Uninformed and Informed Search Techniques in Artificial Intelligence

Khaled A. O. Algasi

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science
in
Applied Mathematics and Computer Science

Eastern Mediterranean University
November 2017
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Assoc. Prof. Dr. Ali Hakan Ulusoy
Acting Director

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Applied Mathematics and Computer Science.

Prof. Dr. Nazim Mahmudov
Chair, Department of Mathematics

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Applied Mathematics and Computer Science.

Prof. Dr. Rashad Aliyev
Supervisor

Examining Committee

1. Prof. Dr. Rashad Aliyev
2. Asst. Prof. Dr. Ersin Kuset Bodur
3. Asst. Prof. Dr. Tolgay Karanfiller

ABSTRACT

In this master thesis the search techniques in Artificial Intelligence are analyzed. The search techniques are grouped into two main categories which are uninformed search techniques and informed search techniques.

Such uninformed search techniques as breadth-first search, depth-first search, depth-limited search, iterative deepening search, uniform cost search, and bidirectional search are considered.

The best-first search, greedy best-first search, A* search and hill climbing techniques as paradigms of informed search techniques are studied.

The completeness, optimality, time complexity, and space complexity properties of all above mentioned search techniques are discussed.

The Dijkstra’s algorithm is used to find the shortest paths from the initial node to all other nodes in a weighted digraph.

Keywords: Breadth-first search, Depth-first search, Depth-limited search, Iterative deepening search, Uniform cost search, Bidirectional search, Best-first search, Greedy best-first search, A* search, Hill climbing search, Dijkstra’s algorithm

ÖZ

Bu master tezinde yapay zekanın arama yöntemleri incelenir. Yapay zekada arama yöntemleri sezgisel olmayan ve sezgisel arama yöntemleri olarak iki esas kategoriye ayrılır.

Önce genişliğine arama, önce derinliğine arama, derinlik sınırlandırmalı arama, yineli derinleştirmeli arama, düzenli maliyet arama ve çift yönlü arama yöntemleri sezgisel olmayan yöntemler olarak dikkate alınır.

En iyi öncelikli arama, açgözlü en iyi öncelikli arama, A* arama ve tepe tırmanma yöntemleri sezgisel arama yöntemlerinin paradigmaları olarak incelenir.

Yukarıda adı geçen tüm arama yöntemlerinin tamlık, optimallik, zaman karmaşıklığı ve bellek karmaşıklığı özellikleri tartışılır.

Dijkstra algoritması kullanarak ağırlıklı yönlü grafikte başlangıç düğümünden diğer dügümlere gidilebilecek en kısa yol bulunur.

Anahtar Kelimeler: Önce genişliğine arama, Önce derinliğine arama, Derinlik sınırlandırmalı arama, Yineli derinleştirmeli arama, Düzenli maliyet arama, Çift yönlü arama, En iyi öncelikli arama, Açgözlü en iyi öncelikli arama, A* arama, Tepe tirmanma arama, Dijkstra algoritması

ACKNOWLEDGMENT

I would like to express my sincere thanks to my supervisor Prof. Dr. Rashad Aliyev for his guidance and support to complete my master thesis.

My special thanks go to my wife who always encouraged me during my studies. She knew how to keep the family’s love even when I used to be busy.

I am grateful to all my family’s members who provided all kind of help and support.

TABLE OF CONTENTS

ABSTRACT iii

ÖZ iv

ACKNOWLEDGMENT v

LIST OF FIGURES vii

LIST OF TABLES ix

1 INTRODUCTION 1

2 REVIEW OF EXISTING LITERATURE ON UNINFORMED AND INFORMED SEARCH TECHNIQUES 5

3 UNINFORMED SEARCH TECHNIQUES 11

3.1 Breadth-first search 11

3.2 Depth-first search 15

3.3 Depth-limited search 18

3.4 Iterative deepening search 19

3.5 Uniform cost search 20

3.6 Bidirectional search 22

4 INFORMED SEARCH TECHNIQUES 24

4.1 Best-first search 24

4.2 Greedy best-first search 27

4.3 A* search strategy 30

4.4 Hill climbing search 35

4.5 Dijkstra’s Algorithm 36

5 CONCLUSION 46

REFERENCES 48

LIST OF FIGURES

Figure 1: Breadth-first traversal of tree…………………………………………..12

Figure 2: The 16-node graph used for uninformed search techniques………………13

Figure 3: Graphical representation of order of node expansion in breadth-first search from the initial node N_1 to the goal node N_9…………………………………......14

