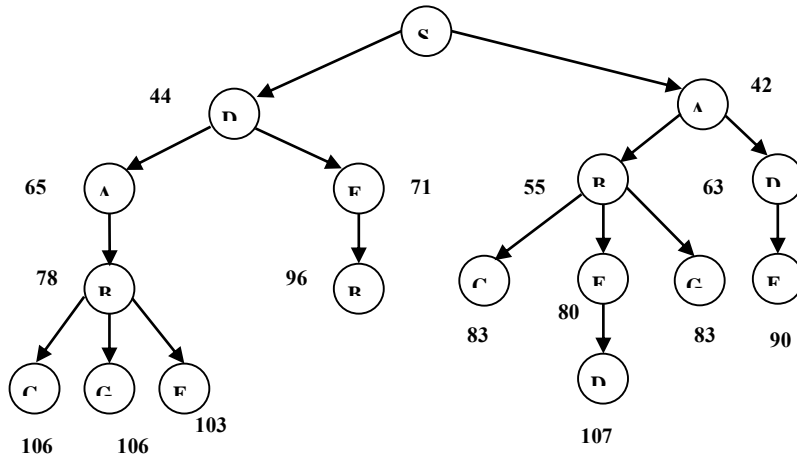


9- Finally, expanding S-A-B-E, leads to partial path S-A-B-E-D, with a partial-path distance of 107. Then there is no need to extend any partial path, because their partial-path distances exceed the complete path (83).

In this particular example, little work is avoided relative to exhaustive search, British Museum style.

Figure (3.3) will explain the algorithm of Branch-and-Bound search according to the previous example shown in Figure (3.2), where the numbers beside the nodes are the length of each path.

Figure3.3: Branch and Bound Search algorithm explanation





### 3.2.1 Adding Underestimates to (Branch and Bound) Search

In some cases, branch-and-bound search can be improved greatly by using guesses about distances remaining, as well as facts about distances already accumulated. After all, if a guess about distance remaining is suitable, then that guessed distance added to the definitely known distance already traversed should be a good estimate of total path length, **e (total path length)**:

$$e(\text{total path length}) = d(\text{already traveled}) + e(\text{distance remaining}),$$

where **d (already traveled)** is the known distance already traveled, and where **e (distance remaining)** is an estimate of the distance remaining.

Surely it makes sense to work hardest on developing the path with the shortest estimated path length until the estimate is revised upward enough to make some other path be the one with the shortest estimated path length. After all, if the guesses were perfect, this approach would keep you on the optimal path at all times.

In general, however, guesses are not perfect, and a bad overestimate somewhere along the true optimal path may cause you to wander away from that optimal path permanently. Note, however, that **underestimates** cannot cause the right path to be overlooked. An underestimate of the distance remaining yields an underestimate of total path length, **u(total path length)**:

$$u(\text{total path length}) = d(\text{already traveled}) + u(\text{distance remaining}),$$

where **d(already traveled)** is the known distance already traveled, and **u(distance remaining)** is an underestimate of the distance remaining.

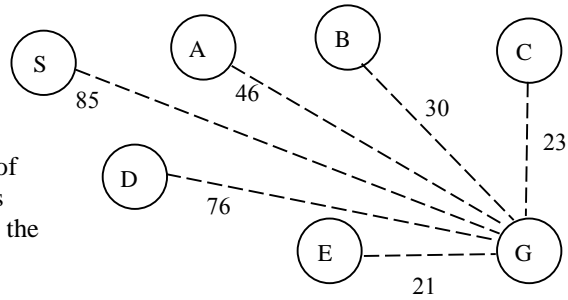
Now, if one finds a total path by extending the path with the smallest underestimate repeatedly, he needs to do no further work once all partial- path distance estimates are longer than the best complete path distance so far encountered. One can stop because the real distance along a complete path cannot be less than an underestimate of that distance. If all estimates of remaining distance can be guaranteed to be underestimates, searcher cannot blunder.

When one is working out a path on a highway map, straight-line distance is guaranteed to be an underestimate. Figure (3.4) shows the straight-line distances from each city to the goal which are considered as underestimates of distances remaining (**ur**). Figure (3.5) shows the already traveled distances at each city (**d**). **Figure (3.6)** shows how straight- line distance helps to make the search efficient, where branch-and- bound search augmented by underestimates determines that the path S-A-B-C-G is optimal. The numbers beside the nodes are underestimate of total path length (**ut**) which is calculated as follows:

**(ut)** = accumulated **d**istances(**d**) + **u**nderestimates of distances remaining(**ur**).

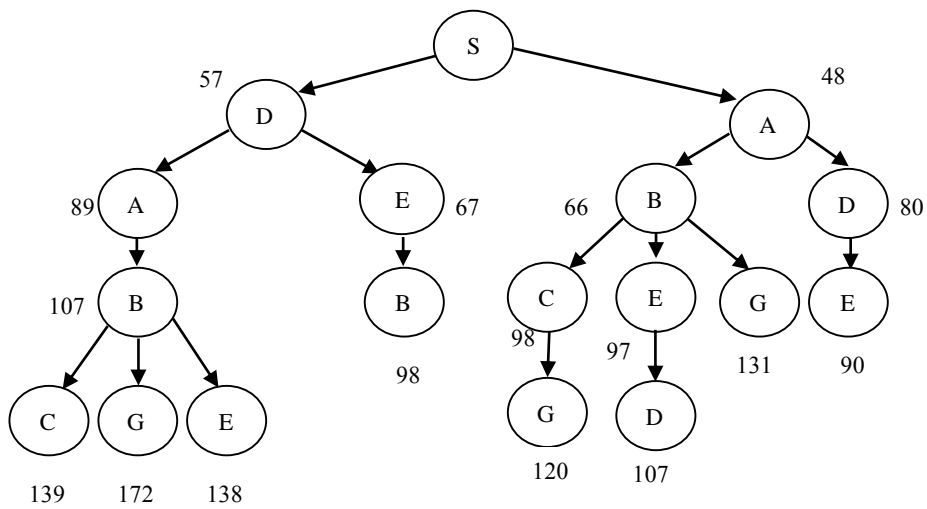
Underestimates quickly push up the lengths associated with bad paths. In this example, many fewer nodes are expanded than would be expanded with

branch-and-bound search operating without underestimates.

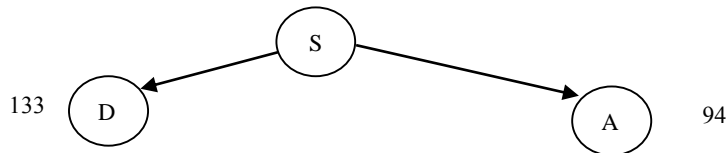


**Figure 3.4:** Example of straight-line distances between each city and the goal = (ur).

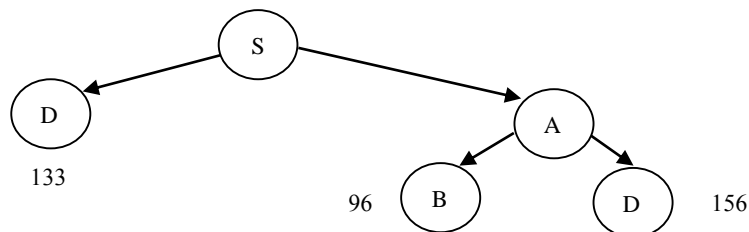
**Figure 3.5:** Example of already traveled distances at each city = (d).



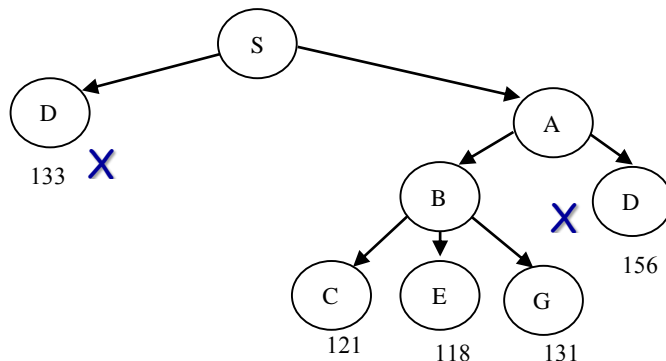
**Figure 3.6:** Branch and Bound Search augmented by underestimates Example



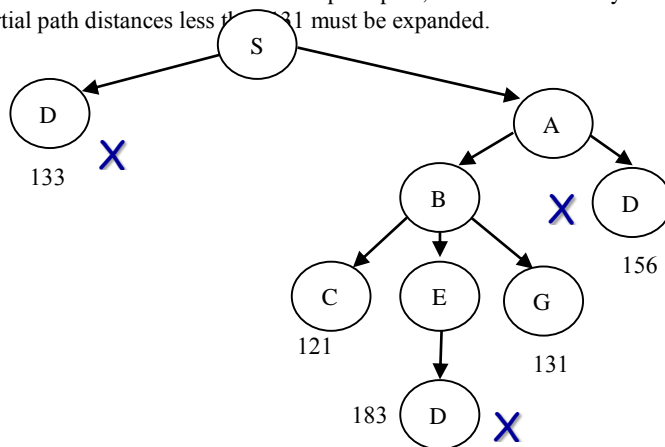
**1-** In the first step, as before, D and A are generated from S, at node A,  $A_{ut} = d(48) + ur(46) = 94$ , at node D,  $D_{ut} = d(57) + ur(76) = 133$ , A is the node from which to search, because A's underestimated path length is 94, which is shorter than that for D, 133.



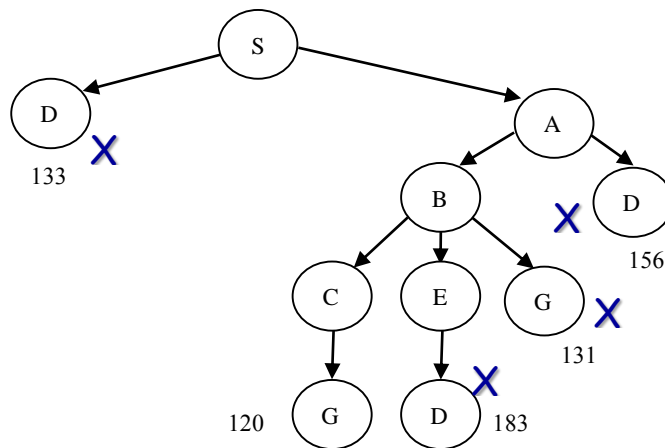
2-Expanding A leads to partial paths S-A-B, with an underestimated path length  $=d(66)+ur(30)=96$ , and to partial path S-A-D, with a underestimated path length  $=d(80)+ur(76)=156$ .



3-Now S-A-B is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to partial paths S-A-B-C, with an underestimated path length  $=d(98)+ur(23)=121$ , S-A-B-E with an underestimated path length  $=d(97)+ur(21)=118$ , and to partial path S-A-B-G, with a underestimated path length  $=d(131)=131$ , where S-A-B-G is the shortest complete path, but to be absolutely sure, all partial paths with partial path distances less than 131 must be expanded.



4- Now S-A-B-E is the partial path to extend, as it is the partial path with the minimum underestimated path length  $=118$ . This expansion leads to partial path S-A-B-E-D, with an underestimated path length  $=d(107)+ur(76)=183$ .



5- Finally S-A-B-C is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to a complete path, S-A-B-C-G, with a total distance of 120. No partial path has a lower-bound distance, so low, so no further search is required.

In the previous example, a great deal of work is avoided. Here is the modified procedure, with the modification in *italic*:

---

To conduct a branch-and-bound search with a lower-bound estimate,

- 1-Form a one-element queue consisting of a zero-length path that contains only the root node.
- 2-Until the first path in the queue terminates at the goal node or the queue is empty,
  - 2.1- Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
  - 2.2- Reject all new paths with loops.
  - 2.3- Add the remaining new paths, if any, to the queue.
  - 2.4- Sort the entire queue by ***the sum of the path length and a lower-***

***bound estimate of the cost remaining, with least-cost paths in front.***

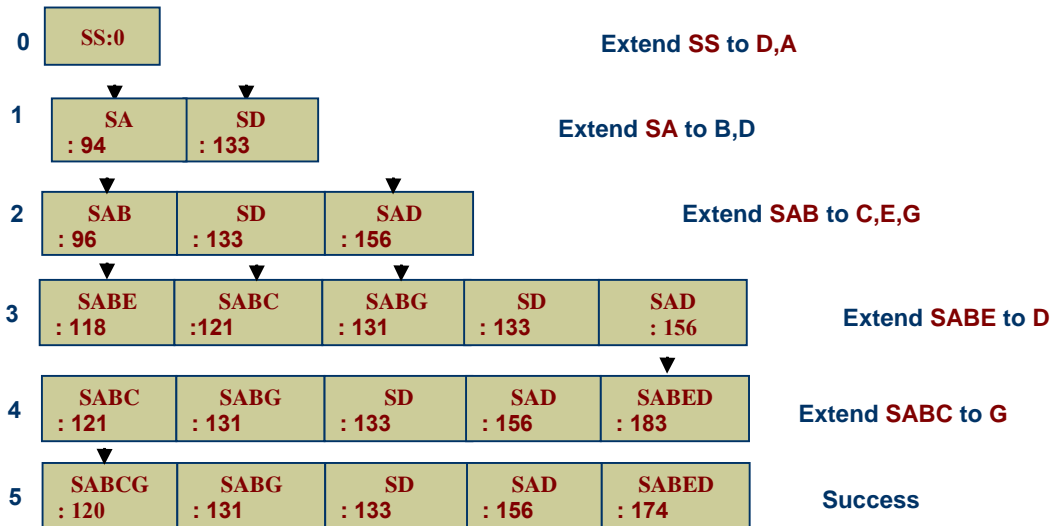
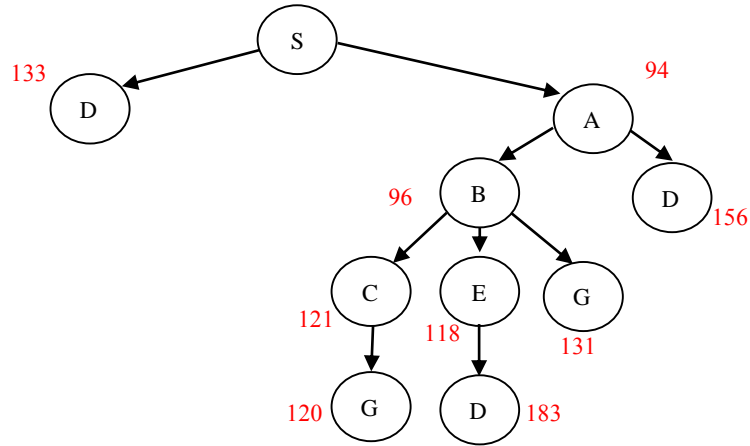
**3-** If the goal node is found, announce success; otherwise, announce failure.

---

Of course, the closer an underestimate is to the true distance, the more efficiently the search because, if there is no difference at all, there is no chance of developing any false movement. At the other extreme, an underestimate may be so poor as to be hardly better than a guess of zero, which certainly must always be the ultimate underestimate of remaining distance. In fact, ignoring estimates of remaining distance altogether can be viewed as the special case in which the underestimate used is uniformly zero.

Figure (3.7) will explain the algorithm of Branch-and-Bound search augmented by underestimate, according to the previous example:

**Figure 3.7:** Branch and Bound Search augmented by underestimate/algorithm explanation.



### 3.3. Branch and Bound Search : Fuzzy Method

The suggested 'Branch and Bound Searching Technique Using Fuzzy Underestimate' is the same as the existing 'Branch and Bound Searching Technique Using Crisp Underestimate' in all steps, unless that the suggested method deals with underestimation of the remaining distance as a fuzzy data , taking into consideration the assumption that “the underlying graph is crisp and the parameters related with its arcs are fuzzy numbers.” (Blue *et al*,2002).

A fuzzy underestimate of the distance remaining yields a fuzzy underestimate of total path length, ***utf*** (total path length):

$$\mathbf{utf} \text{ (total path length)} = \mathbf{d} \text{ (already traveled)} + \mathbf{urf} \text{ (distance remaining)},$$

where ***d*** (already traveled) is the known distance already traveled, and ***urf*** (distance remaining) is a fuzzy underestimate of the distance remaining.

Fuzzy underestimation for the remaining distance (fuzzy data) can be processed according to the following steps as shown in Figure (3.8):



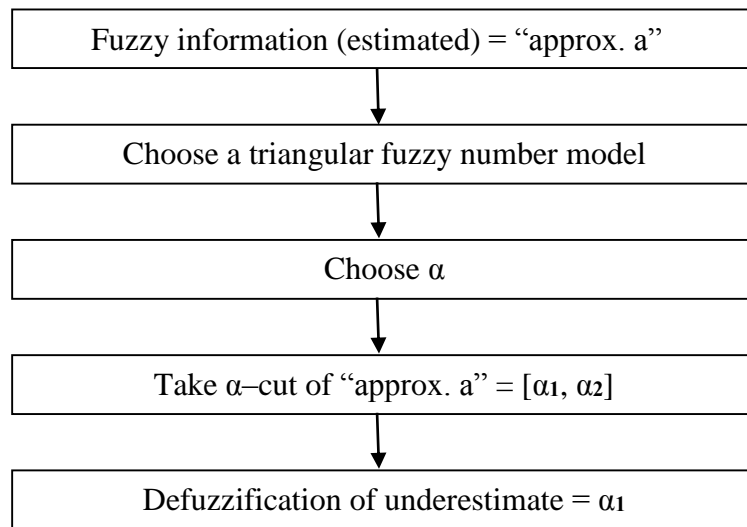


Figure 3.8: Fuzzy data processing steps.

### 3.3.1 Fuzzy underestimate

The following fuzzy data expressions (as estimation of the remaining distance) can be provided to the searcher:

- Examples:-
  - 1) underestimate may be “approx. 24”
  - 2) Underestimate may be “at least 24”
  - 3) Underestimate may be “More than 24”
  - 4) Underestimate may be “not less than 24” ....Etc.

Searcher knows that :

- These all are fuzzy numbers, then.
- Choose a Triangular Fuzzy Number (TFN) model.

### 3.3.1.1 Using a TFN

Each of the above data “estimate” can be modeled as a triangular fuzzy number, where a triangular fuzzy number specifications can be determined by a decision maker.

A triangular fuzzy number for “approx. a” can be as shown in Figure (3.9) :

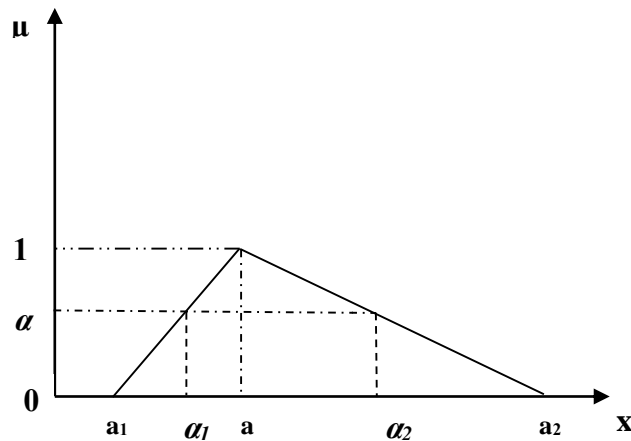


Figure 3.9: TFN model for “approximately a” or “approx. a”

This TFN “approx. a” is denoted by the notation  $(a_1, a, a_2)$ . The membership function of the fuzzy set “approx. a” is given by the following function:-

$$\mu_{\text{“approx. a”}}(x) = \begin{cases} 0 & , \text{ if } x \leq a_1 \\ \left( \frac{x - a_1}{a - a_1} \right) & , \text{ if } a_1 \leq x \leq a \\ \left( \frac{a_2 - x}{a_2 - a} \right) & , \text{ if } a \leq x \leq a_2 \\ 0 & , \text{ if } x \geq a_2 \end{cases}$$

### 3.3.1.2 Choosing $\alpha$

A fixed "decision parameter"  $\alpha$  must be chosen according to confidence in the source of estimation, or confidence in the value of estimation.  $\alpha \in (0,1)$ , according to the following:

- \* If the decision maker (searcher) is **highly confident**, then he can choose **high value of  $\alpha$** , such as: 0.9, or 0.95. ...etc.
- \* But if he is not highly confident, better not to choose a high value of  $\alpha$ , choose.  $\alpha = 0.8, 0.75, \text{ or } 0.82. \dots\text{etc.}$

### 3.3.1.3 $\alpha$ -cut of "approx. a" = $[\alpha_1, \alpha_2]$

The  $\alpha$ -cut of "approx. a" will be the interval  $[\alpha_1, \alpha_2]$ , where for any

$x \in [\alpha_1, \alpha_2]$  one must have:

$$\mu_{\text{"approx. a"}}(x) \leq \alpha .$$

It can be computed as:

$$\begin{cases} \alpha_1 = a_1 + \alpha \cdot (a - a_1) \\ \alpha_2 = a_2 - \alpha \cdot (a_2 - a) \end{cases}$$

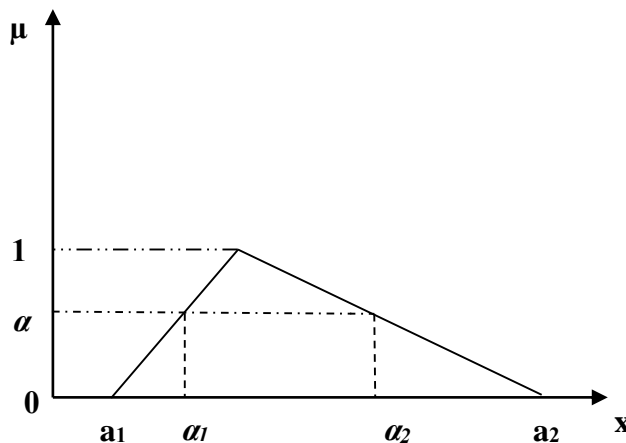


Figure 3.9: TFN model for "approximately a" or "app. a"

### 3.3.1.4 Defuzzification

There are two choices for  $\alpha$ :

$\alpha_1$  = underestimate of the fuzzy number “app. a”

$\alpha_2$  = overestimate of the fuzzy number “app. a”

Then searcher must choose the underestimated value  $\alpha_1$  .

#### Example 3.1:-

1- The provided fuzzy data expressions for the searcher (as estimation of the remaining distance) may be “approx. 27”

2- Consider a TFN “approx. 27” = (24 , 27, 35), as shown in Figure (3.10):

Note: we will consider  $a_1 = (a - 3)$ , and  $a_2 = (a + 8)$  in our examples in order to explain the main idea of our work in a simple way.

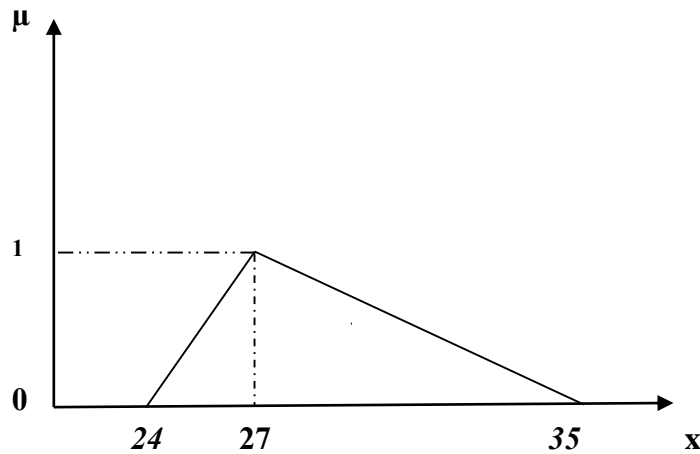


Figure 3.10: TFN model for “approx. 27”

3- Consider any choice–parameters  $\alpha = 0.9$  (say).

4-  $\alpha$ -cut of “approx. 27” will be:

$$\begin{aligned}\alpha_1 &= a_1 + \alpha \cdot (a - a_1) \\ &= 24 + 0.9 \cdot (27 - 24) = 24 + 2.7 = 26.7\end{aligned}$$

$$\begin{aligned}\alpha_2 &= a_2 + \alpha \cdot (a_2 - a) \\ &= 35 - 0.9 \cdot (35 - 27) = 35 - 7.2 = 27.8\end{aligned}$$

Therefore, 0.9-cut of “approx. 27” is the interval [26.7,27.8] as shown in Figure (3.11).

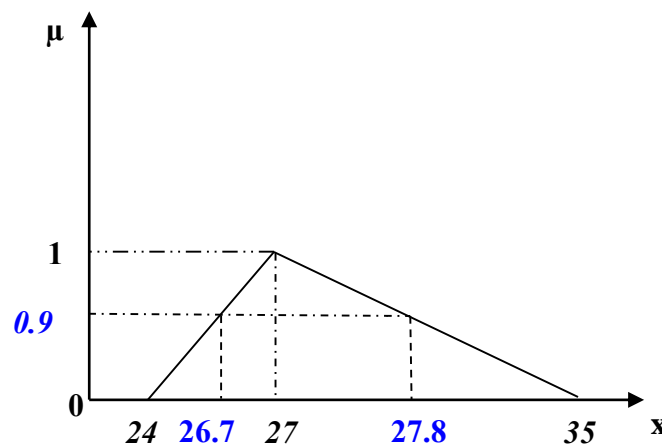


Figure 3.11: TFN model for “approx. 27” with  $\alpha = 0.9$

5- Then, we choose finally  $\alpha_1$  = underestimate of the fuzzy number “app. 27” = **26.7**

The procedure of branch-and-bound search with a fuzzy lower-bound estimate differs from the basic branch-and-bound search procedures only in the steps shown in *italic* bellow:

---

To conduct a branch-and-bound search with a **fuzzy** lower-bound estimate,

**1-**Form a one-element queue consisting of a zero-length path that contains only the root node.

**2-**Until the first path in the queue terminates at the goal node or the queue is empty,

**2.1-** Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.

2.2- Reject all new paths with loops.

2.3- **Determine fuzzy lower-bound estimate of the cost remaining as follow:**

2.3.1- **Take fuzzy estimate of new paths = “approx. a”.**

2.3.2- **Denote TFN for “ approx. a” by the appropriate notation ( $a_1, a_2$ ).**

2.3.3- **Choose a fixed “decision - parameter “  $\alpha$  in (0,1), according to degree of searcher confidence in estimation.**

2.3.4- **Take a fuzzy lower-bound estimate of the cost remaining ( lower  $\alpha$ -cut of “approx. a” ) =  $\alpha_1$ , where:**

$$\alpha_1 = a_1 + \alpha * (a - a_1)$$

2.4- Add the remaining new paths, if any, to the queue.

2.5- Sort the entire queue by **the sum of the path length and a fuzzy lower-bound estimate of the cost remaining**, with least-cost paths in front.

3- If the goal node is found, announce success; otherwise, announce failure.

---

The following flowchart also explains the procedure of branch-and-bound search with a fuzzy lower-bound estimate as shown in Figure (3.12):

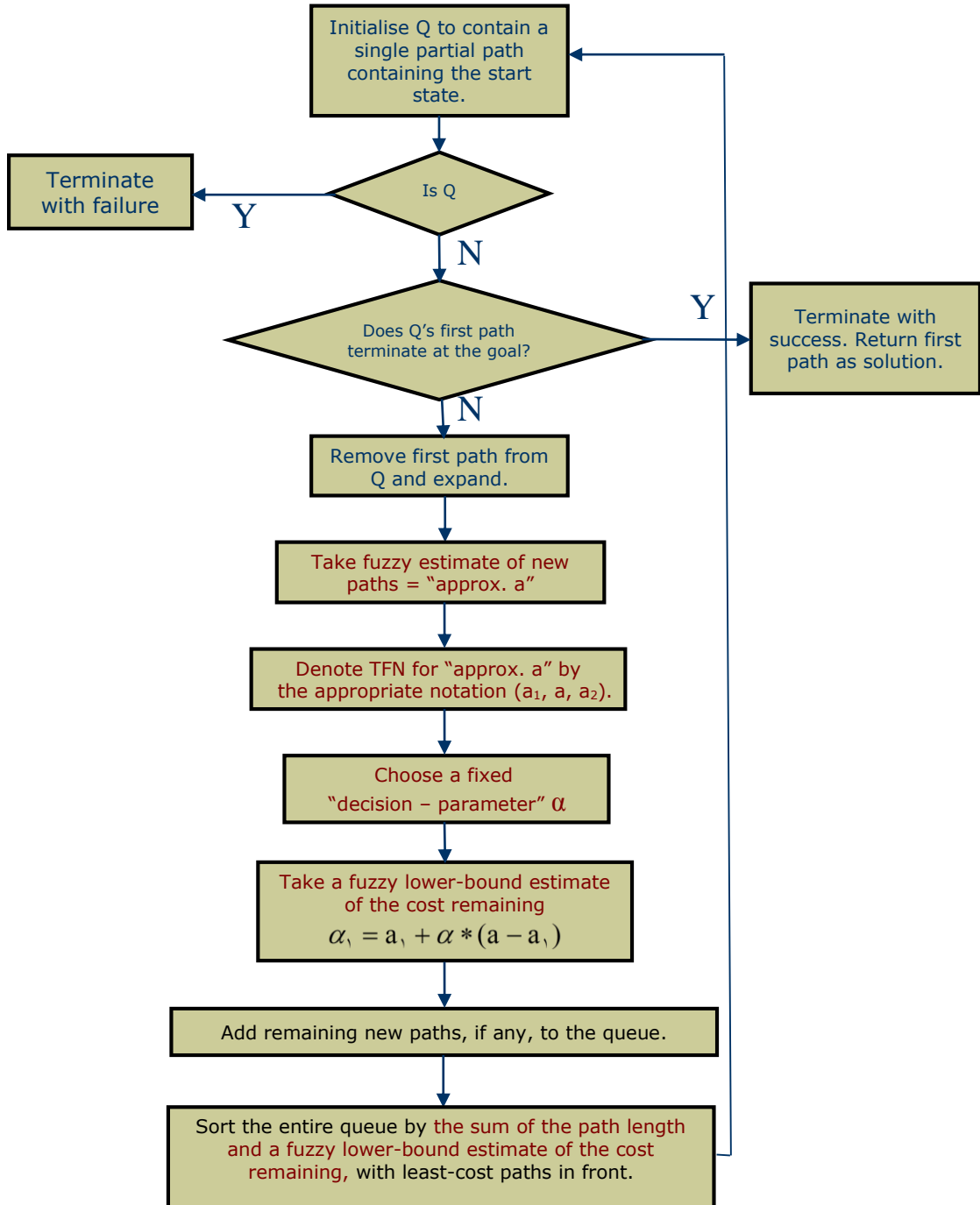


Figure 3.12: flow chart procedure of branch-and-bound search with a fuzzy lower-bound estimate.

When a searcher is working out of a path on a highway map, straight-line distance is guaranteed to be an underestimate, but if searcher has better source of information about the remaining distance estimation, then search procedure will be more efficient.

Confidence in underestimation of remaining distance may vary from case to case; so  $\alpha$  also will vary according to the degree of searcher confidence in the value of underestimation.

### 3.4 Applications

We will take two applications as examples for the proposed branch and bound with fuzzy underestimate algorithm.

The first application will adopt the previous example which was explained for branch and bound augmented by crisp underestimate algorithm (figure 3.5) as a random net application which will be explained in section 3.4.1.

The second application will adopt the real roads between two major Jordanian cities as an example, which will be explained in section 3.4.2.

#### 3.4.1 Random net Application

In the following example, if a decision-maker takes a fixed TFN model slope as  $(a-3, a, a+8)$ , at each node he/she will take fuzzy estimate of remaining distance for new paths = "approx.  $a$ " from different information sources = **IS**, he/she will choose  $\alpha$  according to the degree of searcher confidence in underestimation, take a fuzzy lower-bound estimate of the cost remaining;  $\alpha_1 = a_1 + \alpha * (a - a_1)$ , and finally he/she will add new paths and sort the paths by the sum of the already traveled path length =  $d$  and the fuzzy lower-bound



estimate of the remaining distance =  $ufr$  to choose the shortest path.

Consider the following net, which is shown in Figure (3.13) The traveler is at the node S and intends to go to node G. All possible routs are shown in the net graph, the number against each edge gives the actual distance of that route (node to node) in some unit . The traveler has no knowledge about the distance information; but the traveler records the distance he completed.

**Figure 3.13:** net graph with actual distance of each route.

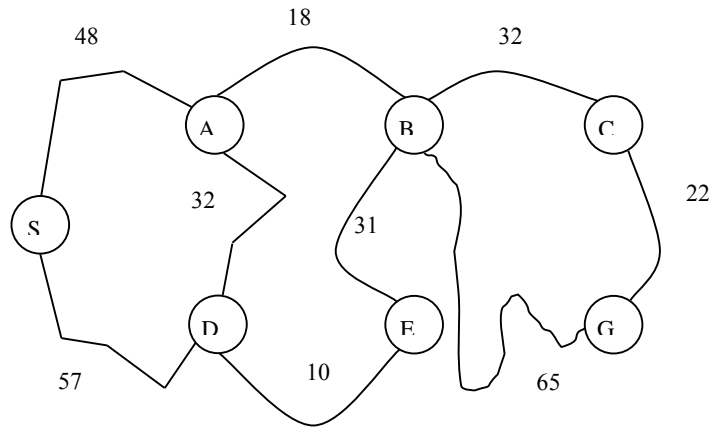
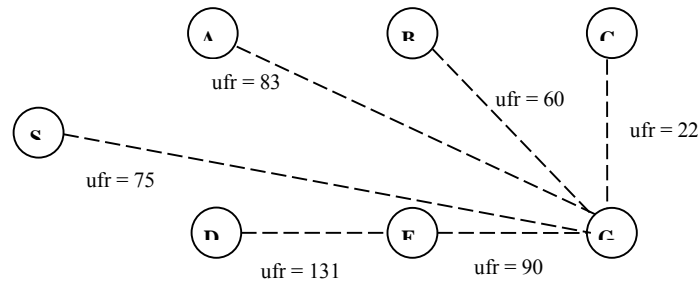


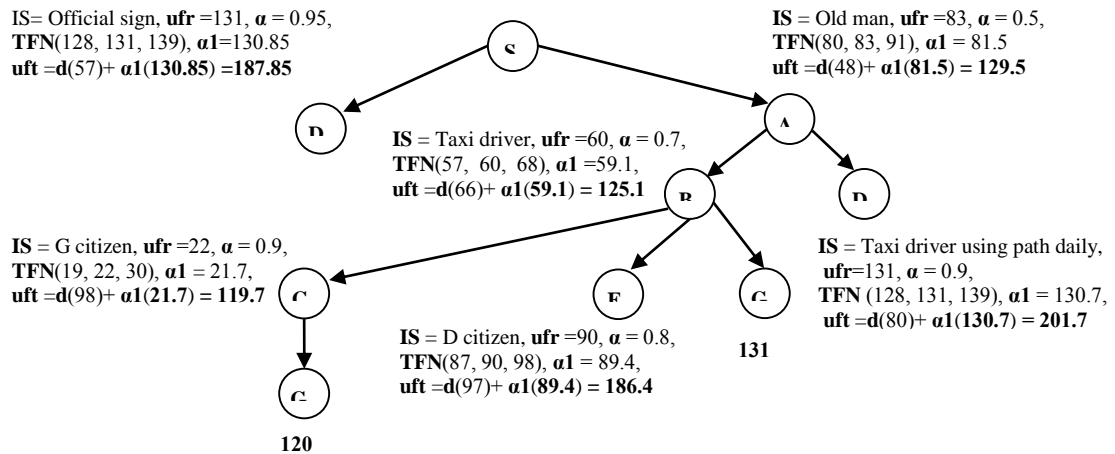
Figure (3.14) shows the fuzzy estimates of distances remaining ( $ufr$ ) from each city to the goal; Figure (3.16) shows how fuzzy underestimates of distances remaining helps to make the search more efficient, where Branch-and- bound search augmented by fuzzy underestimates determines that the path S-A-B-C-G is optimal. The numbers beside the nodes are underestimate of total path length ( $uft$ ) = accumulated distances( $d$ ) + *fuzzy lower-bound estimate of the cost remaining* ( $\alpha_1$ ).

Fuzzy underestimates quickly push up the lengths associated with bad paths. In this example, fewer nodes are expanded than would be expanded with branch-and- bound search operating with crisp underestimates.



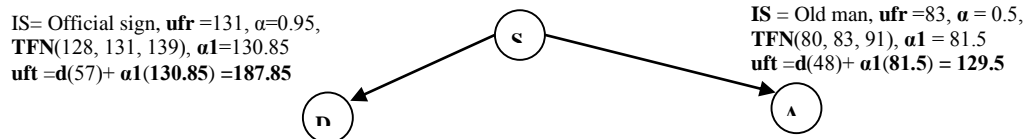
**Figure 3.14:** Example for fuzzy estimates of remaining distances (ufr) between each city and the goal .

Part of the tree, which must be explored by the proposed algorithm will be as shown in Figure (3.15):-



**Figure 3.15:** The explored part of the tree. At each node there is, specific source for estimated information =IS, who (which) will give fuzzy cost underestimate of remaining distance = "approx. a" = a, decision maker will choose an appropriate  $\alpha$ , TFN model, & lower  $\alpha$ -cut for that node.

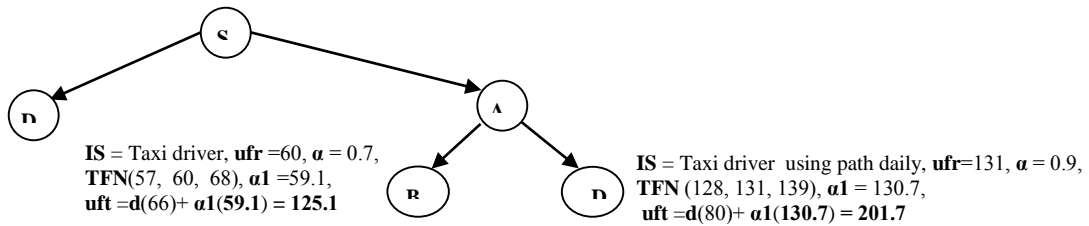
The problem can be solved by applying the proposed algorithm of section 3.2 as shown in the following example:-



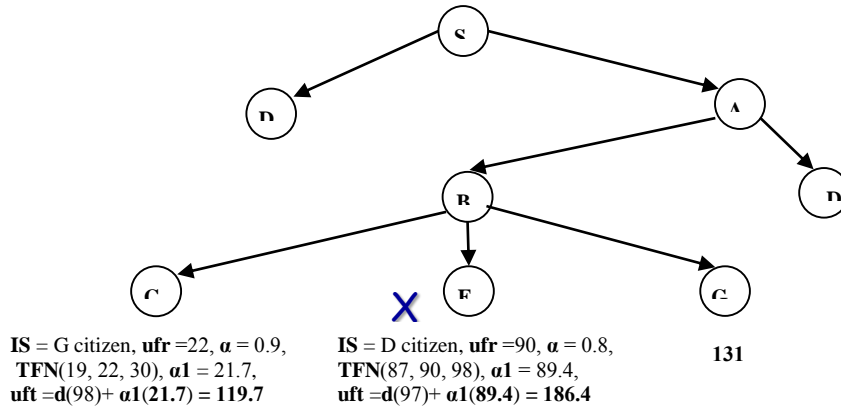
**1-** In the first step, D and A are generated from S, at node A if the information source (IS) was an old man, who gives a fuzzy estimate of remaining distance as a = "approx. 83", decision maker can choose TFN as (80, 83, 91), and  $\alpha = 0.5$  to produce  $\alpha 1 = 81.5$  which will be add to the distance already traveled  $d = 48$ , to produce the fuzzy underestimate of total path length  $uft = 129.5$ .

While node D information source about remaining distance (IS) was an Official sign with a = "approx. 131", decision maker can choose TFN as (128, 131, 139), and  $\alpha = 0.95$  to produce  $\alpha 1 = 130.85$  which will be add to the distance already traveled  $d = 57$ , to produce the fuzzy underestimate of total path length  $uft = 187.85$ .

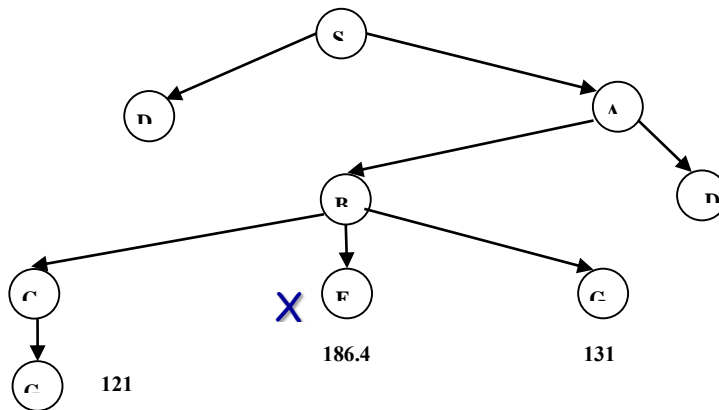
A is the node from which to search, because A's fuzzy underestimated path length is 129.5, which is shorter than that for D, 187.85.



2- Expanding A leads to partial paths **S-A-B**, with a fuzzy underestimated path length of **125.1**, and to partial path **S-A-D**, with a fuzzy underestimated path length of **201.7**



3- Now S-A-B is the partial path to extend, as it is the partial path with the minimum fuzzy underestimated path length. This expansion leads to partial paths **S-A-B-C**, with a fuzzy underestimated path length of **119.7**, partial path **S-A-B-E**, with a fuzzy underestimated path length of **186.4**, and to the shortest complete path, **S-A-B-G**, with a total distance of **131**, but to be absolutely sure, all partial paths with partial path distances less than **131** must be expanded. There is no need to extend the partial path **S-A-B-E**, because its partial-path distance of **186.4** is more than that of the complete path.



4- Finally, **S-A-B-C** is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to a complete path, **S-A-B-C-G**, with a total distance of **121**. No partial path has a lower-bound distance so low, so no further search is required.

Figure 3.16: Example for Branch and Bound Search augmented by fuzzy underestimates

Figure (3.17) explains the algorithm of Branch-and-Bound search augmented by fuzzy underestimate, according to the previous example:

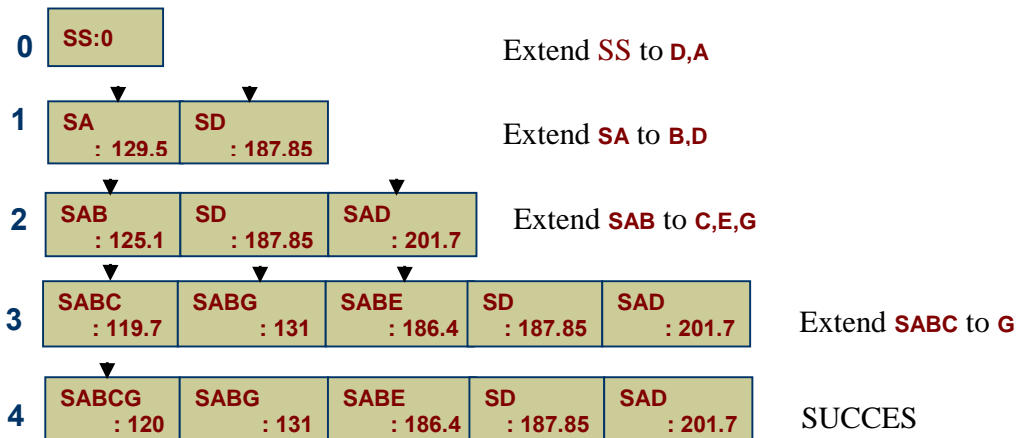
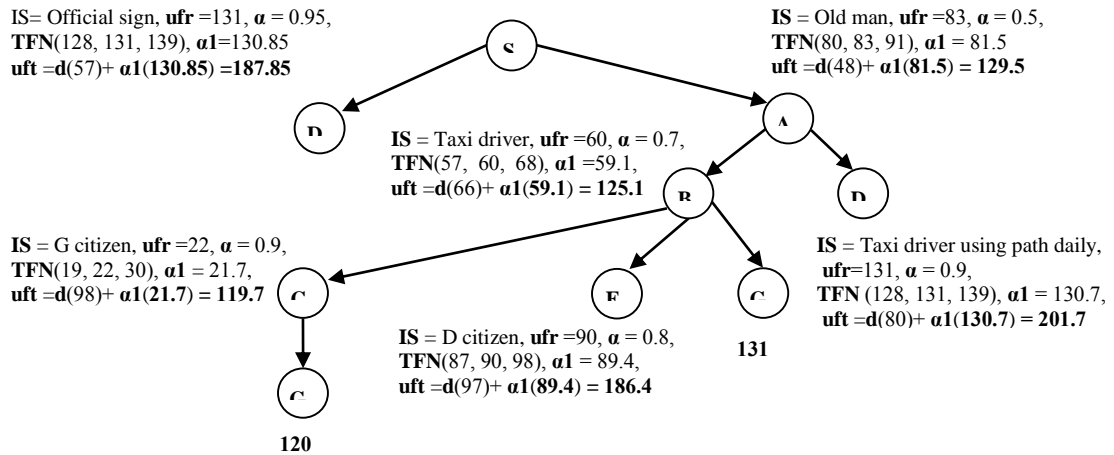
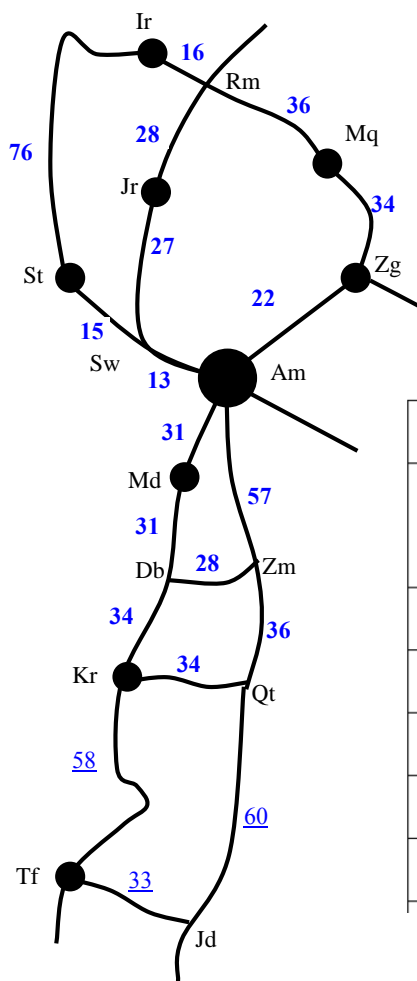


Figure 3.17: Branch and Bound Search augmented by fuzzy underestimate/ algorithm explanation

### 3.4.2 Roads between Two Jordanian Cities Application

In the following example, we will adopt a real life example; i.e. real roads between two major Jordanian cities; say from **Al Karak** to **Irbid** according to an official map of Jordan, as shown in Figure (3.18), where one can plan a route from **Al Karak** as a start node (S) to **Irbid** as a goal node (G) in the following map.

Suppose that Mr. X is trying to find some path from **Al Karak** to **Irbid** using a highway map such as the one shown in Figure (3.18). The starting point in **Al Karak** denoted as **Kr**, which might be called (start node), and ending point in **Irbid** denoted as **Ir**, which might be called (goal node), other cities (nodes) are denoted as shown in the key table of Figure (3.18).



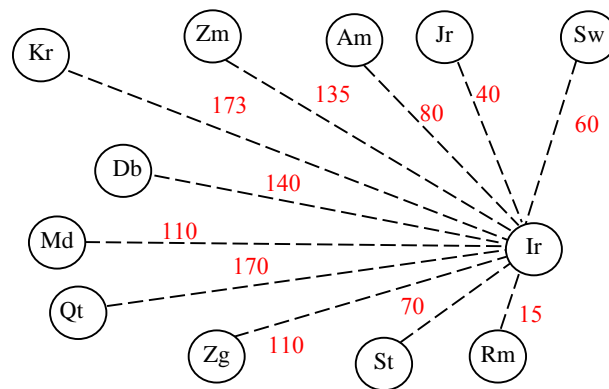
**Figure 3.18:** net graph from Karak to Irbid with actual distance of each route.  
Cities (nodes) are denoted as shown in the key table

**Key table**

City Name	Notation	City Name	Notation
Jurf Al		Seweleh	Sw
Daraweesh	Jd		
Qutraneh	Qt	Al Salt	St
Zmailih	Zm	Jarash,	Jr
Dhiban	Db	Ramtha	Rm
Madaba	Md	Mafraq	Mq
Aamman	Am	Zarga	Zg
		Azraq	Az

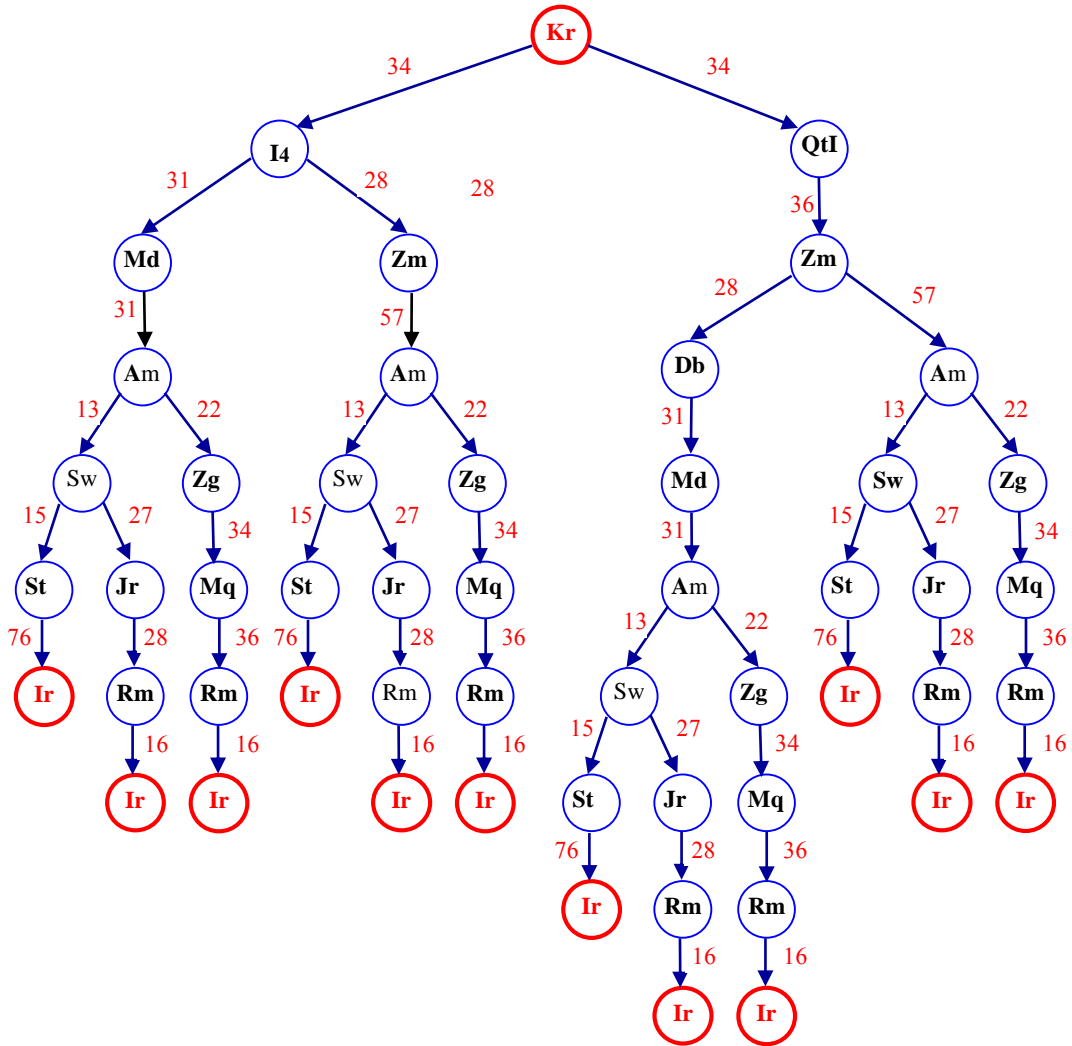
The traveler is at **Al Karak**, and intends to go to **Irbid**. All possible routs are shown in the net graph, the number against each edge gives the actual distance of that route (node to node) in **kilometers**. The traveler has no knowledge about the distance information, but the traveler records the distance he/she completed.

Figure (3.19) shows the **fuzzy estimates** of remaining distances from each city (or intersection node) to **Irbid** (the goal) , where the number against each edge gives that estimated distances in **kilometers** = (efr).



**Figure 3.19:** fuzzy estimates of remaining distances from each city (or intersection node) to Irbid =efr. Cities (nodes) are denoted as shown in the key table of figure (3.18).

With looping paths eliminated, one can arrange all possible paths from the start node (Kr) in a search tree. Figure (3.20) shows a search tree that consists of nodes denoting the possible paths that lead outward from the start node Al Karak (S) of the net shown in Figure (3.18), the number against each edge gives the actual distance of that route (node to node) in **kilometers**.



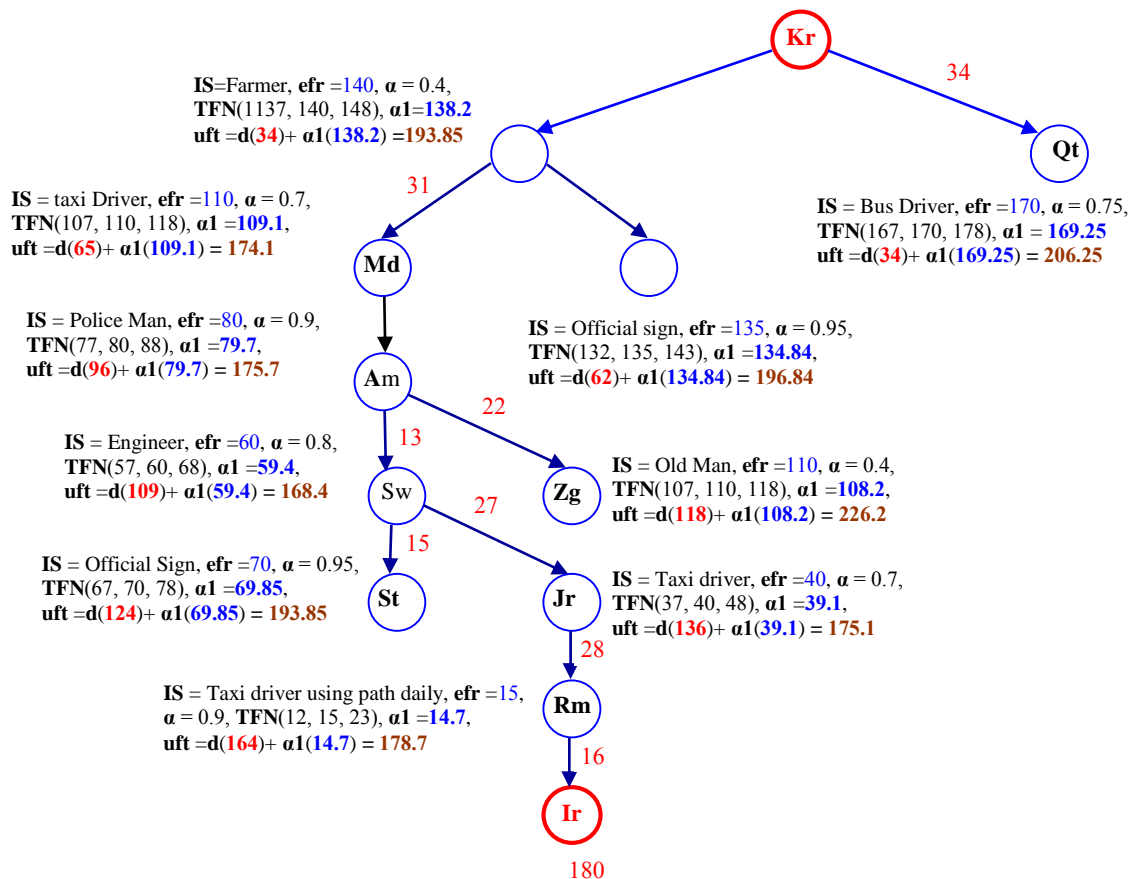
**Figure 3.20:** A search tree that consists of nodes denoting all possible paths those lead outward from the start node Al Karak (S) of the net shown in figure (3.18)

If a decision-maker takes a fixed TFN model slope as  $(a-3, a, a+8)$ , at each node he/she will take fuzzy estimate of remaining distance for new paths = efr = "approx. a" from different information sources = **IS**, choose  $\alpha$  according to degree of searcher confidence in estimation, take a fuzzy lower-bound estimate of the cost remaining;  $\alpha_1 = a_1 + \alpha * (a - a_1)$ , and finally he/she will add new paths, and sort the paths by the sum of the already traveled path length =  $d$  and the fuzzy lower-bound estimate of the remaining distance =  $ufr$  to choose the shortest path.

Figure (3.21) shows how fuzzy underestimates of distances remaining helps to make the search more efficient, where Branch-and-bound search augmented by fuzzy underestimates determines that the path **Kr- I<sub>4</sub>- Md- Am- Sw- Jr- R<sub>m</sub>- Ir** is optimal. The numbers beside the nodes are underestimate of total path length (uft) = accumulated distances(d) + fuzzy lower-bound estimate of the cost remaining ( $\alpha 1$ ).

Fuzzy underestimates quickly push up the lengths associated with bad paths. In this example, fewer nodes are expanded than would be expanded with branch-and-bound search operating with crisp underestimates.

Part of the tree, which must be explored by the proposed algorithm will be as shown in figure (3.21):-

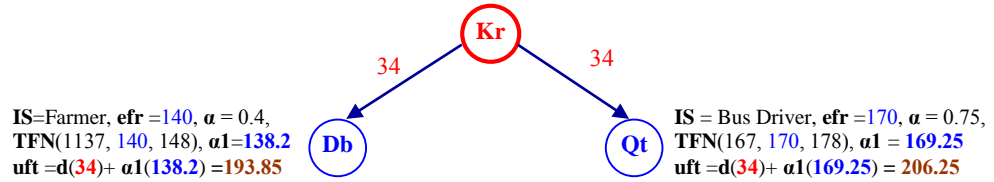


**Figure 3.21:** The explored part of the tree. At each node there is, specific source for estimated information =IS, who (which) will give fuzzy cost estimate of remaining distance = “approx. a” =  $\alpha$ , decision maker will choose an appropriate  $\alpha$ , TFN model, & lower  $\alpha$ -cut for that node.



The problem can be solved by applying the proposed algorithm of section 4.2 as in the following example:-

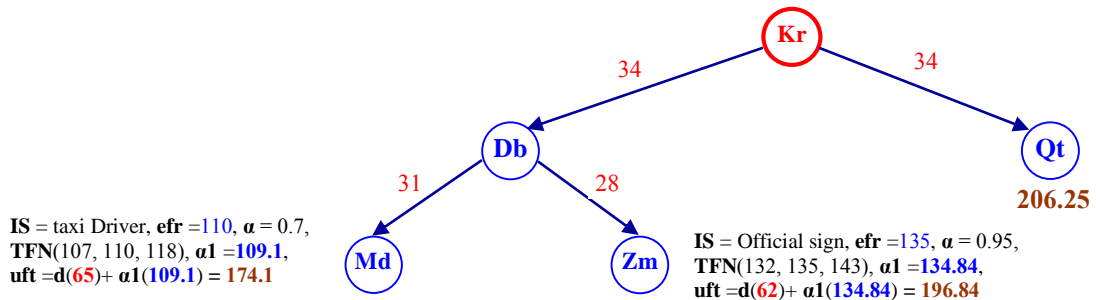
**Figure 3.22:** Example for Branch and Bound Search augmented by fuzzy underestimates



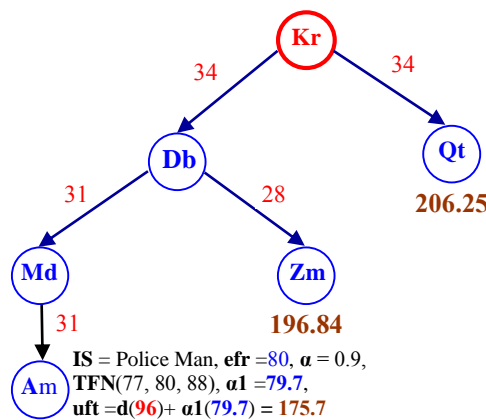
1- In the first step, as before, **Db** and **Qt** are generated from **Kr(S)**, at node **Db** if the information source (**IS**) was a **Farmer**, who gives a fuzzy estimate of remaining distance as **a** = "approx. 140", **decision maker can choose TFN** as (137, 140, 148), and  $\alpha=0.4$  to produce  $\alpha1=138.2$  which will be added to the distance already traveled  $d = 34$ , to produce the fuzzy underestimate of total path length  $uft = 193.85$ .

While at node **Qt** information source about remaining distance (**IS**) was a **Bus Driver** with **a** = "approx. 170", **decision maker can choose TFN** as (167, 170, 178), and  $\alpha = 0.75$  to produce  $\alpha1=169.25$  which will be add to the distance already traveled  $d = 34$ , to produce the fuzzy underestimate of total path length  $uft = 206.25$ .

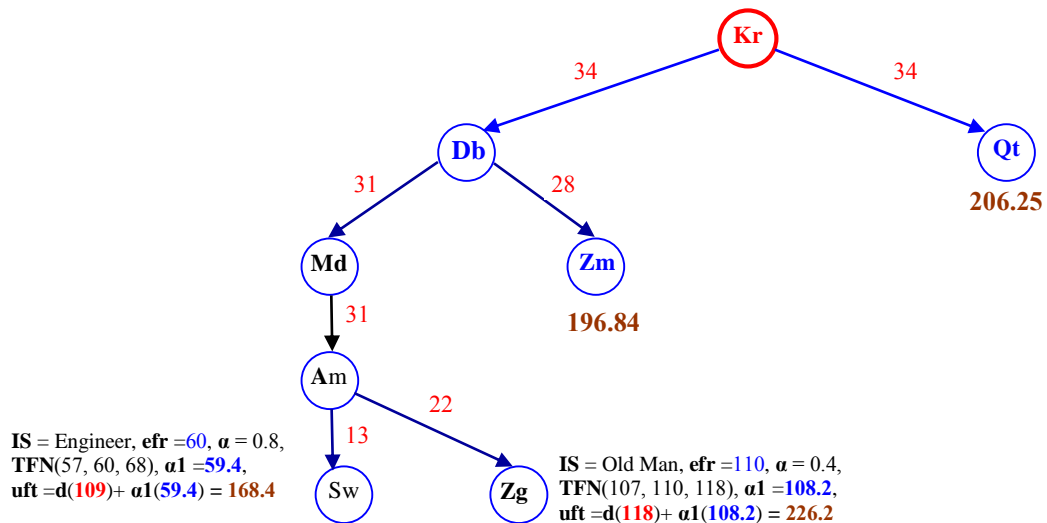
**Db** is the node from which to search, because **Db**'s fuzzy underestimated path length is **193.85**, which is shorter than that for **Qt**, **206.25**.



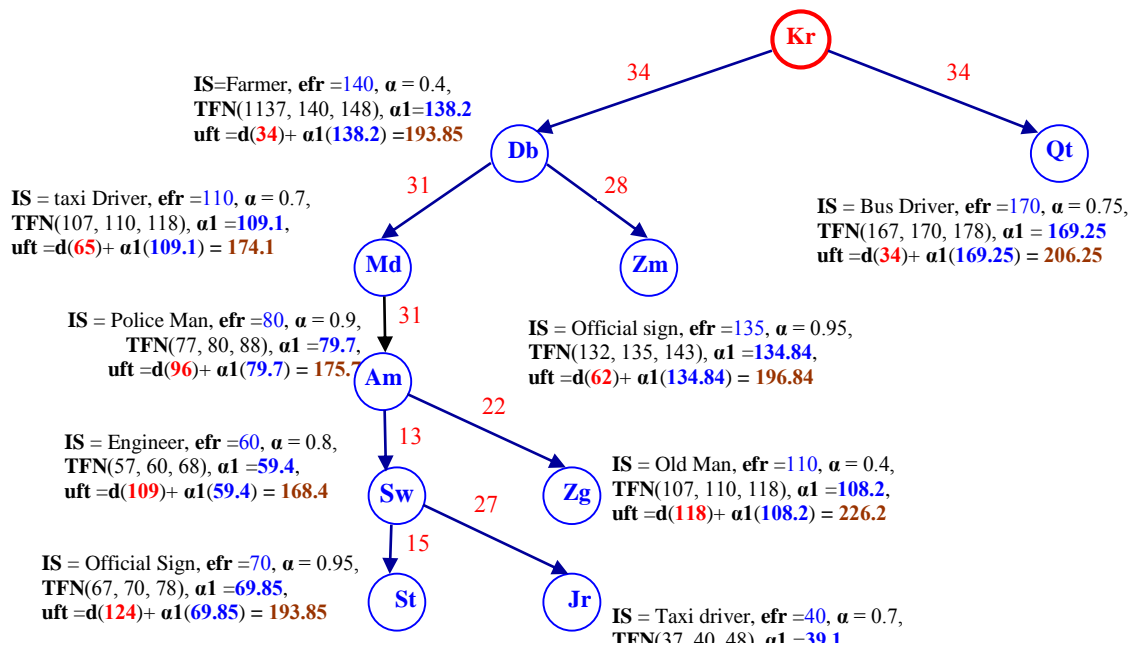
2-Expanding **Db** eads to partial paths **Kr- Db - Md**, with a fuzzy underestimated path length of **174.1**, and to partial path **Kr- Db - Zm**, with a fuzzy underestimated path length of **196.84**



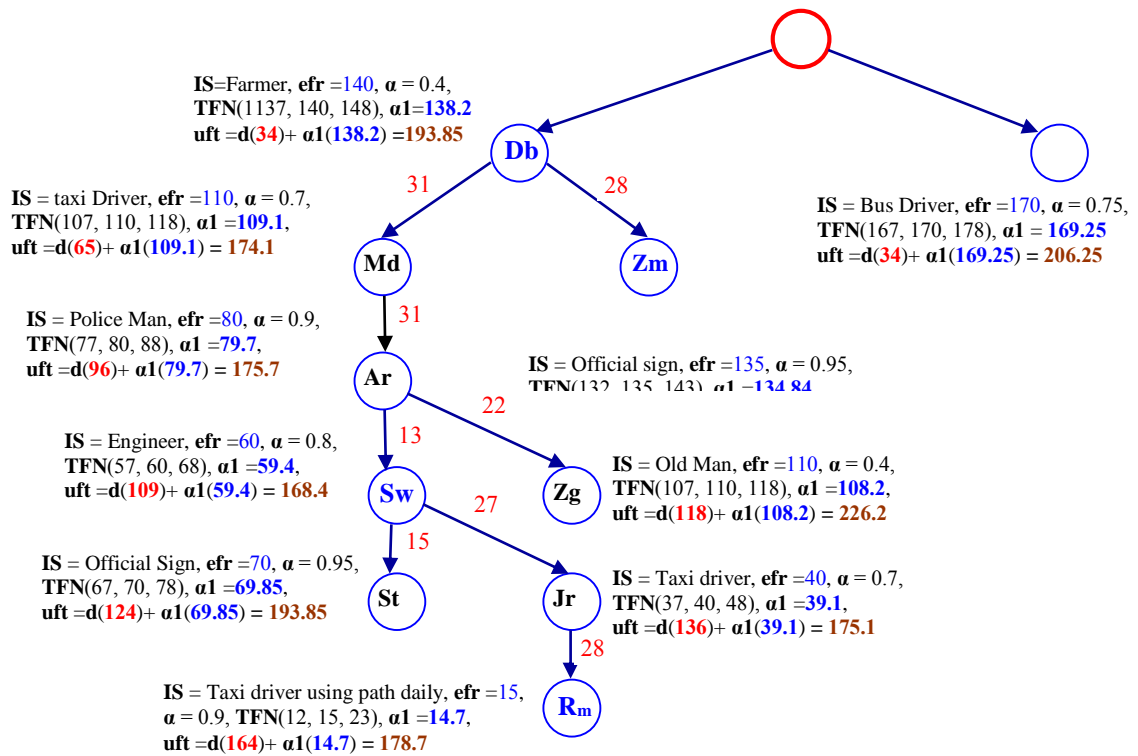
3-Now **Kr- Db - Md** is the partial path to extend, as it is the partial path with the minimum fuzzy underestimated path length. This expansion leads to partial paths **Kr- Db - Md- Am** with a fuzzy underestimated path length of **175.7**.



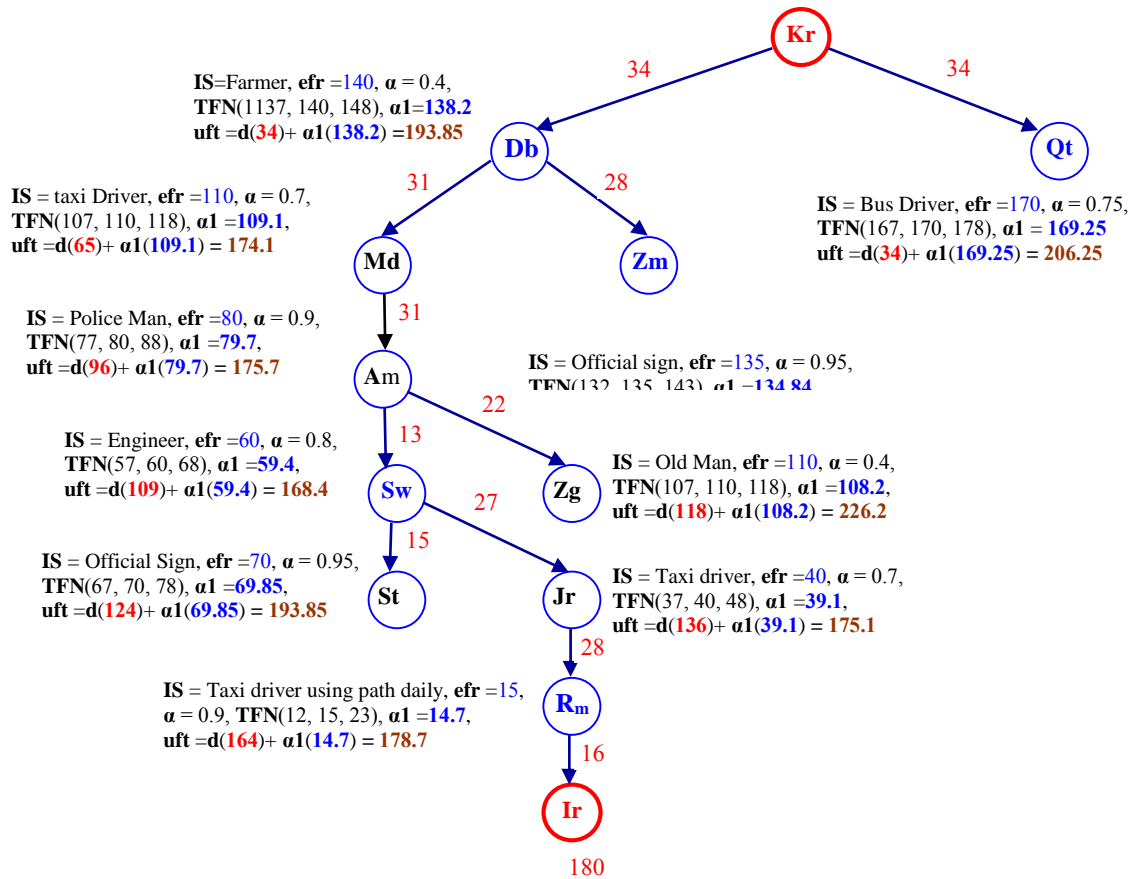
4- Now **Kr- Db - Md- Am** is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to partial paths **Kr- Db - Md- Am- Sw**, with a fuzzy underestimated path length of **168.4**, and to partial **Kr- Db - Md- Am- Zg**, with a fuzzy underestimated path length of **226.2**.  
**Kr- Db - Md- Am- Sw** is the partial path to extend, as it is the partial path with the minimum underestimated path length.



5-Expanding **Sw** leads to partial paths **Kr- Db- Md- Am- Sw- St**, with a fuzzy underestimated path length of **193.85**, and to partial path **Kr- Db - Md- Am- Sw-Jr**, with a fuzzy underestimated path length of **175.1**



6- Now **Kr- Db - Md- Am- Sw- Jr** is the partial path to extend, because it is the partial path with the minimum underestimated path length. This expansion leads to partial path **Kr- Db - Md- Am - Sw -Jr- R<sub>m</sub>**, with a fuzzy underestimated path length of **178.7**.



7- Finally, **Kr- Db - Md- Am- Sw- Jr- R<sub>m</sub>** is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to a complete path, **Kr- Db - Md- Am- Sw- Jr- R<sub>m</sub>- Ir**, with a total distance of 180. No partial path has a lower-bound distance so low, so no further search is required.

Figure (3.23) explains the algorithm of Branch-and-Bound search augmented by fuzzy underestimate, according to the previous example.

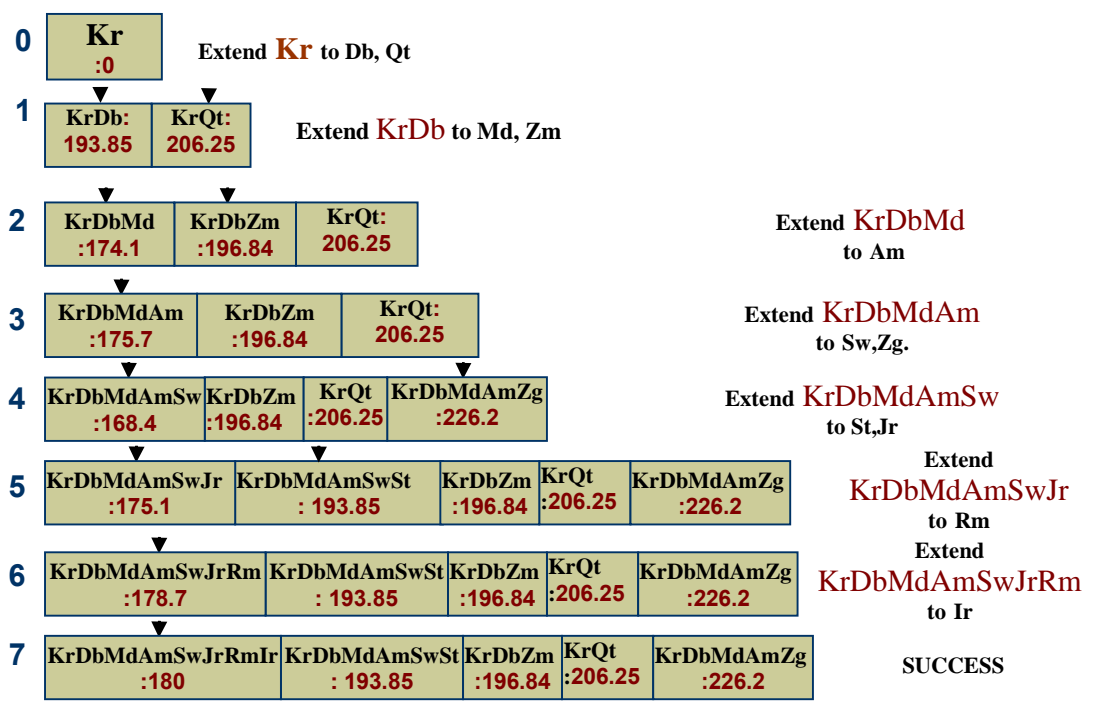
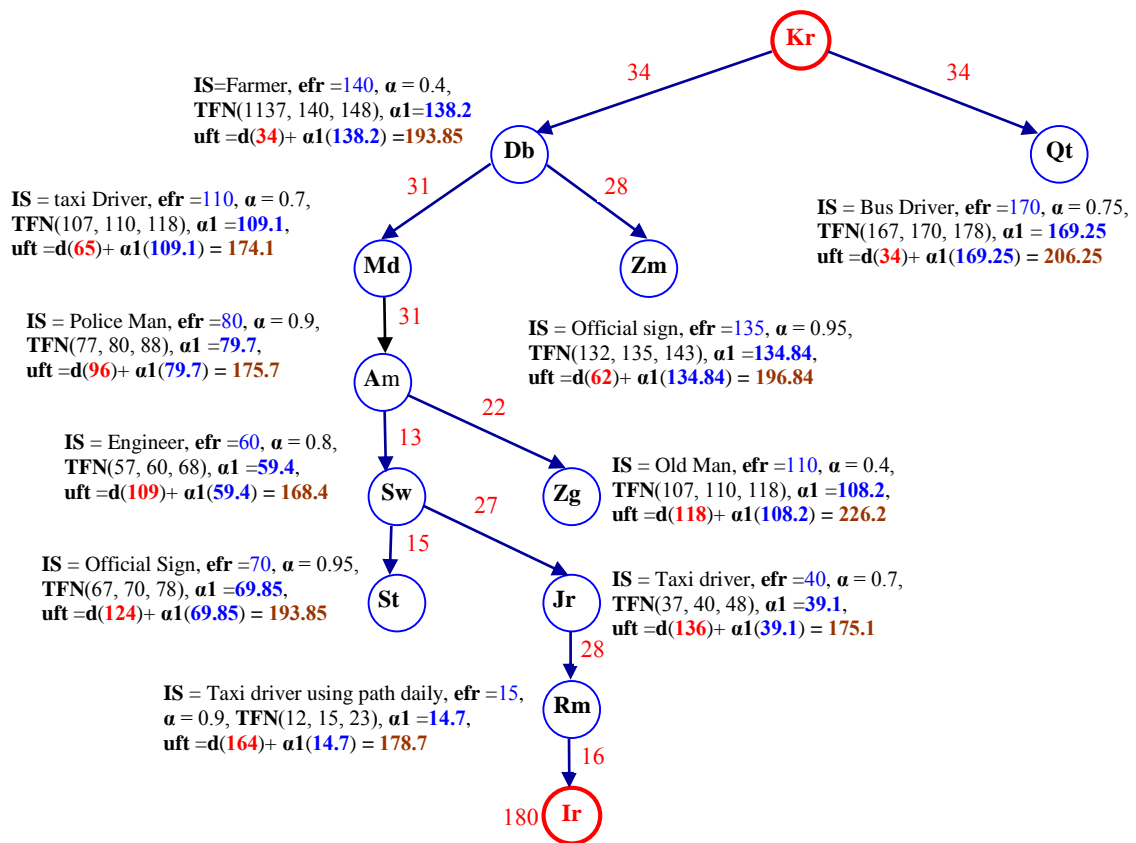


Figure 3.23: Branch and Bound Search augmented by fuzzy underestimate/ algorithm explanation

### 3.5 Conclusion

In this chapter a new type of Branch and Bound searching technique using fuzzy underestimates is proposed. The objective of this work is to deal with the imprecise data involved in different kind of existing searching techniques in a more efficient way.

In general Branch-and-Bound search is suitable when the tree is big and bad paths turn distinctly bad quickly.

Branch-and-Bound search with a guess is suitable when there is a good lower-bound estimate of the distance remaining to the goal, where underestimates quickly push up the lengths associated with bad paths. In Figure (3.6), many fewer nodes are expanded (10 nodes) than which were expanded with branch-and-bound search operating without underestimates (17 nodes) as shown in Figure (3.2) for the same problem.

In choosing heuristics, we usually consider the heuristic that reduces the number of nodes that need to be examined in the search tree. This can be viewed as the reduction of effective branching of search.

In analyzing search methods, it is important to examine the Effective Branching Factor ( $b^*$ ) of each method because it is a suitable way to characterize the quality of an heuristic, where a well - designed heuristic would have a value of  $b^*$  close to 1, allowing fairly large problem to be solved (Russell& Norvig, 2003; Luger, 2005).

When evaluating B&B heuristic search strategies which are discussed in this chapter in term of EBF ( $b^*$ ) according to the results shown in Table (3.1) and the corresponding chart shown in Figure (3.24), we can note that:

- B&B with crisp underestimate search technique achieve better efficiency than regular B&B search technique, where  $b^*$  was decreased from 1.67 to 1.4 because underestimation increases the efficiency of B&B by enabling it to be more informed.
- Adding Fuzzy underestimate to B&B search technique achieve better efficiency than adding crisp underestimate to B&B search technique, where  $b^*$  was decreased from 1.4 to 1.35 (Figure 3.16) and to 1.13 (Figure 3.21) because fuzzy underestimation increases the efficiency of B&B with crisp underestimate by enabling it to be more informed.
- Some Fuzzy underestimates can achieve better efficiency than other Fuzzy underestimates where the more informed Fuzzy underestimates can achieve the better efficiency. Fuzzy underestimation achieved  $b^* = 1.35$  in case of Figure (3.16) but it achieved a better  $b^* = 1.13$  in the case of Figure (3.21). In general adding Fuzzy underestimates can achieve better efficiency than adding crisp underestimates, where Effective Branching Factor for Fuzzy Underestimated B&B are always better (less) than that for crisp algorithms especially when the number of nodes is high. Obviously the closer the Fuzzy underestimate is to the true remaining solution cost the more efficient the B&B search will be.
- B&B Augmented by Fuzzy Underestimate search technique is complete, optimal, nonredundant, and more informed than other algorithms.

Figure No.	d	N	b*	Type of Search
3.2	4	17	1.673	Crisp B&B
3.6	4	10	1.403	B&B with Crisp Underestimation
3.16	4	9	1.352	B&B with Fuzzy Underestimation ( Application 1)
3.21	7	12	1.135	B&B with Fuzzy Underestimation ( Application 2)

**Table 3.1:** Evaluation of Heuristic B&B search strategies in terms of effective branching factor ( $b^*$ ).

d: is the depth of the solution,

N: is the total number of nodes generated by each strategy for a particular problem.

A well - designed heuristic would have a value of  $b^*$  close to 1, allowing fairly large problem to be solved .

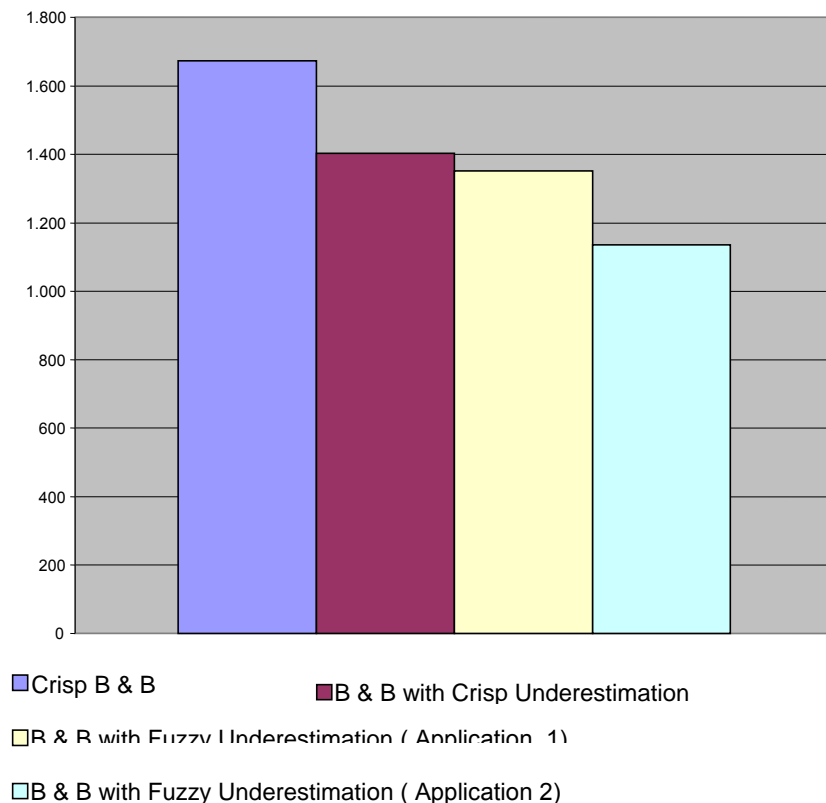


Figure 3.24: Evaluation of Heuristic B&B search strategies in terms of effective branching factor ( $b^*$ ) according to the results shown in table (3.1).



# Chapter four

## searching technique using

### Fuzzy underestimates

#### 4.1 Introduction

A\* algorithm was first presented by Hart *et al* (Hart, *et al* 1968; 1972). (Rich & Knight,2000). A\* is based on one further modification to the Branch and Bound technique augmented by underestimates which again improves search efficiency, by recognizing and removing redundant paths from the search queue according to the Dynamic Programming Principle (Winston, 2000).

A\* has attracted a great deal of attention. It is one of the fundamental algorithms of artificial intelligence. A\* is the name given to the algorithm where the  $e(\text{node})$  function is admissible. In other words, it is guaranteed to provide an underestimate of the true cost to the goal. A\* is optimal and complete. In other words, it is guaranteed to find a solution, and that solution is guaranteed to be the best solution (Coppin, 2004).

A\* may use all the available memory in a matter of minutes. After that the search practically cannot proceed although the user would find it acceptable that the algorithm would run for hours or even days (Bratko, 1998).

In this work a new type of A\* searching technique using fuzzy logic is proposed. The objective of this work is to deal with the imprecise data involved in existing A\* searching techniques in a more efficient way and thus to suggest improved version of A\* searching techniques under uncertainty which will be

helpful in many real life problems of computer science, especially in AI field.

In this chapter we consider a search problem and its solution by the existing crisp method of A\* search. We proved that this method can be enhanced using fuzzy theory. Consequently we proposed a new method of A\* with fuzzy underestimates, introducing the corresponding algorithm, and explaining the algorithm by two applications (examples).

We will explain in details **A\* Search (Crisp Method); by explaining A\* procedure, providing crisp A\* algorithm, and crisp branch-and-bound procedure with dynamic programming algorithm giving : Search Examples, and Search algorithm explanations for both procedures.**

**Then** we are to explain in details **A\* Search (Fuzzy Method) by providing the suggested algorithm explanation giving : A\* Fuzzy Method algorithm, and flow chart explanation.**

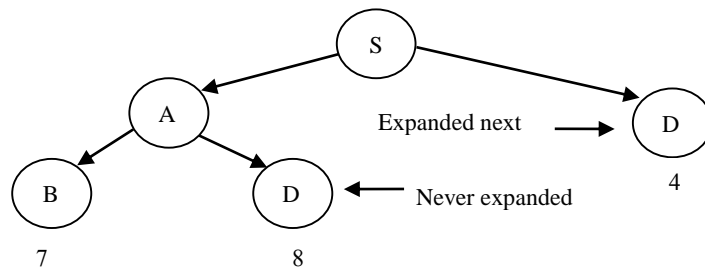
**Finally,** we are to explain in details two **applications (examples) for the suggested algorithm.**

## 4.2 Dynamic Programming

*Dynamic Programming (DP)* is sometimes called the *forward-backward* or, when using probabilities, the *Viterbi* algorithm. Created by Richard Bellman (1956), Dynamic programming addresses the issue of restricted memory search in problems composed of multiple interacting and interrelated subproblems (Luger, 2005).

Now let us consider a different approach to improve on basic branch-and-bound search. Look at Figure (4.1). The root node, S, has been expanded, producing partial paths S-A and S-D. For the moment, let us use no

underestimates for remaining path length.



**Figure 4.1: An illustration of the dynamic programming principle. The numbers beside the node are accumulated distances.**

Because S-A is shorter than S-D. S-A is extended first, leaving three paths: S-A-B, S-A-D, and S-D. Then, S-D will be extended, because it is the partial path with the shortest length.

But, what about the path S-A-D? Will it ever make sense to extend it? Clearly, it will not. Because there is one path to D with length 4, it cannot make sense to work with another path to D with length 8. The path S-A-D should be forgotten forever; it cannot produce a winner.

This example illustrates a general principle. Assume that the path from a starting point, S, to an intermediate point, I, does not influence the choice of paths for traveling from I to a goal point, G. Then the minimum distance from S to G through I is the sum of the minimum distance from S to I and the minimum distance from I to G. Consequently, the strangely named dynamic-programming principle holds that, when you look for the best path from S to G, you can ignore all paths from S to any intermediate node, I, other than the minimum-length path from S to I.

The **dynamic-programming principle** (some times called **Path Deletion**) can

be stated as:

The best **way through** a particular, intermediate place is the **best way to it** from the starting place, followed by the **best way from it** to the goal. There is no need to look at any other paths to or from the intermediate place (Winston, 2000; Doyle, Dec 5, 2005).

So:

**Minimum** distance from **S to G (through I)** = *min.* distance from **S to I**  
+ *min.* distance from **I to G**

**More precisely:**

IF the QUEUE contains:

a path P terminating in I, with cost = cost\_P

a path Q containing I, with cost = cost\_Q

cost\_P ≥ cost\_Q

THEN

delete P

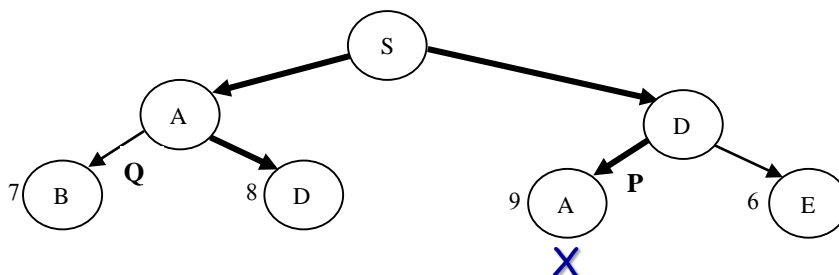


Figure 4.2 A more precisely illustration of the dynamic programming principle. The numbers beside the node are accumulated distances.

The branch-and-bound procedure, with dynamic programming included differs from the basic branch-and-bound procedure only in the steps shown in bold italic bellow:

---

To conduct a branch-and-bound search with dynamic programming,

- 1- Form a one-element queue consisting of a zero-length path that contains only the root node.

2- Until the first path in the queue terminates at the goal node or the queue is empty,

2.1- Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.

2.2- Reject all new paths with loops.

2.3- Add the remaining new paths, if any, to the queue.

2.4- ***If two or more paths reach a common node, delete those paths except the one that reaches the common node with the minimum cost.***

2.5- Sort the entire queue by path length with least-cost paths in front.

3- If the goal node is found, announce success; otherwise, announce failure.

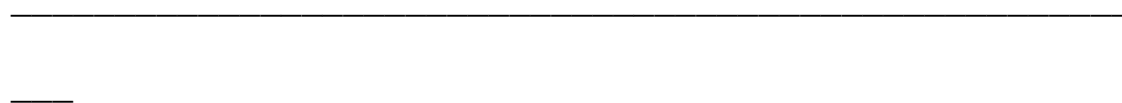
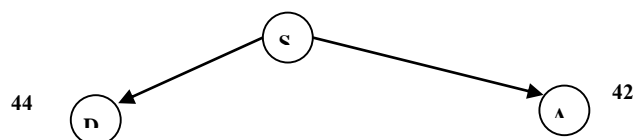
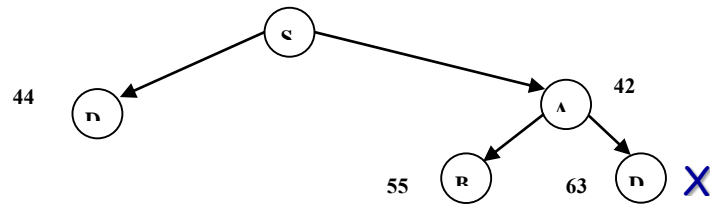


Figure 4.3 shows the effect of using the dynamic-programming principle, together with branch-and-bound search, on the map-traversal problem, where the numbers beside the nodes denotes the length of each path (cost). Four paths are cut off quickly, leaving only the dead-end path to node C and the optimal path, S-D-E-F-G.

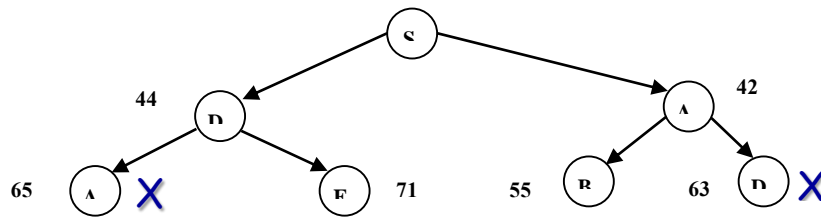
**Figure 4.3:** Branch and Bound with dynamic-programming principle Search Example.



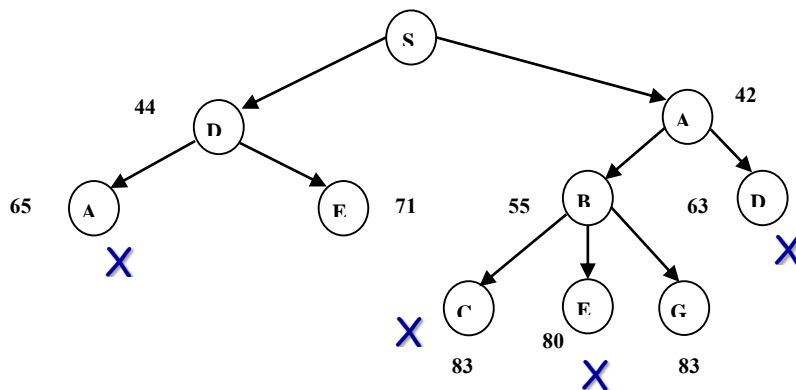
1- In the first step, the partial-path distance of S-A is found to be 42, and that of S-D is found to be 44; partial path S-A is therefore selected for expansion.



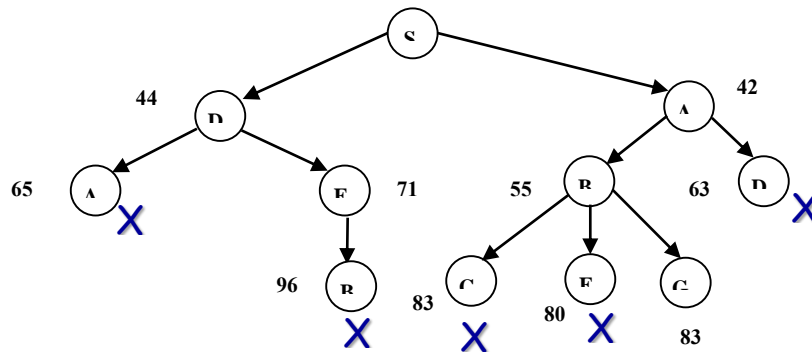
2- Next, S-A-B and S-A-D are generated from S-A with partial path distances of 55 and 63, partial path S-A-D can be deleted because its partial-path distance of 63 is more than that of S-D (44).



3- Now S-D, with a partial path distance of 44, is expanded, leading to partial paths S-D-A and S-D-E. At this point, partial path S-D-A can be deleted as it's partial-path distance of 65 is more than that of S-A-B(55), then there are only two partial paths, with the path S-A-B being the shortest with a partial path distance of 55.