Figure 4: Depth-first traversal of tree…………. ...…………………………………15

Figure 5: Graphical representation of order of node expansion in depth-first search from the initial node N_1 to the goal node N_7………………………………………..17

Figure 6: Graphical representation of order of node expansion in depth-limited search from the initial node N_1 to the goal node N_7……………………………..............18

Figure 7: Graphical representation of order of node expansion in iterative deepening search from the initial node N_1 to the goal node N_9………………………………...20

Figure 8: Weighted digraph used for the uniform cost search………………………21

Figure 9: Graphical representation of steps of best-first search example..…………26

Figure 10: Digraph for the greedy best-first search with straight line distance from each node to the goal node…………………………………………………………..28

Figure 11: Steps of nodes' expansion in digraph for the greedy best-first search…..29

Figure 12: Digraph for A* search with straight line distance from each node to the goal node……………………………………………………………………………31

Figure 13: Optimal solution of the 8-puzzle game for the given initial and goal states…………………………………………………………………………………34

Figure 14: Graphical representation of order of node expansion in Hill climbing search………………………………………………………………………………..35

Figure 15: Initial form of weighted digraph used for Dijkstra's algorithm…………39

Figure 16: Weighted digraph after the step 1 of Dijkstra's algorithm………………40

Figure 17: Weighted digraph after the step 2 of Dijkstra's algorithm……..………..42

Figure 18: Weighted digraph after the step 3 of Dijkstra's algorithm……..………...43

Figure 19: Weighted digraph after the step 4 of Dijkstra's algorithm………………44

LIST OF TABLES

Table 1: Shortest paths after the initial step 0 of Dijkstra’s algorithm……………...39

Table 2: Shortest paths after the step 1 of Dijkstra’s algorithm…………………….41

Table 3: Shortest paths after the step 2 of Dijkstra’s algorithm…………………….42

Table 4: Shortest paths after the step 3 of Dijkstra’s algorithm…………………….43

Table 5: Shortest paths after completing all the steps of Dijkstra’s algorithm……...45

Chapter 1

INTRODUCTION

We are living in an era within which most things we are doing and our daily tasks are being computerized with the goal to enhance their sustainability and reduce the human effort. When considering daily tasks of individual, it appears that these tasks can be classified in subcategories. The first subcategory is made of jobs that require physical power, whereas the second subcategory contains jobs that require brainstorming. In the latest mentioned, one can face complex calculation. Human brain might require several hours or even days to perform such calculation, whereas a computer with “good instructions” might be able to perform the computation within a very short time.

It is known that a computer is a machine which is able to perform tasks given to it only; it is therefore “very important” to deal with a well-defined problem, and to know constraints and solving strategy of this problem in order to use the computer’s abilities.

Artificial Intelligence appears to be a science that studies the simulation of the human brain’s way of thinking and tries to implement it on computers. In other words, in Artificial Intelligence the goal is to emulate “smart computer” which means a computer can act in a similar way like a human brain.

The main difference between human and computer is that a human has a brain, therefore can think and adapt his/her behavior according to circumstances. A computer does not have brain like human but can speedily perform some well-defined instructions.

From a general point of view, various problems in science require to be solved by optimization techniques. The variables or constraints involved in problems are usually non commensurate that make the task much more difficult. The selection of a suitable optimization function becomes more difficult as the number of variables increases.

There is usually a trade-off to be considered while implementing the solution to a complex optimization problem. The robustness of the solution versus the time or space complexity is required to reach the goal. The complexity is considered as the essence of optimization. An approach used for solving the Artificial Intelligence problem might be called optimal approach if and only if it requires less time and optimal space consumption than other approaches.

Solving any optimization problem usually depends on how it is stated or defined. A well-defined problem might be solved much easier than a problem in which it is not so easy to identify the unknown and belonging space.

Artificial Intelligence is the field of computer science intending to solve problems through mathematical modeling. This field also studies the different search techniques. Indeed, the goal is to find a good algorithm to reach the target or solution satisfying the initial conditions of a given problem. Human being is often able to solve a given problem by using his/her intuition although a formal approach to solve that problem might exist. Human may require a lot of time to solve a given problem. However, a computer can help solving a problem if it is written in a language which is understandable by the machine.

In Artificial Intelligence, search algorithms are implemented to solve various problems. There exist two categories of search algorithms: uninformed search also called blind search and informed search also called heuristic search.

An uninformed search method is characterized by the fact that no prior knowledge, specification or hint to the solution of the problem is given in advance.

The informed search strategy is characterized by the fact that some prior knowledge or hint as heuristic function is given. The heuristic helps improve the power and performance of the algorithm.

In the past years the developed search techniques were mostly based on the assumption that the problem has a single, common variable which is considered as a real valued variable. The solution set in this case possesses the capability to be completely lexicographically ordered. Such assumption made things easier for some extensions.

The uninformed and informed search strategies which are considered in this master thesis can be evaluated based on the following four criteria: completeness, optimality, time complexity, and space complexity.

Completeness: This property intends to check whether the algorithm guarantees to find the solution of the problem, if any exists.

Optimality: This property investigates whether or not the strategy provides the optimal solution.

Time complexity: This property evaluates the time required to find the solution of the problem.

Space complexity: This property evaluates the memory, and space required to run the search algorithm.

Chapter 2

REVIEW OF EXISTING LITERATURE ON UNINFORMED AND INFORMED SEARCH TECHNIQUES

This chapter is a general review of some works done in search techniques. We extend our focus on both uninformed and informed search techniques as well as on their applications. Some existing bibliographies are studied and the following appears as important and required to well-done the task we have in this thesis.

The feasible performance evaluation of the proposed method is what one usually attempts to find an optimal solution of the problem. In Artificial Intelligence, it is common to evaluate search methods based on their performances. In [1], for instance, some common and well-known search algorithms as breadth first, depth first, A^* search, greedy best first, and Hill climbing are explored and their performances are compared. The appropriateness of the specific search technique over other techniques is identified for different situations. The evaluations are done by obtaining the correct solutions to the N-puzzle and 8 Queen Puzzle problems. The experimental results show that A^* search is the best algorithm for solving both N puzzle and 8 Queen puzzle.

The approach based on the integration of the breadth-first and depth-first search techniques in a single algorithm to possess the strengths of both approaches is realized for the treewidth problem [2]. The benefits of the proposed novel combination of above search strategies in comparison with using either method alone are justified.

In [3], the Halstead's volume and Cyclomatic number complexity measures are used for the application to the breadth-first and depth-first search techniques. It is defined that in taking into consideration the program volume, the Pascal and C languages are best programming languages for using on breadth-first and depth-first search techniques, respectively. In taking into account the program difficulty and program effect, both search techniques are suitable to be implemented using programming language Pascal. The implementation of Visual Basic programming language is a better choice in computing the values of cyclomatic number metric of the complexity measure.

In [4], the different search algorithms are compared to find the best approach for solving a particular problem, because various algorithms behave differently to solve problems. So the goal is to categorize the search techniques with respect to their performances and specifications about the type of problem either is more powerful to be used.

Several search algorithms to find the low cost solution of the chosen method (MIN problems) are discussed in [5]. Meanwhile the search algorithms are compared in terms of observing the most convenient approach for solving the high reward (MAX problems). It is stated that the uninformed search techniques being effective for MIN problems can be ineffective for MAX problems. At the same time, the heuristic search techniques can be successfully applied to MAX problems.

The uninformed best-first search algorithm developed in [6] can find all the possible group paths which consist of collection of lightpaths. The proposed algorithm can find the optimal (minimum) number of Add/Drop Multiplexers in Wavelength division multiplexing. The results of computational experiments show the efficiency of the developed search technique.