4- Then expanding S-A-B, leads to S-A-B-C, S-A-B-E, and S-A-B-G with partial path distances of 83, 80, and 83, partial path S-A-B-E can be deleted it's partial-path distance of 80 is more than that of S-D-E(71), now S-A-B-G is the shortest complete path, but to be absolutely sure, all partial paths with partial path distances less than 83 must be expanded. Also there is no need to extend the partial path S-A-B-C, because its partial-path distance of 83 is equal to that of the complete path.

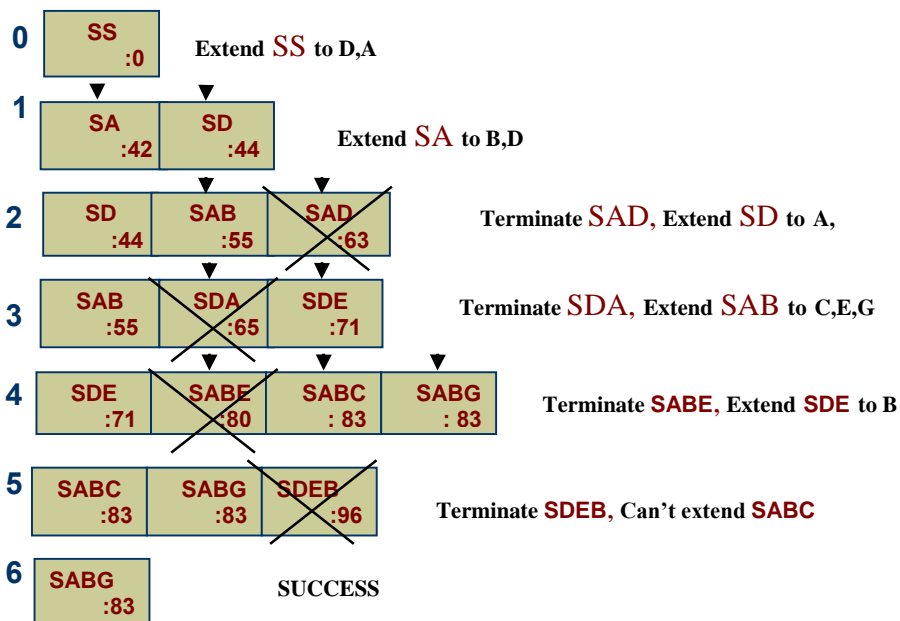
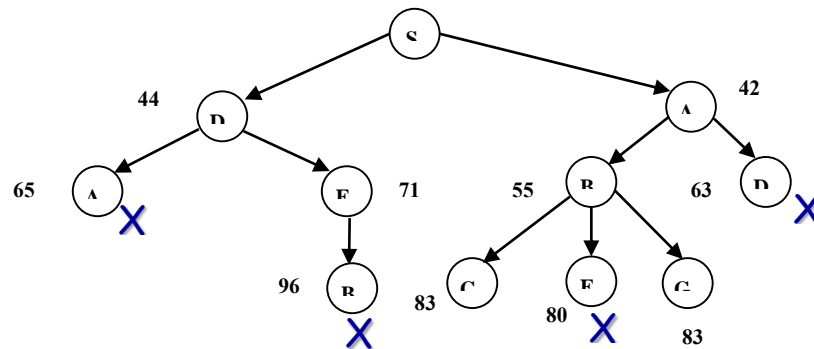


5- Now S-D-E-B is generated from S-D-E with partial path distances 96, partial path S-D-E-B can be deleted it's partial-path distance of 96 is more than that of the shortest complete path S-A-B-G.

In this particular example little work is avoided relative to branch and bound, where number of steps has been reduced to 5 steps instead of 9.

Figure (4.4) explains the algorithm of branch-and-bound search with dynamic programming according to the previous example shown in figure (4.3), where the numbers beside the nodes are the length of each path:

**Figure 4.4:** branch-and-bound search with dynamic programming algorithm explanation.



#### 4.2.1 A\* search

The A\* procedure is Branch-and-Bound search, with an Underestimate of remaining distance, combined with the Dynamic Programming principle. If the estimate of remaining distance is a lower-bound on the actual distance, then A\* produces optimal solutions generally, the estimate may be assumed to be lower-bound estimate, unless specifically stated otherwise, implying that A\* solutions are normally optimal.



The A\* algorithm operates in the same manner as Branch-and-Bound search augmented by underestimate, so it employs the following function to calculate underestimate of total path length,  $u_t = (\text{total path length})$ :

$$u(\text{total path length}) = d(\text{already traveled}) + u(\text{distance remaining}),$$

where  $d = d(\text{already traveled})$  is the known distance already traveled, and where  $u = u(\text{distance remaining})$  is underestimate of the distance remaining (Coppin, 2004).

The A\* procedure differs from the basic branch-and-bound with a lower-bound estimate procedure only in the steps shown in bold italic bellow:

---

To conduct A\* search,

- 1- Form a one-element queue consisting of a zero-length path that contains only the root node.
- 2- Until the first path in the queue terminates at the goal node or the queue is empty,
  - 2.1- Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
  - 2.2- Reject all new paths with loops.
  - 2.3- Add the remaining new paths, if any, to the queue.
  - 2.4- ***If two or more paths reach a common node, delete those paths except the one that reaches the common node with the minimum cost.***

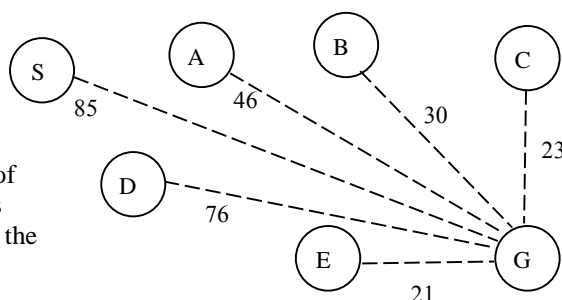
2.5- Sort the entire queue by the sum of the path length and a lower-bound estimate of the cost remaining, with least-cost paths in front.

3- If the goal node is found, announce success; otherwise, announce failure.

---

Now, if the straight-line distances from each city to the goal is as shown in Figure (4.5) are considered as an underestimates of distances remaining (**ur**), and the already traveled distances to each city from the source = (**d**) is considered as shown in Figure (4.6), then **Figure (4.7)** will show how A\* will determine that the path S-A-B-C-G is optimal. The numbers beside the nodes are the underestimate of total path length (**ut**).

(**ut**)= accumulated distances(**d**) + underestimates of distances remaining(**ur**).



**Figure 4.5:** Example of straight-line distances between each city and the goal = (**ur**).

Figure 4.6: Example of already traveled distances at each city = (d).

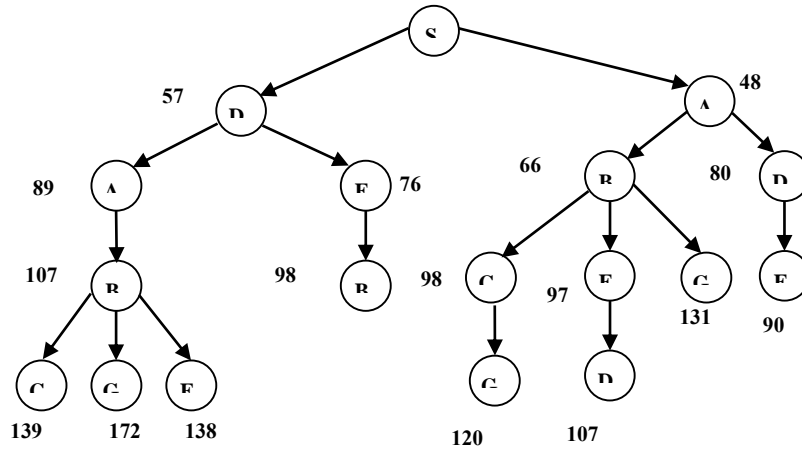
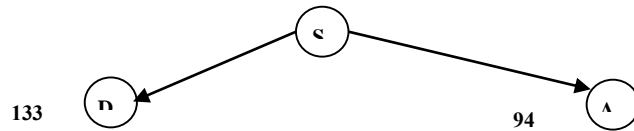
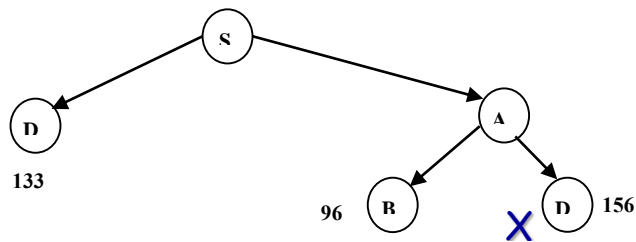


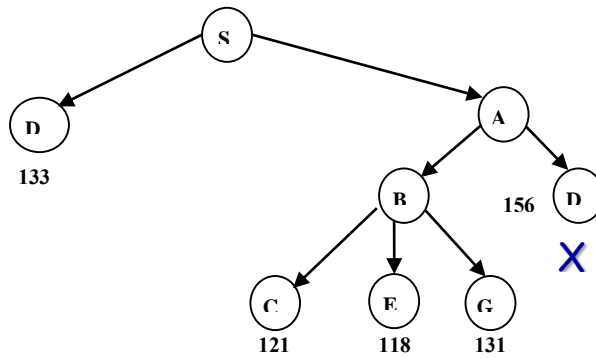
Figure 4.7: A\* Search Example.



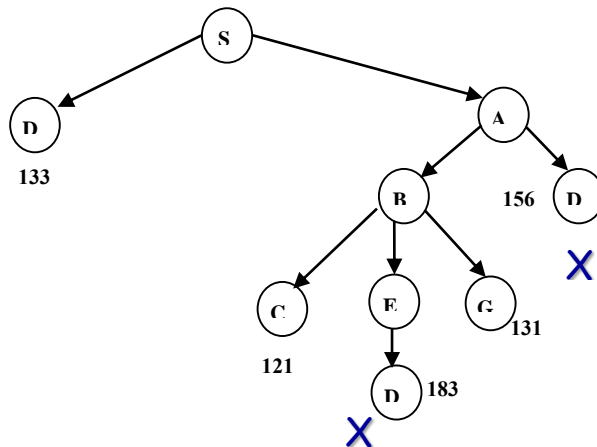
1- In the **first** step, D and A are generated from S, at node A,  $A_{ut} = d(48) + ur(46) = 94$ , at node D,  $D_{ut} = d(57) + ur(76) = 133$ , A is the node from which to search, because A underestimated path length is 94, which is shorter than that for D, 133.



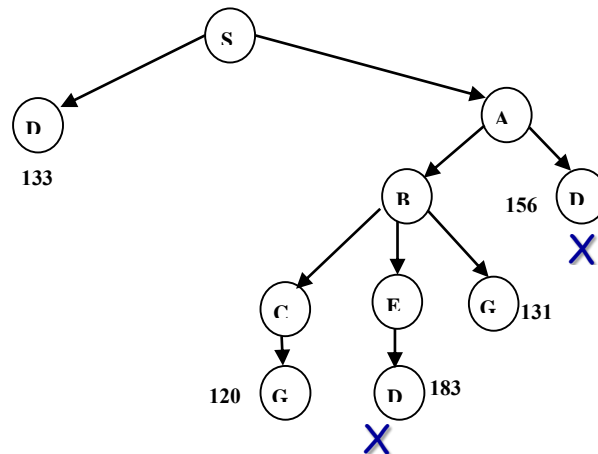
2- Expanding A leads to partial paths S-A-B, with an underestimated path length  $=d(66)+ur(30)=96$ , and to partial path S-A-D, with a underestimated path length  $=d(80)+ur(76)=156$ , partial path S-A-D can be deleted as its partial-path distance of 156 is more than that of. S-D (133).



3- Now S-A-B is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to partial paths S-A-B-C, with an underestimated path length  $=d(98)+ur(23)= 121$ , S-A-B-E with an underestimated path length  $=d(97)+ur(21)= 118$ , and to partial path S-A-B-G, with a underestimated path length  $=d(131)=131$ , where S-A-B-G is the shortest complete path, but to be absolutely sure, all partial paths with partial path distances less than 131 must be expanded.



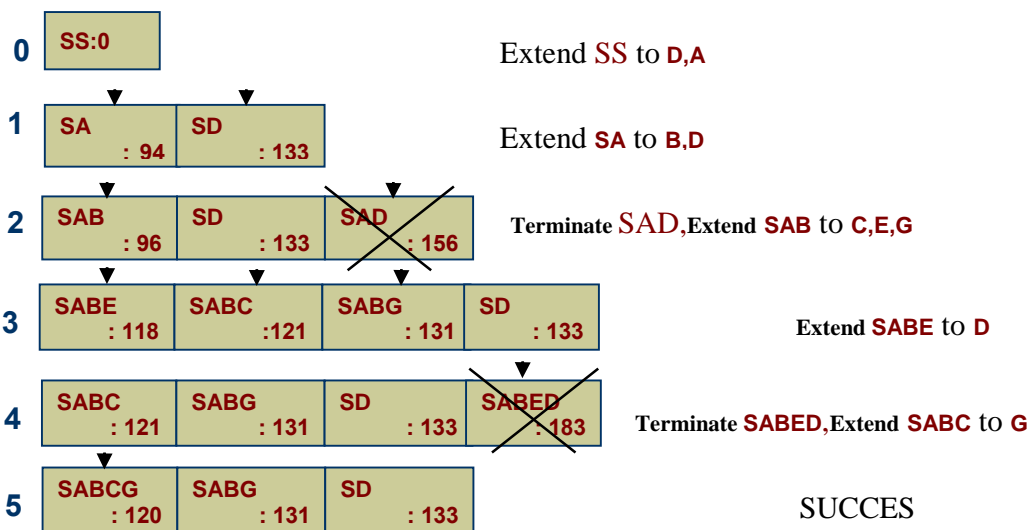
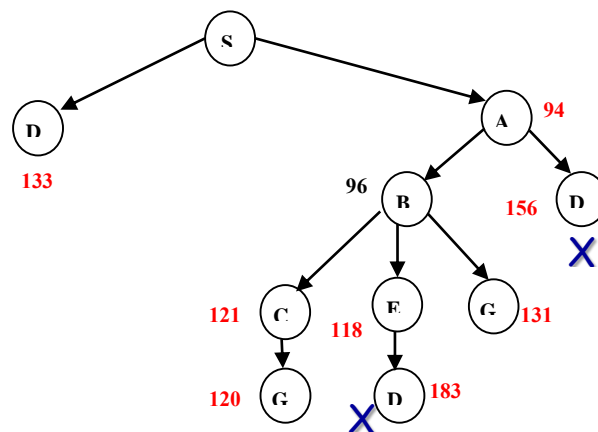
4- Now S-A-B-E is the partial path to extend, as it is the partial path with the minimum underestimated path length  $=118$ . This expansion leads to partial path S-A-B-E-D, with an underestimated path length  $=d(107)+ur(76)= 183$ , partial path S-A-B-E-D can be deleted because its partial-path distance of 183 is more than that of. S-D (133).



5- Finally **S-A-B-C** is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to a complete path, **S-A-B-C-G**, with a total distance of **120**. No partial path has a lower-bound distance so low, so no further search is required.

Figure (4.8) will explain the algorithm of A\* search, according to the previous

example:



**Figure 4.8:** A\* Search / algorithm explanation.

### 4.3 Search : Fuzzy Method

The suggested “A\* Searching Technique Using Fuzzy Underestimate” is the same as the existing “A\* Searching Technique Using Crisp Underestimate” in all its steps, unless the suggested method deals with underestimation of the remaining distance (**Ur**) as a fuzzy data , taking into consideration the assumption that “the underlying graph is crisp and the parameters related to its arcs are fuzzy numbers.” (Blue *et al*,2002).

A fuzzy underestimate of the distance remaining yields a fuzzy underestimate of total path length, **uft** (total path length):

$$\mathbf{uft \text{ (total path length)} = d \text{ (already traveled)} + ufr \text{ (distance remaining)},}$$

Were **d** (already traveled) is the known distance already traveled, and **ufr**(distance remaining) is a fuzzy underestimate of the distance remaining.

Fuzzy underestimation for the remaining distance (fuzzy data) can be processed according to the previous steps, as shown in Fgure (3.8), and as detailed in section (3.2.1).

The suggested procedures of A\* search with a fuzzy lower-bound estimate are different from the basic A\* search procedures only in the steps shown in italic bellow:

---

To conduct A\* search with a *fuzzy* lower-bound estimate,

- 1- Form a one-element queue consisting of a zero-length path that contains only the root node.
- 2- Until the first path in the queue terminates at the goal node or the queue is empty,
  - 2.1- Remove the first path from the queue; create new paths by extending the first path to all the neighbors of the terminal node.
  - 2.2- Reject all new paths with loops.
  - 2.3- ***Determine fuzzy lower-bound estimate of the cost remaining as follows:***
    - 2.3.1- ***Take fuzzy estimate of new paths = "approx. a".***
    - 2.3.2- ***Denote TFN for "approx. a" by the appropriate notation (a<sub>1</sub>, a, a<sub>2</sub>).***
    - 2.3.3- ***Choose a fixed "decision - parameter"  $\alpha$  in (0,1), according to degree of searcher confidence in estimation.***
    - 2.3.4- ***Take a fuzzy lower-bound estimate of the cost remaining (lower  $\alpha$ -cut of "approx. a") =  $\alpha_1$ , where:***
$$\alpha_1 = a_1 + \alpha * (a - a_1)$$
  - 2.4- Add the remaining new paths, if any, to the queue.

- 2.5- If two or more paths reach a common node, delete those paths except the one that reaches the common node with the minimum cost.
- 2.6- Sort the entire queue by ***the sum of the path length and a lower-bound estimate of the cost remaining***, with least-cost paths in front.
- 3- If the goal node is found, announce success; otherwise, announce failure.



The following flow chart also explain the procedure of A\* search with a fuzzy lower-bound estimate as shown in Figure (4.9):

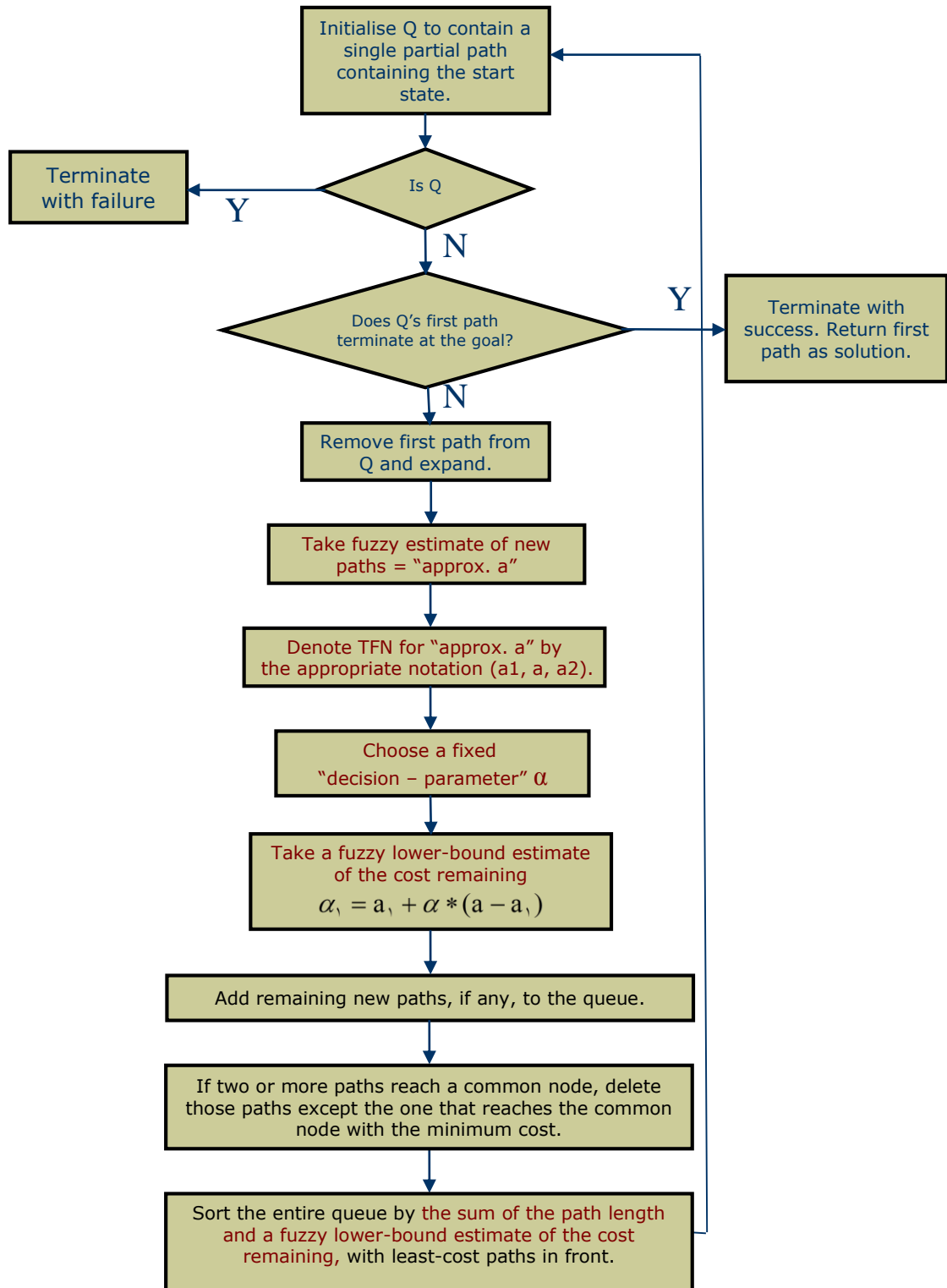


Figure 4.9: flow chart procedure of A\* search with a fuzzy lower-bound estimate.

When a searcher is working out a path on a highway map, straight-line distance is guaranteed to be an underestimate, but if searcher has better source of information about the remaining distance estimation , then search procedure will be more efficient.

Confidence about underestimation of remaining distance may vary from case to case; so  $\alpha$  also will vary according to degree of searcher confidence in value of underestimation.

## 4.4 Applications

We will consider two applications as examples for the proposed A\* with fuzzy underestimate algorithm.

The first application will adopt the previous example which was explained for branch and bound augmented by crisp underestimate algorithm (Figure 4.6) as a random net application which will be explained in section 4.4.1.

The second application will adopt the real roads between two major Jordanian cities as an example, which will be explained in section 4.4.2.

### 4.4.1 Random net Application

In the following example, if a decision-maker takes a fixed TFN model slope as  $(a-3, a, a+8)$ , at each node he/she will take fuzzy estimate of remaining distance for new paths = "approx. a" from different information source = IS, choosing  $\alpha$  according to the degree of searcher confidence in underestimation, taking a fuzzy lower-bound estimate of the cost remaining;  $\alpha_1 = a_1 + \alpha * (a - a_1)$ , and finally he/she will add new paths, and sort the paths by the sum of the already traveled path length =  $d$ , and a fuzzy lower-bound estimate of the remaining

distance = ufr to choose the shortest path.

Consider the following net, as shown in Figure (4.10). The traveler is at the node S, and intends to go to the node G. All possible routes are shown in the net graph, the number against each edge gives the actual distance of that route (node to node) in some unit. The traveler has no knowledge about the distance information, but the traveler records the distance he completed.

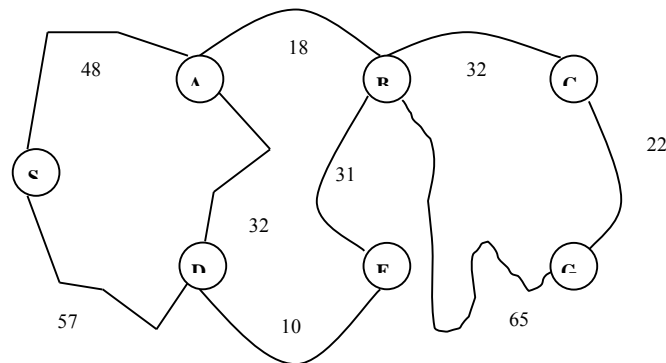


Figure 4.10: net graph with actual distance of each route.

Figure 4.11 shows the fuzzy estimates of distances remaining (ufr) from each city to the goal; Figure 4.13 shows how fuzzy underestimates of distances remaining helps to make the search more efficient, where A\* search augmented by fuzzy underestimates determines that the path S-A-B-C-G is optimal. The numbers beside the nodes are **underestimate of total path length (uft)**.

$(uft) = \text{accumulated distances}(d) + \text{fuzzy lower-bound estimate of the cost remaining} (\alpha_1)$ .

Fuzzy underestimates quickly push up the lengths associated with bad paths. In this example, fewer nodes are expanded than would be expanded with A\* search operating with crisp underestimates.

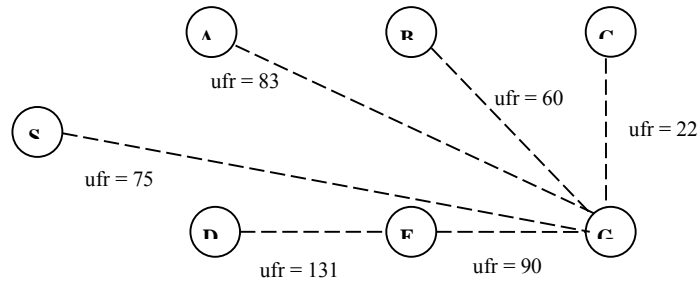


Figure: 4.11 Example of fuzzy underestimates of distances remaining (ufr) between each city and the goal.

Part of the tree, which must be explored by the proposed algorithm, will be as shown in Figure (4.12):-

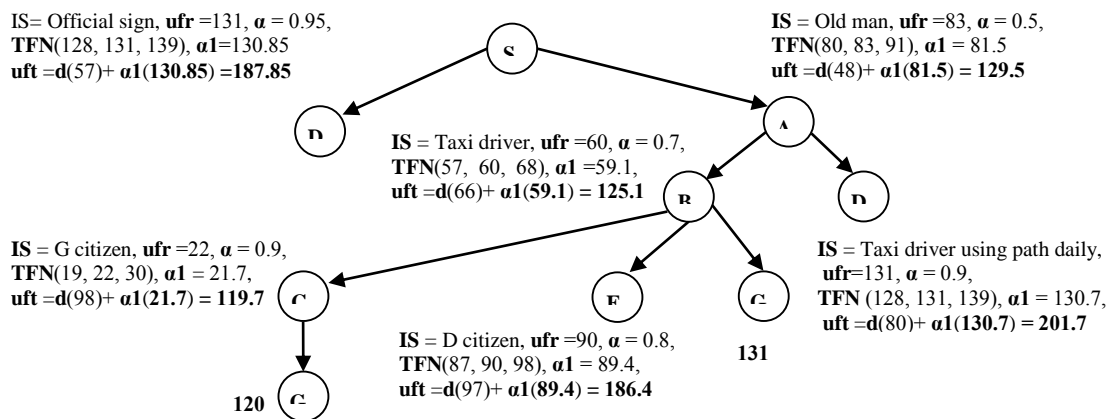
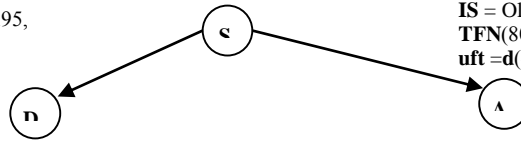


Figure 4.12: The explored Part of the tree. At each node there is, specific source for estimated information =IS, who (which) will give fuzzy cost underestimate of remaining distance = "approx. a". a decision maker will choose an appropriate  $\alpha$ , TFN model, and lower  $\alpha$ -cut for that node.

The problem can be solved by applying the proposed algorithm of section 4.2 as in the following example; Figure 4.13:-

Figure 4.13: Example for A\* Search augmented by fuzzy underestimates.

IS= Official sign,  $ufr=131, \alpha=0.95,$   
 $TFN(128, 131, 139), \alpha1=130.85$   
 $uft =d(57)+ \alpha1(130.85) =187.85$

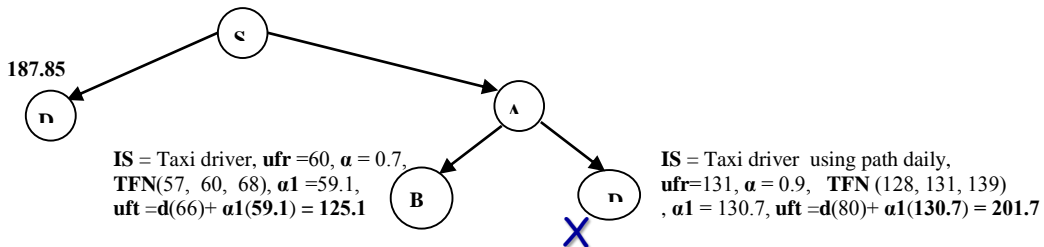


IS = Old man,  $ufr=83, \alpha = 0.5,$   
 $TFN(80, 83, 91), \alpha1 = 81.5$   
 $uft =d(48)+ \alpha1(81.5) = 129.5$

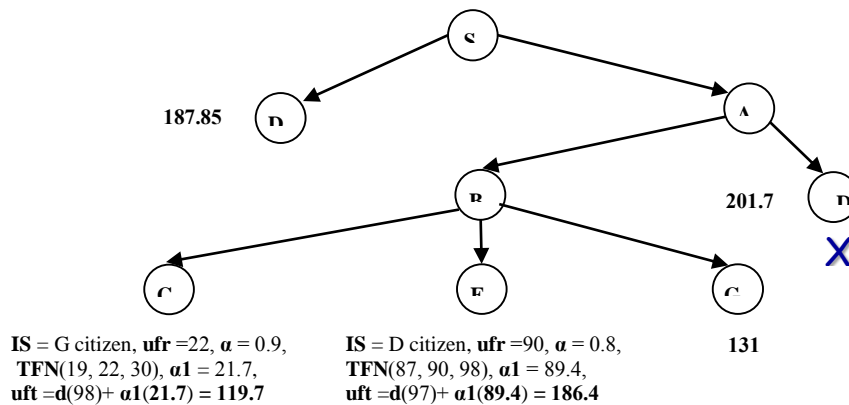
**1-** In the first step, as before, **D** and **A** are generated from **S**, at node **A** if the information source (**IS**) was an old man, who gives a fuzzy estimate of remaining distance as  $a = \text{"approx. 83"}$ , decision maker can choose **TFN** as (80, 83, 91), and  $\alpha=0.5$  to produce  $\alpha1=81.5$  which will be add to the distance already traveled  $d = 48$ , to produce the fuzzy underestimate of total path length  $uft = 129.5$ .

While node **D** information source about remaining distance (**IS**) was an Official sign with  $a = \text{"approx. 131"}$ , decision maker can choose **TFN** as (128, 131, 139), and  $\alpha = 0.95$  to produce  $\alpha1=130.85$  which will be add to the distance already traveled  $d = 57$ , to produce the fuzzy underestimate of total path length  $uft = 187.85$ .

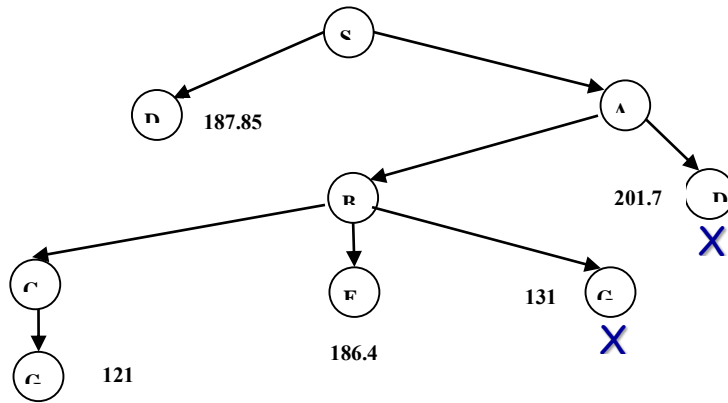
**A** is the node from which to search, as **A**'s fuzzy underestimated path length is **129.5**, which is shorter than that for **D**, **187.85**.



**2-** Expanding **A** leads to partial paths **S-A-B**, with a fuzzy underestimated path length of **125.1**, and to partial path **S-A-D**, with a fuzzy underestimated path length of **201.7**, partial path **S-A-D** can be deleted as its partial-path distance of **201.7** is more than that of. **S-D** (187.85) according to the dynamic programming procedure.

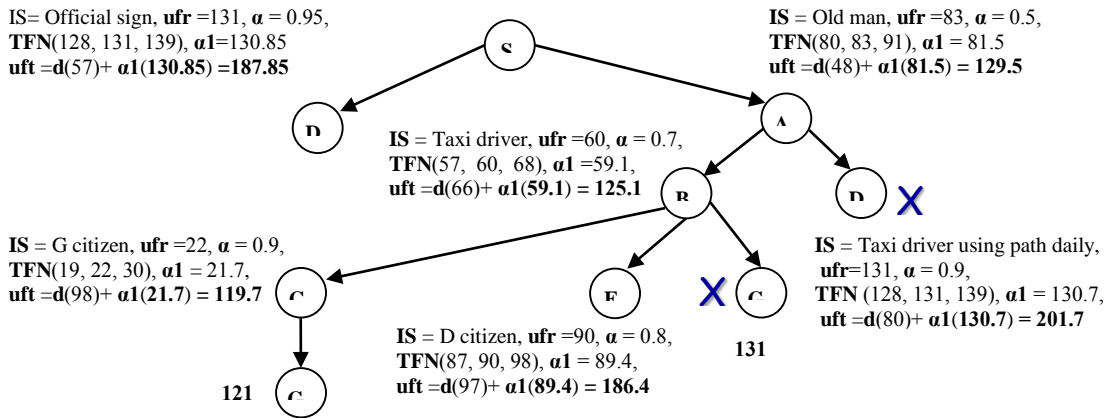


**3-** Now **S-A-B** is the partial path to extend, as it is the partial path with the minimum fuzzy underestimated path length. This expansion leads to partial paths **S-A-B-C**, with a fuzzy underestimated path length of **119.7**, partial path **S-A-B-E**, with a fuzzy underestimated path length of **186.4**, and to the shortest complete path, **S-A-B-G**, with a total distance of **131**, but to be absolutely sure, all partial paths with partial path distances less than **131** must be expanded. There is no need to extend the partial path **S-A-B-E**, as its partial-path distance of **186.4** is more than that of the complete path.



4- Finally S-A-B-C is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to a complete path, S-A-B-C-G, with a total distance of 121. No partial path has a lower-bound distance so low, so no further search is required.

Figure 4.14 will explain the algorithm of A\* search augmented by fuzzy underestimate, according to the previous example:



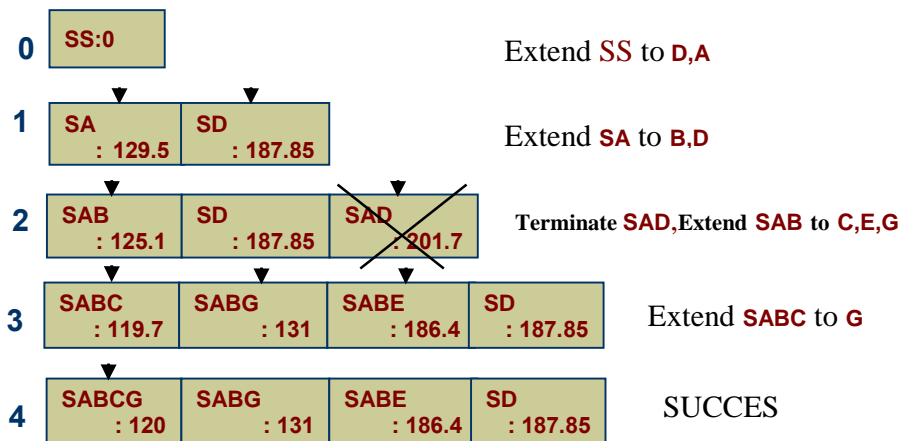


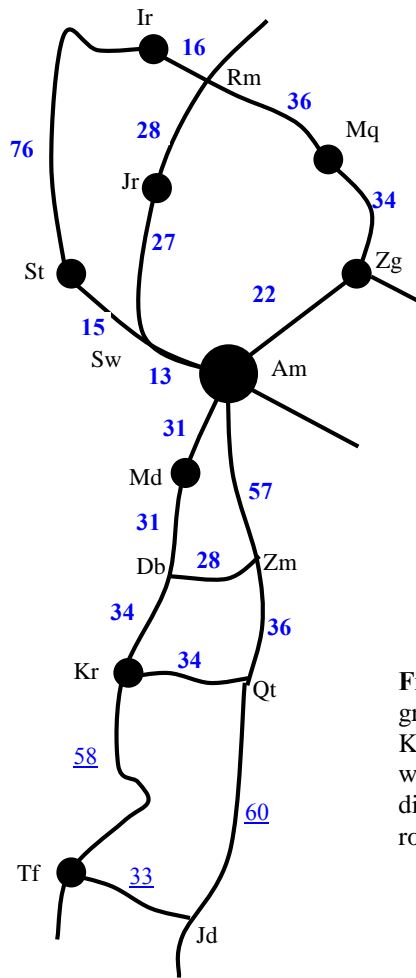
Figure 4.14: A\* Search augmented by fuzzy underestimate/ algorithm explanation.

#### 4.4.2 Roads between Two Jordanian Cities Application

In the following example, we will adopt a real life example; i.e. real roads between two major Jordanian cities; from **Al Karak** to **Irbid** according to an actual map for Jordan, as shown in Figure (4.15), where one can plan a route from **Al Karak** as start node (S) to **Irbid** as goal node (G) in the following map.

Suppose that Mr. X is trying to find some path from **Al Karak** to **Irbid** using a highway map such as the one shown in Figure (4.15). The starting point in **Al Karak** denoted as **Kr**, which might be called (start node), and ending point in **Irbid** denoted as **Ir**, which might be called (goal node).

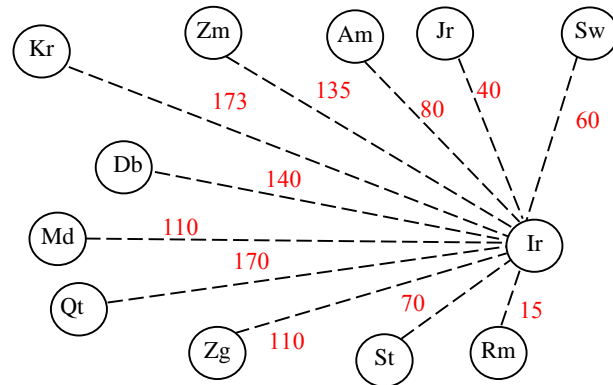
The traveler is at **Al Karak**, and intends to go to **Irbid**. All possible routes are shown in the net graph; the number against each edge gives the actual distance of that route (node to node) in **kilometers**. The traveler has no knowledge about the distance information, but the traveler records the distance he completed.



**Figure 4.15:** net graph from Karak to Irbid with actual distance of each route.

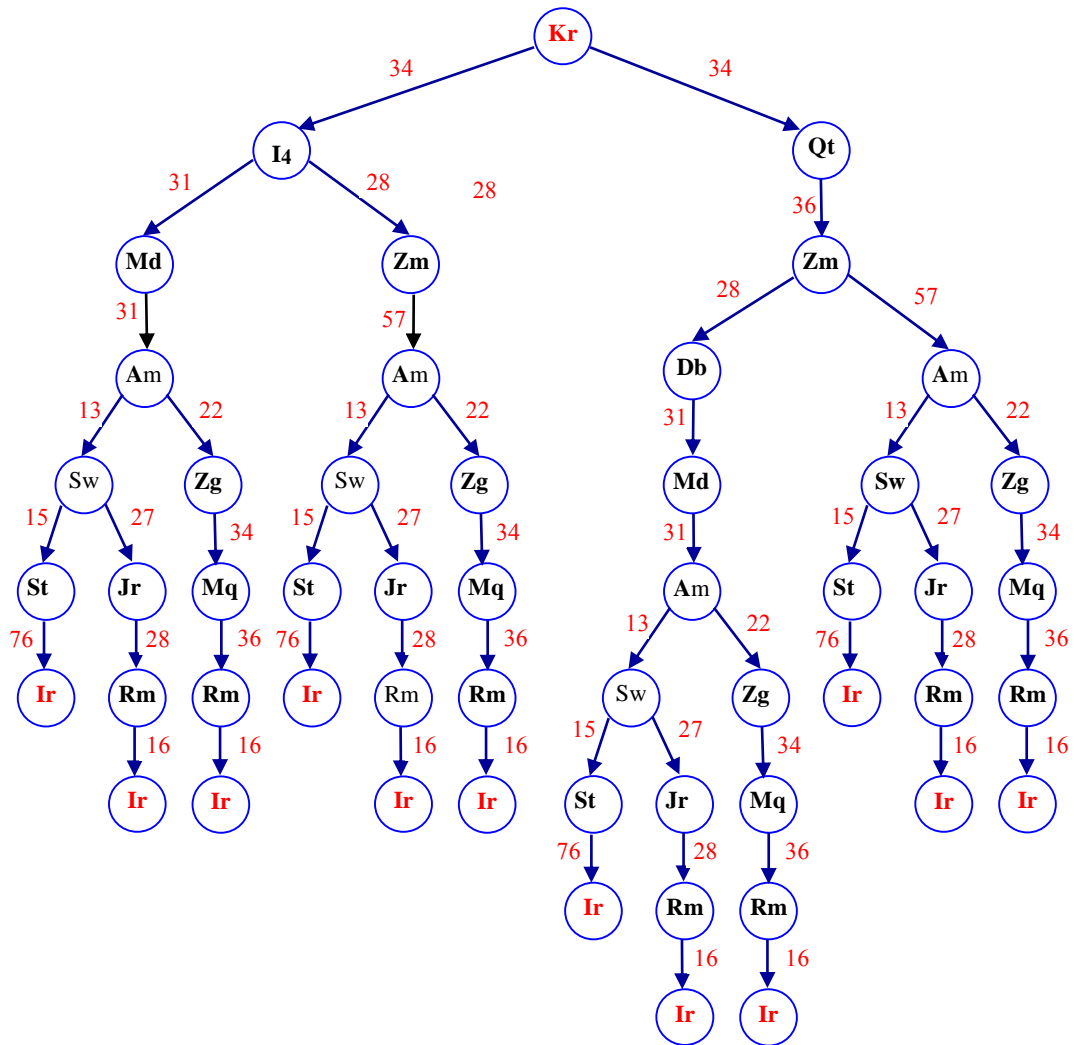


Figure (4.16) shows the fuzzy estimates of remaining distances from each city to **Irbid** (the goal) , where the number against each edge gives the estimated distances in **kilometers** = (efr).



**Figure 4.16:** fuzzy estimates of remaining distances from each city (or intersection node) to Irbid =efr.

With looping paths eliminated, one can arrange all possible paths from the start node S in a search tree. Figure (4.17) shows a search tree that consists of nodes denoting all possible paths that lead outward from the start node Al Karak (S) of the net shown in Figure (4.15) , the number against each edge gives the actual distance of that route (node to node) in **kilometers**.



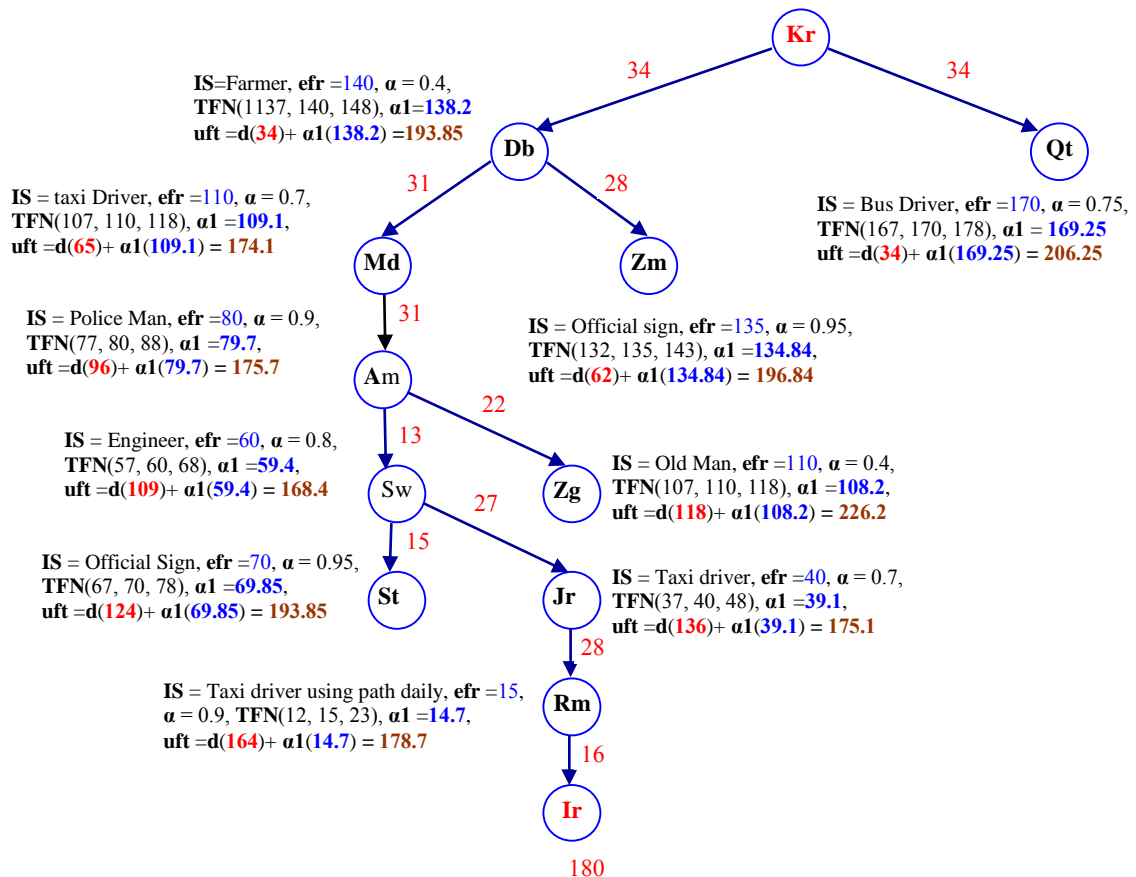
**Figure 4.17:** A search tree that consists of nodes denoting all possible paths that lead outward from the start node Al Karak (S) of the net shown in figure (4.15).

If a decision-maker takes a fixed TFN model slope as  $(a-3, a, a+8)$ , at each node he/she will take fuzzy estimate of remaining distance for new paths = efr = "approx. a" from different information sources = **IS**, choose  $\alpha$  according to degree of searcher confidence in estimation, take a fuzzy lower-bound estimate of the cost remaining;  $\alpha_1 = a_1 + \alpha * (a - a_1)$ , and finally he/she will add new paths, and sort the paths by the sum of the already traveled path length = d and the fuzzy lower-bound estimate of the remaining distance = ufr to choose the shortest path.

Figure (4.18) shows how fuzzy underestimates of distances remaining helps to make the search more efficient, where A\* search augmented by fuzzy underestimates determines that the path Kr- I<sub>4</sub>- Md- Am- Sw- Jr- R<sub>m</sub>- Ir is optimal. The numbers beside the nodes are underestimate of total path length (uft) = accumulated distances(d) + fuzzy lower-bound estimate of the cost remaining ( $\alpha$ 1).

Fuzzy underestimates quickly push up the lengths associated with bad paths. In this example, fewer nodes are expanded than would be expanded with A\* search operating with crisp underestimates.