The paper [7] discusses the replanning algorithm to be used for nondeterministic domains. This algorithm is used as incremental heuristic minimax search algorithm. The efficiency of the algorithm in continuous domain is justified by the implementation of the parti-game reinforcement-learning algorithm which can use the incremental search, uninformed search or informed search techniques. The experiments show that the implementation of the parti-game reinforcement-learning algorithm with Minimax LPA* provides better results than other algorithms.

Three different approaches - uninformed search algorithm, informed search algorithm and evolutionary algorithm for semantic web service compositions are analyzed and compared in [8]. The efficiency of all three approaches to provide the correct results to the requests is investigated.

In [9], three informed and three uninformed search techniques of Artificial Intelligence are considered to solve the optimization problem of the shortest path. The problem is related with investigation of the intelligent travel planning based on some cities in Borneo Island. It is defined that compare to most uninformed search techniques, the best-first and A* search algorithms provide better results to optimize the short useful paths. The improved Dijkstra‘s algorithm is implemented to find the shortest paths.

Three optimization algorithms - Dijkstra, A^*, and Genetic algorithms are considered in [10] for determining the multi-criteria paths in construction sites. The feasibility and performance limitations of all three optimization methods are investigated. The importance of such characteristics as shortest path, low risk path, high visibility path, and moreover, the path reflecting the combination of above mentioned paths is stated. The accuracy of the determined path and time complexity are given.

A deterministic method proposed in [11] is used to find the solution of global optimization problems. The typical local search, the discrete local search and the attractor based search are three phases of the proposed new deterministic method which are used to obtain a local minimum, sup-local minimum, and attractor-based global search. The computer simulation results show the efficiency of this method.

The locally informed search algorithms which are the combination of two complementary methods - local analysis and global search are considered in [12] for playing search-based sum of games in which each subgame is simple. The mentioned algorithms provide better solution quality in terms of increasing the time limits.

In [13], the locally informed gravitational search algorithm (LIGSA) intends to improve the search ability and performance of the original GSA. The important property of LIGSA is that each agent in the population can directly learn from the best agent among local neighbors. Testing of validity of LIGSA demonstrates in most cases its superiority over the original GSA in terms of convergence ability, computational complexity, and convergence accuracy.

In [14], the authors use informed search technique to solve the triangular board Peg Game. The solution of this problem is reached by the combination of pattern-based and heuristic searches. This combination is very effective to improve the speed of the search and to reduce the complexity of the domains to a reasonable and manageable size.

The authors of the paper [15] analyze the automatic knowledge revision technique which is based on informed tree search method. The revision problem is modeled to analyze the execution of the system logs and to revise knowledge based on these logs. The effectiveness and efficiency of the suggested approach are reached, and the experimental results show that the cartographic generalization is the main application domain of the informed search strategies.

The similarity of such classical search algorithms as single-agent and two-agent heuristic searches is described in [16]. It is stated that there is no essential difference between single-agent and two-agent search techniques in terms of search-space properties and enhancements. The comparative analysis of single-agent and two-agent searches shows that the application of the naïve, simple, breadth, informed, space efficient informed, and the real-time informed search algorithms demonstrate more similarities than differences in single/two-agent search algorithms. All these algorithms enhance the achievement of the high performance of the heuristic search.

The application of the general multi-heuristic search which is similar to multi-heuristic A^* makes it available to reach the significant improvements in fixing and solving the calibration problem [17].

The algorithm for severe weather avoidance using A^* informed search technique is developed to determine the total flight path cost for the aviation aircraft [18]. A^* informed search uses the heuristic function. The modified version of A^* informed search can quickly reach the optimal solution of the problem.

In [19], the dynamic programming and informed search are combined to control the inputs of discrete-time hybrid system, and a new method for the procedure of sequence synthesis of safe inputs of above hybrid system is suggested. The dynamic programming part deals with global information whereas the informed search is used to switch the mode of the system from one state to another one. A Bellman-Ford method is used to determine the appropriate switching mode. The suggested technique is also applied to a nonlinear system.

Chapter 3

UNINFORMED SEARCH STRATEGIES

In this chapter we focus on well-known uniformed search strategies. They refer to a group of search techniques known as searching without any knowledge or information to reach the objective. The uninformed search is also known as a blind search.

3.1 Breadth-first search

This is a search strategy in which the expansion starts from the root node. Afterwards the root node successors are expanded. The process is repeated until the last successor is involved. The First in First out (FIFO) data structure is preferable to implement the breadth-first search strategy.

The completeness criterion of the breadth-first search strategy is used to evaluate the algorithm performance. Assume the number of branching factor is $b < \infty$, and the shallowest goal node is located at the depth $d < \infty$. Then the number of nodes generated by the breadth-first search algorithm is

$$b + b^2 + b^3 + ... + b^d = O(b^d).$$

The completeness of the breadth-first search is clear. If the location of the shallowest goal node is at the depth of $d < \infty$, then the algorithm will find it (shallowest goal node) after it has generated all the shallower nodes which have all failed to the goal test.

The breadth-first search is tree-traversal if the expansion is done from one level to another one.

Figure 1 represents the breadth-first traversal of tree with the levels and corresponding nodes given below: Level 0$\{N_1\}$; Level 1$\{N_2, N_3\}$; Level 2$\{N_4$,N_5 and $N_6\}$, and level 3$\{N_7$,$N_8\}$.

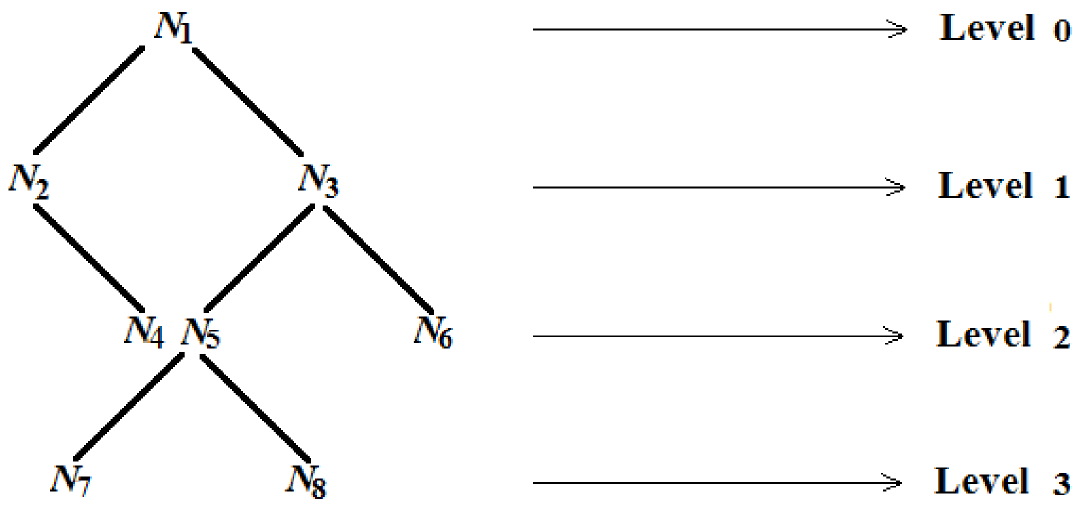


Figure 1: Breadth-first traversal of tree

The order of nodes obtained after the application of the breadth-first search algorithm to explore the tree represented in Figure 1, is $\{N_1$,N_2, N_3,N_4,N_5,N_6,N_7,$N_8\}$.

Figure 2 shows the 16-node graph used for uninformed search techniques.