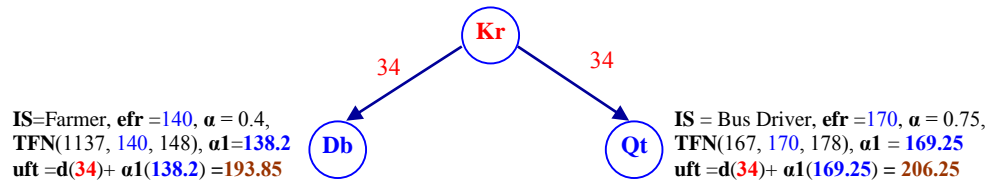
Part of the tree, which must be explored by the proposed algorithm will be as shown in figure (4.18):-



**Figure 4.18:** The explored part of the tree. At each node there is, specific source for estimated information =IS, who (which) will give fuzzy cost estimate of remaining distance = "approx. a" =  $\alpha$ , decision maker will choose an appropriate  $\alpha$ , TFN model, & lower  $\alpha$ -cut for that node.

The problem can be solved by applying the proposed algorithm of section 4.2 as in the following example:-

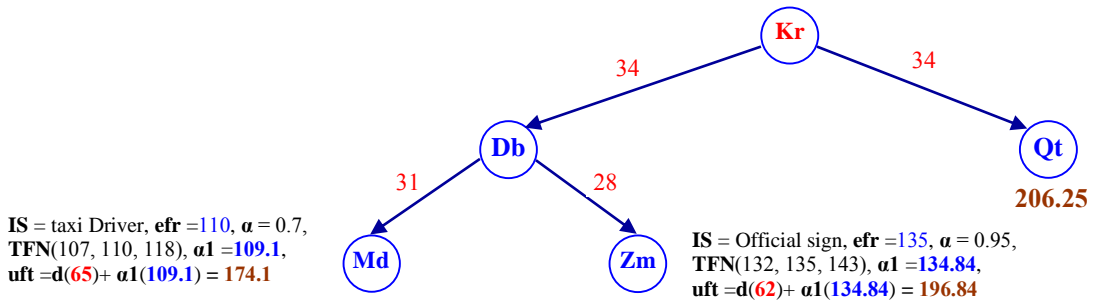
**Figure 4.19:** Example for A\* Search augmented by fuzzy underestimates



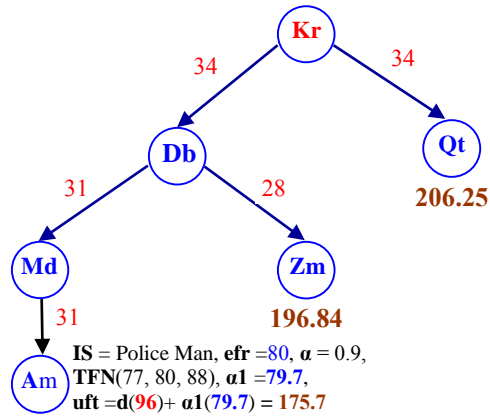
**1-** In the first step, as before, **Db** and **Qt** are generated from **Kr(S)**, at node **Db** if the information source (**IS**) was a **Farmer**, who gives a fuzzy estimate of remaining distance as **a** = "approx. 140", **decision maker can choose TFN as (137, 140, 148)**, and  $\alpha=0.4$  to produce  $\alpha1=138.2$  which will be added to the distance already traveled  $d = 34$ , to produce the fuzzy underestimate of total path length  $uft = 193.85$ .

While at node **Qt** information source about remaining distance (**IS**) was a **Bus Driver** with **a** = "approx. 170", **decision maker can choose TFN as (167, 170, 178)**, and  $\alpha = 0.75$  to produce  $\alpha1=169.25$  which will be added to the distance already traveled  $d = 34$ , to produce the fuzzy underestimate of total path length  $uft = 206.25$ .

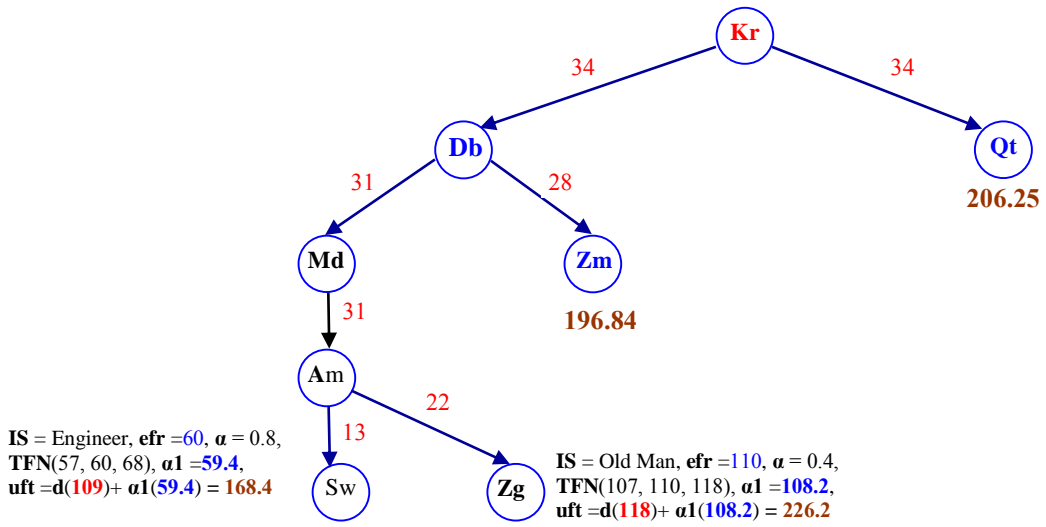
**Db** is the node from which to search, as **Db**'s fuzzy underestimated path length is **193.85**, which is shorter than that for **Qt**, **206.25**.



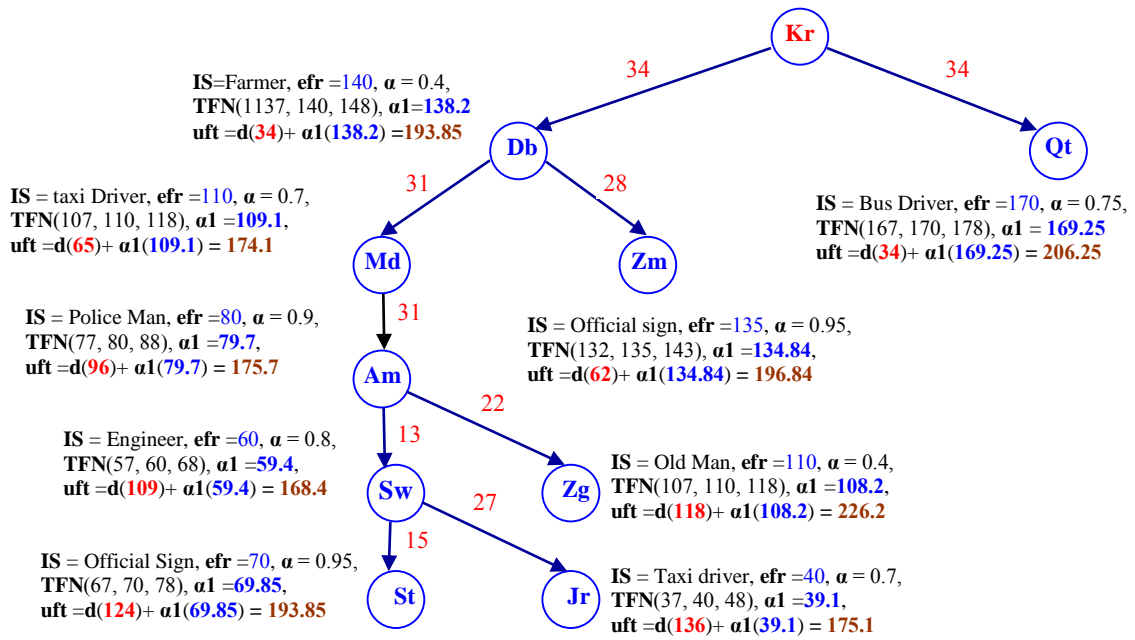
**2-** Expanding **Db** leads to partial paths **Kr- Db - Md**, with a fuzzy underestimated path length of **174.1**, and to partial path **Kr- Db - Zm**, with a fuzzy underestimated path length of **196.84**



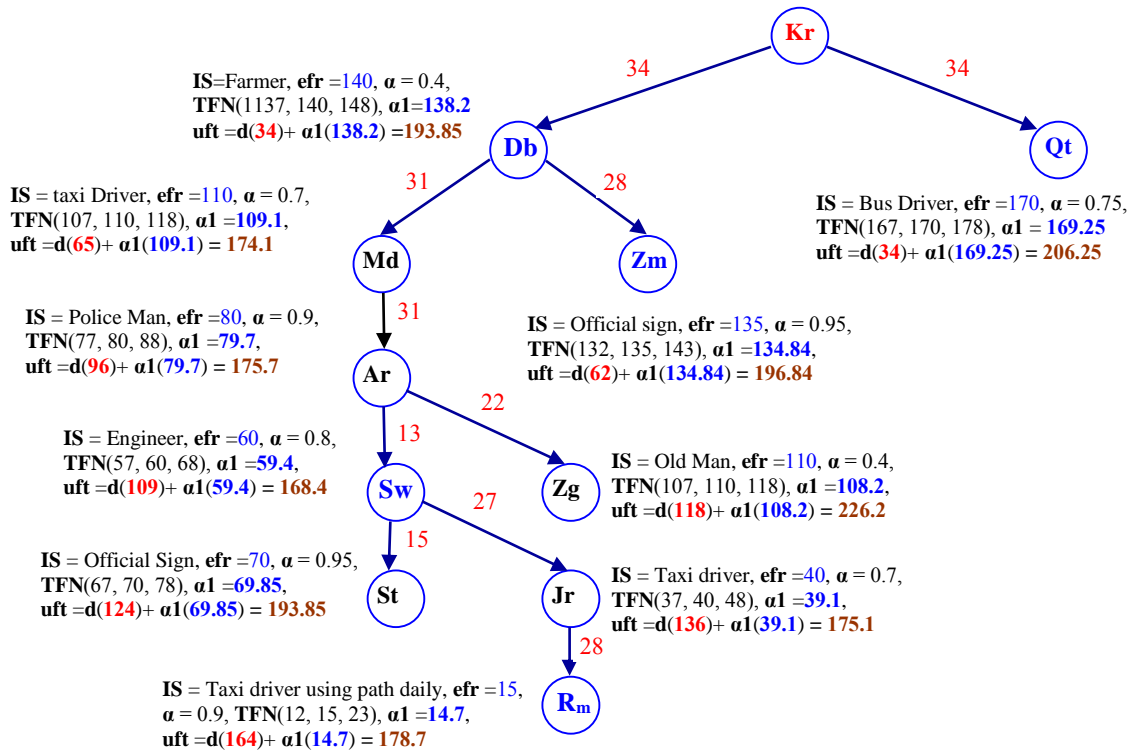
3- Now **Kr- Db - Md** is the partial path to extend, as it is the partial path with the minimum fuzzy underestimated path length. This expansion leads to partial paths **Kr- Db - Md- Am** with a fuzzy underestimated path length of **175.7**.



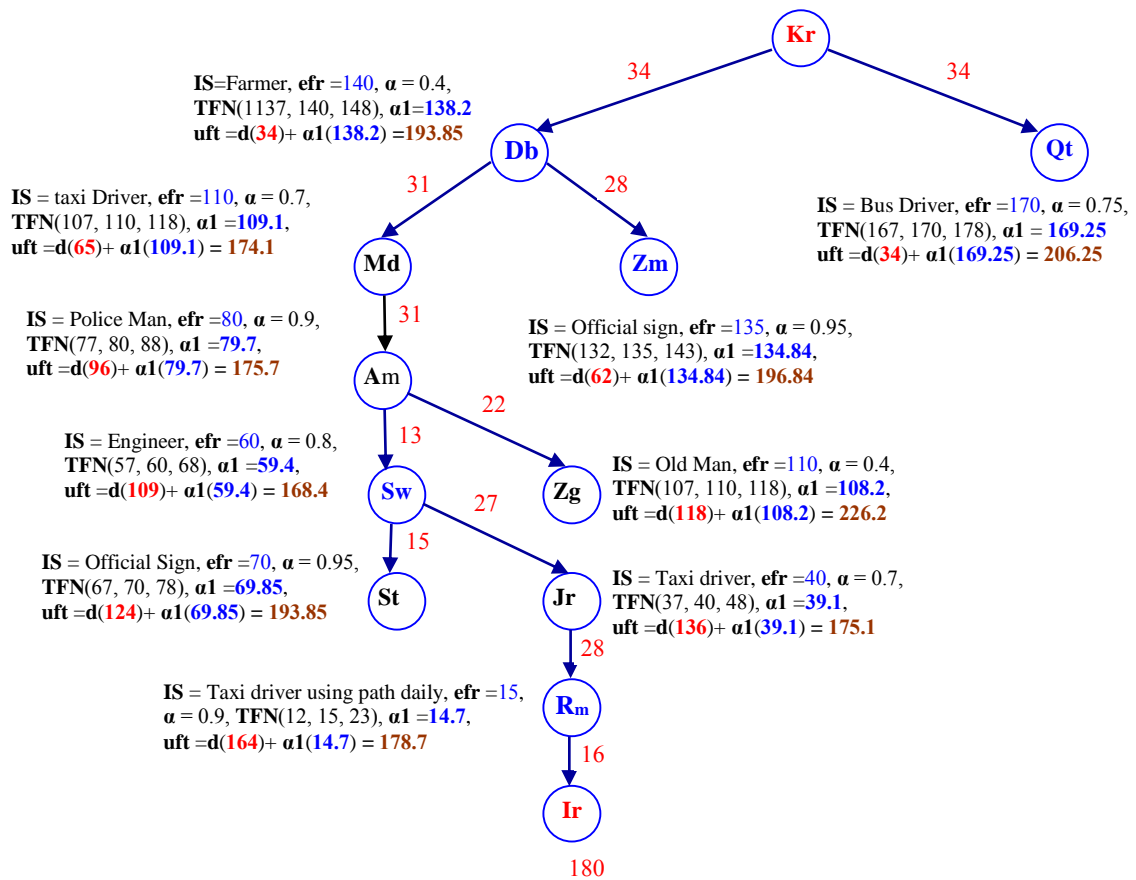
4- Now **Kr- Db - Md- Am** is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to partial paths **Kr- Db - Md- Am- Sw**, with a fuzzy underestimated path length of **168.4**, and to partial **Kr- Db - Md- Am- Zg**, with a fuzzy underestimated path length of **226.2**.  
**Kr- Db - Md- Am- Sw** is the partial path to extend, as it is the partial path with the minimum underestimated path length.



**5-** Expanding **Sw** leads to partial paths **Kr-Db-Md-Am-Sw-St**, with a fuzzy underestimated path length of **193.85**, and to partial path **Kr-Db-Md-Am-Sw-Jr**, with a fuzzy underestimated path length of **175.1**



**6-** Now **Kr-Db-Md-Am-Sw-Jr** is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to partial path **Kr-Db-Md-Am-Sw-Jr-Rm**, with a fuzzy underestimated path length of **178.7**.



7- Now **Kr- Db - Md- Am- Sw- Jr- R<sub>m</sub>** is the partial path to extend, as it is the partial path with the minimum underestimated path length. This expansion leads to a complete path, **Kr- Db - Md- Am- Sw- Jr- R<sub>m</sub>- Ir**, with a total distance of 180. No partial path has a lower-bound distance so low, so no further search is required.

Figure (4.20) will explain the algorithm of A\* search augmented by fuzzy underestimate, according to the previous example:

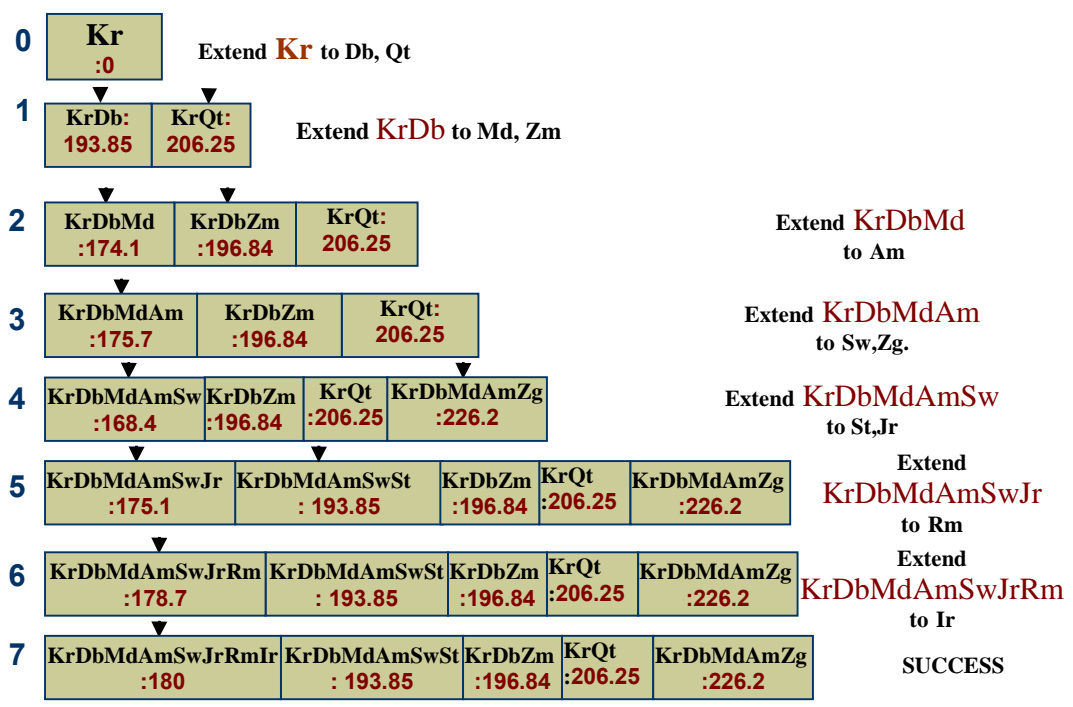
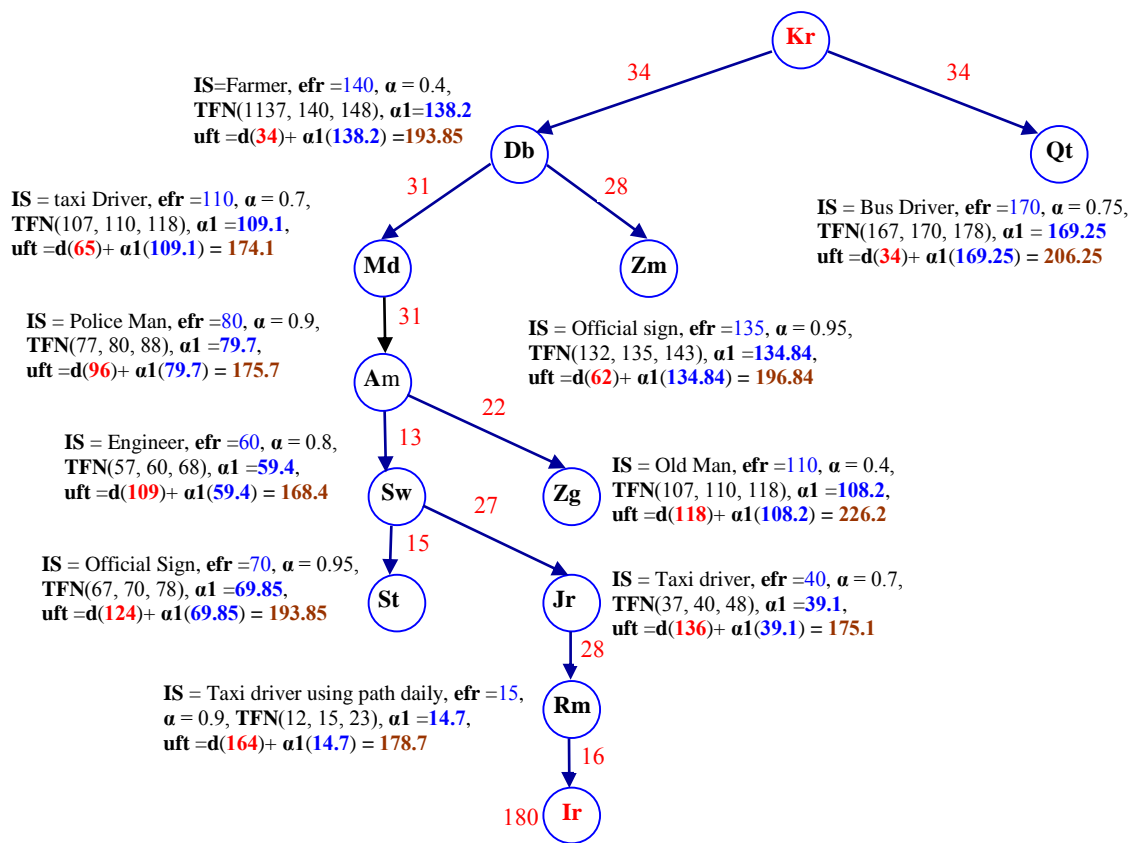


Figure 4.20: A\* Search augmented by fuzzy underestimate/ algorithm explanation.



## 4.5 Conclusion

In this chapter a new type of A\* searching technique using fuzzy underestimates is proposed.

Branch and Bound with dynamic programming is suitable when many paths converge on the same place.

□ The A\* procedure is suitable when both branch-and bound search with a guess and dynamic programming are good, where underestimates quickly push up the lengths associated with bad paths and dynamic programming drop redundant paths from the search queue. In figure (4.7), fewer nodes are expanded (10 nodes) than which may be expanded with branch-and- bound with dynamic programming search operating without underestimates.

As in the last chapter, when analyzing search methods, Effective Branching Factor ( $b^*$ ) of each method is important to be examined to characterize the quality of a heuristic, where a well - designed heuristic would have a value of  $b^*$  close to 1, by reducing the number of nodes that need to be examined in the search tree.

When evaluating heuristic search strategies which are discussed in this chapter in term of EBF ( $b^*$ ) according to the results shown in table (4.1) and the corresponding chart shown in figure (4.21), we can note that:

- DP principle has a high value of Effective Branching Factor when it is used without other algorithms, where  $b^* = 1.81$  in case of figure (4.3).

- A\* search technique achieves better efficiency than B&B with dynamic programming search technique, where  $b^*$  was decreased from 1.81 to 1.4 because underestimation increases the efficiency of Branch and Bound with dynamic programming by enabling it to be more informed.
- Using Fuzzy underestimate with A\* search technique achieves better efficiency than using crisp underestimate, where  $b^*$  was decreased from 1.4 to 1.35 (in example of figure 4.13), and to 1.135 (in example of figure 4.19) because fuzzy underestimation increases the efficiency of A\* by enabling it to be more informed.
- A\* with Fuzzy Underestimate search technique achieves better efficiency than Branch and Bound Augmented by Fuzzy Underestimate search technique, because dynamic programming principle will drop redundant paths.
- Using Fuzzy Underestimate with A\* search technique as shown in case of figures (4.13) and (4.19) may not decrease  $b^*$  much as using Fuzzy underestimate with B B search technique as shown in the case of figures (3.16) and (3.21), because A\* search technique already achieves higher efficiency than B&B search technique.
- In general adding Fuzzy underestimates can achieve better efficiency than adding crisp underestimates, where Effective Branching Factor for Fuzzy Underestimated A\* is always better (less) than that for crisp algorithms especially when the number of nodes is high. Obviously the closer the Fuzzy underestimate is to the true remaining solution cost the more efficient the A\* search will be.

- Fuzzy Underestimated A\* search technique is complete (guaranteed to find a solution), optimal ( solution is guaranteed to be the best solution), nonredundant, and more informed than other algorithms, and have higher memory requirements (space complexity) comparable to other search techniques because it maintains all the generated nodes in the memory.

Figure No.	d	N	b*	Type of Search
4.3	3	11	1.81	B&B with Dynamic Programming
4.7	4	10	1.403	A*
4.13	4	9	1.352	A* with fuzzy Underestimation ( Application 1)
4.19	7	12	1.135	A* with fuzzy Underestimation ( Application 2)

Table 4.1: Evaluation of Heuristic A\* search strategies in terms of effective branching factor ( $b^*$ ).

d: is the depth of the solution,

N: is the total number of nodes generated by each strategy for a particular problem.

A well - designed heuristic would have a value of  $b^*$  close to 1.

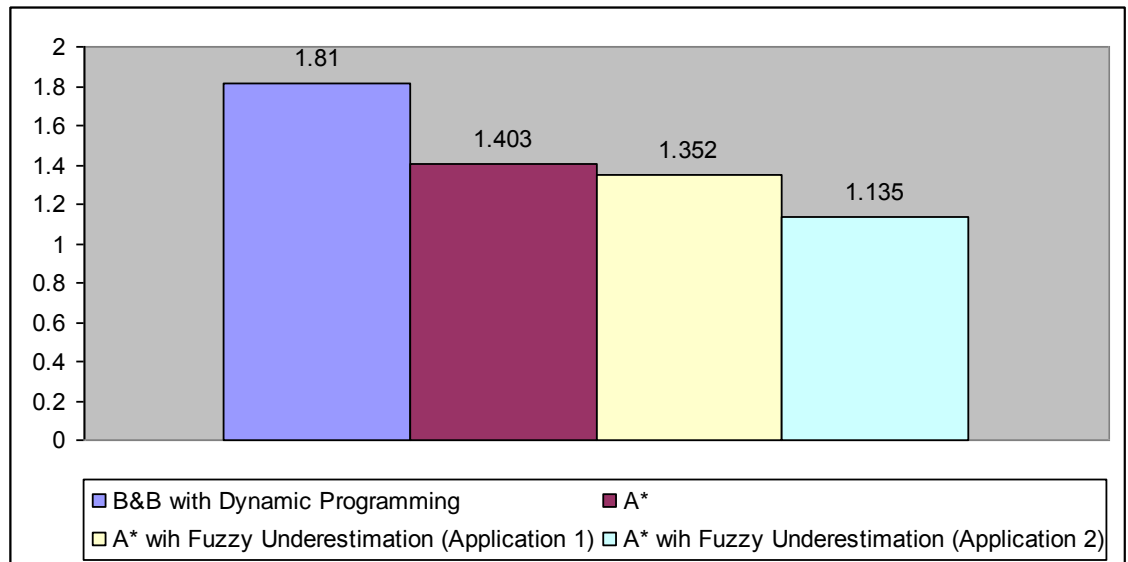


Figure 4.21: Evaluation of Heuristic A\* search strategies in terms of effective branching factor ( $b^*$ ) according to the results shown in table (4.1)

# chapter five

## performance evaluation and simulation

### 5.1 Introduction

In order to evaluate the performance of the proposed algorithms we will introduce a visual interface program called Searching Performance Analyzer (SPA), the source code for this program is shown in appendix (2).

In this chapter six Searching techniques are to be analyzed using the simulation program (SPA) to compute the Number of Iterations, Time Complexity, Space Complexity, and Effective Branching Factor for each algorithm.

This analysis process is to be carried out using six searching techniques, these techniques are Branch & Bound, Branch & Bound with Underestimation, Branch & Bound with dynamic programming, A\*, Branch & Bound with Fuzzy Underestimation, and A\* with Fuzzy Underestimation.

Appendix (1) presents the pseudocode for each algorithm. Section (5.2) presents the simulated examples which goes through the simulation procedures step by step.

### 5.2 Simulation

This section presents the simulation for the proposed Searching techniques which are Branch & Bound with Fuzzy Underestimation, A\* with Fuzzy Underestimation, and other related Searching techniques (Branch & Bound, Branch & Bound with Underestimation, Branch & Bound with dynamic programming, and A\*).

This simulation is programmed in Visual Basic Object Oriented Programming (OOP) language.

### 5.2.1 Simulator Description

Simulated search problems can be represented by a state space as a directed graph whose nodes correspond to problem situations and arcs to possible moves. The particular problem is defined by a start node and a goal condition. The solution of the problem corresponds to a path on the graph. Thus problem solving is reduced to searching for a path in a graph.

In order to explain the many techniques available, one can look at the problem of route planning, in a particular route from a start node (S) to a goal node (G) as was shown previously in Figure (2.9), or a highway map such as the one shown in Figure (3.18) which represents the real life example between two major Jordanian cities (from **Al Karak** to **Irbid**) according to an official map of Jordan, where one can plan a route from **Al Karak** as a start node (S) to **Irbid** as a goal node (G).

The graph data must be entered in special Nodes and Edges Screens, where the nodes information to be entered are: Start, Goal, and other Intermediate nodes, according to the net graph, then node's cost information are entered using the Edges Screen as: expansion edges for each node with the corresponding cost (between each two nodes), straight line distance from each node to goal (as the crisp underestimated value), and fuzzy estimated distance from each node to goal with the corresponding Information Source type.

Then with looping paths eliminated, the program will display all possible paths from the start node S in a search tree that consists of nodes denoting all the possible paths that lead outward from the start node S of the net.

Finally the six search procedures will be executed by the program for that specific net, and the results of those procedures execution will be recorded and displayed (by Result Screen) in a table and four bar charts.

Simulation steps and procedures are presented in the following sections.

### **5.2.2 Examples**

In this sub section all examples will be introduced by adopting real roads between some major Jordanian cities; according to an official map of Jordan, which is shown in Figure (5.1), one can plan a route from a start node (S) to a goal node (G) using the map taken from (rjgc, April. 23, 2007).

Results of each example are presented using a table and four bar charts representing (Number of Iterations, Time Complexity, Space Complexity, and Effective Branching Factor) for the six algorithms.

The simulation program gives us the opportunity of choosing number of nodes to be used (6-20 nodes).

All examples will use two main screens to enter node's information and node's cost information. The net will be converted automatically into a search tree by tracing out all possible paths until searcher cannot extend any of them without creating a loop.

Each type of algorithm has two execution methods (Manual or Automatic) which can be chosen from the tree screen as shown in figure (5.8). When choosing Automatic run (A), all algorithms (techniques) will be displayed automatically and the results will be given directly in the result screen.

Manual run (M) can be chosen in cases where we would like to see detailed description of each step during the execution of a specific algorithm.

By choosing Manual run (M) for one of the six algorithms, the type of search which has been chosen will be executed in steps to show the procedures of the chosen algorithm, while simultaneously the other algorithms (techniques) will be executed automatically and the comparison results for all six algorithms will be displayed in the result screen after completion of the manual run.

المملكة الأردنية الهاشمية – المواصلات والآثار ٢٧

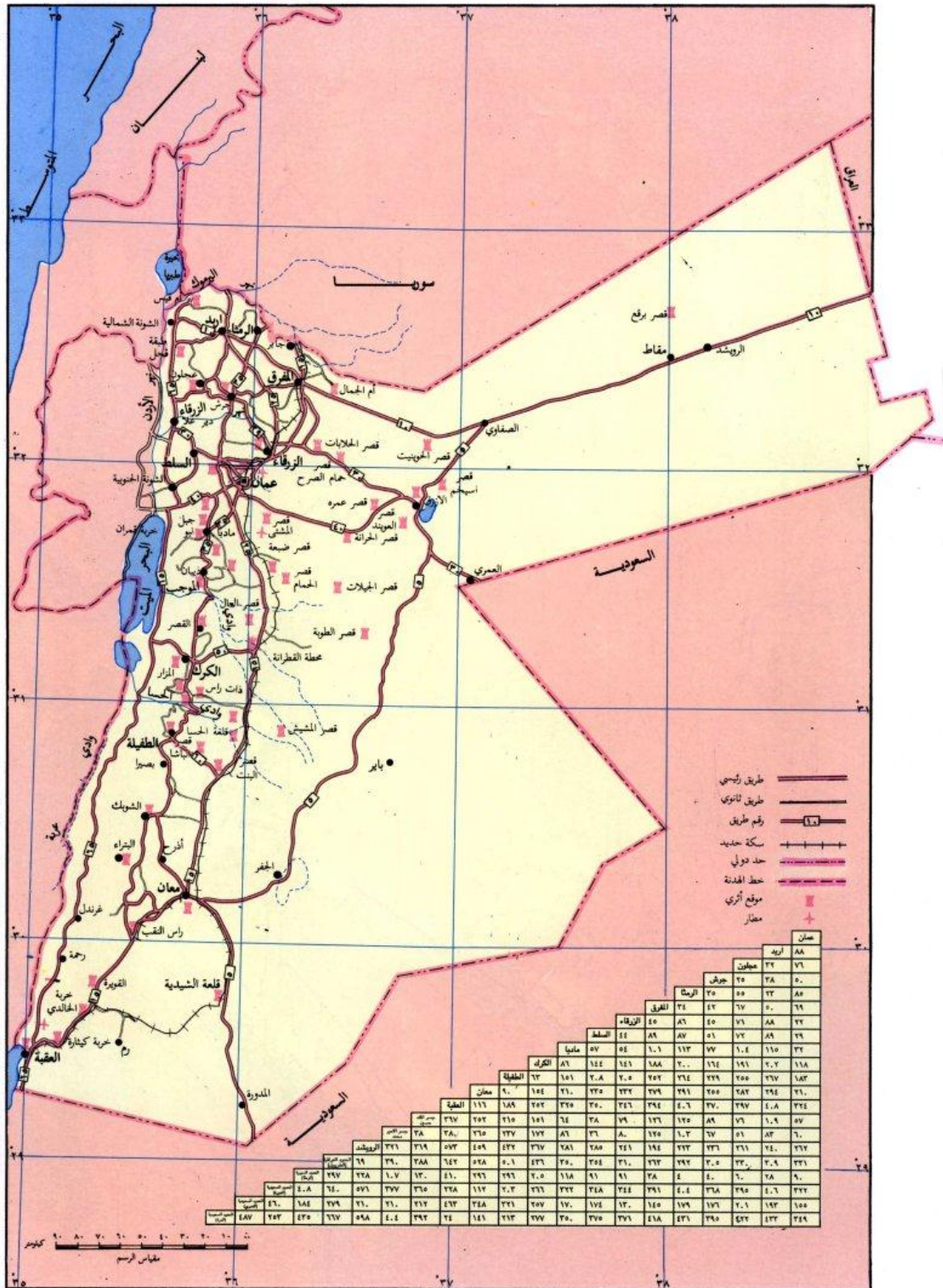
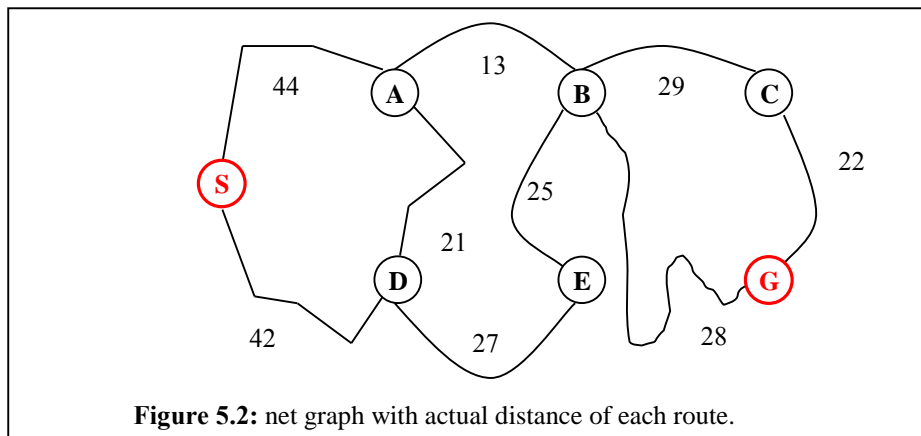


Figure 5.1: Jordan Map.



### Example one:

In the following example, we will consider the net shown in figure (5.2) The traveler is at node S, and intends to go to node G. All possible routes are shown in the net graph; the numbers shown represent the actual distance of each route (node to node) in some unit. The traveler has no knowledge about the distance information, but the traveler records the distance he completed.



To solve this net graph problem, we can use the simulation program (SPA) as shown in the following steps.

#### Step 1-

When the program starts, a splash screen will appear as shown in figure (5.3).

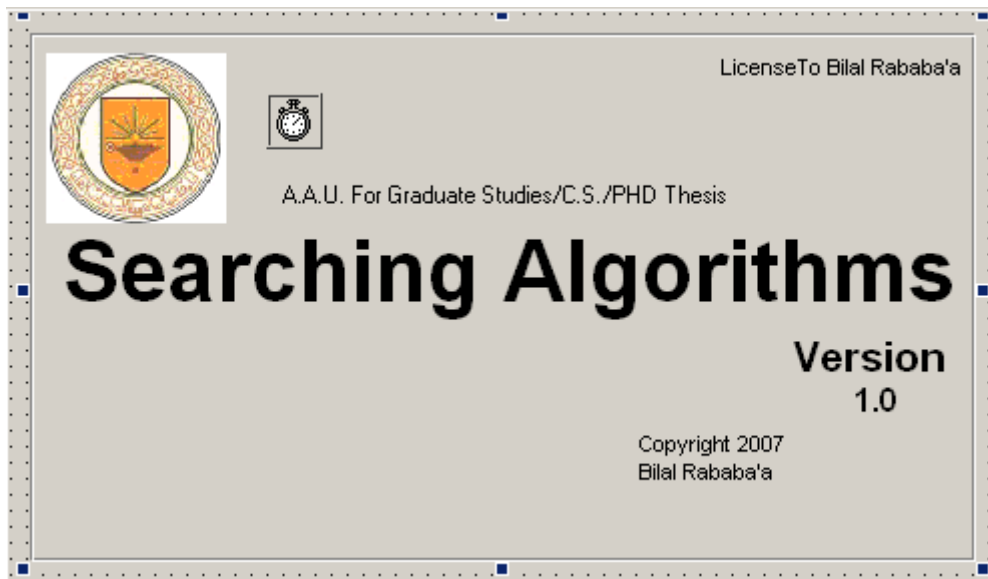
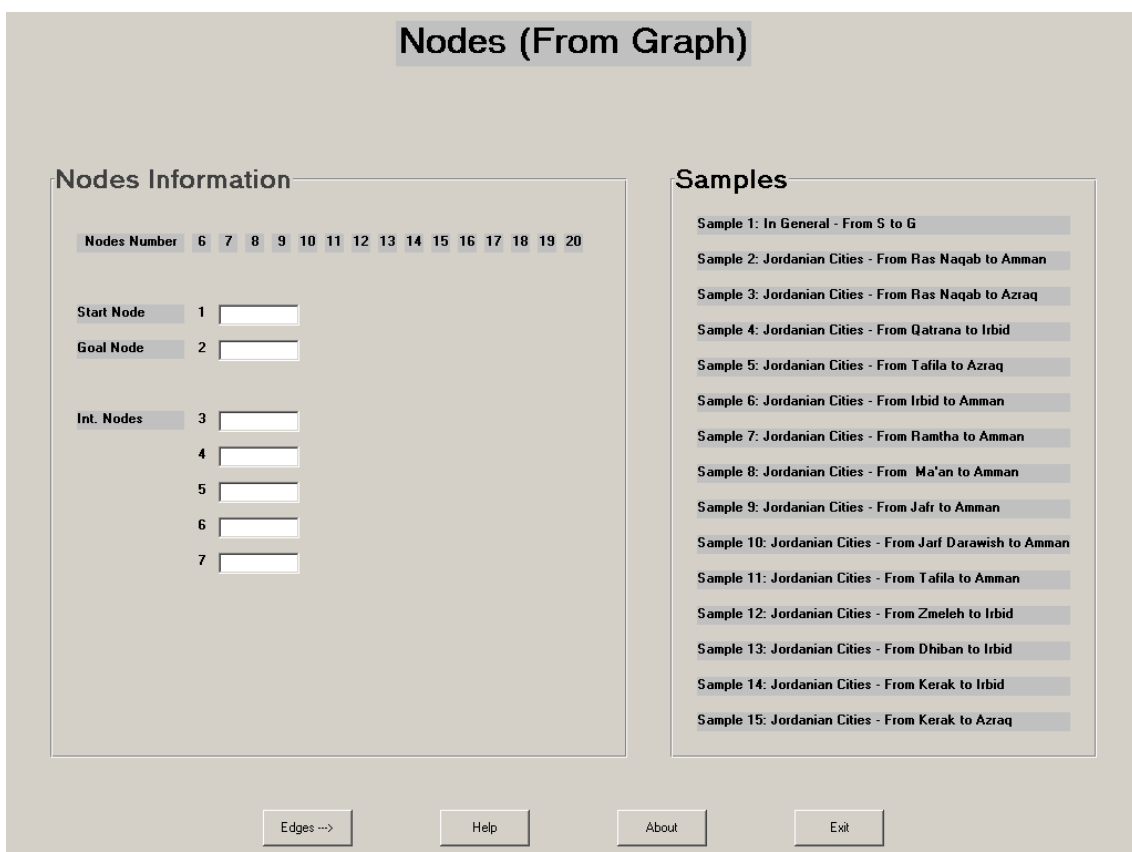


Figure 5.3: Splash Screen.

After completing the necessary operations the Nodes Screen will be displayed as shown in figure (5.4).



**Figure 5.4** Nodes Screen without Nodes Information.

In the Nodes Screen the user must assign (Nodes Information) these are; number of nodes (from 6 to 20 nodes), start node, goal node, and intermediate nodes according to the required net graph, or he/she can use one of the available samples (sample 1 - sample 15).

In this example we will choose (Sample 1) to deal with the graph given in figure (5.2).

### Step 2-

On choosing (Sample 1), the empty fields (Nodes Information) will be filled by the program according to (Sample 1) pre-entered data as shown in figure (5.5).

**Nodes (From Graph)**

#### Nodes Information

Nodes Number: 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Start Node: 1

Goal Node: 2

Int. Nodes:

- 3
- 4
- 5
- 6
- 7

#### Samples

- Sample 1: In General - From S to G
- Sample 2: Jordanian Cities - From Ras Naqab to Amman
- Sample 3: Jordanian Cities - From Ras Naqab to Azraq
- Sample 4: Jordanian Cities - From Qatrana to Irbid
- Sample 5: Jordanian Cities - From Tafila to Azraq
- Sample 6: Jordanian Cities - From Irbid to Amman
- Sample 7: Jordanian Cities - From Ramtha to Amman
- Sample 8: Jordanian Cities - From Ma'an to Amman
- Sample 9: Jordanian Cities - From Jafr to Amman
- Sample 10: Jordanian Cities - From Jarf Darawish to Amman
- Sample 11: Jordanian Cities - From Tafila to Amman
- Sample 12: Jordanian Cities - From Zmeleh to Irbid
- Sample 13: Jordanian Cities - From Dhiban to Irbid
- Sample 14: Jordanian Cities - From Kerak to Irbid
- Sample 15: Jordanian Cities - From Kerak to Azraq

Edges -->    Help    About    Exit

**Figure 5.5:** Nodes Screen after choosing (Sample 1).

Note that fields 1 to 7 have been filled with letters which represent (Sample 1) net graph nodes.

At the bottom of (Nodes Screen) four commands are available which can be used to display the following:

Edges: to display the next screen (Edges Screen), as shown in figure (5.7).

Help: to display the Help file (Read me) file.

About: to display (About Screen), which gives a brief description about the program.

Exit: to end the program.

### Step 3-

After choosing Edges command, the next screen (Edges Screen) will appear as shown in figure (5.6).

The screenshot shows the 'Edges (From Graph)' interface. It features a table with columns for 'From Node', 'To Node', 'Edge Cost', and 'e. to Goal S.L. est.'. The table is split into two panes. The left pane shows nodes 1 through 7, each with two 'To Node' options and an 'Edge Cost' field. The right pane is empty. Below the table, there are three buttons: '<-- Nodes', 'Tree -->', and 'Exit'.

**Figure 5.6:** Edges Screen without node's cost information.

In the (Edges Screen), the user must assign node's cost information in the empty fields which are:

- 1- (To Node) field : expansion edges for each node
- 2- (Edge Cost) field: cost of the corresponding edge., , and with the corresponding Information Source type.
- 3- (e. to Goal-S.L.) field: Straight Line distance from each node to goal as the crisp underestimated value.
- 4- (e. to Goal-est.) field: fuzzy estimated distance from each node to goal.
- 5- (Information Source) field: choose one of ten choices which are pre-determined as:

“Map” which corresponds to  $\alpha = 0.95$ , “Official Sign” which corresponds to  $\alpha = 0.95$ ,

“Police Man” who corresponds to  $\alpha = 0.95$ , “G Citizen” who corresponds to  $\alpha = 0.90$ ,

“Person Using The Path Daily” who corresponds to  $\alpha = 0.90$ , “Taxi Driver using The Path Daily” who corresponds to  $\alpha = 0.90$ , “Previous Expert” who corresponds to  $\alpha = 0.80$ , “Other City Citizen” who corresponds to  $\alpha = 0.60$ , “Taxi Driver” who corresponds to  $\alpha = 0.70$ , and “Old Man” who corresponds to  $\alpha = 0.50$ .

In our example the empty fields will be filled by the program according to (Sample 1) pre-entered data as shown in figure (5.7).

### Edges (From Graph)

From Node	To Node	Edge Cost	e. to Goal S.L. est.	Information Source
S	D	= 44		
	A	= 42		
G	Goal Node			
D	A	= 21	55 60	Official Sign
	E	= 27		
A	B	= 13	35 40	Old Man
	D	= 21		
E	B	= 25	45 50	Other City Citizen
		=		
B	C	= 29	20 25	Taxi Driver
	G	= 28		
C	G	= 22	15 20	Taxi Driver Using The Path Daily
		=		

**Figure 5.7:** Edges Screen after choosing (Sample 1).

Note that at the bottom of (Edges Screen) three choices are available which can be used as follows:

Nodes: to display the previous screen (Nodes Screen).

Tree: to display the next screen (Tree Screen).

Exit: to end the program.

In case of modification of data user can go back to (Nodes Screen) by choosing (Nodes) command.

#### Step 4 -

To continue, choose (Tree) command, where the net will be converted automatically into a search tree as shown in figure (5.8) by tracing out all possible paths from the start node S of the net until program cannot extend any of them without creating a loop.

Note that the numbers illustrated on the tree have been explained as shown in the associated text boxes.

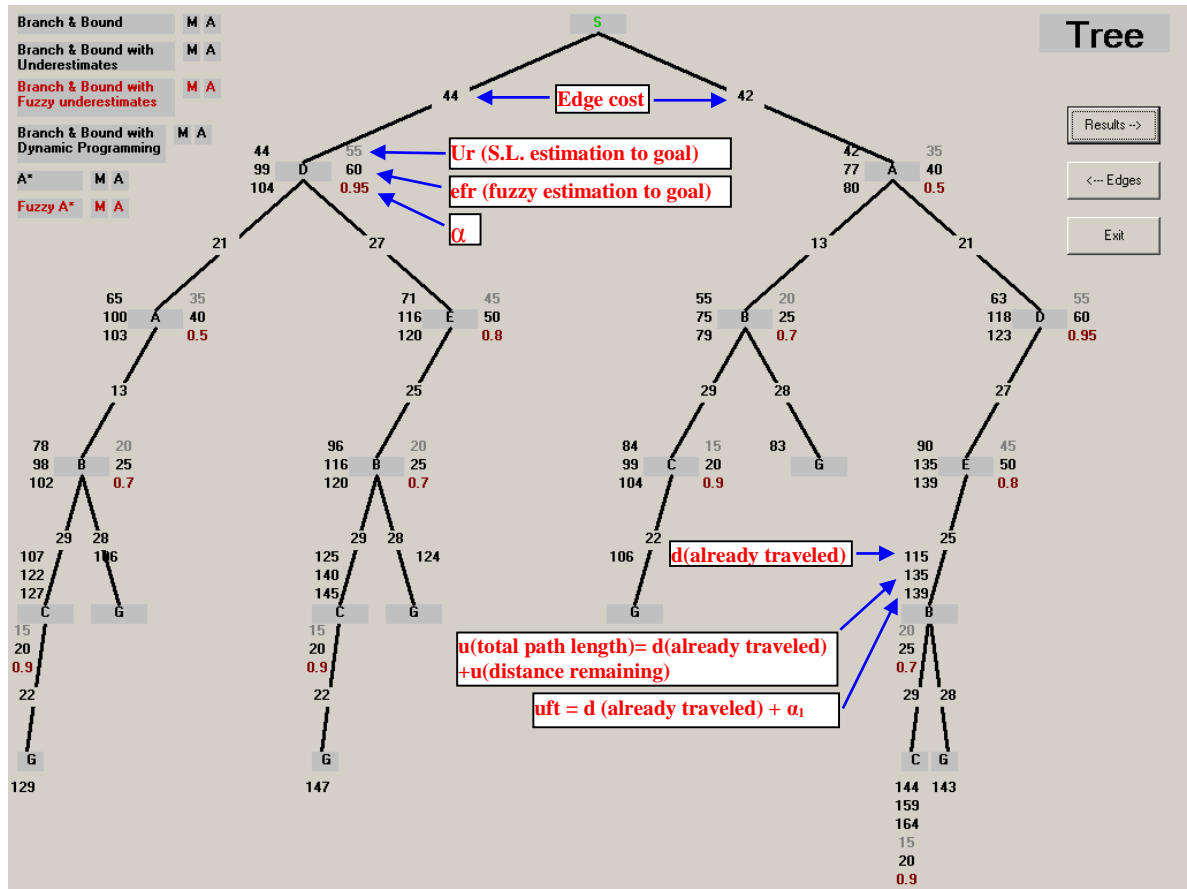


Figure 5.8: Tree Screen after choosing (Sample 1).