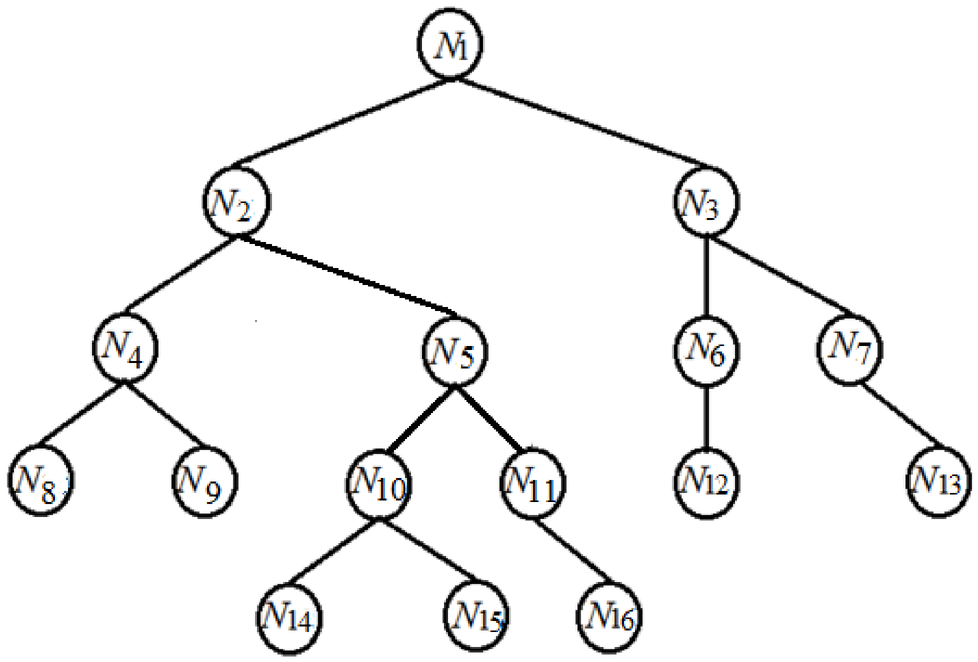


Figure 2: The 16-node graph used for uninformed search techniques

Suppose the purpose is to move from the initial node N_1 to the goal node N_9. Figure 3 illustrates the graphical representation of order of node expansion in breadth-first search from the initial node N_1 to the goal node N_9 for the graph represented in Figure 2.

The breadth-first search has the following properties:

Complete: Yes;

Optimal: Yes;

Time Complexity: $O(b^d)$, where b is branching factor and d is depth of the solution;

Space Complexity: $O(b^d)$.

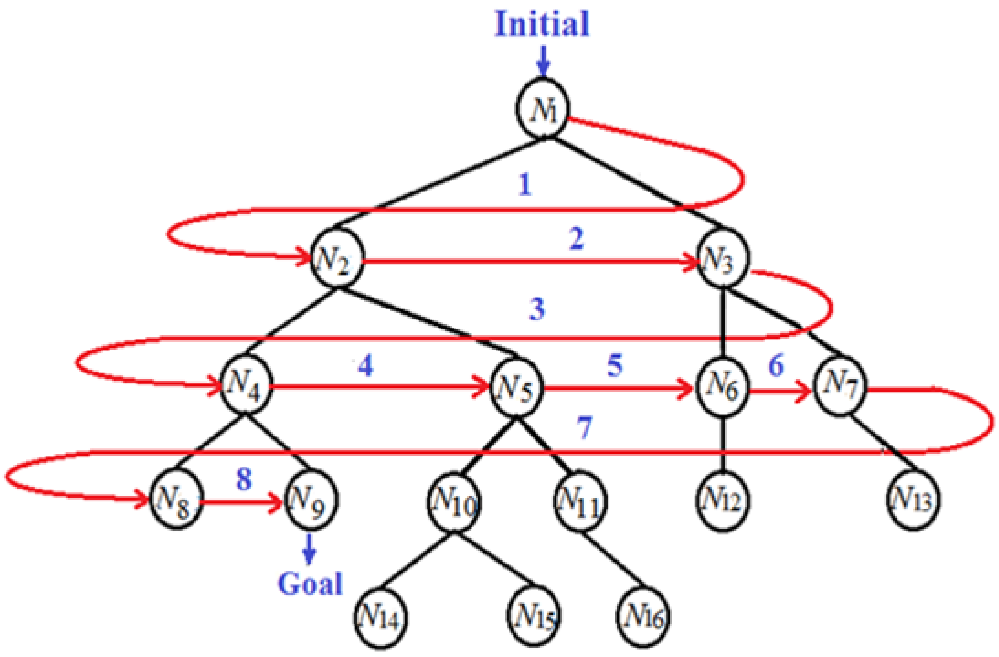


Figure 3: Graphical representation of order of node expansion in breadth-first search from the initial node N_1 to the goal node N_9

The breadth-first search always finds a solution of the problem if any exists. If the problem has more than one solution, then the algorithm finds the solutions in ascending order of their costs which means that the cheapest solution will be found at first. Since this search strategy is traversal, a situation where the algorithm will explore a branch which is indefinitely deep can never occur.