To the top left corner of the screen you will find names of the six algorithms which will be executed and type of execution for each technique, Manual or Automatic, where type of algorithm and execution method must be chosen.

On choosing Automatic run (A), all algorithms (techniques) will be executed automatically and the results will be given directly in the result screen.

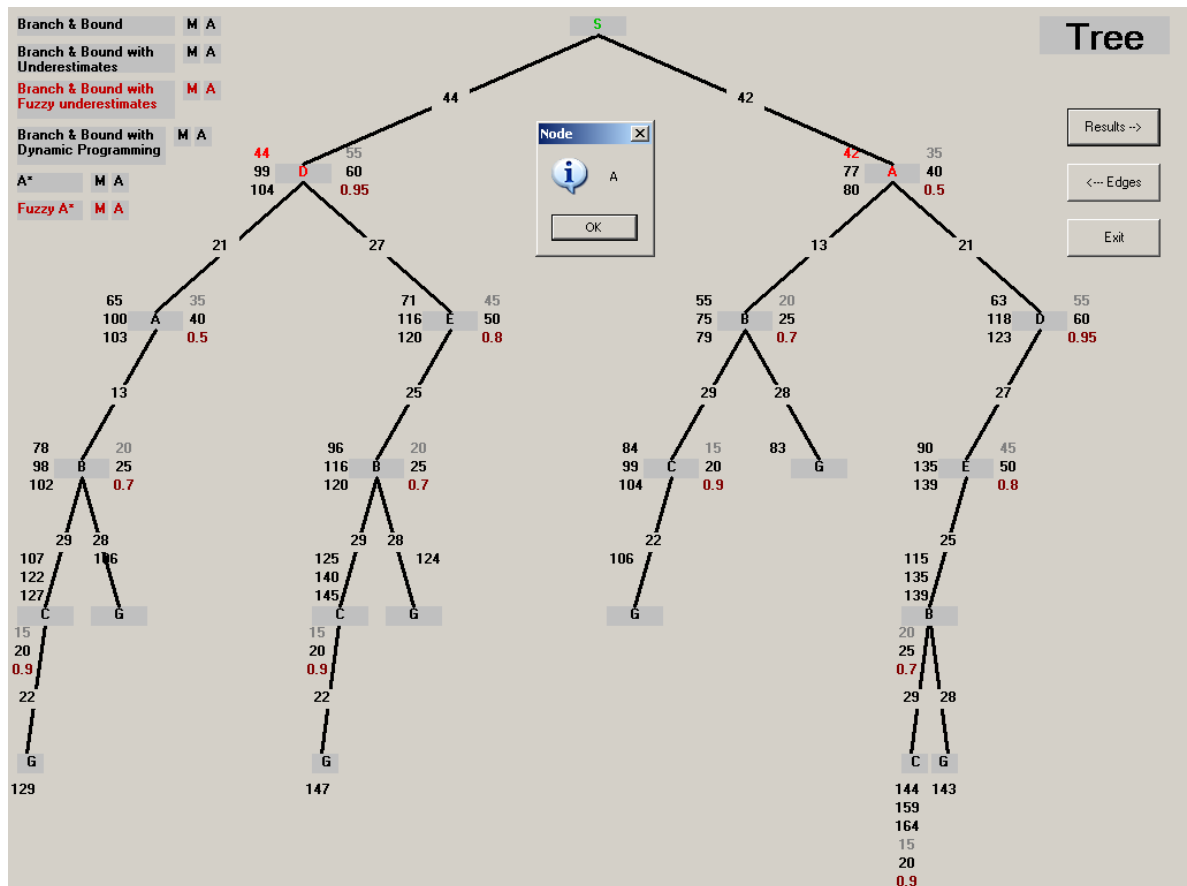
When choosing Manual run (M) for any of the six algorithms, the type of search which has been chosen will be executed step by step to show the algorithm procedures, while all other algorithms (techniques) will be carried out simultaneously. The comparison results will be given in the result screen after the

completion of the manual run steps.

In this example (Sample 1) we will choose the Manual (M) run of “Branch & Bound Search with dynamic programming”.

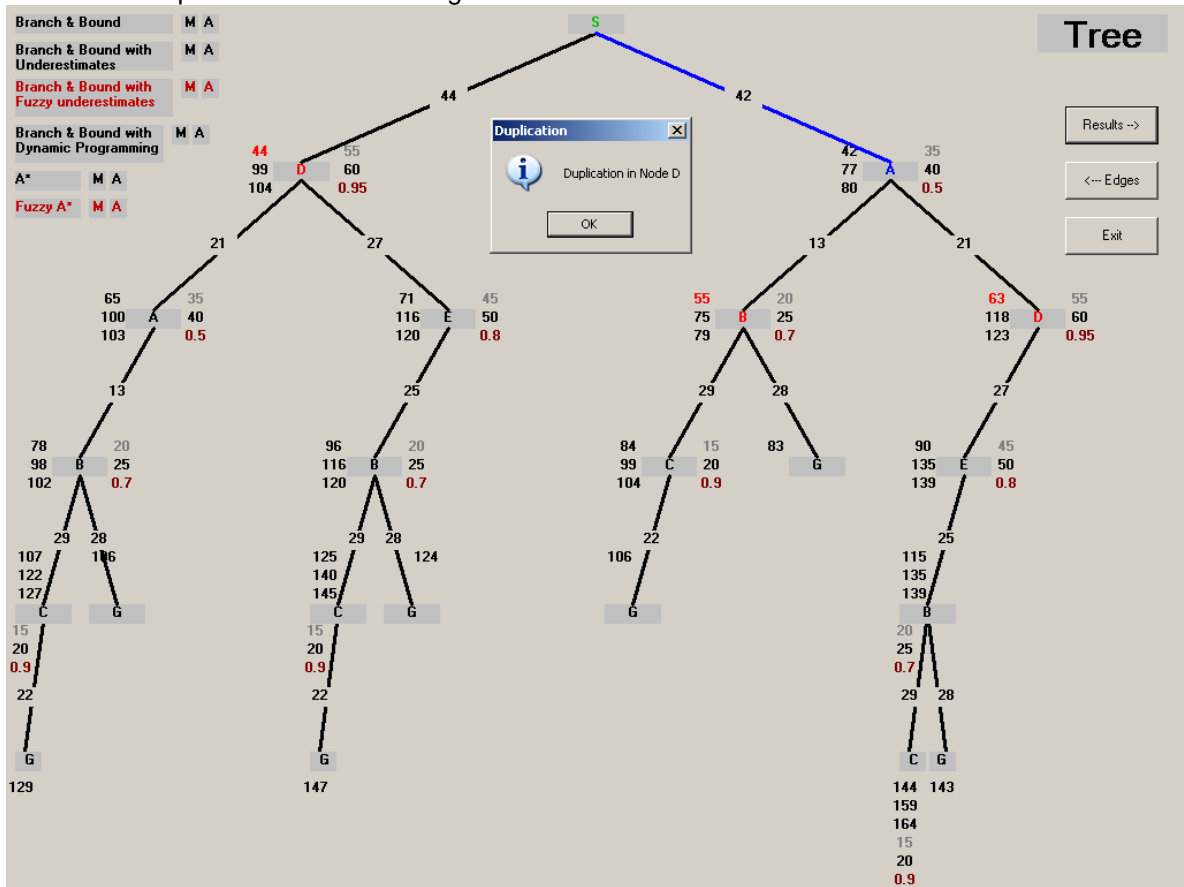
Figures (5.9 – 5.20) show the effect of using “Branch & Bound Search with dynamic programming”, on the map-traversal problem (Sample 1), where the red colored numbers beside the nodes denote the length of each path (cost) =  $d$  (already traveled), red colored nodes denote explored nodes, blue colored lines denote paths of reached nodes, dark red colored **X** denotes canceled nodes, and green colored lines denote the final path.

A short message will appear on each screen to inform users which node to be considered or to be canceled next. All paths are cut off quickly, leaving only the optimal path, S-D-E-F-G.

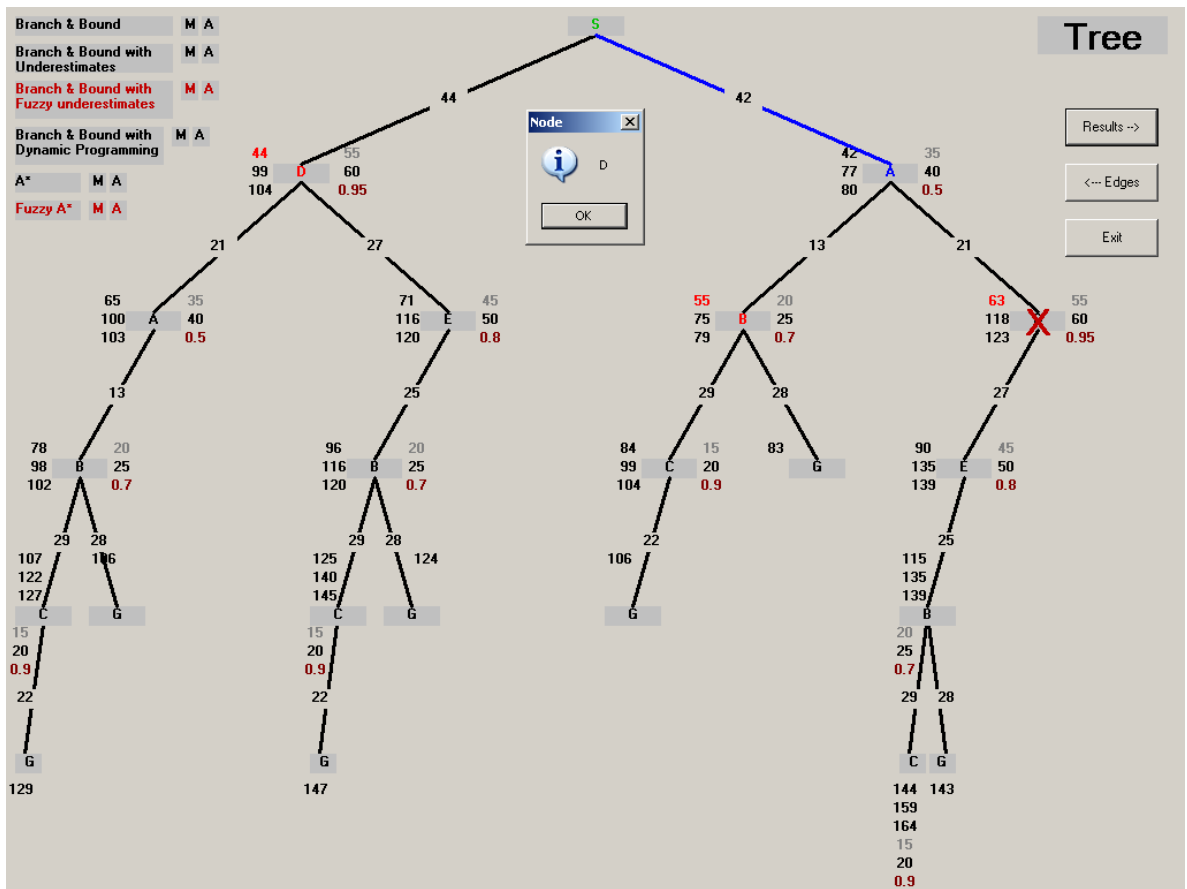




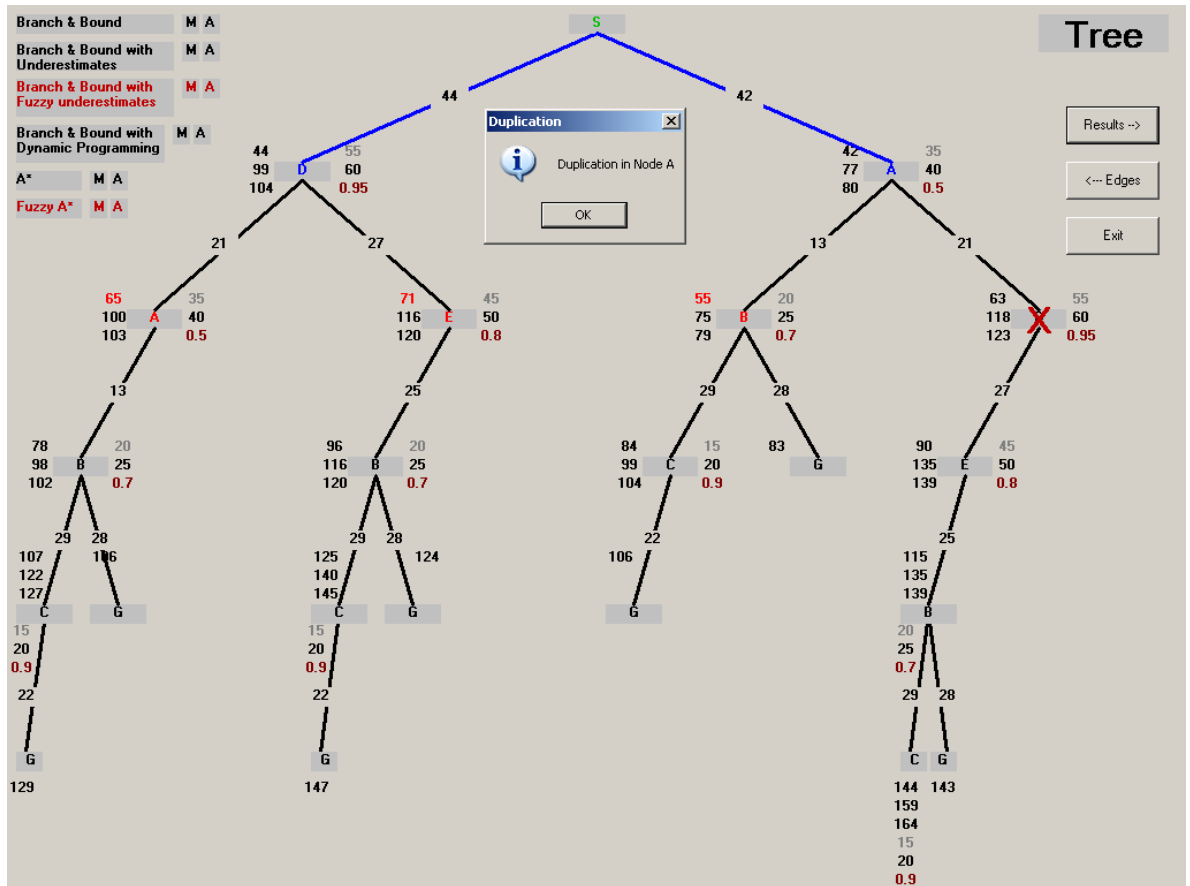
**Figure 5.9:** Tree Screen using Branch & Bound Search with dynamic programming, on the map-traversal problem (Sample 1). 1-In the first step, the red colored nodes (A and D) and the associated red colored numbers (42 and 44) denote the explored nodes and the length of each path. The Short message informs users that node A is to be considered next.



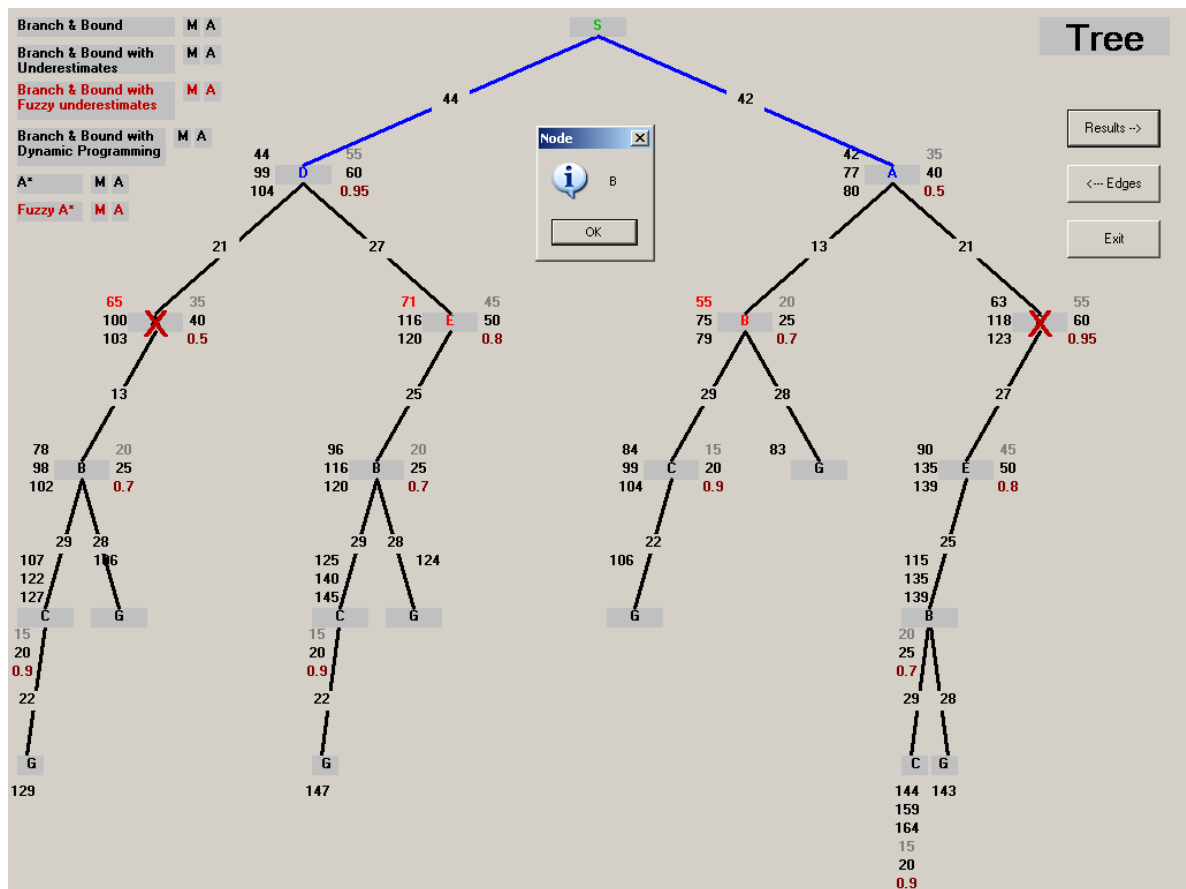
**Figure 5.10:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1). 2-Partial **path S-A** denoted by the **blue colored line** is therefore **selected for expansion** . The Short message informs users that node D is to be cancelled next.



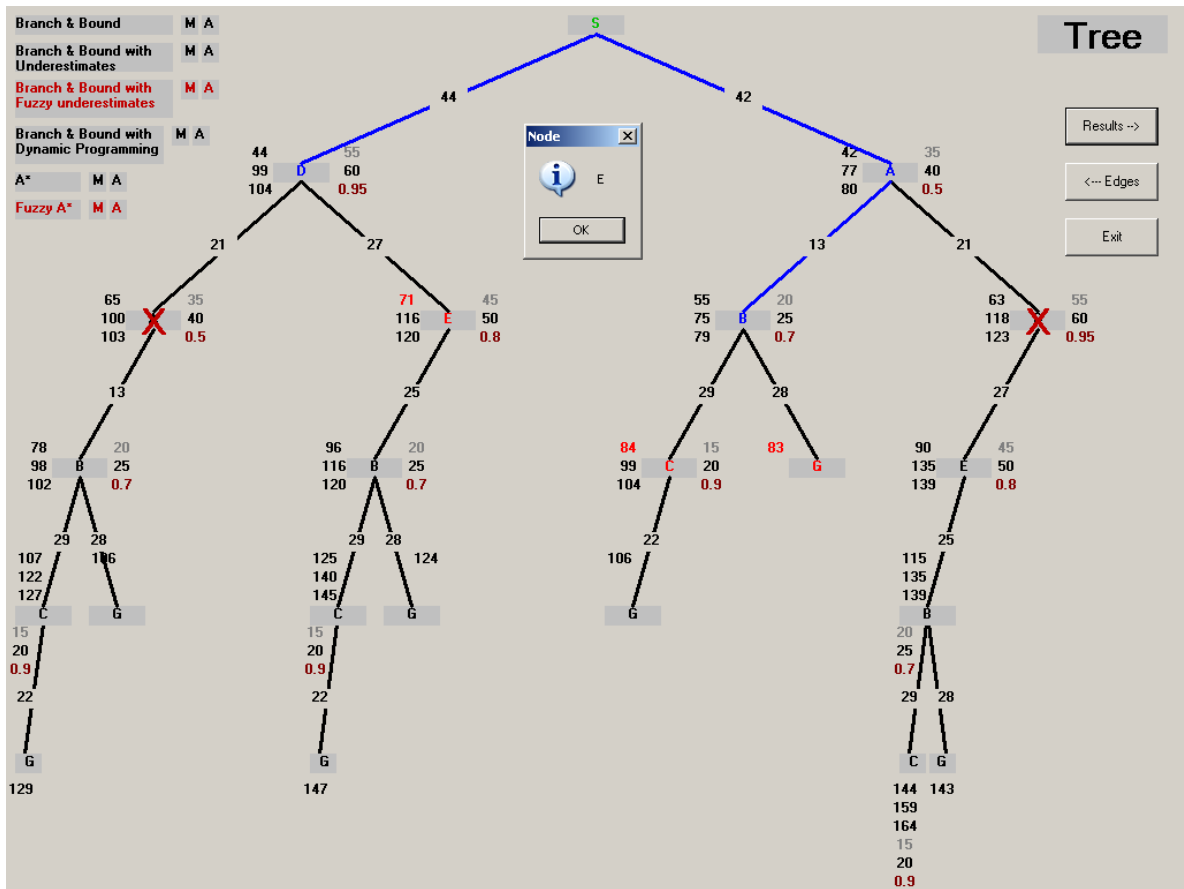
**Figure 5.11:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 3- Next, S-A-B and S-A-D are generated from S-A with partial path distances of 55 and 63, partial **path S-A-D** will be **deleted** because its partial-path distance of **63** is more than that of S-D (44). The **dark red colored X** denotes **canceled node**. The Short message informs users that node D is to be considered next.



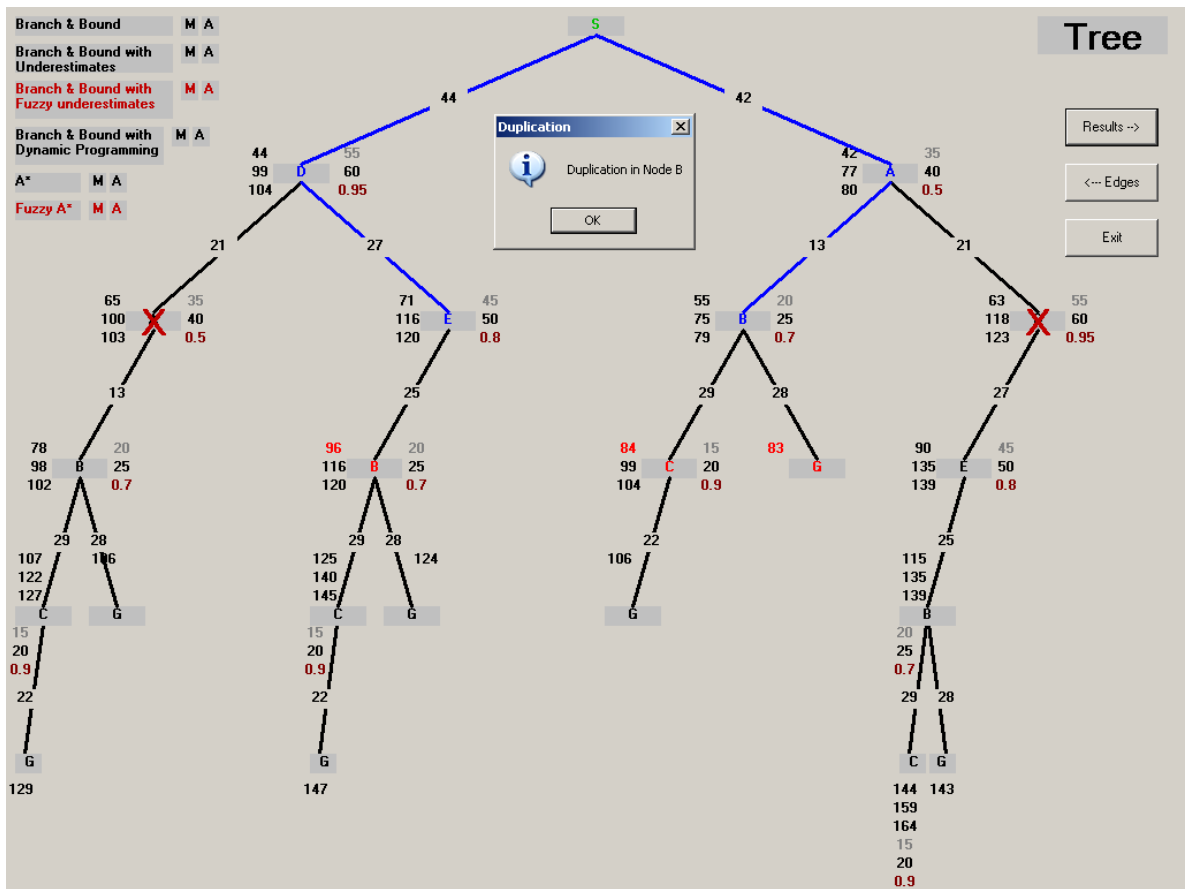
**Figure 5.12:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 4- Now S-D, with a partial path distance of 44, is expanded, leading to partial paths S-D-A and S-D-E. The Short message informs users that node A is to be cancelled next.



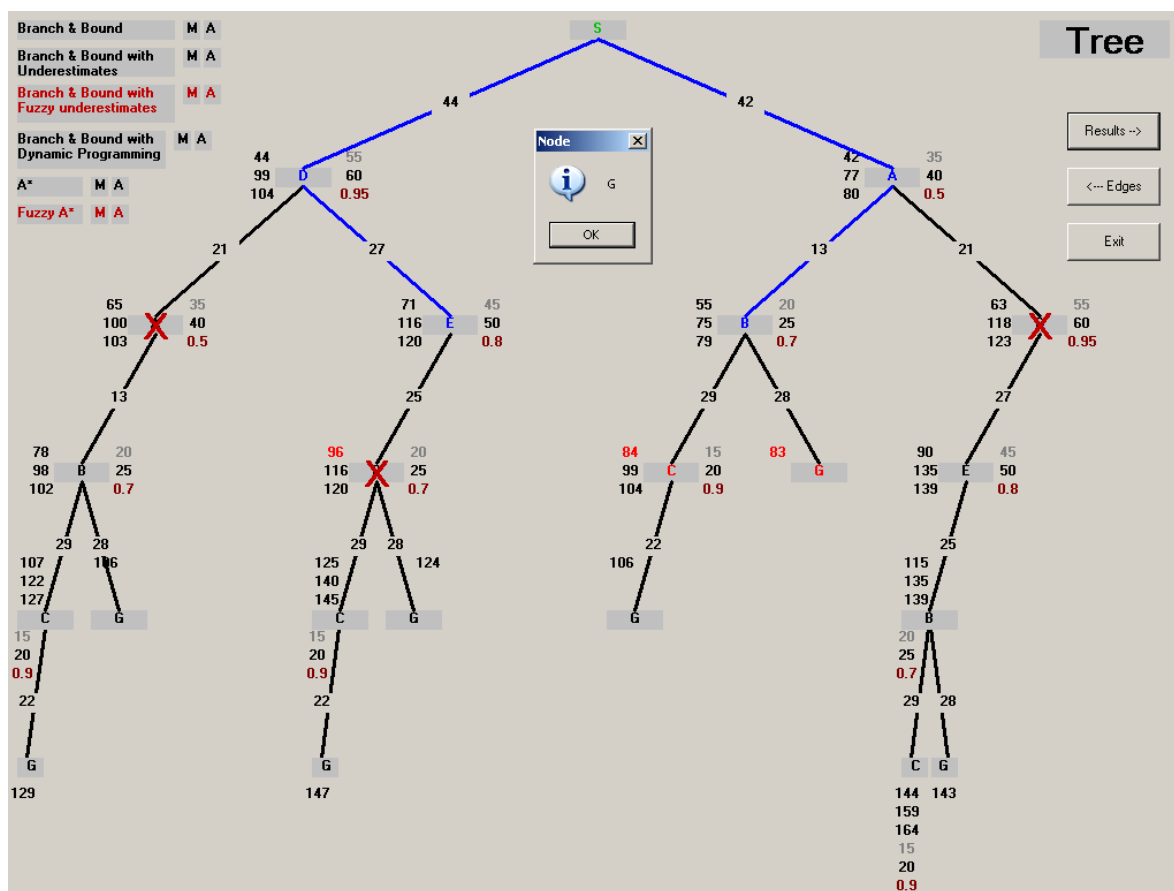
**Figure 5.13:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 5- At this point, partial path S-D-A can be deleted because its partial-path distance of 65 is more than that of S-A-B (55). The Short message informs users that node B is to be considered next.



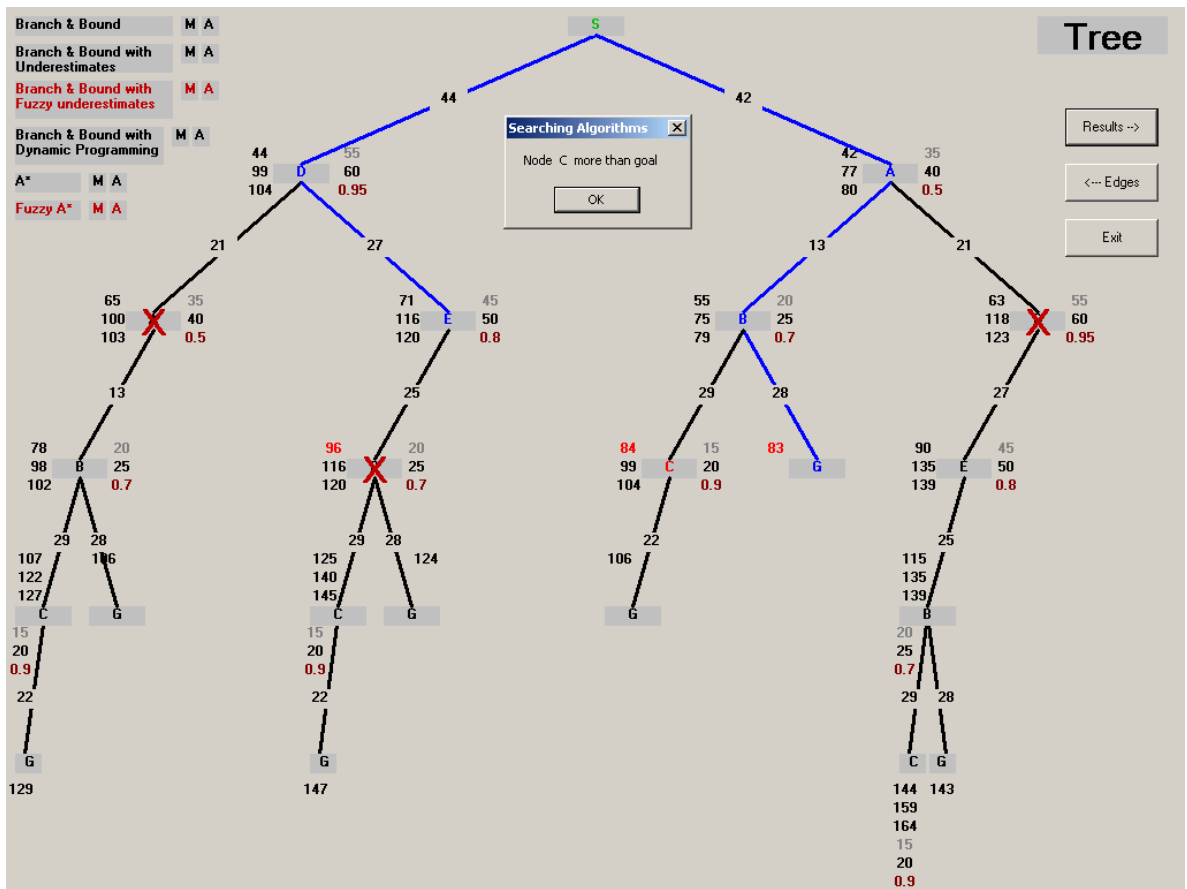
**Figure 5.14:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 6-Then expanding S-A-B , leads to S-A-B-C and S-A-B-G with partial path distances of 84, and 83. The Short message informs users that node E is to be considered next.



**Figure 5.15:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 7- Then expanding S-D-E , leads to S-D-E-B with partial path distance of 96. The Short message informs users that node B is to be cancelled next.

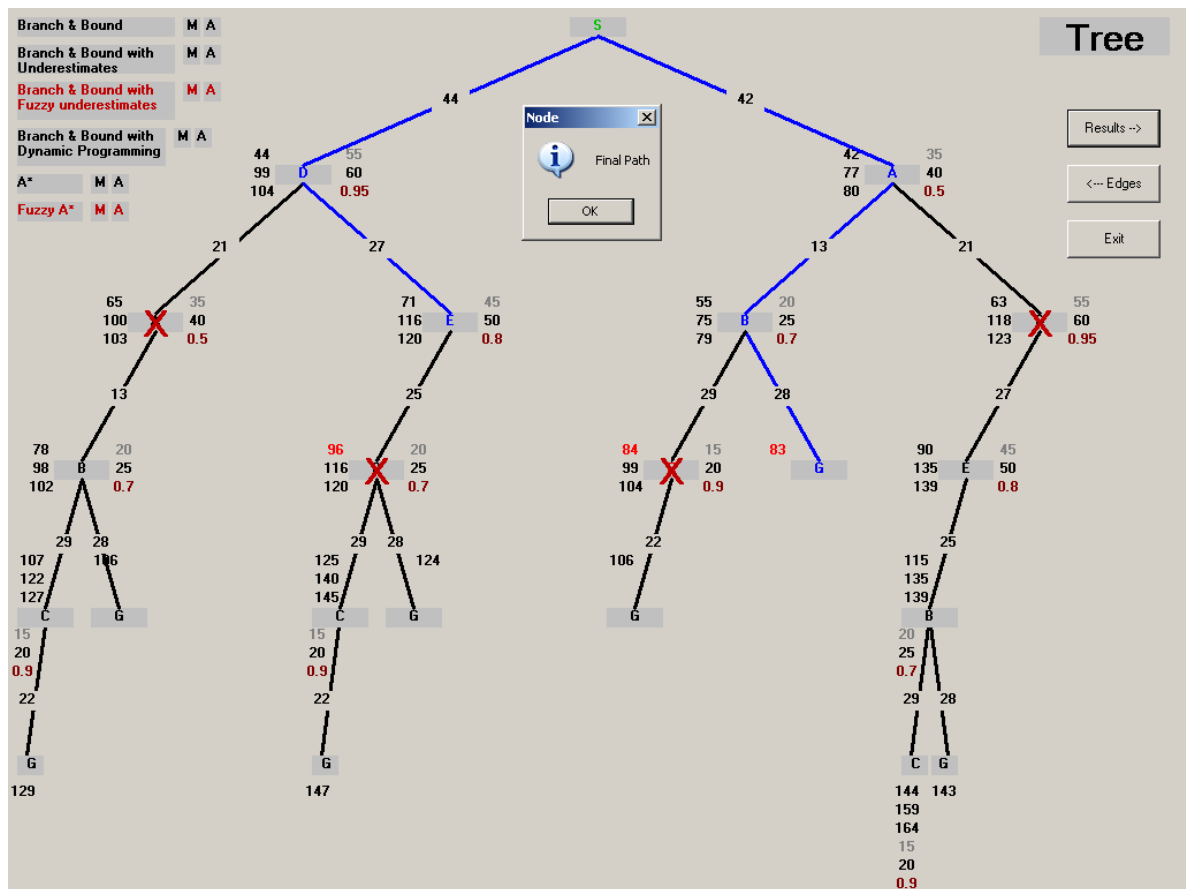


**Figure 5.16:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 8- Now S-D-E-B can be deleted because its partial-path distance of 96 is more than that S-A-B (55).The Short message informs users that node G is to be considered next.

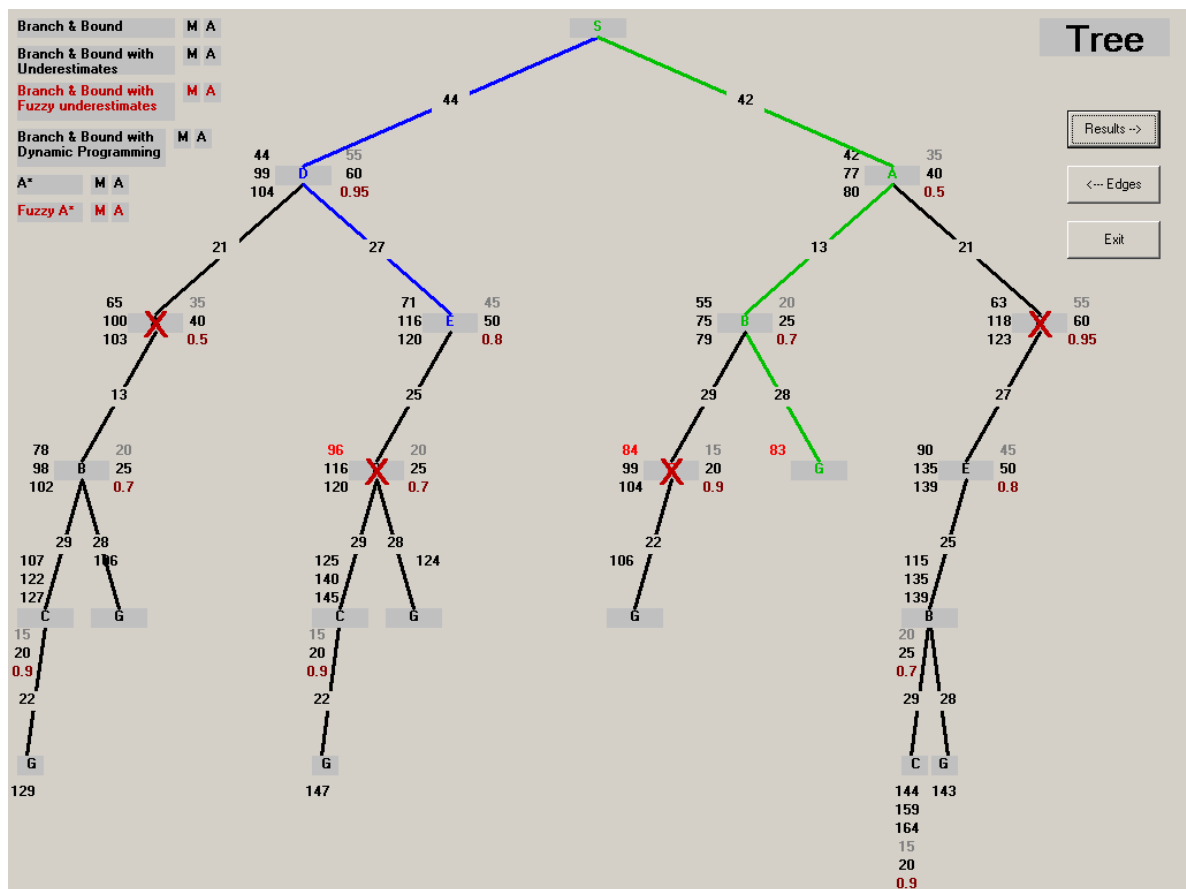


**Figure 5.17:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 9- Now the complete path S-A-B-G is generated from S-A-B. The Short message informs users that node C is to be cancelled next.





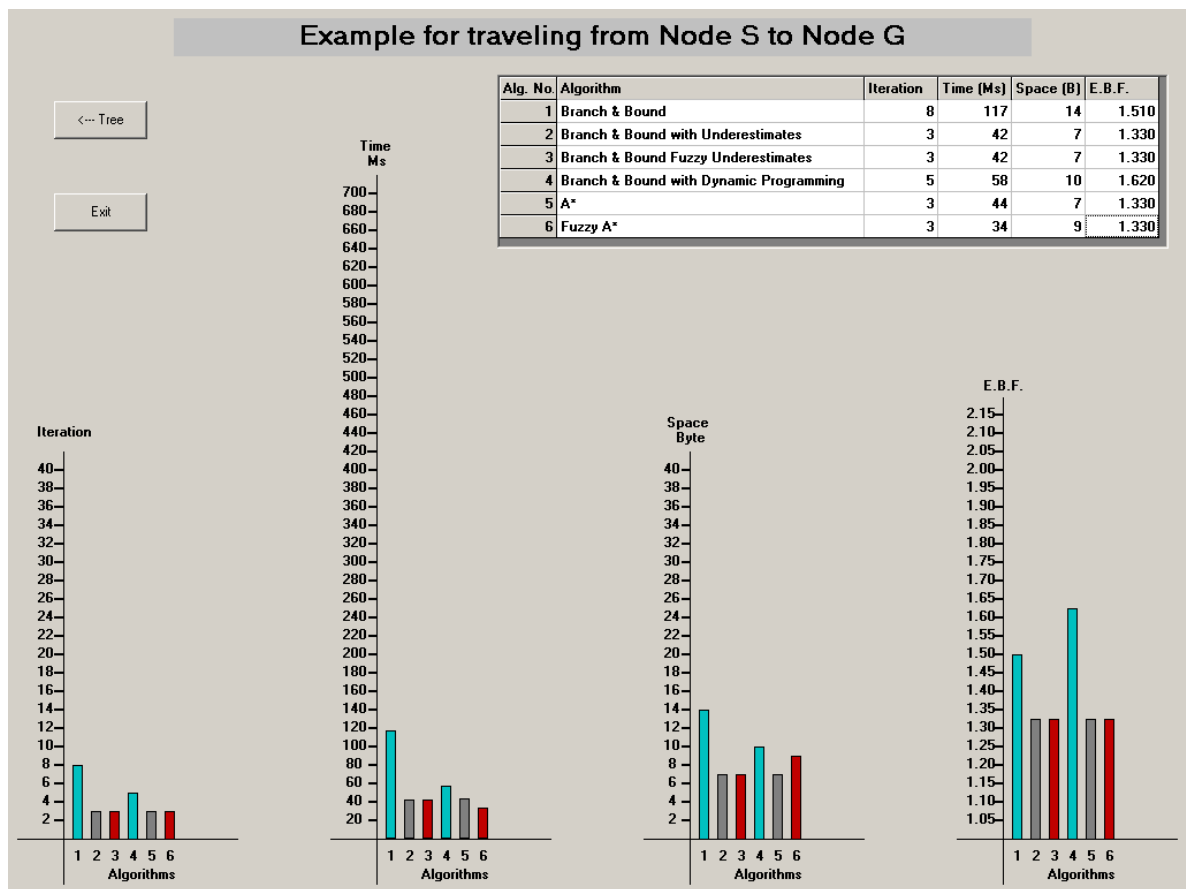
**Figure 5.18:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), 10-Then partial path S-A-B -C can be deleted because its partial-path distance of 84 is more than that of the shortest complete path S-A-B-G (83). The Short message informs users that the final path S-A-B-G is to be considered next.



**Figure 5.19:** Tree Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1), , 11- Finally S-A-B-G is the final shortest complete path because there are no other partial paths to be expanded, where the green colored line denotes the final path.

At the top right corner of the screen you can find three commands

On choosing (Results) command, the next screen (Results Screen) will display the results as shown in figure (5.20).



**Figure 5.20:** Results Screen using Branch & Bound Search with dynamic programming , on the map-traversal problem (Sample 1).

On completion of the manual run steps the six searching techniques will run automatically and the comparison results will be given in the result screen, where the six searching techniques will be analyzed by computing the Number of Iterations, Time Complexity (ms), Space Complexity (B), and Effective Branching Factor for each algorithm.

Results of these procedures will be displayed in a table and four bar charts as shown in figure (5.20).

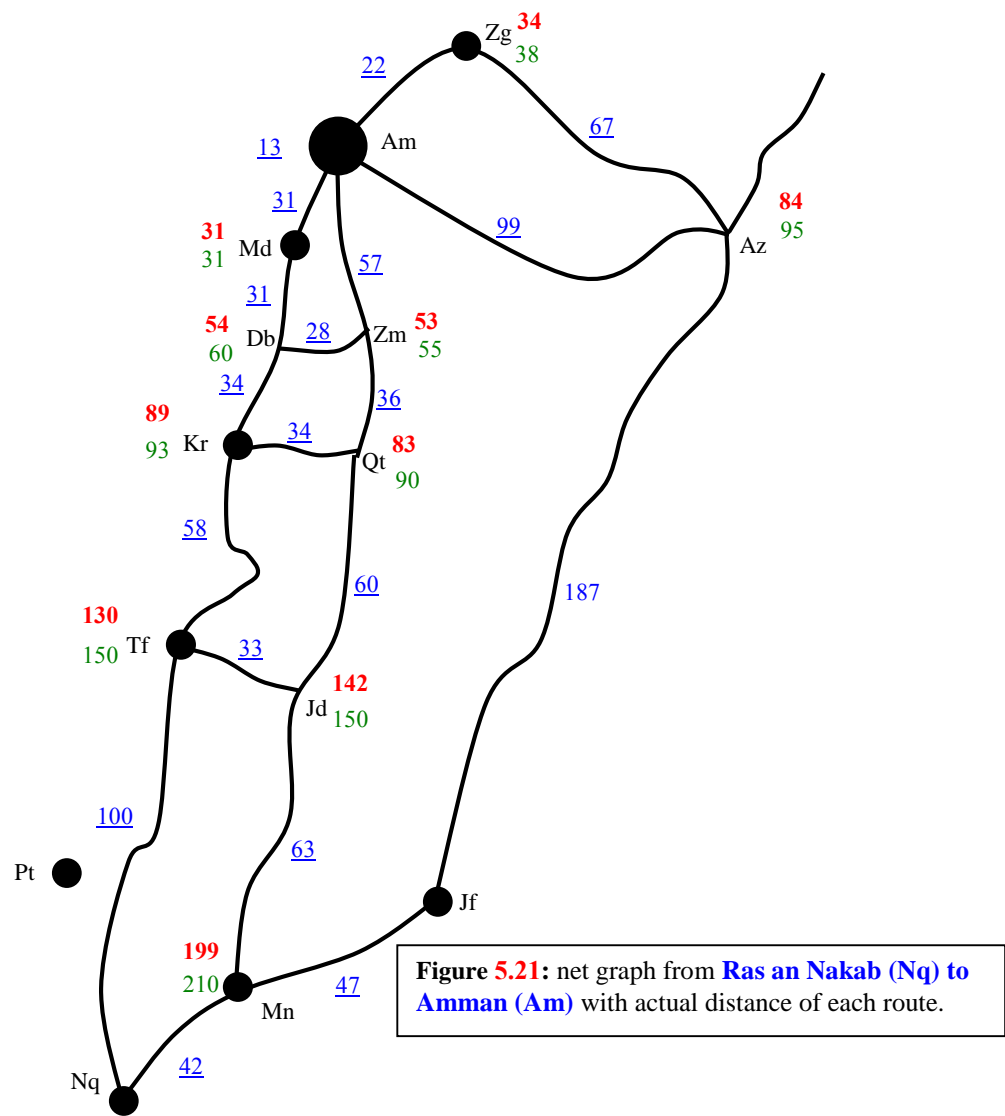
## Example Two:

In the following example, we will consider the following highway map as shown in figure (5.21) which represents the real life roads between two major Jordanian cities from **Ras an Nakab (Nq)** to **Amman (Am)** according to the official map of Jordan which is shown in figure (5.1), where one can plan a route from **Ras an Nakab** as start node (S) to **Amman** as goal node (G)

All possible routes are shown in the graph; the number against each edge gives the actual distance of that route (node to node) in some unit. The traveler has no knowledge about the distance information, but the traveler records the distance he completed.

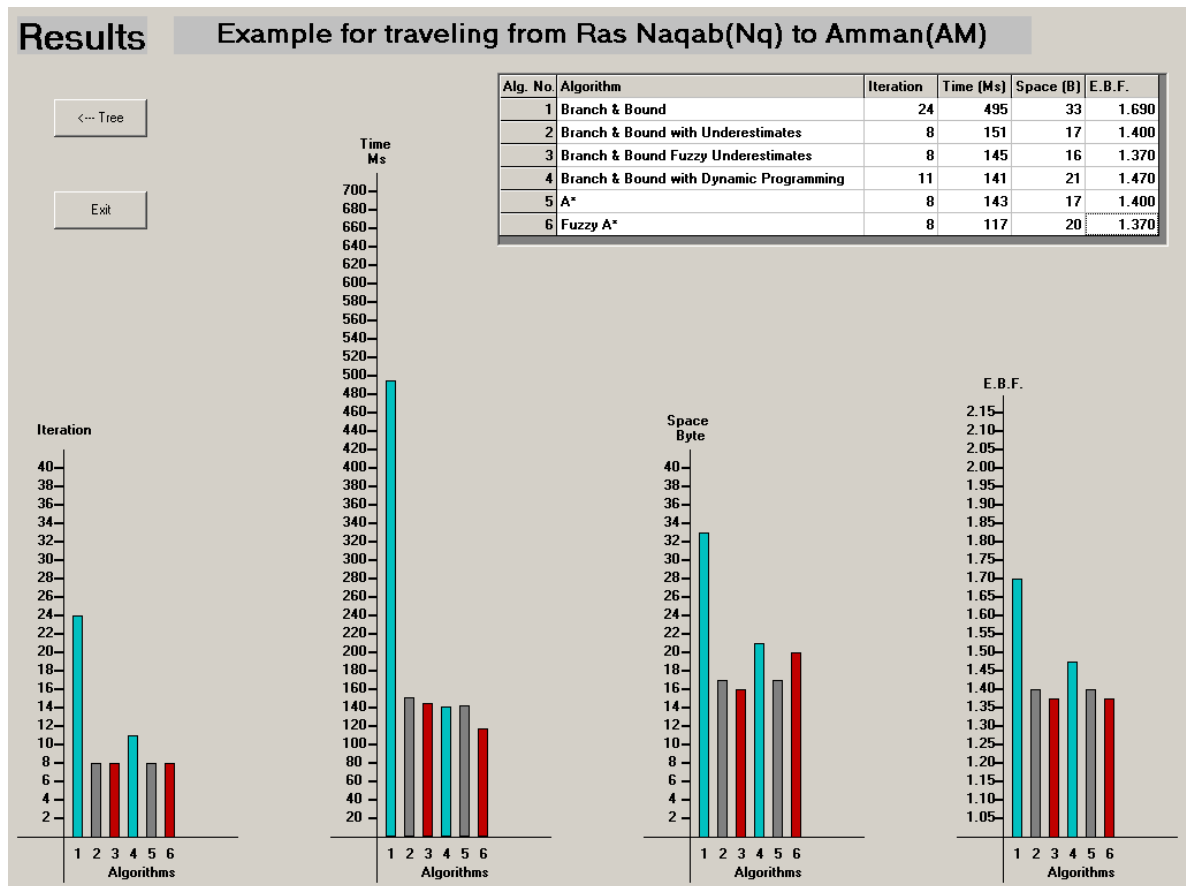
When the program starts, a splash screen will appear temporarily, then (Nodes Screen) will be displayed as shown previously in figures (5.3) and (5.4).

In this example (Sample 2: Jordanian Cities - From Ras Nakab to Amman) represented by the net graph of figure (5.21) has been chosen.



On choosing (Sample 2), data of net graph of Figure (5.21) will be loaded into the program, then all necessary steps were followed using the setup screens to activate the example.

The results graph of this run are as given in Figure 5.22.



**Figure 5.22:** Results Screen using Branch & Bound Search with Fuzzy underestimation, on the map-traversal problem (Sample 2).

### Example Three:

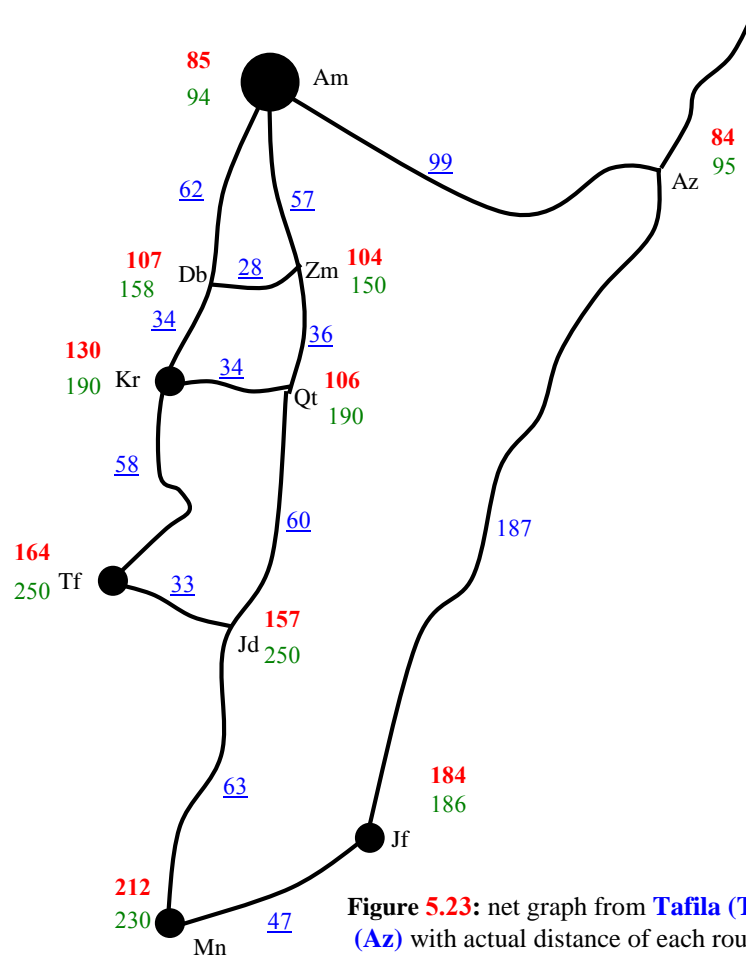
In the following example, we will consider the following highway map as shown in figure (5.23) which represents the real roads between two major Jordanian cities from **Tafila (Tf)** to **Azraq (Az)** according to the official map of Jordan which is shown in figure (5.1), where one can plan a route from **Tafila** as start node (S) to **Azraq** as goal node (G)

All possible routes are shown in the graph; the number against each edge gives the actual distance of that route (node to node) in some unit. The traveler has no knowledge about the distance information, but the traveler records the

distance he completed.

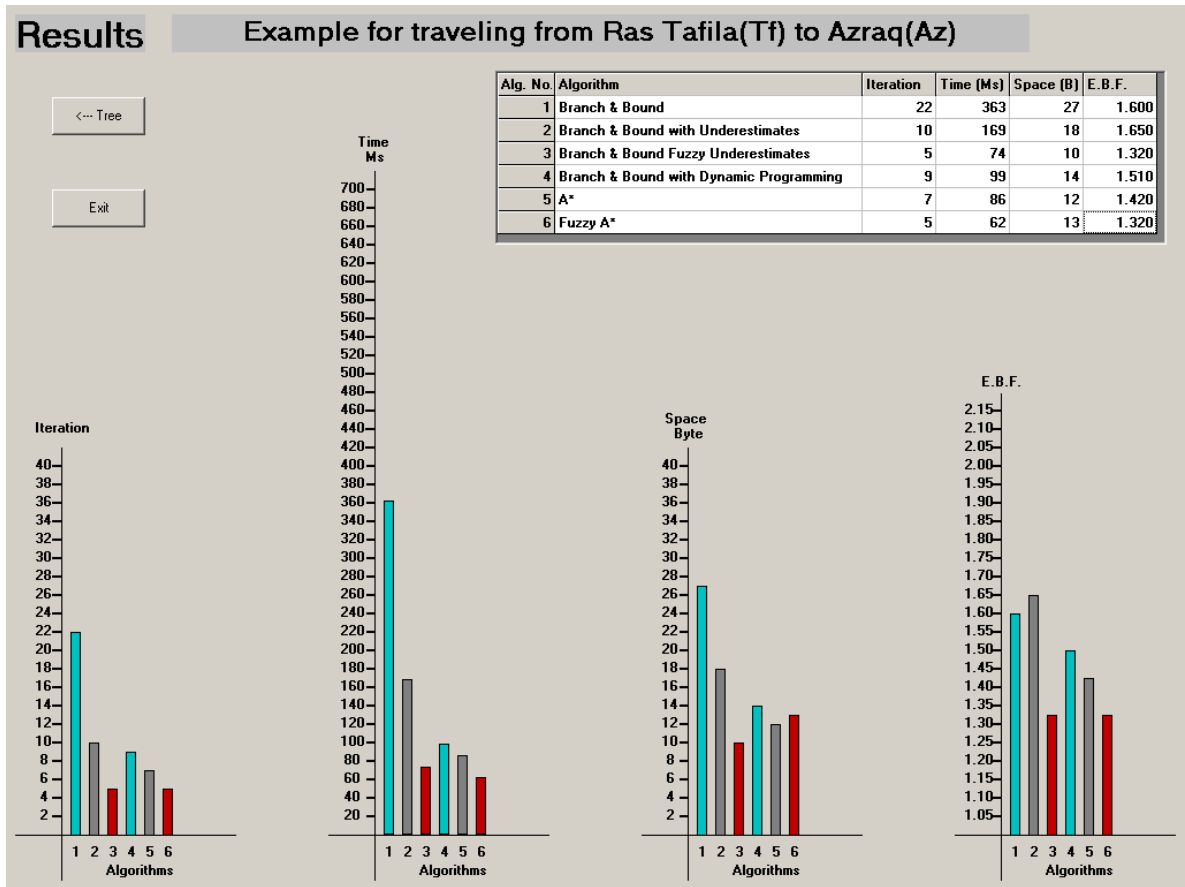
When the program starts, a splash screen will appear temporarily then (Nodes Screen) will be displayed as shown previously in figures (5.3) and (5.4).

In this example (Sample 5: Jordanian Cities - From **Tafila** to **Azraq**) represented by the net graph of figure (5.23) has been chosen.



On choosing (Sample 5), data of net graph of figure (5.23) will be loaded into the program, then all necessary steps were followed using the setup screens to activate the example.

The results graph of this run are as given bellow (figure 5.24):



**Figure 5.24:** Results Screen using A\* Search with Fuzzy underestimation, on the map-traversal problem (Sample 5).

### Example Four:

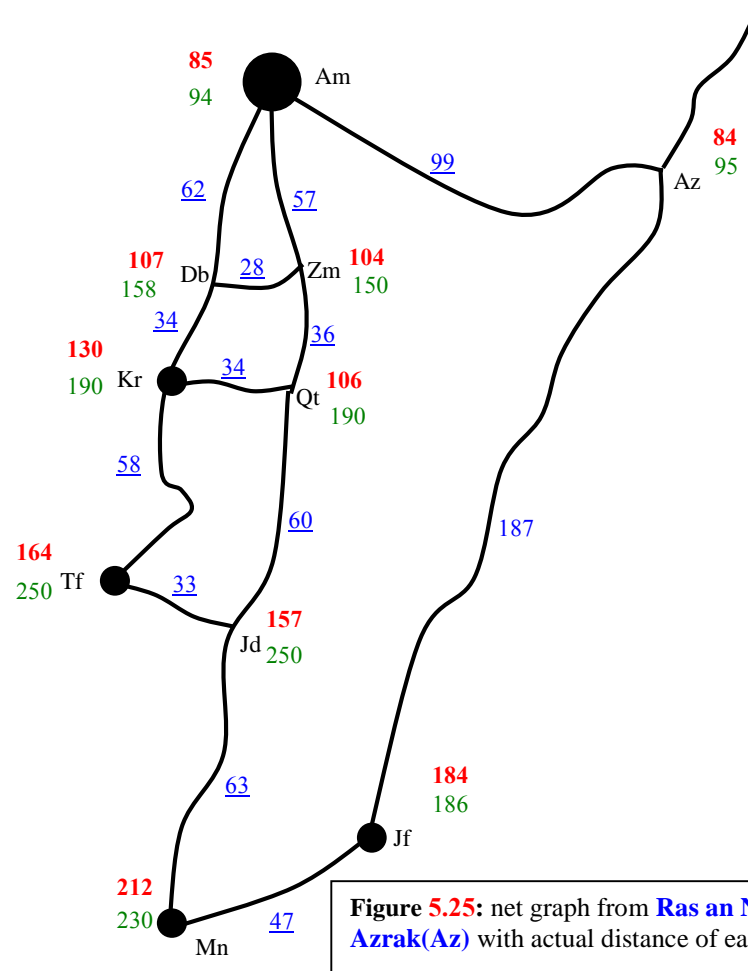
In the following example, we will consider the following highway map as shown in figure (5.25) which represents the real roads between two major Jordanian cities from **Ras an Nakab (Rn)** to **Azraq (Az)** according to the official map of Jordan which is shown in figure (5.1), where one can plan a route from **Ras an Nakab** as start node (S) to **Azraq** as goal node (G)



All possible routes are shown in the graph; the number against each edge gives the actual distance of that route (node to node) in some unit. The traveler has no knowledge about the distance information, but the traveler records the distance he completed.

When the program starts, a splash screen will appear temporarily then (Nodes Screen) will be displayed as shown previously in figures (5.3) and (5.4).

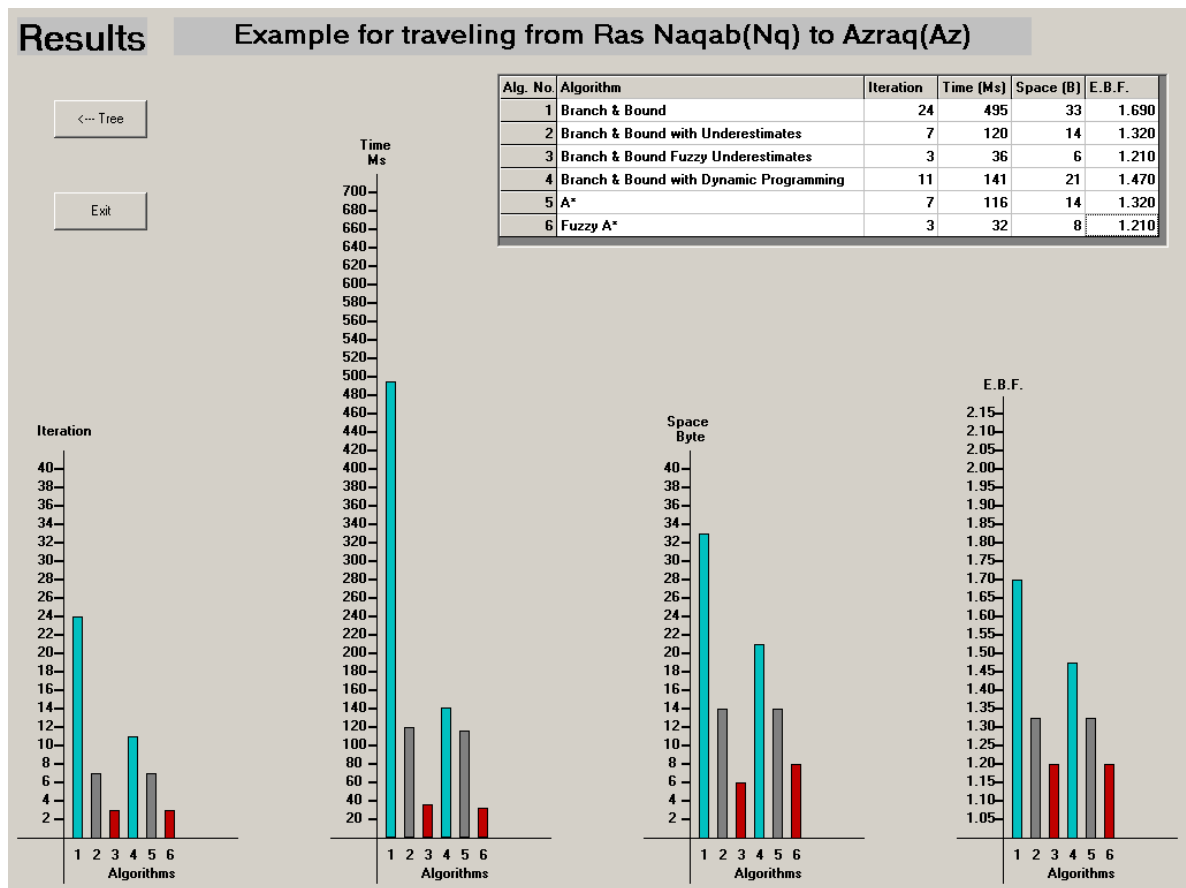
In this example (Sample 5: Jordanian Cities - From **Ras Nakab** to **Azraq**) represented by the net graph of figure (5.25) has been chosen.



**Figure 5.25:** net graph from **Ras an Nakab (Nq)** to **Azrak(Az)** with actual distance of each route.

On choosing (Sample 3), data of net graph of figure (5.25) will be loaded into the program, then all necessary steps were followed using the setup screens to activate the example.

The results graph of this run are as given bellow (figure 5.26):



**Figure 5.26:** Results Screen using Branch and Bound Search with Fuzzy underestimation, on the map-traversal problem (Sample 3).

### Example Five:

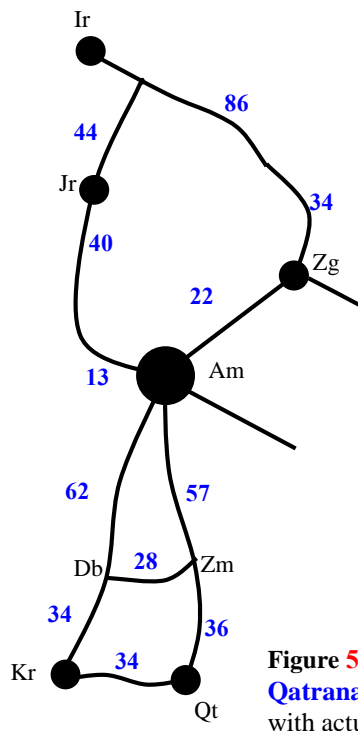
In the following example, we will consider the following highway map as shown in figure (5.27) which represents the real roads between two major Jordanian cities from **Qatrana (Qt)** to **Irbid (Ir)** according to the official map of Jordan which is shown in figure (5.1), where one can plan a route from **Qatrana**

as start node (S) to **Irbid** as goal node (G)

All possible routes are shown in the graph; the number against each edge gives the actual distance of that route (node to node) in some unit. The traveler has no knowledge about the distance information, but the traveler records the distance he completed.

When the program starts, a splash screen will appear temporarily then (Nodes Screen) will be displayed as shown previously in figures (5.3) and (5.4).

In this example (Sample 4: Jordanian Cities - From **Qatrana** to **Irbid**) represented by the net graph of figure (5.27) has been chosen.



**Figure 5.27:** net graph from **Qatrana (Qt)** to **Irbid (Ir)** with actual distance of each route.

On choosing (Sample 4), data of net graph of figure (5.27) will be loaded into the program, then all necessary steps were followed using the setup screens to activate the example.

The results graph of this run are as given bellow (figure 5.28):

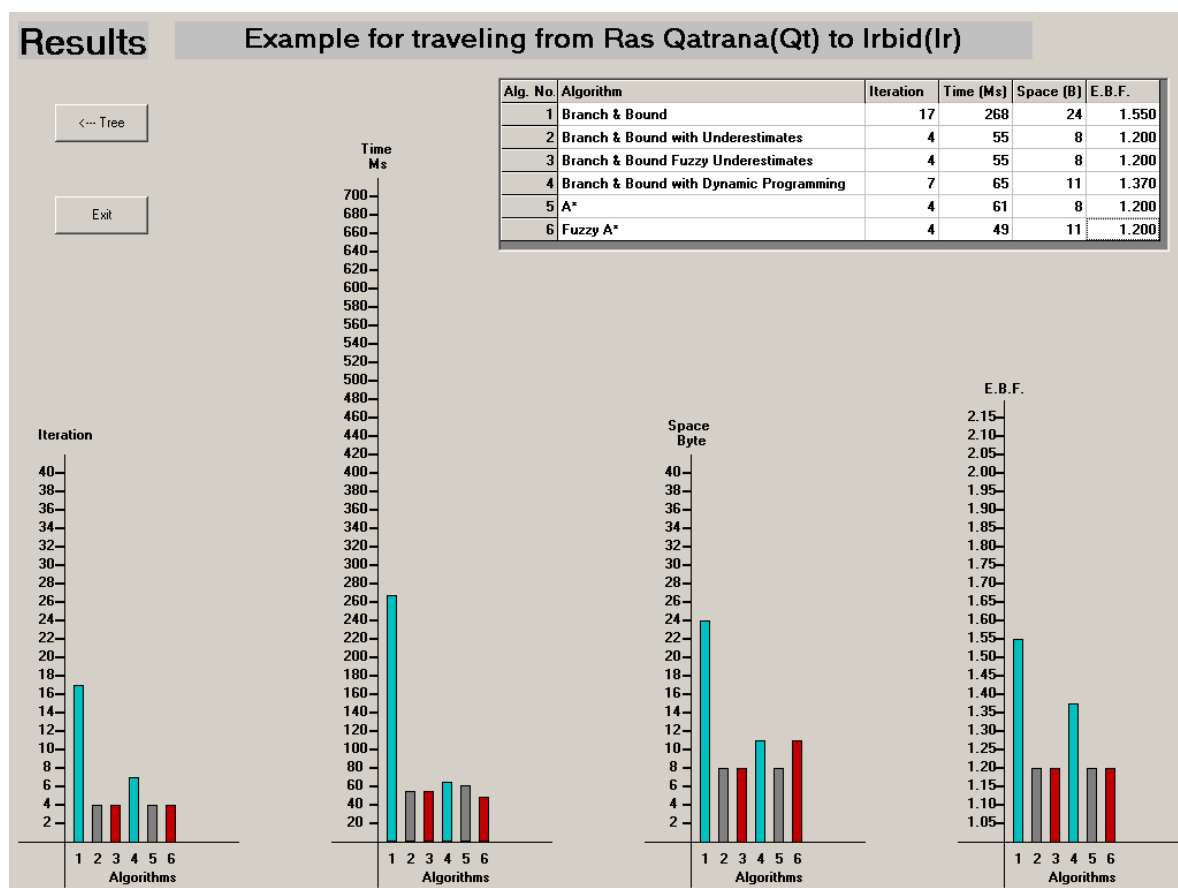


Figure 5.28: Results Screen using A\* Search, on the map-traversal problem (Sample 4).

### Example Six:

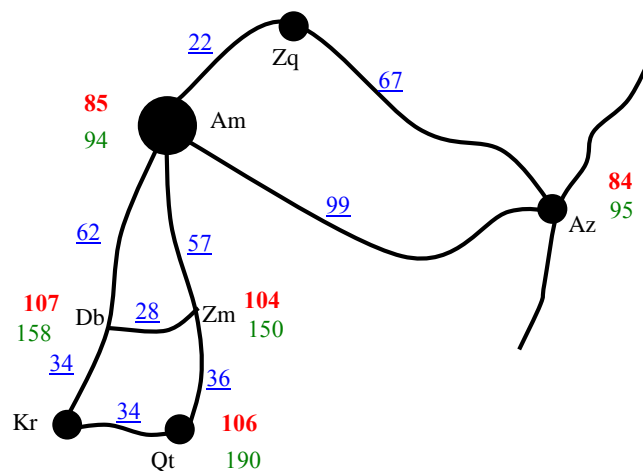
In the following example, we will consider the following highway map as shown in figure (5.29) which represents the real roads between two major Jordanian cities from **Karak (Kr)** to **Azraq (Az)** according to the official map of Jordan which is shown in figure (5.1), where one can plan a route from **Karak** as

start node (S) to **Azraq** as goal node (G)

All possible routes are shown in the graph; the number against each edge gives the actual distance of that route (node to node) in some unit. The traveler has no knowledge about the distance information, but the traveler records the distance he completed.

When the program starts, a splash screen will appear temporarily then (Nodes Screen) will be displayed as shown previously in figures (5.3) and (5.4).

In this example (Sample 15): Jordanian Cities - From **Karak** to **Azraq**) represented by the net graph of figure (5.29) has been chosen.



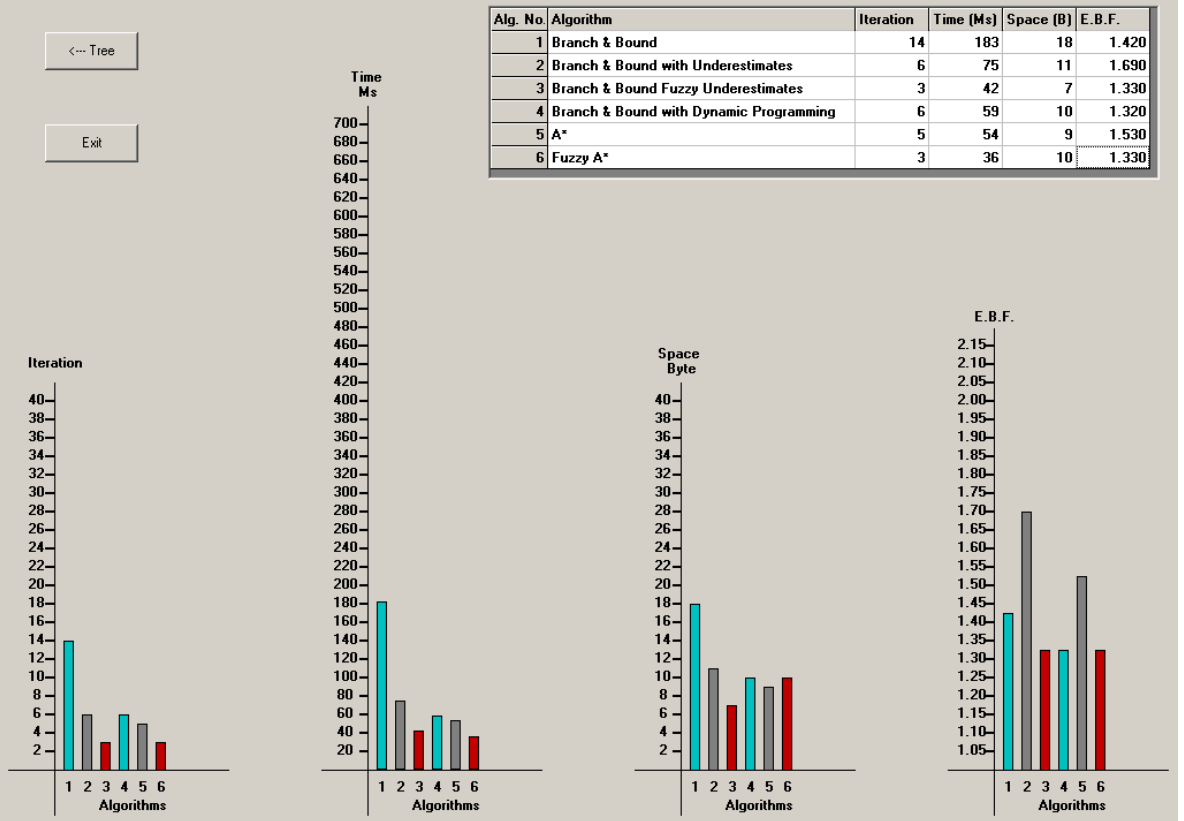
**Figure 5.29:** net graph from **Karak (Kr)** to **Azraq (Az)** with actual distance of each route.

On choosing (Sample 15), data of net graph of figure (5.29) will be loaded into the program, then all necessary steps were followed using the setup screens to activate the example.

The results graph of this run are as given bellow (figure 5.30):

# Results

## Example for traveling from Kerak(Kr) to Azraq(Az)



**Figure 5.30:** Results Screen using Branch and Bound Search with Crisp Underestimation, on the map-traversal problem (Sample 15).

### 5.3. Simulation Survey:

The following tables and line charts show results when executing the program for different examples; according to an official map of Jordan, which was shown in Figure (5.1).

Six Searching techniques were analyzed by computing the Number of Iterations, Time Complexity, Space Complexity, and Effective Branching Factor for each algorithm.

The analysis process is carried out for six searching techniques, which are Branch & Bound, Branch & Bound with Underestimation, Branch & Bound with dynamic programming, A\*, Branch & Bound with Fuzzy Underestimation, and A\* with Fuzzy Underestimation.

- **Number of Iterations:** Table(5.1) shows **Number of Iterations** for six Searching techniques when the simulation was run for different examples.

**Table 5.1:** Comparison of Number of Iterations for various search techniques, where **B&B** denotes Branch & Bound, **DB&B** denotes Branch & Bound with dynamic programming, **UB&B** denotes Branch & Bound with Underestimation, **FB&B** denotes Branch & Bound with Fuzzy Underestimation, **A\*** denotes A\*, and **FA\*** denotes A\* with Fuzzy Underestimation.

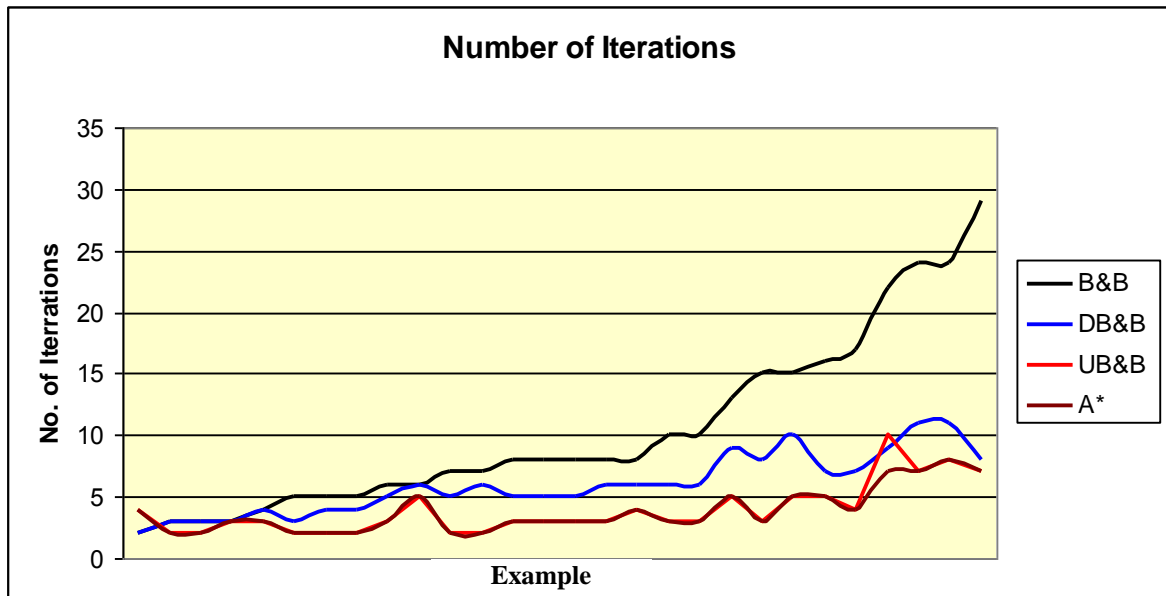
Number of Iterations						
Example	Name of search technique					
	B&B	DB&B	UB&B	FB&B	A*	FA*
S-G	8	5	3	3	3	3
Nq-Am	24	11	8	8	8	8
Nq-AZ	24	11	7	3	7	3
Qt-Ir	17	7	4	4	4	4
Tf-Az	22	9	10	5	7	5
Ir-Am	6	6	5	4	5	4
Rm-Am	4	4	3	3	3	3
Mn-Am	13	9	5	5	5	5
Jf-Am	15	10	5	5	5	5
Jd-Am	10	6	3	3	3	3
Tf-Am	10	6	3	3	3	3
Zm-Ir	8	5	3	3	3	3
Db-Ir	8	5	3	3	3	3
Kr-Ir	16	7	5	4	5	4
kr-Az	145	6	6	3	5	3
Jr-Az	7	5	2	2	2	2
Jd-Az	8	6	4	4	4	4
Db-Az	7	6	2	2	2	2
Ir-Az	5	4	2	2	2	2
Az-Ir	5	3	2	2	2	2
Tf-Ir	29	8	7	5	7	5
Jf-Ir	15	8	3	3	3	3
Am-Ir	3	3	2	2	2	2
Kr-Am	6	5	3	3	3	3
Zm-Am	2	2	4	4	4	4
Qt-Am	5	4	2	2	2	2
Db-Am	3	3	2	2	2	2
Az-Zg	3	3	3	3	3	3
Md-Ir	8	6	3	3	3	3



The following line chart shows comparison of Number of Iterations for various search techniques according to different independent examples.

Figure 5.31 shows that Number of iterations for B&B increases as number of expanded nodes has been increased.

**Figure 5.31 (Line chart):** Comparison of Number of Iterations for four crisp search techniques.



- **Time Complexity:** Table(5.2) shows **Time Complexity** (ms)for six Searching techniques when the simulation was run for different examples.

**Table 5.2:** Comparison of Time Complexity for various search techniques.

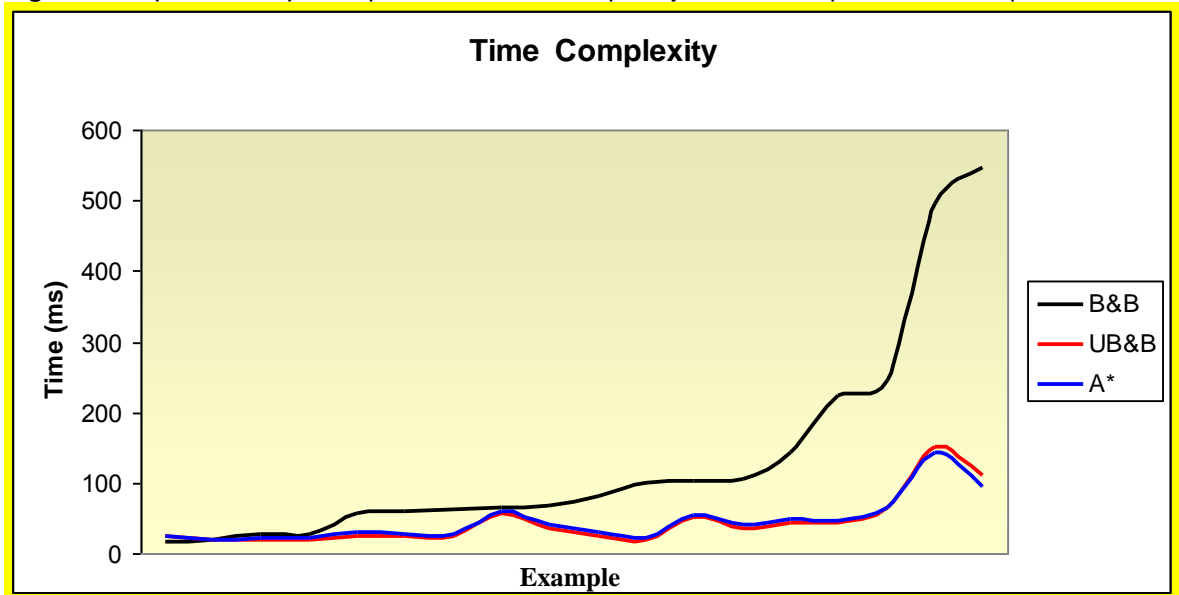
Search time (ms)						
Example	Name of search technique					
	B&B	DB&B	UB&B	FB&B	A*	FA*
S-G	117	58	42	42	44	34
Nq-Am	495	141	151	145	143	117
Nq-AZ	495	141	120	36	116	32
Qt-Ir	269	65	55	55	61	49
Tf-Az	363	99	169	74	86	62
Ir-Am	65	66	56	47	60	43
Rm-Am	35	35	28	28	30	26
Mn-Am	210	101	68	68	76	60
Jf-Am	196	114	70	70	78	62
Jd-Am	143	62	42	42	48	36
Tf-Am	141	59	42	42	48	36
Zm-Ir	91	45	36	36	40	42
Db-Ir	91	45	36	36	40	32
Kr-Ir	245	66	64	55	64	49
kr-Az	183	59	75	42	54	36
Jr-Az	82	52	25	25	29	21
Jd-Az	103	67	51	43	53	37
Db-Az	100	75	19	19	21	17
Ir-Az	62	46	25	25	27	21
Az-Ir	60	33	25	25	27	21
Tf-Ir	546	82	110	76	94	62
Jf-Ir	224	80	42	42	46	36
Am-Ir	26	27	19	19	21	17
Kr-Am	67	48	36	36	40	32
Zm-Am	17	17	25	25	25	25
Qt-Am	56	38	25	25	29	21
Db-Am	26	27	19	19	21	17
Az-Zg	18	20	18	18	18	18
Md-Ir	105	64	36	36	40	32

The following line charts show comparisons of **Time Complexity** for various search techniques according to different independent examples, where Time complexity for all search techniques increases as number of expanded nodes has been increased.

Figure 5.32 shows that UB&B and A\* achieve less Time complexity than B&B search technique.

Figure 5.33 shows that FB&B and FA\* achieve less Time complexity than UB&B and A\* search techniques for all examples.

**Figure 5.32 (Line chart):** Comparison of Time Complexity for three crisp search techniques.



**Figure 5.33 (Line chart):** Comparison of Time Complexity for two crisp and two fuzzy search techniques.

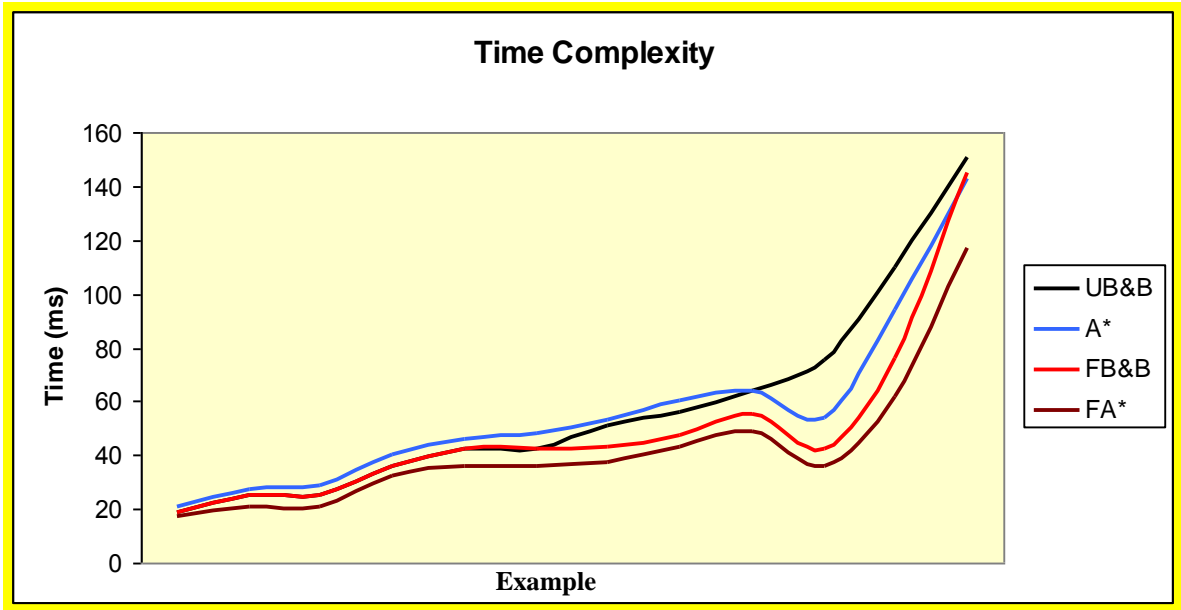


Figure 5.34 shows that FB&B achieves less Time complexity than UB&B search technique, while figure 5.35 shows that FA\* achieves less Time complexity than A\* search technique for all examples.

Figure 5.34 (Line chart): Comparison of Time Complexity for two search techniques.

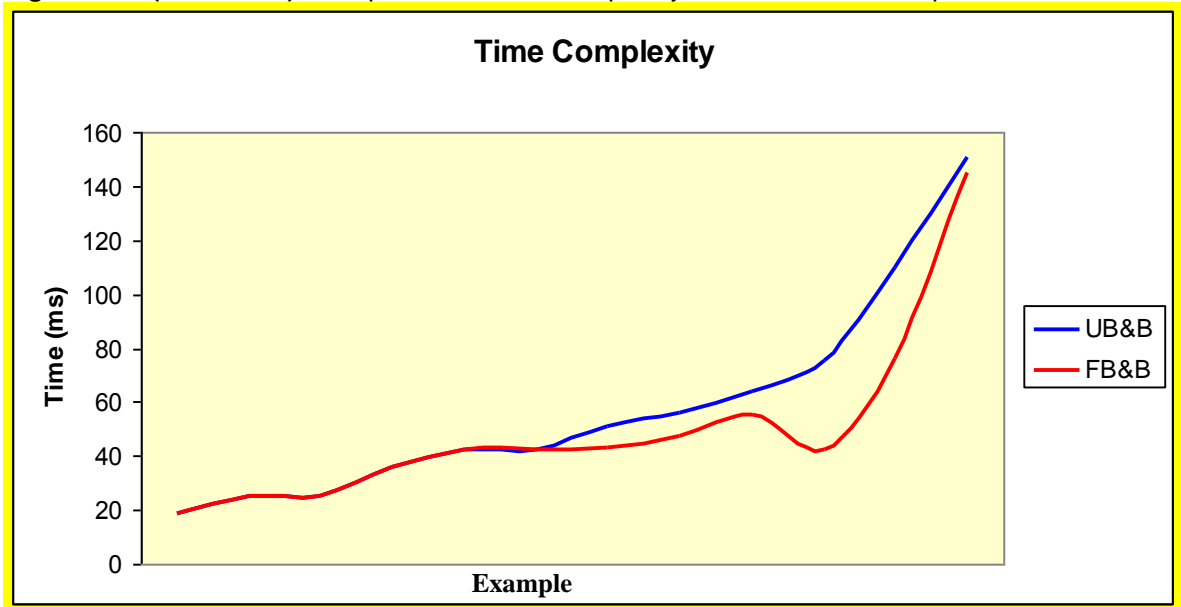
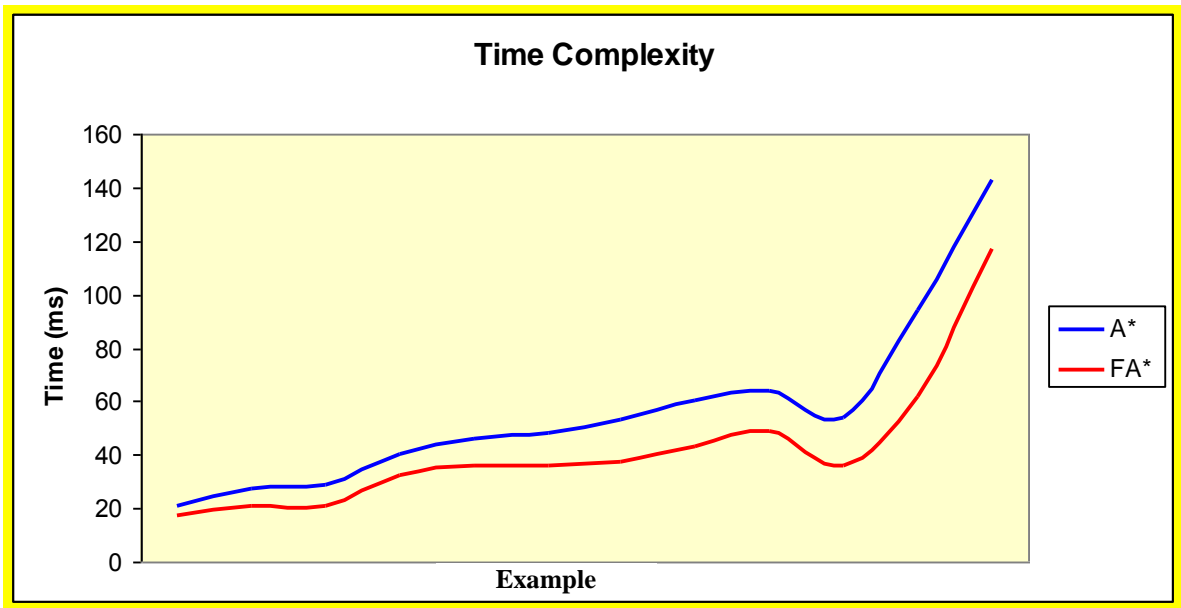


Figure 5.35 (Line chart): Comparison of Time Complexity for two search techniques.



- **Space Complexity:** Table (5.3) shows **Space Complexity** (Byte) for six Searching techniques when the simulation was run for different examples.

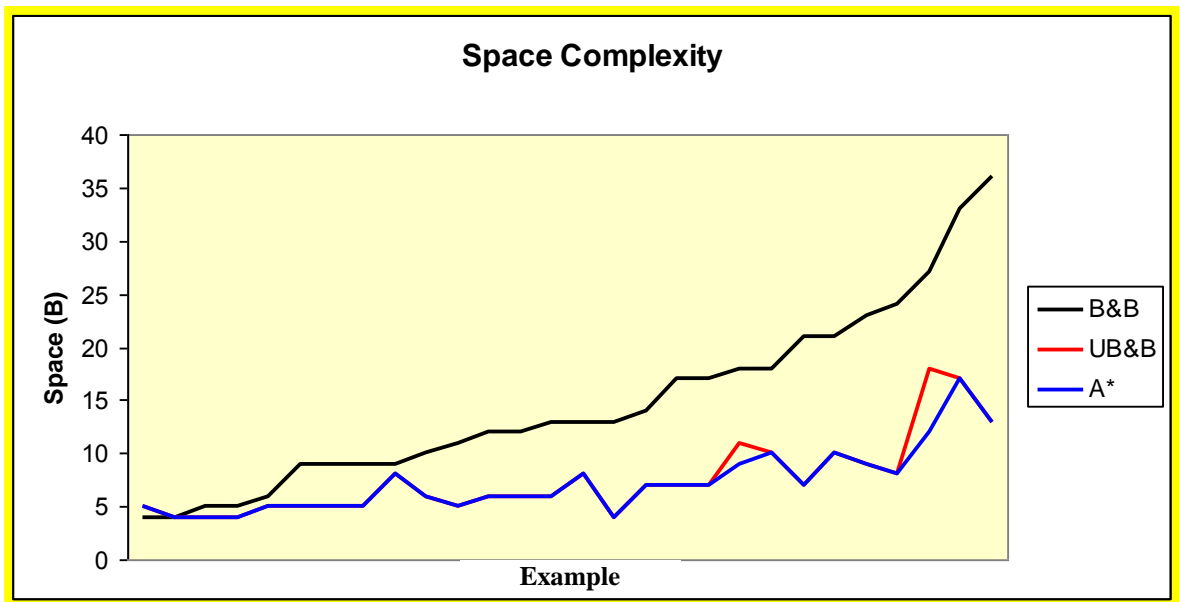
**Table 5.3:** Comparison of **Space Complexity** for various search techniques.

Search space(Byte)						
Example	Name of search technique					
	B&B	DB&B	UB&B	FB&B	A*	FA*
S-G	14	10	7	7	7	9
Nq-Am	33	21	17	16	17	20
Nq-AZ	33	21	14	16	14	8
Qt-Ir	24	11	8	8	8	11
Tf-Az	27	14	18	10	12	13
Ir-Am	9	9	8	7	8	9
Rm-Am	6	6	5	5	5	6
Mn-Am	21	16	10	10	10	14
Jf-Am	18	16	10	10	10	14
Jd-Am	17	11	7	7	7	10
Tf-Am	17	10	7	7	7	10
Zm-Ir	12	8	6	6	6	8
Db-Ir	12	8	6	6	6	8
Kr-Ir	23	11	9	8	9	11
kr-Az	18	10	11	7	9	10
Jr-Az	11	8	5	5	5	7
Jd-Az	13	10	8	7	8	8
Db-Az	13	12	4	4	4	5
Ir-Az	9	9	5	5	5	6
Az-Ir	9	6	5	5	5	6
Tf-Ir	36	13	13	10	13	13
Jf-Ir	21	12	7	7	7	9
Am-Ir	5	5	4	4	4	5
Kr-Am	10	8	6	6	6	8
Zm-Am	4	4	5	5	5	5
Qt-Am	9	7	5	5	5	7
Db-Am	5	5	4	4	4	5
Az-Zg	4	5	4	4	4	4
Md-Ir	13	10	6	6	6	8

The following line charts show comparisons of **Space Complexity** for various search techniques according to different independent examples, where Space complexity for all search techniques increases as number of expanded nodes has been increased.

Figure 5.36 shows that UB&B and A\* achieve less Space complexity than B&B search technique, and figure 5.37 shows that FB&B achieves less Space complexity than UB&B search technique for all examples.

**Figure 5.36 (Line chart):** Comparison of Space Complexity for three crisp search techniques.



**Figure 5.37 (Line chart):** Comparison of **Space Complexity** for two search techniques.

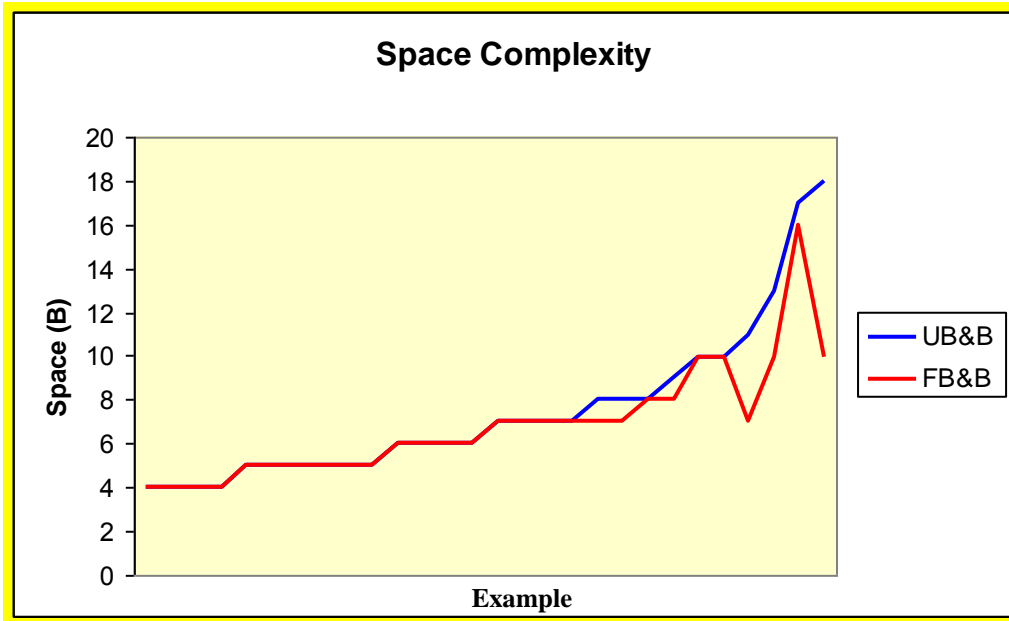
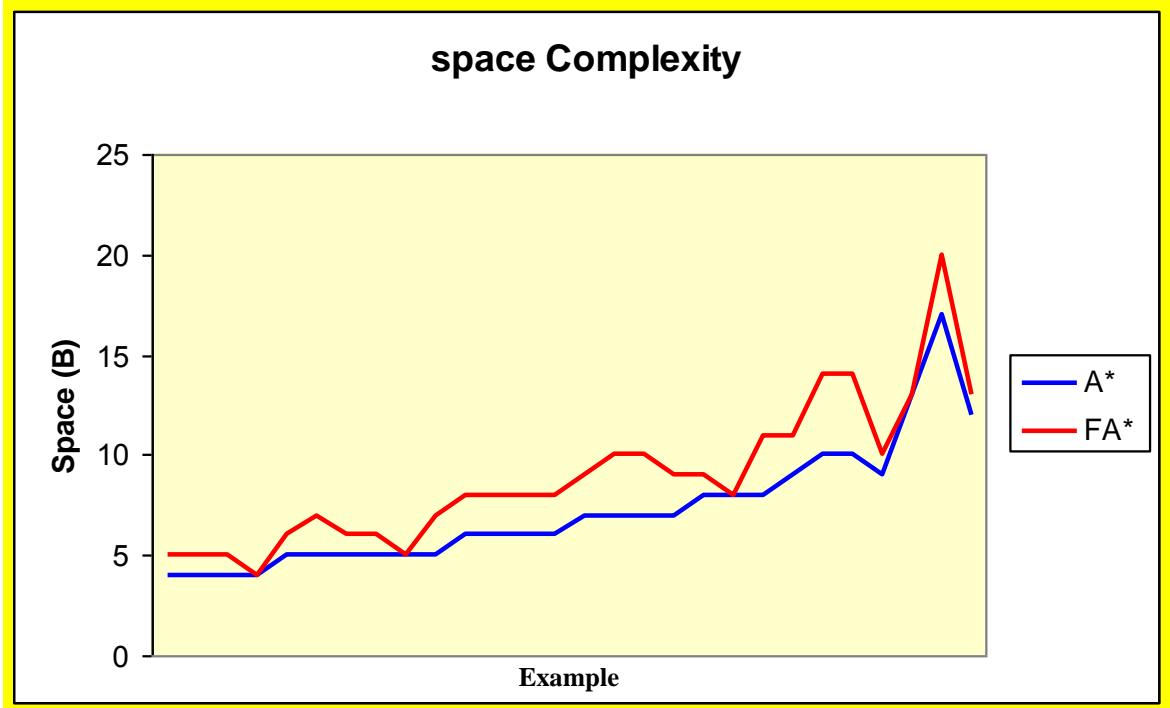


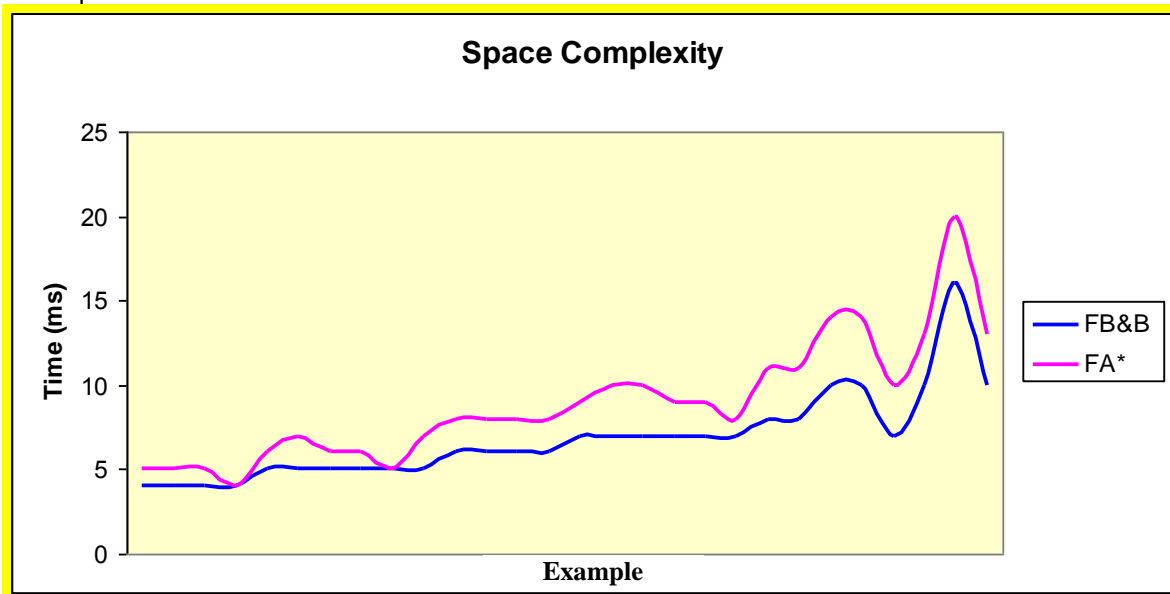


Figure 5.38 shows that FA\* achieves less Space complexity than A\* search technique, and figure 5.39 shows that FA\* achieves less Space complexity than FB&B search technique for all examples.

**Figure 5.38 (Line chart):** Comparison of **Space Complexity** for two search techniques.



**Figure 5.39 (Line chart):** Comparison of **Space Complexity** for the proposed two fuzzy search techniques.



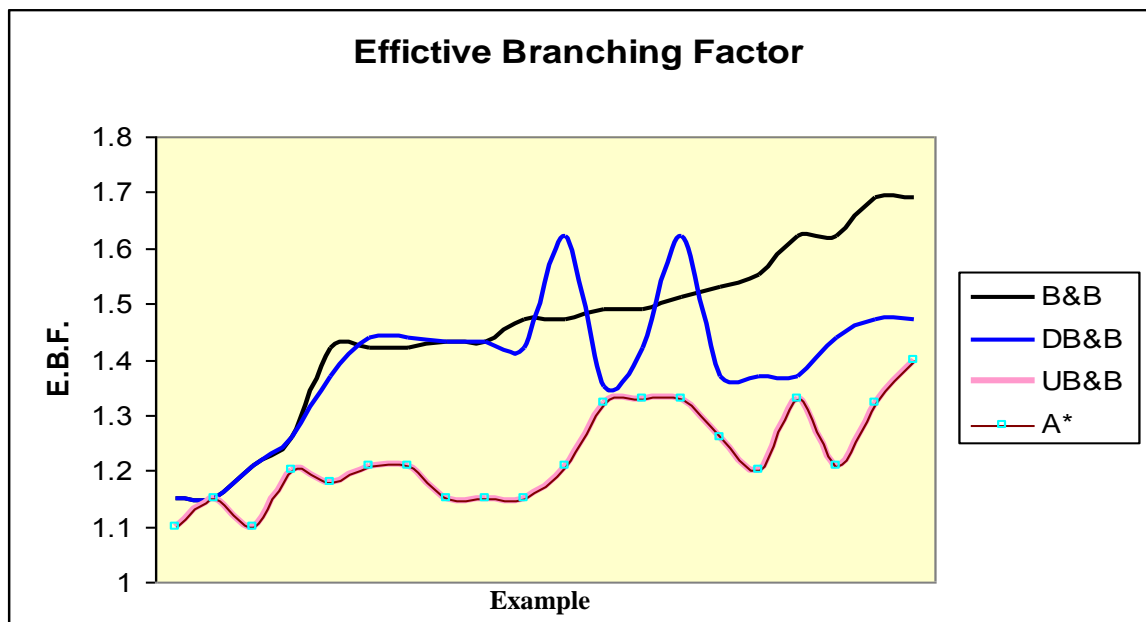
- **Effective Branching Factor:** Table (5.4) shows **Effective Branching Factor** for six Searching techniques when the simulation was run for different examples.

**Table 5.4:** Comparison of **Effective Branching Factor** for various search techniques.

<b>Effective Branching Factor</b>						
<b>Example</b>	<b>Name of search technique</b>					
	<b>B&amp;B</b>	<b>DB&amp;B</b>	<b>UB&amp;B</b>	<b>FB&amp;B</b>	<b>A*</b>	<b>FA*</b>
S-G	0.51	0.62	0.33	0.33	0.33	0.33
Nq-Am	0.69	0.47	0.4	0.37	0.4	0.37
Nq-AZ	0.69	0.47	0.32	0.21	0.32	0.21
Qt-Ir	0.55	0.37	0.2	0.2	0.2	0.2
Tf-Az	0.6	0.51	0.65	0.32	0.42	0.32
Ir-Am	0.26	0.26	0.2	0.13	0.2	0.13
Rm-Am	0.21	0.21	0.1	0.1	0.1	0.1
Mn-Am	0.49	0.35	0.32	0.32	0.32	0.32
Jf-Am	0.42	0.37	0.18	0.18	0.18	0.18
Jd-Am	0.62	0.37	0.33	0.33	0.33	0.33
Tf-Am	0.62	0.32	0.33	0.33	0.33	0.33
Zm-Ir	0.42	0.44	0.21	0.21	0.21	0.21
Db-Ir	0.42	0.44	0.21	0.21	0.21	0.21
Kr-Ir	0.53	0.37	0.26	0.2	0.26	0.2
kr-Az	0.42	0.32	0.69	0.32	0.53	0.33
Jr-Az	0.37	0.44	0.43	0.43	0.43	0.43
Jd-Az	0.29	0.32	0.44	0.33	0.44	0.33
Db-Az	0.47	0.42	0.15	0.15	0.15	0.15
Ir-Az	0.53	0.44	0.43	0.43	0.43	0.43
Az-Ir	0.53	0.21	0.43	0.43	0.43	0.43
Tf-Ir	0.73	0.29	0.29	0.18	0.29	0.18
Jf-Ir	0.49	0.42	0.33	0.33	0.33	0.33
Am-Ir	0.43	0.43	0.15	0.15	0.15	0.15
Kr-Am	0.62	0.44	0.21	0.21	0.21	0.21
Zm-Am	0.15	0.15	0.1	0.1	0.1	0.1
Qt-Am	0.53	0.33	0.43	0.43	0.43	0.43
Db-Am	0.43	0.43	0.15	0.15	0.15	0.15
Az-Zg	0.15	0.15	0.15	0.15	0.15	0.15
Md-Ir	0.47	0.62	0.21	0.21	0.21	0.21

The following line charts show comparisons of **Effective Branching Factor** for various search techniques according to different independent examples, where EBF for B&B, UB&B, and A\* search techniques increases as number of expanded nodes has been increased. Figure 5.40 shows that UB&B and A\* achieve less EBF than B&B and DB&B search techniques, and figure 5.41 shows that FB&B achieves less EBF than UB&B search technique for all examples.

**Figure 5.40 (Line chart):** Comparison of **Effective Branching Factor** for four crisp search techniques.



**Figure 5.41 (Line chart):** Comparison of **Effective Branching Factor** for two search techniques.

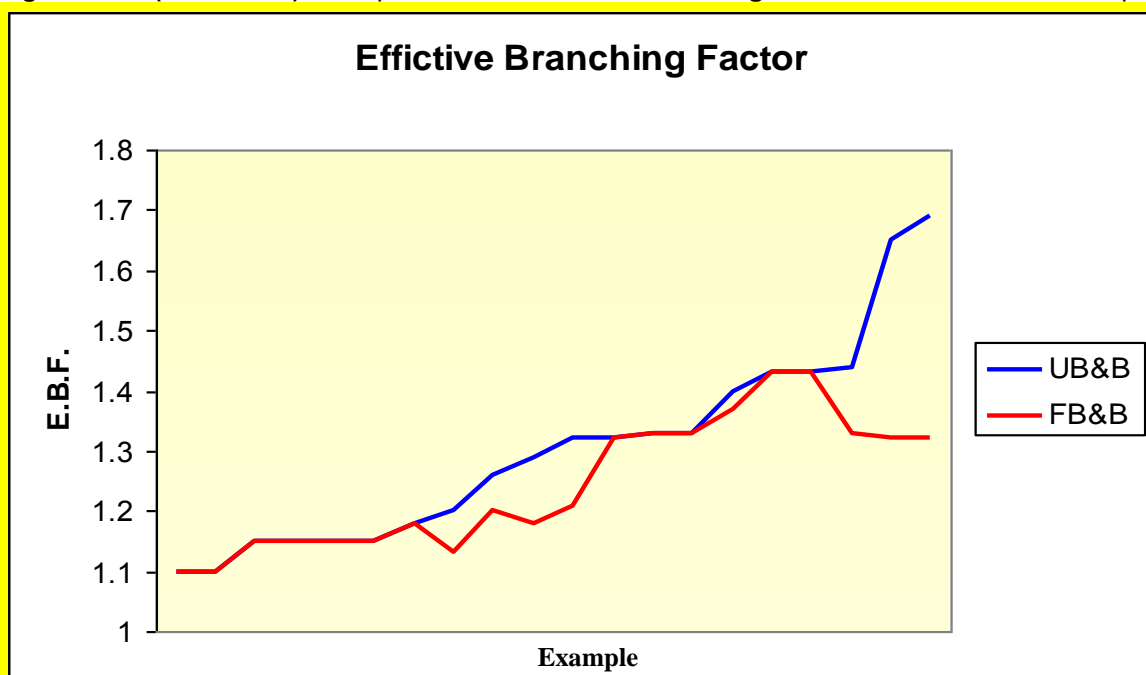
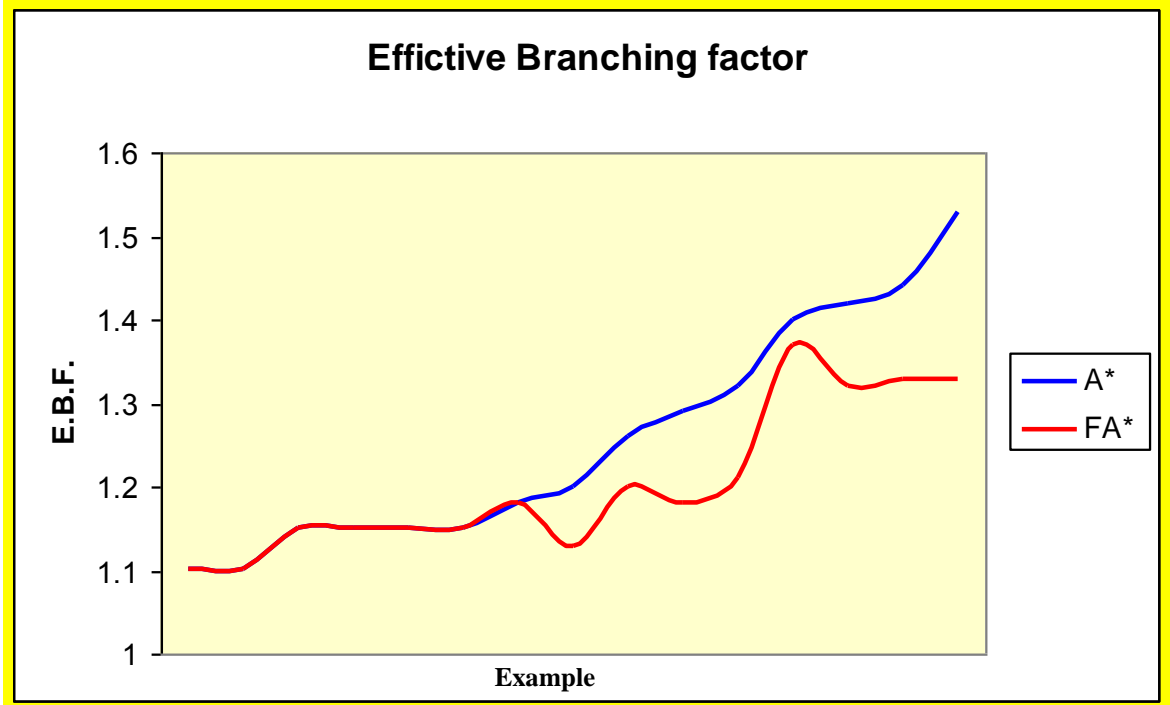


Figure 5.42 shows that FA\* achieves less EBF than A\* search technique for all examples.

**Figure 5.42 (Line chart):** Comparison of **Effective Branching Factor** for two search techniques.



#### 5.4 Conclusion.

This chapter introduced and described the practical aspect of this research, where the presented simulation program forms the core of the practical part to evaluate this work. The simulation is considered enough to evaluate, and compare the proposed algorithms with other well-known algorithms.

This chapter introduced and described the simulation program which has been used to evaluate the performance of the proposed algorithms with a number of practical examples to show the execution steps and procedures according to an official map of Jordan with actual distances.

Simulation program has been explained for the proposed Searching techniques which are Branch & Bound with Fuzzy Underestimation, and A\* with Fuzzy Underestimation, and other related Searching techniques which are: B&B, B&B with Underestimation, B&B with dynamic programming, and A\*.

The results of each comparison when executing the program for a specific problem (example) will be shown in a table for Number of Iterations, Time Complexity, Space Complexity, and Effective Branching Factor for each of the six algorithms, and the results also will be shown in four bar charts each one of them represents one of the four factors for the six searching techniques.

The analysis and simulation results show that Fuzzy A\* and Fuzzy Underestimate B&B search techniques achieve better efficiency, Time Complexity, Number of Iteration, and Effective Branching Factor are better than all other searching techniques, while.

Space Complexity for A\* and Underestimated B &B are always more (worse) than those of other algorithms, while Space complexity for Fuzzy A\* and Fuzzy Underestimated B&B are always less (better) than those of crisp A\* and Underestimated B&B.

Time complexity for Fuzzy A\* is better (less) than that for Fuzzy Underestimated B&B, while Space complexity for Fuzzy A\* is worse (more) than that for Fuzzy Underestimated B&B.

## Chapter six

### Conclusions and future work

#### 6.1 Introduction

Well-designed heuristic functions can play an important part in efficiently guiding a search process toward a solution. The purpose of a heuristic function is to guide the search process in the most profitable direction by suggesting which path to follow first when more than one is available: The more accurately the heuristic function estimates the true merits of each node in the search tree (or graph), the more direct the solution process. In the extreme, the heuristic function would be so good that essentially no search would be required. The system would move directly to a solution (Rich & Knight, 2000).

Branch and Bound search augmented by underestimate and A\* searching technique are effective heuristic principle guided Artificial Intelligence Problem-Solving Techniques.

The existing Branch & Bound augmented by underestimate and A\* searching techniques are techniques that work well on precise data, but not on imprecise data, whereas data available are not always crisp in real life.

The aim of this research is to deal with such data type and to deal with the imprecise data involved in different kinds of existing searching techniques in a more efficient way by applying fuzzy logic on Branch & Bound augmented by underestimate and A\* searching techniques, where fuzzy logic is an appropriate tool to deal with uncertain and imprecise information, because fuzzy logic has a capability to express knowledge in the form of linguistic rules.

A search problem and its solution by the existing crisp method of branch and bound and A\* search has been considered in this dissertation. We have proved that these methods can be improved by using fuzzy theory. Consequently a new method of B&B and A\* searching techniques with fuzzy underestimation to the available fuzzy information (using Triangular Fuzzy Number model) has been proposed to add Fuzzy Underestimation to the existing algorithms, thus a new improved version of searching techniques under uncertainty has been suggested to be helpful in many real life problems of computer science, especially in AI field. The corresponding algorithms have been given, and the algorithms have been explained by examples and applications.

This chapter concludes the dissertation and presents comparisons with previous work. It provides analysis for the simulation results of the proposed algorithms. Future work will be explained in the last section.

## 6.2 Conclusions

The proposed solution has been implemented and tested; the conclusions and recommendations of the researcher are as follows:

### 6.2.1 Comparison between the Crisp Algorithms.

1. Dynamic programming is preferable when many paths converge on the same place.
2. Branch-and-bound search is preferable when the tree is big and bad paths turn distinctly bad quickly.
3. Branch-and-bound search with a guess is preferable when there is a good lower-bound estimate of the distance remaining to the goal.
4. The A\* procedure is preferable when both branch-and bound search with a guess and dynamic programming are effective.
5. Crisp B&B has high values for Number of iteration, Time complexity, Space complexity, and Effective Branching Factor when it is used without other algorithms as shown in Figures (5.31,33,36,40).
6. Crisp B&B with Dynamic Programming principle also has high values for Number of iteration, Time complexity, Space complexity and Effective Branching Factor when it is used without other algorithms but in general it is still superior to Crisp B&B, as shown in Figure (5.31).
7. B&B Augmented by Underestimate search technique achieve better efficiency than regular B&B search technique as shown in Figure (5.32), where underestimation increases the efficiency of B&B by enabling it to be more informed.



8. A\* algorithm can be considered as one type of branch-and-bound algorithm but it is better than B&B, because dynamic programming improves its efficiency.

### **6.2.2 Effect of Using the Underestimated Value.**

When analyzing the effect of using the Underestimated value we can observe that:

1. Maximum underestimated values means more efficient search because the closer an underestimate to the true distance, the more efficiently the search (because, if there is no difference at all, there is no chance of developing any false movement); in other words if the guesses were perfect, this approach would keep you on the optimal path at all times.

However, guesses are not perfect, and a bad overestimate somewhere along the true optimal path may cause you to wander away from that optimal path permanently.

2. At the other extreme, an underestimate may be so poor as to be hardly better than a guess of zero (underestimate = 0), which certainly must always be the ultimate underestimate of remaining distance (it has no heuristic power and does not provide any guidance for the search). In fact; ignoring estimates of remaining distance altogether can be viewed as the special case in which the underestimate used is uniformly zero (underestimate of close to zero is of little Value).

### **6.2.3 Effect of Using the Fuzzy Underestimated Value ( $\alpha$ )**

When analyzing the effect of using the Fuzzy Underestimated value ( $\alpha$ ) we can observe that:

1. Maximum value of  $\alpha = 1$  (which must not be so) means that we take the

estimated value as a crisp one without fuzzification, which turns Fuzzy A\* or Fuzzy B&B to crisp ones.

2. Minimum value of  $\alpha = 0$  means that there is no estimation, which turns Fuzzy B&B or Fuzzy A\* to crisp B&B or crisp B&B with D.P. principle consequently.
3. Good DM can make  $\alpha$  more close to real value (optimal value), then solution will go directly to the goal (high efficiency) but as  $\alpha$  decreases the efficiency will decrease but the goal guaranteed not to be overlooked, taking into consideration that in the worst case fuzzy underestimate will not be less than straight line distance , then it will be better than crisp underestimate in all cases( $u \text{ straight line distance} \leq \alpha < 1$ ), where fuzzy underestimate is more informed than crisp underestimate.

#### **6.2.4 Comparison between the Proposed Algorithms and Previous Works.**

The analysis and simulation results show that:

- 1- B&B Augmented by Fuzzy Underestimate search technique achieve better efficiency than B&B Augmented by Crisp Underestimate search technique as shown in Figures (5.34, 41), where fuzzy underestimation increases the efficiency of B&B Augmented by Underestimate enabling it to be more informed.
2. A\* with Fuzzy Underestimate search technique achieve better efficiency than A\* with Crisp Underestimate search technique as shown in Figures (5.35, 42), where fuzzy underestimation increases the efficiency of A\* enabling it to be more informed.

3. The analysis and simulation results show that Fuzzy Underestimated B&B and Fuzzy A\* search techniques achieve a search time (Time complexity) less than that of the B&B and A\* with crisp underestimate as shown in Figures (5.34, 35).
4. Space complexity for Fuzzy A\* and Fuzzy Underestimated B&B are always better (less) than those of crisp algorithms, as shown in Figures (5.37, 38) where fuzzy methods decrease Space complexity by enabling the algorithms to be more informed.
5. Effective Branching Factor for Fuzzy A\* and Fuzzy Underestimated B&B are always better (less) than those of crisp algorithms especially when number of nodes are high as shown in Figures (5.41, 42); fuzzy methods decrease Effective Branching Factor, where its value becomes closer to 1.

### **6.2.5 Comparison between the two Proposed Algorithms.**

The analysis and the simulation results show that:

1. B&B Augmented by Fuzzy Underestimate search technique is simpler than A\* with Fuzzy Underestimate search technique with less space complexity, but with less efficiency (more time complexity) as shown in Figures (5.39, 33).
2. A\* with Fuzzy Underestimate search technique is more efficient than B&B Augmented by Fuzzy Underestimate search technique because we add D.P. but with more space complexity as shown in Figures (5.33, 39).
3. Fuzzy A\* achieves better(less) Time complexity than Fuzzy Underestimated B&B as shown in figure (5.33).
4. Fuzzy A\* has memory requirements (space complexity) comparable to Fuzzy Underestimated B&B as shown in Figure (5.39) as it maintains all the

generated nodes in the memory.

5. Both A\* with Fuzzy Underestimate and B&B Augmented by Fuzzy Underestimate search techniques are complete, optimal, nonredundant, and well informed among all other algorithms.
6. High efficiency Fuzzy A\* or Fuzzy Underestimated B&B has no difference as both achieved nearly the maximum efficiency (goes directly to the goal).

### **6.2.6 Effects of Time complexity, Space complexity, and Effective Branching Factor on Efficiency.**

When analyzing the effects of Time complexity, Space complexity, and Effective Branching Factor on Efficiency we observed the following:

1. Efficiency is related directly to Time complexity, where efficiency increases as time complexity decreases; less time complexity means more efficiency for specific technique.
2. Space complexity increased as Time complexity decreased in most cases, where higher efficiency related to higher Space complexity.
3. One heuristic is more informed than another heuristic if a search method that uses it needs to examine fewer nodes to reach a goal.
4. Less Effective Branching Factor for a specific search technique means that the search technique can find optimal paths with less work.

### 6.2.7 Expected drawbacks for the two Proposed Algorithms.

- 1- A Bad Decision Maker can cause bad overestimated values, where a bad overestimate somewhere along the true optimal path may cause you to wander away from the optimal path permanently. Note, however, that a good Decision Maker with good underestimates cannot cause the right path to be overlooked, taking into consideration that in the worst case, fuzzy underestimate will not be less than straight line distance , then it must be better than crisp underestimate in all cases ( $u$  straight line distance  $\leq \alpha < 1$ ).
- 2- Fuzzy A\* has a high Space complexity as crisp A\* , due to adding Dynamic Programming Principle, where it may use all the available memory in a matter of minutes, but after that the search practically cannot proceed although the user would find it acceptable that the algorithm would run for hours or even days (Bratko, 1998).
- 3- Unfortunately, although Fuzzy A\* or Fuzzy Underestimated B&B algorithms are more efficient than others, they still require exponential time (like Crisp A\* and Crisp Underestimated B&B). The exact amount of time they save for a particular problem depends on the order in which the paths are explored (Rich & Knight, 2000).

### 6.3 Future Work.

Even though the research in this dissertation proved that the proposed fuzzy algorithms improve the efficiency of the existing crisp algorithms, further research can be done to improve or support the presented solution such as:

- 1- Interval Valued Fuzzy Number model can be used in addition to Triangular Fuzzy Number model, where the Overestimated Value ( $\alpha_2$ ) can be taken into consideration, to exclude some choices , in order to decrease search efforts.
- 2- Other fuzzy models than Triangular Fuzzy Number like Trapezoidal Fuzzy Number can be suggested.
- 3- Fuzzy logic can be used to consider more than one Source of Information and convert it to a single crisp value, where fuzzy logic can be used to associate different metrics so as to produce one corresponding crisp value.
- 4- Triangular Fuzzy Number can be considered to be dynamic variable according to Source of Information or other variable conditions or parameters (rule of the thumb), while we considered it in our work as  $(a-3, a, a+3)$  to simplify the suggested algorithms.
- 5-  $\alpha$ -cut can be considered to be dynamic variable according to Decision Maker confidence in Source of Information, or other variable conditions, while we considered  $\alpha$  as a constant value according to each specific Source of Information = (IS).
- 7- Statistics can be combined with fuzzy logic for some uncertainties.

## References

1. Atanassov, K. T. (1999). Intuitionistic Fuzzy Sets: theory and application. Physica Verlag, 1999.
2. Baldwin, J.F. (1981). Fuzzy logic and fuzzy reasoning. In: Mamdani, E.H. And Gaines, B.R. (eds.), **Fuzzy Reasoning and Its Applications**, Academic Press, London, PP. 133-148.
3. Bangoli, C. And & Smith, H. C.(1998). The Theory of Fuzzy Logic and its Application to Real Estate Valuation. In: **Journal of Real Estate Research**, volume 16, number 2, PP. 169-199.
4. Bhatkar, D. P. (1994).**Methods And Tools for Applied Artificial Intelligence**.(NA): CRC Press
5. Bigus, J. P. And Bigus, J. (2001). **Constructing Intelligent Agents Using Java™**, Second Edition. Canada: Willy Computer Publishing
6. Black, P. E. (eds.) (2005, Sep. 12). **Branch and Bound**. in Dictionary of Algorithms and Data Structures (online), Paul E. Black, (ed.), **U.S. National Institute of Standards and Technology**.Retrieved Feb 22,.2007, Available from: <http://www.nist.gov/dads/HTML/branchNbound.html>
7. Blue, M. Bush, B.W. And Puckett, J. (2002). Unified approach to fuzzy graph problems. In **Fuzzy sets and systems**,125(3), pp.355-368
8. Bolloju, N. (1995). Design of an Adaptive Fuzzy Logic Controller for Knowledge Discovery. In Steele, N.C. (eds.). **Proceedings of the International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions
9. Bratko, I. (1998).**Prolog Programming for Artificial Intelligence**, 3<sup>rd</sup> ed. England: Addison-Wesely.

10. Bratko, I. (2001). **Prolog Programming for Artificial Intelligence**. Pearson Education.
11. Bremner, H. And Postlethwaite, B. (1995). THE APPLICATION OF A RELATIONAL FUZZY MODEL BASED CONTROL SYSTEM TO AN INDUSTRIAL DRYER. In Steele, N.C. (eds.). **Proceedings of the International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions
12. Buffalo, (2007). **Libraries**. (on-line). Retrieved Feb. 27, 2007, Available from: [ublib.buffalo.edu/libraries/help/glossary.html](http://ublib.buffalo.edu/libraries/help/glossary.html).
13. Bustince, H. And Burillo, P. (1995). Interval-valued Fuzzy Ordering Relation. In Steele, N.C. (eds.). **Proceedings of the International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions
14. Chandwani, M. And Chaudhari, N.S. (1993). An algorithm for fuzzy control based on shortest-path framework. In Industrial Electronics, Control, and Instrumentation, 1993. **Proceedings of the IECON '93. International Conference on 15-19 Nov. 1993, Vol.1, PP. 254 – 257.**
15. Clay, P. , Crispin, A. And Crossley, S. (2000). A Comparative Analysis of Search Methods as Applied to Shearographic Fringe Modelling. In Loganantharaj, R. , Palm,G. And Ali, M. (eds.). **Intelligent Problem Solving Methodologies and Approaches**, 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2000, June 19-22, 2000 Proceedings. USA: Springer.
16. Colton, S. (ed.) (2005). **Search in Problem Solving**, Lecture 3. Retrieved from [www.doc.ic.ac.uk/~sgc/teaching/v231/lecture3.html](http://www.doc.ic.ac.uk/~sgc/teaching/v231/lecture3.html) on 25/4/2007
17. Cooman, G. (1995). GENERALIZED POSSIBILITY AND NECESSITY MEASURES ON FIELDS OF SETS. In Steele, N.C. (eds.). **Proceedings of the International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions



18. Coppin, B. (2004). **Artificial Intelligence Illuminated**, first edition. USA: Jones and Bartlett publishers.
19. Doyle, P. (Dec.,2005). **Search Methods**. (on-line). Retrieved Dec. 5, 2005, Available from: [pdoyle@cs.stanford.edu](mailto:pdoyle@cs.stanford.edu).
20. Dubious, D. And Prade, H. (1987). **A Theory of Possibility**. Plenum Publishing, 1987
21. Dubious, D. And Prade, H. (1990). **Fuzzy Sets and Systems: Theory and Application**. New York : Academic Press.
22. Elaine, R. (1983). **Artificial Intelligence**. New York: McGraw- Hill.
23. Foley, H. And Petry, F. (2000). Fuzzy Knowledge-Based System for Performin Conflation in Geographical Information Systems. In Loganantharaj, R. , Palm,G. And Ali, M. (eds.). **Intelligent Problem Solving Methodologies and Approaches**, 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2000, June 19-22, 2000 Proceedings. USA: Springer.
24. Fujisawa, Y. And Etal (1993). Control method of manipulator/vehicle system with fuzzy inference. In L.' Bezdek (editor), **Fuzzy Logic Technology and Application**. 1993.
25. Gaines, B.R. (1976). Foundations of fuzzy reasoning. In Man t. J. (eds.), **Machine Studies**, 6: PP. 623-668.
26. Godjevac, J. (1995). A learning procedure for a fuzzy system: application to obstacle avoidance. In Steele, N.C. (eds.). **Proceedings of the International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions
27. Goldberg, E. D. (1989). **Genetic Algorithms in Search Optimization and Machine Learning**. Reading, MA: Addison Wesley.

28. Gorzalczany M. B. (1987). A method of inference in approximate reasoning based on interval-valued fuzzy sets. In **Fuzzy Sets and Systems**, 21: (1987) 1-17
29. Gottwald, S., And Treatise, A. (2001). **On Many-Valued Logics**. Baldock, Hertfordshire, England: Research Studies Press LTD.
30. Hansen,P. Beckmann, M. And Kunzi, H.P. (1980). Multiple criteria decision making. In: **Theory and applications, Lecture Note in Economics and in Mathematical Systems**, vol. 177. Berlin: Springer; 1980, pp. 109-27.
31. Hart, P. E., Nilsson , N. J. And Raphael, B. (1968). **A formal Basis for the Heuristic Determination Cost Paths**: In IEEE Transactions on SSC 4. PP. 100-107
32. Hart, P. E., Nilsson, N. J. And Raphael, B. (1972).**Correction to "A formal Basis for the Heuristic Determination Cost Paths"**: In SIGART Newsletter 37. PP. 28-29
33. Hellendoorn, H. (1992). The generalized models considered as a fuzzy relation. In **Fuzzy Sets and Systems** 46: PP. 29-48.
34. Klein, C.M. (1991). Fuzzy shortest paths. **Fuzzy Sets and Systems** 1991; Vol 39:27–41.
35. Klir, G. And Yuan, B. (1995) . **Fuzzy Sets And Fuzzy Logic: Theory and Applications**. India: Prentice Hall.
36. Klir, G. And Yuan, B. (1995). **Fuzzy Sets and Fuzzy Logic**. ISBN 0-13-101171-5
37. Klir, G., Clair, U. H. St. And Yuan, B. (1997). **Fuzzy Set Theory Foundations and Applications**. (on-line). Retrieved Feb. 20, 2007, Available: [www.wikipedia.org/wiki](http://www.wikipedia.org/wiki), This page was last modified 1997.
38. Kruse, L.; Schmidt, E.; Jochens, G.; Stammermann, A.; And Nebel, W. (2000).Lower bound estimation for low power high-level synthesis. System Synthesis, 2000. Proceedings. **The 13th International Symposium** on 20-22 Sept. 2000, PP. 180 - 185

39. KU Libraries. (2007). **Data base search techniques** (on-line). Retrieved Feb. 27, 2007, Available: [www.lib.ku.edu](http://www.lib.ku.edu).
40. Li, X. And Wu, J. (2002). **Searching techniques in peer-to-peer networks** (on-line). Retrieved Feb. 27, 2007, Available: [www.cse.fau.edu](http://www.cse.fau.edu).
41. Lin, K. And Chern, M. (1994). The fuzzy shortest path problem and its most vital arcs. In **Fuzzy Sets and Systems** 1994;58:343-53.
42. Loganantharaj, R. And Thomas, B.(2000). An Overview of a Synergetic Combination of Local Search with Evolutionary Learning to Solve Optimization Problems. . In Loganantharaj, R., Palm, G. And Ali, M. (eds.). **Intelligent Problem Solving Methodologies and Approaches**, 13th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2000, June 19-22, 2000 Proceedings. USA: Springer.
43. Luger, G. F. (2005). **Artificial intelligence Structures and Strategies for Complex Problem Solving**, Fifth Edition. England: Addison-Wesley
44. Guangwu, M. Basic Theory for interval-valued fuzzy sets. In **Mathematica Applicata**, Vol.2 1993, pp. 129-135.
45. Ma, W.M. And Chen, G.Q. (2005). Competitive analysis for the on-line fuzzy shortest path problem. In **Proceedings of the Fourth International Conference on Machine Learning and Cybernetics**, Vol. 2, 18-21 August 2005. PP. 862-867.
46. Magdalena, L. And Monasterio, F. (1993). Fuzzy controlled gait synthesis for a biped walking machine. In L Bezdek (editor), **Fuzzy Logic Technology and Application**. 1993.
47. Mares, M. And Horak, J. (1983). Fuzzy quantities in networks. In **Fuzzy Sets and Systems**, Vol. 10, 1983, pp. 135-155.
48. Moazeni, S.(2005). Fuzzy shortest path problem with finite fuzzy quantities. In NAFIPS 2005-Annual Meeting of the North American **Fuzzy Information Processing Society**, 26-28 June, 2005 PP. 664 - 669.

49. Moore, R. E. (1969). **Interval Analysis**. New Jersey: Prentice Hall.
50. Ncsu, (2007). (on-line). Retrieved Feb. 27, 2007, Available from: [www.ces.ncsu.edu/depts/pp/soybeanrust/netterms.php](http://www.ces.ncsu.edu/depts/pp/soybeanrust/netterms.php)
51. Negnevitsky, M. (2002). **Artificial Intelligence A Guide to Intelligent Systems**, England: Pearson Education Limited.
52. Newell, A. And Simon, H. A. (1976). Computer science as Empirical inquiry: symbols and search. In **Communications of the ACM**, 19(3): 113-126.
53. Nilsson, N. (1971). **Problem-Solving Methods in Artificial Intelligence**. New York: McGraw-Hill.
54. Okada, S. (2001). Interactions among Paths in Fuzzy Shortest Path Problems. In **9th International Fuzzy Systems Association (IFSA) World Congress and 20th NAFIPS International Conference: Vancouver, Canada, Vol.1** pp.41-46
55. Okada, S. And Soper, T.A. (2005). A shortest path problem on a network with fuzzy arc lengths. In **Fuzzy Sets and Systems 2000**; Vol. 109:129-140.
56. Pearl, J. (1984). **Heuristic: Intelligent Strategies for Computer Problem Solving**. Reading MA: Addison-Wisley
57. Pearl, J. (1984). **Heuristics**. England: Addison-Wesely.
58. Polya, G. (1945). **How to Solve It**. Princeton, NJ: Princeton University Press.
59. Qadi, Z. And Rasras, R. (2003) **Artificial Intelligence**. Amman: Arab Community Library.
60. Rajagopalan, A., Washington, G., Rizzoni, G. And Guezenec. Y. (2006) **Development of Fuzzy Logic & neural network Control a Advance Emissions Modeling for Parallel Hybrid Vehicles**. USA: Nation Renewable Energy laboratory.

61. Rich, E. And Knight, K. (2000). **Artificial Intelligence**, Second Edition. New Delhi: Tata McGraw-Hill Publishing Company
62. Rjgc, (2007). المملكة الاردنية الهاشمية-المواصلات والاثار. (on-line). Retrieved April. 23, 2007, Available: [www.rjgc.gov.jo](http://www.rjgc.gov.jo), Royal Jordanian Geographic Centre.
63. Roy M. K. And Biswas, R. (1992). I-v fuzzy relations and Sanchez's approach for medical diagnosis. In **Fuzzy Sets and Systems**, 47: (1992) 35-38.
64. Ruan, D. (1995). FLINS, a Bridge between Fuzzy Logic and the Nuclear Industrial World. In Steele, N.C. (eds.). **Proceedings of the International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions
65. Russell, S. And Norvig, P. (2003). **Artificial Intelligence, A Modern Approach, Second Edition**. New Jersey, USA: Pearson Education, Inc.
66. Saffiotti, A. Ruspini, E. And Kurt, K. (1993). Blending reactivity and goal-directedness in a fuzzy controller. In L. Bezdek (editor), **Fuzzy Logic Technology and Application**. 1993.
67. Sambuc R. (1975). Fonctions  $\Phi$ -Floues. Application a l'aide au Diagnostic en Pathologie Thyroïdienne. In **This de Doctoral en Medecine**, Marseille, 1975.
68. San, (2007). **Glossary**. (on-line). Retrieved Feb. 27, 2007, Available from: [home.san.rr.com/denbeste/glossary.html](http://home.san.rr.com/denbeste/glossary.html)

69. Schaeffer, J. And Plant, A. (2000). Unifying Single-Agent and Two-Player Search. In Hamilton, H. J. (eds.). **Advances in Artificial in Intelligence**, Lecture Notes in Artificial Intelligence 1822, Sub series of lecture Notes in computer science. Canada: 13th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, AI 2000
70. Shafer G. (1976). **A Mathematical Theory of Evidence**. Princeton University Press, 1976
71. Silva, P. C. (1995). FUZZY SITUATED-AUTOMATA APPROACH. In Steele, N.C. (eds.). **Proceedings of the International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions
72. Siue, (2007). **Library research guides glossary of terms** (on-line). Retrieved Feb. 27, 2007, Available from: [www.library.siu.edu/lib/research\\_tools.html](http://www.library.siu.edu/lib/research_tools.html).
73. Slagle, J.R. And Lee, R.C.T.(1971). **Applications of Game Tree Searching Techniques to Sequential Pattern Recognition**. In CACM (14), 1971, pp. 103-110
74. Sun, K. And Hadipriono, F.C. (1995). RATING SPACE FOR FUZZY REASONING. In Steele, N.C. (eds.). Proceedings of the **International ICSC Symposium on FUZZY LOGIC**, May, 1995. Canada: Academic press International Computer Science Conventions
75. Takahashi, M. T. And Yamakami, A. (2005). On Fuzzy Shortest Path Problems with Fuzzy Parameters: an Algorithmic Approach. In NAFIPS 2005 - **2005 Annual Meeting of the North American Fuzzy Information Processing Society**, PP. 654-657.
76. Thomas, E. And Portegys, A. (1995). **A Search Technique for Pattern Recognition Using Relative Distances**. In IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 17, No. 9, September 1995, pp. 910-912

77. Turksen, I.B. (1986). Interval valued fuzzy sets based on normal form. In **Fuzzy Sets and Systems**, Vol.20, 1986, pp. 191-210.
78. UBC Lib.,The university of British Columbia Library (2007).**Web search techniques** (on-line). Retrieved Feb. 27, 2007, Available: [www.library.bcu.ca/home/websearch/quick.html](http://www.library.bcu.ca/home/websearch/quick.html).
79. ucsd, (2007). **Search Cognitive Science** 108b Lecture Notes. (on-line). Retrieved April. 25, 2007, Available: <http://cogsci.ucsd.edu/~batali/108b/lectures/heuristic.html>
80. Wikipedia Org. (2007). **Fuzzy Logic** (on-line). Retrieved Feb. 25, 2007, Available: [www.wikipedia.org/wiki/Fuzzy\\_logic](http://www.wikipedia.org/wiki/Fuzzy_logic), This page was last modified 22 February 2007.
81. Wikipedia Org. (2007).**Define: Search** (on-line). Retrieved Feb. 27, 2007, Available: [www.wikipedia.org/wiki/Searching](http://www.wikipedia.org/wiki/Searching)
82. Winston, P. H.(2000).**Artificial Intelligence**, 3<sup>rd</sup> Edition. India: Addison- Wesley.
83. Yager, R.R. (1980). An approach to inference in approximate reasoning. In Man, t. J., **Machine Studies**, 13:PP. 323-338.
84. Yao, J.S. And Lin, F.T. (2003). Fuzzy Shortest-Path Network Problems With Uncertain Edge Weights. In **Journal of Information Science and Engineering** 19, 329-351 (2003)
85. Zadeh L.A. (1965). Fuzzy sets. In **Information and Control**, Vol. 8, pp. 338-353.
86. Zadeh L.A. (1968). Fuzzy Algorithm. In **Inform. And Control** 12, 94-102.
87. Zadeh L.A. (1973). Outline of a new approach to the analysis of complex systems and decision processes, **interval-valued fuzzy sets**. In IEEE Trans. Systems Man Cybernet, 3: (1973) 2844.
88. Zadeh L.A. (1975). Fuzzy logic and approximate reasoning. In **Syntheses** 30:PP. 407- 428

89. Zadeh L.A. (1975). The concept of a linguistic variable and its application to approximate reasoning. In **Information Sciences**, Vol. 8, pp. 199–249, 301–357; Vol. 9, pp. 43–80.
90. Zadeh L.A. (1978). Fuzzy Sets as a Basis for a Theory of Possibility. In **Fuzzy Sets and Systems**, Vol. 1, No. 1, pp. 3–28
91. Zadeh L.A. (1985). **Syllogistic Reasoning in Fuzzy Logic and its Application to Usuality and Reasoning with Dispositions**. In IEEE, 6, pp754 -763, 1985
92. Zhou , B. And Mouftah, H. T. (2004). Adaptive Shortest Path Routing in GMPLS-based Optical Networks Using Fuzzy Link Costs. In **Electrical and Computer Engineering, 2004. Canadian Conference CCECE 2004- CCGEI 2004**, Vol.3 , May 2004, PP. 1653 - 1657
93. Zimmermann, H. J. (1991). **Fuzzy Set Theory and Its Applications**. Dordrecht Kluwer: Academic Publication.



# Appendices

## Appendix 1 Simulation Pseudocodes:

The pseudocodes for the six algorithms are presented below:

### 1 Branch & Bound Search pseudocode

The **pseudocode** for Branch & Bound Search technique will be introduced in **this appendix**, while the corresponding **algorithm** (procedure) was presented in **section (3.2)**, Branch and Bound Search **algorithm explanation** was detailed in **figure(3.3)**, and a Branch and Bound Search **Example** was detailed in **figure(3.2)**.

---

To conduct a branch-and-bound search:  
Initialize all vertices to "undiscovered."  
goal = 0  
While goal = 0  
    min = 1000  
    For j = 1 To 63  
        If w(j) = True Then  
            lbl(j).ForeColor = vbRed  
        Else  
            lbl(j).ForeColor = vbBlack  
        End If  
    Next j  
    For i = 2 To 63  
        If (w(i) = True And v(i) < min) Or ((w(i) = True And v(i) < min) And (lbl(i).Caption = a(2)))  
Then  
        min = v(i)  
        order = i  
        End If  
    Next i  
    w(order) = False  
    lin(order - 1).BorderColor = vbBlue  
    lbl(order).ForeColor = vbBlue  
    If order <= 31 Then  
        If lbl(2 \* order).Visible = True Then  
            w(2 \* order) = True  
            lbl(2 \* order).ForeColor = vbRed  
        End If  
        If lbl(2 \* order + 1).Visible = True Then  
            w(2 \* order + 1) = True  
            lbl(2 \* order + 1).ForeColor = vbRed  
        End If  
    End If  
    If lbl(order).Caption = a(2) Then  
        For r = 2 To 63  
            If w(r) = True And v(r) >= v(order) Then  
                lblx(r).Visible = True  
            End If  
        End For  
    End If  
End While

```

        goal = 0
    ElseIf w(r) = True And v(r) < v(order) Then
        order1 = order
        order = r
        goal = 0
    Else
        goal = 1
    End If
Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
    Wend
End If

```

---

## 1 Branch & Bound Search with Underestimates pseudocode

The **pseudocode** for Branch & Bound with Underestimates Search technique will be introduced in **this appendix**, while the corresponding **algorithm** (procedure) was presented in **section (3.2.1)**, **algorithm explanation** was detailed in **figure(3.7)**, and an **example** was detailed in **figure(3.6)**.

---

To conduct a branch-and-bound search with underestimates:

Initialize all vertices to "undiscovered."

```

goal = 0
While goal = 0
    min = 1000
    For j = 1 To 63
        If w(j) = True Then
            lblgg(j).ForeColor = vbRed
        Else
            lblgg(j).ForeColor = vbBlack
        End If
    Next j
    For i = 2 To 63
        If (w(i) = True And g(i) < min) Or ((w(i) = True And g(i) < min) And (lbl(i).Caption = a(2)))
        Then
            min = g(i)
            order = i
        End If
    Next i
    w(order) = False
    lin(order - 1).BorderColor = vbBlue
    lbl(order).ForeColor = vbBlue
    If order <= 31 Then
        If lbl(2 * order).Visible = True Then
            w(2 * order) = True
            lbl(2 * order).ForeColor = vbRed
        End If
    End If

```

```

    If lbl(2 * order + 1).Visible = True Then
        w(2 * order + 1) = True
        lbl(2 * order + 1).ForeColor = vbRed
    End If
End If
If lbl(order).Caption = a(2) Then
    For r = 1 To 63
        If w(r) = True And v(r) >= g(order) Then
            lblx(r).Visible = True
            goal = 0
        ElseIf w(r) = True And v(r) < g(order) Then
            goal = 0
        Else
            goal = 1
        End If
    Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
    Wend
End If

```

---

## 2 B&B Search with Fuzzy Underestimation pseudocode

The **pseudocode** for Branch & Bound with Fuzzy Underestimates Search technique will be introduced in **this appendix**, while the corresponding **algorithm** (procedure) was presented in **section (3.3)**, **flow chart procedure** is shown in **figure (3.12)**, **algorithm explanation** is detailed in **figure(3.17)**, and **two applications** (examples) is detailed in **figure(3.16)** as a **random net example** and in **figure(3.22)** as an **example for real roads between two major Jordanian cities**; from **Al Karak** to **Irbid** according to an official map of Jordan.

To conduct a branch-and-bound search with fuzzy Underestimates:

```

Initialize all vertices to "undiscovered."
goal = 0
While goal = 0
    min = 1000
    For j = 1 To 63
        If w(j) = True Then
            lbluft(j).ForeColor = vbRed
        Else
            lbluft(j).ForeColor = vbBlack
        End If
    Next j

```

```

    For i = 2 To 63
    If (w(i) = True And uft(i) < min) Or ((w(i) = True And uft(i) < min) And (lbl(i).Caption = a(2)))
Then
    min = uft(i)
    order = i
    End If
Next i
w(order) = False
lin(order - 1).BorderColor = vbBlue
lbl(order).ForeColor = vbBlue
If order <= 31 Then
    If lbl(2 * order).Visible = True Then
        w(2 * order) = True
        lbl(2 * order).ForeColor = vbRed
    End If
    If lbl(2 * order + 1).Visible = True Then
        w(2 * order + 1) = True
        lbl(2 * order + 1).ForeColor = vbRed
    End If
End If
If lbl(order).Caption = a(2) Then
    For r = 1 To 63
        If w(r) = True And v(r) >= uft(order) Then
            lblx(r).Visible = True
            goal = 0
        ElseIf w(r) = True And v(r) < uft(order) Then
            goal = 0
        Else
            goal = 1
        End If
    Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
    Wend
End If

```

---

## 4 B&B Search with Dynamic Programming pseudocode

The **pseudocode** for Branch & Bound with **Dynamic Programming** Search technique will be introduced in **this appendix**, while the corresponding **algorithm** (procedure) was presented in **section (4.2)**, **algorithm explanation** is detailed in **figure(4.4)**, and an **example** is detailed in **figure(4.3)**.

To conduct a Branch-and-Bound search with dynamic programming:  
 Initialize all vertices to "undiscovered."  
 goal = 0

```

While goal = 0
  min = 1000
  For j = 1 To 63
    If w(j) = True Then
      lblt(j).ForeColor = vbRed
    Else
      lblt(j).ForeColor = vbBlack
    End If
  Next j
  For i = 2 To 63
    If (w(i) = True And v(i) < min) Or ((w(i) = True And v(i) < min) And (lbl(i).Caption = a(2)))
  Then
    min = v(i)
    order = i
  End If
  Next i
  For p = 1 To 62
    For q = 2 To 63
      If ((w(p) = True And w(q) = True) Or (w(p) = True And lbl(q).ForeColor = vbBlue) Or (w(q)
= True And lbl(p).ForeColor = vbBlue)) And lbl(p).Caption = lbl(q).Caption And v(p) < v(q) Then
vbInformation, "Duplication")
        w(q) = False
        lblx(q).Visible = True
        lbl(q).ForeColor = vbBlack
      ElseIf ((w(p) = True And w(q) = True) Or (w(p) = True And lbl(q).ForeColor = vbBlue) Or
(w(q) = True And lbl(p).ForeColor = vbBlue)) And lbl(p).Caption = lbl(q).Caption And v(p) > v(q)
Then
vbInformation, "Duplication")
        w(p) = False
        lblx(p).Visible = True
        lbl(p).ForeColor = vbBlack
      End If
    Next q
  Next p
  w(order) = False
  lin(order - 1).BorderColor = vbBlue
  lbl(order).ForeColor = vbBlue
  iteration6 = iteration6 + 1
  If order <= 31 Then
    If lbl(2 * order).Visible = True Then
      w(2 * order) = True
      lbl(2 * order).ForeColor = vbRed
    End If
    If lbl(2 * order + 1).Visible = True Then
      w(2 * order + 1) = True
      lbl(2 * order + 1).ForeColor = vbRed
    End If
  End If
  End If
  If lbl(order).Caption = a(2) Then
    For r = 1 To 63
      If w(r) = True And v(r) >= v(order) Then
        lblx(r).Visible = True
        goal = 0
      ElseIf w(r) = True And v(r) < v(order) Then
        goal = 0
      Else

```

```

        goal = 1
    End If
Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
    Wend
End If

```

---

## 5 A\* Search Technique pseudocode

The **pseudocode** for A\* Search Technique will be introduced in **this appendix**, while the corresponding **algorithm** (procedure) was presented in **section (4.2.1)**, **algorithm explanation** is detailed in **figure(4.8)**, and an **example** is detailed in **figure(4.7)**.

---

To conduct the A\* search technique:

Initialize all vertices to "undiscovered."

```

goal = 0
While goal = 0
    min = 1000
    For j = 1 To 63
        If w(j) = True Then
            lblgg(j).ForeColor = vbRed
        Else
            lblgg(j).ForeColor = vbBlack
        End If
    Next j
    For i = 2 To 63
        If (w(i) = True And g(i) < min) Or ((w(i) = True And g(i) < min) And (lbl(i).Caption =
a(2))) Then
            min = g(i)
            order = i
        End If
    Next i
    For p = 1 To 62
        For q = 2 To 63
            If w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And g(p) <
g(q) Then
                w(q) = False
                lblx(q).Visible = True
            Elseif w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And g(p)
> g(q) Then
                w(p) = False
                lblx(p).Visible = True
            End If
        Next q
    Next p

```

```

w(order) = False
lin(order - 1).BorderColor = vbBlue
lbl(order).ForeColor = vbBlue
If order <= 31 Then
    If lbl(2 * order).Visible = True Then
        w(2 * order) = True
        lbl(2 * order).ForeColor = vbRed
    End If
    If lbl(2 * order + 1).Visible = True Then
        w(2 * order + 1) = True
        lbl(2 * order + 1).ForeColor = vbRed
    End If
End If
If lbl(order).Caption = a(2) Then
    For r = 1 To 63
        If w(r) = True And v(r) >= g(order) Then
            lblx(r).Visible = True
            goal = 0
        ElseIf w(r) = True And v(r) < g(order) Then
            goal = 0
        Else
            goal = 1
        End If
    Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
    Wend
End If

```

---

## 6 A\* with Fuzzy Underestimates Search technique

The **pseudocode** for A\* with Fuzzy Underestimates Search technique will be introduced in **this appendix**, while the corresponding **algorithm (procedure)** was presented in **section (4.3)**, **flow chart procedure** is shown in **figure (4.9)**, **algorithm explanation** is detailed in **figure(4.14)**, and **two applications (examples)** was detailed in **figure(4.13)** as a **random net example** and in **figure(4.19)** as an **example for real roads between two major Jordanian cities**; from **Al Karak** to **Irbid** according to an official map of Jordan.

---

To conduct the A\* with Fuzzy Underestimates search technique use the following:

Initialize all vertices to "undiscovered."

goal = 0

While goal = 0

min = 1000

For j = 1 To 63

If w(j) = True Then

lbluft(j).ForeColor = vbRed

Else

lbluft(j).ForeColor = vbBlack

End If

Next j

For i = 2 To 63

If (w(i) = True And uft(i) < min) Or ((w(i) = True And uft(i) < min) And (lbl(i).Caption = a(2)))

Then

min = uft(i)

order = i

End If

Next i

For p = 1 To 62

For q = 2 To 63

If w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And uft(p) < uft(q) Then

w(q) = False

lblx(q).Visible = True

Elsif w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And uft(p) > uft(q)

Then

w(p) = False

lblx(p).Visible = True

End If

Next q

Next p

w(order) = False

lin(order - 1).BorderColor = vbBlue

lbl(order).ForeColor = vbBlue

If order <= 31 Then

If lbl(2 \* order).Visible = True Then

w(2 \* order) = True

lbl(2 \* order).ForeColor = vbRed

End If

If lbl(2 \* order + 1).Visible = True Then

w(2 \* order + 1) = True

lbl(2 \* order + 1).ForeColor = vbRed

End If

End If

If lbl(order).Caption = a(2) Then

For r = 1 To 63

If w(r) = True And v(r) >= uft(order) Then

lblx(r).Visible = True

goal = 0

Elsif w(r) = True And v(r) < uft(order) Then

goal = 0

Else

goal = 1

End If



```
    Next r
  End If
Wend
If goal = 1 Then
  check = order
  While check > 1
    lin(check - 1).BorderColor = &HC000&
    lbl(check).ForeColor = &HC000&
    check = Int(check / 2)
  Wend
End If
```

---

## Appendix 2 Simulation code:

### 'Go results screen with the six algorithms

```
iteration3 = 0
iteration4 = 0
iteration5 = 0
iteration6 = 0
iteration7 = 0
iteration8 = 0
time3 = 0
time4 = 0
time5 = 0
time6 = 0
time7 = 0
time8 = 0
space3 = 0
space4 = 0
space5 = 0
space6 = 0
space7 = 0
space8 = 0
ebf3 = 0
ebf4 = 0
ebf5 = 0
ebf6 = 0
ebf7 = 0
ebf8 = 0
```

### 'Branch & Bound Search

```
For i = 2 To 63
    lin(i - 1).BorderColor = vbBlack
    lbl(i).ForeColor = vbBlack
Next i
node = a(1)
flag = False
i = 2
steps = 0
lbl(1).ForeColor = &HC000&
For j = 1 To 63
    w(j) = False
Next j
For j = 1 To 63
    lblt(j).ForeColor = vbBlack
    lblgg(j).ForeColor = vbBlack
    lbluft(j).ForeColor = vbBlack
Next j
For j = 1 To 63
    lblx(j).Visible = False
Next j
iteration3 = 0
```

```

If lbl(2).Visible = True And lbl(3).Visible = True Then
    lbl(2).ForeColor = vbRed
    lbl(3).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lbl(3).ForeColor = vbRed
    time3 = time3 + 3
    If v(2) <= v(3) Then
        w(3) = True
        lbl(3).ForeColor = vbRed
        lin(1).BorderColor = vbBlue
        lbl(2).ForeColor = vbBlue
        time3 = time3 + 3
        If lbl(4).Visible Then
            w(4) = True
            lbl(4).ForeColor = vbRed
            time3 = time3 + 2
        End If
        If lbl(5).Visible Then
            w(5) = True
            lbl(5).ForeColor = vbRed
            time3 = time3 + 2
        End If
    Else
        w(2) = True
        lbl(2).ForeColor = vbRed
        lin(2).BorderColor = vbBlue
        lbl(3).ForeColor = vbBlue
        time3 = time3 + 3
        If lbl(6).Visible Then
            w(6) = True
            lbl(6).ForeColor = vbRed
            time3 = time3 + 2
        End If
        If lbl(7).Visible Then
            w(7) = True
            lbl(7).ForeColor = vbRed
            time3 = time3 + 2
        End If
    End If
ElseIf lbl(2).Visible = True And lbl(3).Visible = False Then
    lbl(2).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lin(1).BorderColor = vbBlue
    lbl(2).ForeColor = vbBlue
    time3 = time3 + 3
    If lbl(4).Visible Then
        w(4) = True
        lbl(4).ForeColor = vbRed
        time3 = time3 + 3
    End If
    If lbl(5).Visible Then
        w(5) = True
        lbl(5).ForeColor = vbRed
        time3 = time3 + 3
    End If
End If

```

```

iteration3 = iteration3 + 1
goal = 0
While goal = 0
  min = 1000
  For j = 1 To 63
    If w(j) = True Then
      lbl(j).ForeColor = vbRed
      time3 = time3 + 2
    Else
      lbl(j).ForeColor = vbBlack
    End If
  Next j
  For i = 2 To 63
    If (w(i) = True And v(i) < min) Or ((w(i) = True And v(i) < min) And (lbl(i).Caption = a(2)))
  Then
    min = v(i)
    order = i
  End If
  Next i
  w(order) = False
  lin(order - 1).BorderColor = vbBlue
  lbl(order).ForeColor = vbBlue
  iteration3 = iteration3 + 1
  time3 = time3 + 1
  If order <= 31 Then
    If lbl(2 * order).Visible = True Then
      w(2 * order) = True
      lbl(2 * order).ForeColor = vbRed
      time3 = time3 + 2
    End If
    If lbl(2 * order + 1).Visible = True Then
      w(2 * order + 1) = True
      lbl(2 * order + 1).ForeColor = vbRed
      time3 = time3 + 2
    End If
  End If
  If lbl(order).Caption = a(2) Then
    For r = 2 To 63
      If w(r) = True And v(r) >= v(order) Then
        lblx(r).Visible = True
        time3 = time3 + 2
        space3 = space3 + 1
        goal = 0
      ElseIf w(r) = True And v(r) < v(order) Then
        order1 = order
        order = r
        goal = 0
      Else
        goal = 1
      End If
    Next r
  End If
End While
If goal = 1 Then
  check = order
  While check > 1

```

```

        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
        time3 = time3 + 2
    Wend
End If
For i = 1 To 63
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        ebf3 = ebf3 + 1
    End If
Next i
For i = 2 To 3
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath3 = 1
    End If
Next i
For i = 4 To 7
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath3 = 2
    End If
Next i
For i = 8 To 15
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath3 = 3
    End If
Next i
For i = 16 To 31
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath3 = 4
    End If
Next i
For i = 32 To 63
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath3 = 5
    End If
Next i

```

## 'Branch & Bound Search with Crisp Underestimates

```

For i = 2 To 63
    lin(i - 1).BorderColor = vbBlack
    lbl(i).ForeColor = vbBlack
Next i
node = a(1)
flag = False
i = 2
steps = 0
lbl(1).ForeColor = &HC000&
For j = 1 To 63
    w(j) = False
Next j
For j = 1 To 63

```

```

    lblt(j).ForeColor = vbBlack
    lblgg(j).ForeColor = vbBlack
    lbluft(j).ForeColor = vbBlack
Next j
For j = 1 To 63
    lblx(j).Visible = False
Next j
iteration4 = 0
If lbl(2).Visible = True And lbl(3).Visible = True Then
    lblgg(2).ForeColor = vbRed
    lblgg(3).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lbl(3).ForeColor = vbRed
    time4 = time4 + 3
    If g(2) <= g(3) Then
        w(3) = True
        lbl(3).ForeColor = vbRed
        lin(1).BorderColor = vbBlue
        lbl(2).ForeColor = vbBlue
        time4 = time4 + 3
        If lbl(4).Visible Then
            w(4) = True
            lbl(4).ForeColor = vbRed
            time4 = time4 + 2
        End If
        If lbl(5).Visible Then
            w(5) = True
            lbl(5).ForeColor = vbRed
            time4 = time4 + 2
        End If
    Else
        w(2) = True
        lbl(2).ForeColor = vbRed
        lin(2).BorderColor = vbBlue
        lbl(3).ForeColor = vbBlue
        time4 = time4 + 3
        If lbl(6).Visible Then
            w(6) = True
            lbl(6).ForeColor = vbRed
            time4 = time4 + 2
        End If
        If lbl(7).Visible Then
            w(7) = True
            lbl(7).ForeColor = vbRed
            time4 = time4 + 2
        End If
    End If
Elseif lbl(2).Visible = True And lbl(3).Visible = False Then
    lblgg(2).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lin(1).BorderColor = vbBlue
    lbl(2).ForeColor = vbBlue
    time4 = time4 + 3
    If lbl(4).Visible Then
        w(4) = True

```

```

        lbl(4).ForeColor = vbRed
        time4 = time4 + 2
    End If
    If lbl(5).Visible Then
        w(5) = True
        lbl(5).ForeColor = vbRed
        time4 = time4 + 2
    End If
End If
iteration4 = iteration4 + 1
goal = 0
While goal = 0
    min = 1000
    For j = 1 To 63
        If w(j) = True Then
            lblgg(j).ForeColor = vbRed
            time4 = time4 + 2
        Else
            lblgg(j).ForeColor = vbBlack
        End If
    Next j
    For i = 2 To 63
        If (w(i) = True And g(i) < min) Or ((w(i) = True And g(i) < min) And (lbl(i).Caption = a(2)))
        Then
            min = g(i)
            order = i
        End If
    Next i
    w(order) = False
    lin(order - 1).BorderColor = vbBlue
    lbl(order).ForeColor = vbBlue
    iteration4 = iteration4 + 1
    time4 = time4 + 1
    If order <= 31 Then
        If lbl(2 * order).Visible = True Then
            w(2 * order) = True
            lbl(2 * order).ForeColor = vbRed
            time4 = time4 + 2
        End If
        If lbl(2 * order + 1).Visible = True Then
            w(2 * order + 1) = True
            lbl(2 * order + 1).ForeColor = vbRed
            time4 = time4 + 2
        End If
    End If
    If lbl(order).Caption = a(2) Then
        For r = 1 To 63
            If w(r) = True And v(r) >= g(order) Then
                lblx(r).Visible = True
                time4 = time4 + 2
                space4 = space4 + 1
                goal = 0
            ElseIf w(r) = True And v(r) < g(order) Then
                goal = 0
            Else
                goal = 1
            End If
        Next r
    End If
End While

```

```

    Next r
  End If
Wend
If goal = 1 Then
  check = order
  While check > 1
    lin(check - 1).BorderColor = &HC000&
    lbl(check).ForeColor = &HC000&
    check = Int(check / 2)
    time4 = time4 + 2
  Wend
End If
For i = 1 To 63
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    ebf4 = ebf4 + 1
  End If
Next i

For i = 2 To 3
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath4 = 1
  End If
Next i
For i = 4 To 7
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath4 = 2
  End If
Next i
For i = 8 To 15
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath4 = 3
  End If
Next i
For i = 16 To 31
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath4 = 4
  End If
Next i
For i = 32 To 63
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath4 = 5
  End If
Next i

```

## 'Branch & Bound Search with Fuzzy Underestimates.

```

For i = 2 To 63
  lin(i - 1).BorderColor = vbBlack
  lbl(i).ForeColor = vbBlack
Next i
node = a(1)
flag = False
i = 2
steps = 0
lbl(1).ForeColor = &HC000&

For j = 1 To 63
  w(j) = False

```



```

Next j
For j = 1 To 63
    lblt(j).ForeColor = vbBlack
    lblgg(j).ForeColor = vbBlack
    lbluft(j).ForeColor = vbBlack
Next j
For j = 1 To 63
    lblx(j).Visible = False
Next j
iteration5 = 0
If lbl(2).Visible = True And lbl(3).Visible = True Then
    lbluft(2).ForeColor = vbRed
    lbluft(3).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lbl(3).ForeColor = vbRed
    time5 = time5 + 3
    If uft(2) <= uft(3) Then
        w(3) = True
        lbl(3).ForeColor = vbRed
        lin(1).BorderColor = vbBlue
        lbl(2).ForeColor = vbBlue
        time5 = time5 + 3
        If lbl(4).Visible Then
            w(4) = True
            lbl(4).ForeColor = vbRed
            time5 = time5 + 2
        End If
        If lbl(5).Visible Then
            w(5) = True
            lbl(5).ForeColor = vbRed
            time5 = time5 + 2
        End If
    Else
        w(2) = True
        lbl(2).ForeColor = vbRed
        lin(2).BorderColor = vbBlue
        lbl(3).ForeColor = vbBlue
        time5 = time5 + 3
        If lbl(6).Visible Then
            w(6) = True
            lbl(6).ForeColor = vbRed
            time5 = time5 + 2
        End If
        If lbl(7).Visible Then
            w(7) = True
            lbl(7).ForeColor = vbRed
            time5 = time5 + 2
        End If
    End If
ElseIf lbl(2).Visible = True And lbl(3).Visible = False Then
    lbluft(2).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lin(1).BorderColor = vbBlue
    lbl(2).ForeColor = vbBlue
    time5 = time5 + 3
    If lbl(4).Visible Then
        w(4) = True
        lbl(4).ForeColor = vbRed
        time5 = time5 + 2
    End If

```

```

End If
If lbl(5).Visible Then
    w(5) = True
    lbl(5).ForeColor = vbRed
    time5 = time5 + 23
End If
End If
iteration5 = iteration5 + 1
goal = 0
While goal = 0
    min = 1000
    For j = 1 To 63
        If w(j) = True Then
            lbluft(j).ForeColor = vbRed
            time5 = time5 + 2
        Else
            lbluft(j).ForeColor = vbBlack
        End If
    Next j

    For i = 2 To 63
        If (w(i) = True And uft(i) < min) Or ((w(i) = True And uft(i) < min) And (lbl(i).Caption = a(2)))
        Then
            min = uft(i)
            order = i
        End If
    Next i
    w(order) = False
    lin(order - 1).BorderColor = vbBlue
    lbl(order).ForeColor = vbBlue
    iteration5 = iteration5 + 1
    time5 = time5 + 1
    If order <= 31 Then
        If lbl(2 * order).Visible = True Then
            w(2 * order) = True
            lbl(2 * order).ForeColor = vbRed
            time5 = time5 + 2
        End If
        If lbl(2 * order + 1).Visible = True Then
            w(2 * order + 1) = True
            lbl(2 * order + 1).ForeColor = vbRed
            time5 = time5 + 2
        End If
    End If
    If lbl(order).Caption = a(2) Then
        For r = 1 To 63
            If w(r) = True And v(r) >= uft(order) Then
                lblx(r).Visible = True
                time5 = time5 + 2
                space5 = space5 + 1
                goal = 0
            ElseIf w(r) = True And v(r) < uft(order) Then
                goal = 0
            Else
                goal = 1
            End If
        Next r
    End If
End While
Wend

```

```

If goal = 1 Then
  check = order
  While check > 1
    lin(check - 1).BorderColor = &HC000&
    lbl(check).ForeColor = &HC000&
    check = Int(check / 2)
    time5 = time5 + 2
  Wend
End If
For i = 1 To 63
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    ebf5 = ebf5 + 1
  End If
Next i
For i = 2 To 3
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath5 = 1
  End If
Next i
For i = 4 To 7
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath5 = 2
  End If
Next i
For i = 8 To 15
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath5 = 3
  End If
Next i
For i = 16 To 31
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath5 = 4
  End If
Next i
For i = 32 To 63
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath5 = 5
  End If
Next i

```

## 'Branch & Bound Search with Dynamic Programming.

```

For i = 2 To 63
  lin(i - 1).BorderColor = vbBlack
  lbl(i).ForeColor = vbBlack
Next i
node = a(1)
flag = False
i = 2
steps = 0
lbl(1).ForeColor = &HC000&
For j = 1 To 63
  w(j) = False
Next j
For j = 1 To 63

```

```

    lblt(j).ForeColor = vbBlack
    lblgg(j).ForeColor = vbBlack
    lbluft(j).ForeColor = vbBlack
Next j
For j = 1 To 63
    lblx(j).Visible = False
Next j
iteration6 = 0
If lbl(2).Visible = True And lbl(3).Visible = True Then
    lblt(2).ForeColor = vbRed
    lblt(3).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lbl(3).ForeColor = vbRed
    time6 = time6 + 3
    If v(2) <= v(3) Then
        w(3) = True
        lbl(3).ForeColor = vbRed
        lin(1).BorderColor = vbBlue
        lbl(2).ForeColor = vbBlue
        time6 = time6 + 3
        If lbl(4).Visible Then
            w(4) = True
            lbl(4).ForeColor = vbRed
            time6 = time6 + 2
        End If
        If lbl(5).Visible Then
            w(5) = True
            lbl(5).ForeColor = vbRed
            time6 = time6 + 2
        End If
    Else
        w(2) = True
        lbl(2).ForeColor = vbRed
        lin(2).BorderColor = vbBlue
        lbl(3).ForeColor = vbBlue
        time6 = time6 + 3
        If lbl(6).Visible Then
            w(6) = True
            lbl(6).ForeColor = vbRed
            time6 = time6 + 2
        End If
        If lbl(7).Visible Then
            w(7) = True
            lbl(7).ForeColor = vbRed
            time6 = time6 + 2
        End If
    End If
Elseif lbl(2).Visible = True And lbl(3).Visible = False Then
    lblt(2).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lin(1).BorderColor = vbBlue
    lbl(2).ForeColor = vbBlue
    time6 = time6 + 3
    If lbl(4).Visible Then
        w(4) = True

```

```

        lbl(4).ForeColor = vbRed
        time6 = time6 + 2
    End If
    If lbl(5).Visible Then
        w(5) = True
        lbl(5).ForeColor = vbRed
        time6 = time6 + 2
    End If
End If
iteration6 = iteration6 + 1
goal = 0
While goal = 0
    min = 1000
    For j = 1 To 63
        If w(j) = True Then
            lblt(j).ForeColor = vbRed
            time6 = time6 + 2
        Else
            lblt(j).ForeColor = vbBlack
        End If
    Next j
    For i = 2 To 63
        If (w(i) = True And v(i) < min) Or ((w(i) = True And v(i) < min) And (lbl(i).Caption = a(2)))
Then
            min = v(i)
            order = i
        End If
    Next i
    For p = 1 To 62
        For q = 2 To 63
            If ((w(p) = True And w(q) = True) Or (w(p) = True And lbl(q).ForeColor = vbBlue) Or (w(q)
= True And lbl(p).ForeColor = vbBlue)) And lbl(p).Caption = lbl(q).Caption And v(p) < v(q) Then
                w(q) = False
                lblx(q).Visible = True
                time6 = time6 + 2
                space6 = space6 + 1
                lbl(q).ForeColor = vbBlack
            ElseIf ((w(p) = True And w(q) = True) Or (w(p) = True And lbl(q).ForeColor = vbBlue) Or
(w(q) = True And lbl(p).ForeColor = vbBlue)) And lbl(p).Caption = lbl(q).Caption And v(p) > v(q)
Then
                w(p) = False
                lblx(p).Visible = True
                time6 = time6 + 3
                space6 = space6 + 1
                lbl(p).ForeColor = vbBlack
            End If
        Next q
    Next p
    w(order) = False
    lin(order - 1).BorderColor = vbBlue
    lbl(order).ForeColor = vbBlue
    iteration6 = iteration6 + 1
    time6 = time6 + 1
    If order <= 31 Then
        If lbl(2 * order).Visible = True Then
            w(2 * order) = True

```

```

        lbl(2 * order).ForeColor = vbRed
        time6 = time6 + 2
    End If
    If lbl(2 * order + 1).Visible = True Then
        w(2 * order + 1) = True
        lbl(2 * order + 1).ForeColor = vbRed
    End If
End If
If lbl(order).Caption = a(2) Then
    For r = 1 To 63
        If w(r) = True And v(r) >= v(order) Then
            lblx(r).Visible = True
            time6 = time6 + 2
            space6 = space6 + 1
            goal = 0
        ElseIf w(r) = True And v(r) < v(order) Then
            goal = 0
        Else
            goal = 1
        End If
    Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
        time6 = time6 + 2
    Wend
End If
For i = 1 To 63
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        ebf6 = ebf6 + 1
    End If
Next i
For i = 2 To 3
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath6 = 1
    End If
Next i
For i = 4 To 7
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath6 = 2
    End If
Next i
For i = 8 To 15
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath6 = 3
    End If
Next i
For i = 16 To 31
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath6 = 4
    End If

```

```

Next i
For i = 32 To 63
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath6 = 5
    End If
Next i

```

## 'A\* Search.

```

For i = 2 To 63
    lin(i - 1).BorderColor = vbBlack
    lbl(i).ForeColor = vbBlack
Next i
node = a(1)
flag = False
i = 2
steps = 0
lbl(1).ForeColor = &HC000&
For j = 1 To 63
    w(j) = False
Next j
For j = 1 To 63
    lblt(j).ForeColor = vbBlack
    lblgg(j).ForeColor = vbBlack
    lbluft(j).ForeColor = vbBlack
Next j
For j = 1 To 63
    lblx(j).Visible = False
Next j
iteration7 = 0
If lbl(2).Visible = True And lbl(3).Visible = True Then
    lblgg(2).ForeColor = vbRed
    lblgg(3).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lbl(3).ForeColor = vbRed
    time7 = time7 + 3
    If g(2) <= g(3) Then
        w(3) = True
        lbl(3).ForeColor = vbRed
        lin(1).BorderColor = vbBlue
        lbl(2).ForeColor = vbBlue
        time7 = time7 + 3
        If lbl(4).Visible Then
            w(4) = True
            lbl(4).ForeColor = vbRed
            time7 = time7 + 2
        End If
        If lbl(5).Visible Then
            w(5) = True
            lbl(5).ForeColor = vbRed
            time7 = time7 + 2
        End If
    Else

```

```

w(2) = True
lbl(2).ForeColor = vbRed
lin(2).BorderColor = vbBlue
lbl(3).ForeColor = vbBlue
time7 = time7 + 3
If lbl(6).Visible Then
    w(6) = True
    lbl(6).ForeColor = vbRed
    time7 = time7 + 2
End If
If lbl(7).Visible Then
    w(7) = True
    lbl(7).ForeColor = vbRed
    time7 = time7 + 2
End If
End If
Elseif lbl(2).Visible = True And lbl(3).Visible = False Then
    lblgg(2).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lin(1).BorderColor = vbBlue
    lbl(2).ForeColor = vbBlue
    time7 = time7 + 3
    If lbl(4).Visible Then
        w(4) = True
        lbl(4).ForeColor = vbRed
        time7 = time7 + 2
    End If
    If lbl(5).Visible Then
        w(5) = True
        lbl(5).ForeColor = vbRed
        time7 = time7 + 2
    End If
End If
iteration7 = iteration7 + 1
goal = 0
While goal = 0
    min = 1000
    For j = 1 To 63
        If w(j) = True Then
            lblgg(j).ForeColor = vbRed
            time7 = time7 + 2
        Else
            lblgg(j).ForeColor = vbBlack
        End If
    Next j
    For i = 2 To 63
        If (w(i) = True And g(i) < min) Or ((w(i) = True And g(i) < min) And (lbl(i).Caption = a(2)))
        Then
            min = g(i)
            order = i
        End If
    Next i
    For p = 1 To 62
        For q = 2 To 63

```



```

    If w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And g(p) < g(q) Then
        w(q) = False
        lblx(q).Visible = True
        time7 = time7 + 2
        space7 = space7 + 1
    ElseIf w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And g(p) > g(q)
Then
        w(p) = False
        lblx(p).Visible = True
        time7 = time7 + 2
        space7 = space7 + 1
    End If
Next q
Next p
w(order) = False
lin(order - 1).BorderColor = vbBlue
lbl(order).ForeColor = vbBlue
iteration7 = iteration7 + 1
time7 = time7 + 1
If order <= 31 Then
    If lbl(2 * order).Visible = True Then
        w(2 * order) = True
        lbl(2 * order).ForeColor = vbRed
        time7 = time7 + 2
    End If

    If lbl(2 * order + 1).Visible = True Then
        w(2 * order + 1) = True
        lbl(2 * order + 1).ForeColor = vbRed
        time7 = time7 + 2
    End If
End If
If lbl(order).Caption = a(2) Then
    For r = 1 To 63
        If w(r) = True And v(r) >= g(order) Then
            lblx(r).Visible = True
            time7 = time7 + 4
            space7 = space7 + 1
            goal = 0
        ElseIf w(r) = True And v(r) < g(order) Then
            goal = 0
        Else
            goal = 1
        End If
    Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
        time7 = time7 + 2
    Wend
End If

```

```

For i = 1 To 63
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    ebf7 = ebf7 + 1
  End If
Next i
For i = 2 To 3
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath7 = 1
  End If
Next i
For i = 4 To 7
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath7 = 2
  End If
Next i
For i = 8 To 15
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath7 = 3
  End If
Next i
For i = 16 To 31
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath7 = 4
  End If
Next i
For i = 32 To 63
  If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
    dpath7 = 5
  End If
Next i

```

### 'A\* Search with Fuzzy Underestimation.

```

For i = 2 To 63
  lin(i - 1).BorderColor = vbBlack
  lbl(i).ForeColor = vbBlack
Next i
node = a(1)
flag = False
i = 2
steps = 0
lbl(1).ForeColor = &HC000&
For j = 1 To 63
  w(j) = False
Next j
For j = 1 To 63
  lblt(j).ForeColor = vbBlack
  lblgg(j).ForeColor = vbBlack
  lbluft(j).ForeColor = vbBlack
Next j
For j = 1 To 63
  lblx(j).Visible = False
Next j

```

```

iteration8 = 0
If lbl(2).Visible = True And lbl(3).Visible = True Then
    lbluft(2).ForeColor = vbRed
    lbluft(3).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lbl(3).ForeColor = vbRed
    time8 = time8 + 3
    If uft(2) <= uft(3) Then
        w(3) = True
        lbl(3).ForeColor = vbRed
        lin(1).BorderColor = vbBlue
        lbl(2).ForeColor = vbBlue
        time8 = time8 + 3
        If lbl(4).Visible Then
            w(4) = True
            lbl(4).ForeColor = vbRed
            time8 = time8 + 2
        End If
        If lbl(5).Visible Then
            w(5) = True
            lbl(5).ForeColor = vbRed
            time8 = time8 + 2
        End If
    Else
        w(2) = True
        lbl(2).ForeColor = vbRed
        lin(2).BorderColor = vbBlue
        lbl(3).ForeColor = vbBlue
        time8 = time8 + 3
        If lbl(6).Visible Then
            w(6) = True
            lbl(6).ForeColor = vbRed
            time8 = time8 + 2
        End If
        If lbl(7).Visible Then
            w(7) = True
            lbl(7).ForeColor = vbRed
            time8 = time8 + 2
        End If
    End If
ElseIf lbl(2).Visible = True And lbl(3).Visible = False Then
    lbluft(2).ForeColor = vbRed
    lbl(2).ForeColor = vbRed
    lin(1).BorderColor = vbBlue
    lbl(2).ForeColor = vbBlue
    time8 = time8 + 3
    If lbl(4).Visible Then
        w(4) = True
        lbl(4).ForeColor = vbRed
        time8 = time8 + 2
    End If
    If lbl(5).Visible Then
        w(5) = True
        lbl(5).ForeColor = vbRed
        time8 = time8 + 2
    End If
End If

```

```

iteration8 = iteration8 + 1
goal = 0
While goal = 0
  min = 1000
  For j = 1 To 63
    If w(j) = True Then
      lbluft(j).ForeColor = vbRed
      time8 = time8 + 2
    Else
      lbluft(j).ForeColor = vbBlack
    End If
  Next j
  For i = 2 To 63
    If (w(i) = True And uft(i) < min) Or ((w(i) = True And uft(i) < min) And (lbl(i).Caption = a(2)))
  Then
    min = uft(i)
    order = i
  End If
  Next i
  For p = 1 To 62
    For q = 2 To 63
      If w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And uft(p) < uft(q) Then
        w(q) = False
        lblx(q).Visible = True
        space8 = space8 + 1
      Elseif w(p) = True And w(q) = True And lbl(p).Caption = lbl(q).Caption And uft(p) > uft(q)
  Then
        w(p) = False
        lblx(p).Visible = True
        space8 = space8 + 1
      End If
    Next q
  Next p
  w(order) = False
  lin(order - 1).BorderColor = vbBlue
  lbl(order).ForeColor = vbBlue
  iteration8 = iteration8 + 1
  time8 = time8 + 1
  If order <= 31 Then
    If lbl(2 * order).Visible = True Then
      w(2 * order) = True
      lbl(2 * order).ForeColor = vbRed
      time8 = time8 + 2
    End If
    If lbl(2 * order + 1).Visible = True Then
      w(2 * order + 1) = True
      lbl(2 * order + 1).ForeColor = vbRed
      time8 = time8 + 2
    End If
  End If
  If lbl(order).Caption = a(2) Then
    For r = 1 To 63
      If w(r) = True And v(r) >= uft(order) Then
        lblx(r).Visible = True
        space8 = space8 + 2
        goal = 0
      End If
    Next r
  End If
End While

```

```

ElseIf w(r) = True And v(r) < uft(order) Then
    goal = 0
Else
    goal = 1
End If
Next r
End If
Wend
If goal = 1 Then
    check = order
    While check > 1
        lin(check - 1).BorderColor = &HC000&
        lbl(check).ForeColor = &HC000&
        check = Int(check / 2)
        time8 = time8 + 2
    Wend
End If
For i = 1 To 63
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        ebf8 = ebf8 + 1
    End If
Next i
For i = 2 To 3
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath8 = 1
    End If
Next i
For i = 4 To 7
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath8 = 2
    End If
Next i
For i = 8 To 15
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath8 = 3
    End If
Next i
For i = 16 To 31
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath8 = 4
    End If
Next i
For i = 32 To 63
    If lbl(i).ForeColor = &HC000& Or lbl(i).ForeColor = vbBlue Or lblx(i).Visible = True Then
        dpath8 = 5
    End If
Next i

```

## Appendix 3

### Fuzzy Logic

#### Examples where fuzzy logic is used

- Automobile and other vehicle subsystems, such as ABS and cruise control (e.g. Tokyo monorail)
- Air conditioners
- The MASSIVE engine used in the Lord of the Rings films, which helped show huge scale armies create random, yet orderly movements
- Cameras
- Digital image processing, such as edge detection
- Rice cookers
- Dishwashers
- Elevators
- Washing machines and other home appliances
- Video game artificial intelligence
- Language filters on message boards and chat rooms for filtering out offensive text
- Pattern recognition in Remote Sensing
- Gambit System in Final Fantasy XII

Fuzzy logic has also been incorporated into some microcontrollers and microprocessors, for instance, the Freescale 68HC12

#### Fuzzy Logic Bibliography

- Ahmad M. Ibrahim, *Introduction to Applied Fuzzy Electronics*, **ISBN 0-13-206400-6**
- Von Altrock C., *Fuzzy Logic and NeuroFuzzy Applications Explained* (2002), **ISBN 0-13-368465-2**
- Biacino L., Gerla G., Fuzzy logic, continuity and effectiveness, *Archive for Mathematical Logic*, 41, (2002), 643-667.
- Cignoli R., D'Ottaviano I. M. L. , Mundici D. , "Algebraic Foundations of Many-Valued Reasoning". Kluwer, Dordrecht, 1999.
- Cox E., *The Fuzzy Systems Handbook* (1994), **ISBN 0-12-194270-8**
- Elkan C.. *The Paradoxical Success of Fuzzy Logic*. November 1993. Available from Elkan's home page.
- Hájek P., *Metamathematics of fuzzy logic*. Kluwer 1998.
- Hájek P., Fuzzy logic and arithmetical hierarchy, *Fuzzy Sets and Systems*, 3, (1995), 359-363.
- Höppner F., Klawonn F., Kruse R. and Runkler T., *Fuzzy Cluster Analysis* (1999), **ISBN 0-471-98864-2**.
- Klir G. and Folger T., *Fuzzy Sets, Uncertainty, and Information* (1988), **ISBN 0-13-345984-5**.
- Klir G. , UTE H. St. Clair and Bo Yuan *Fuzzy Set Theory Foundations and Applications*, 1997.

- 
- Klir G. and Bo Yuan, *Fuzzy Sets and Fuzzy Logic* (1995) **ISBN 0-13-101171-5**
- **Bart Kosko**, *Fuzzy Thinking: The New Science of Fuzzy Logic* (1993), Hyperion. **ISBN 0-7868-8021-X**
- Gerla G., Effectiveness and Multivalued Logics, *Journal of Symbolic Logic*, 71 (2006) 137-162.
- Montagna F., Three complexity problems in quantified fuzzy logic. *Studia Logica*, 68,(2001), 143-152.
- Scarpellini B., Die Nichaxiomatisierbarkeit des unendlichwertigen Prädikatenkalküls von Łukasiewicz, *J. of Symbolic Logic*, 27,(1962), 159-170.
- Yager R. and Filev D., *Essentials of Fuzzy Modeling and Control* (1994), **ISBN 0-471-01761-2**
- Zimmermann H., *Fuzzy Set Theory and its Applications* (2001), **ISBN 0-7923-7435-5**.
- Kevin M. Passino and Stephen Yurkovich, *Fuzzy Control*, Addison Wesley Longman, Menlo Park, CA, 1998.
- Wiedermann J. , Characterizing the super-Turing computing power and efficiency of classical fuzzy Turing machines, *Theor. Comput. Sci.* 317, (2004), 61-69.
- Zadeh L.A., Fuzzy algorithms, *Information and Control*, 5,(1968), 94-102.
- Zadeh L.A., Fuzzy Sets, "Information and Control", 8 (1965) 338353.
- Zemankova-Leech, M., *Fuzzy Relational Data Bases* (1983), Ph. D. Dissertation, Florida State University.

## Fuzzy Logic Sample applications

- **Agriculture**
- **GIS**
- **Image Processing** (in **Internet Archive**)
- **Machine Learning**
- **Machine Vision**
- **Medicine**
- **Model Validation**
- **OCR**
- **Robot Navigation**
- **Shape Recognition**
- **Telecommunications**

•