The time consumption is very high for breadth-first strategy. This factor can affect the algorithm performance if the number of branching factor increases. In fact, if the number of branching factors increases, the time complexity increases exponentially. This strategy is highly space consuming. Actually, it requires a lot of space in terms of memory to save the nodes generated at a particular depth before the next depth can proceed.

3.2 Depth-first search

The idea of this search algorithm is to go as deeper as possible in the search tree but from one neighbor node to another neighbor node before a backtracking process starts. The depth-first search algorithm is also called a Last In First Out (LIFO) algorithm.

Figure 4 represents the depth-first traversal of tree:

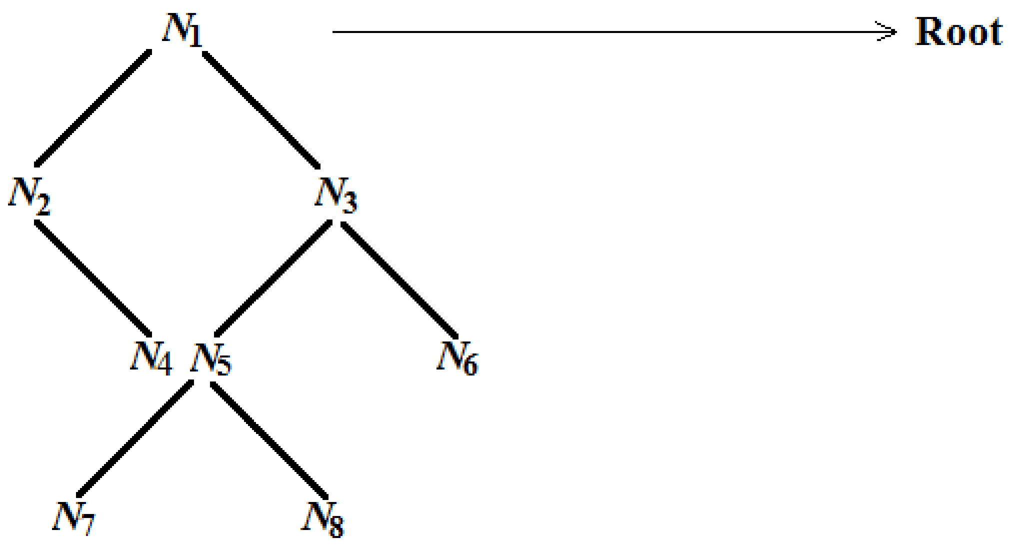


Figure 4: Depth-first traversal of tree

The depth-first search tries to go deeper from a node before it would come back to explore the sibling node.

The order of nodes obtained after the application of the depth-search algorithm to explore the tree represented in Figure 4, is $\{N_1, N_2, N_4, N_3, N_5, N_7, N_8, N_6\}$.

The depth-first search algorithm is defined and implemented depending on whether we are using a tree-search version or a graph-search version. The former version is incomplete whereas the latest is complete.

Both tree-search and graph-search versions of the depth-first search are non-optimal. Moreover, the graph-search version of the depth-first search has a time complexity which is bounded by state space size. This size can be infinity in some cases whereas a tree-search version of depth-first search has complexity of order $O(b^d)$. This complexity may be unbounded as well.

It would seem that the depth-first search has only disadvantages toward the breadth-first search. This is not the case. Actually, with branching factor b and maximum depth m, the algorithm requires a storage space of complexity $O(bm)$.

Figure 5 illustrates the graphical representation of order of node expansion in depth-first search from the initial node N_1 to the goal node N_7 for the graph represented in Figure 2.

Depth-first search has the following properties:

Complete: No;

Optimal: No;

Time Complexity: $O(b^m)$; where b is branching factor, and m is maximum depth of the tree;

Space Complexity: $O(bm)$.

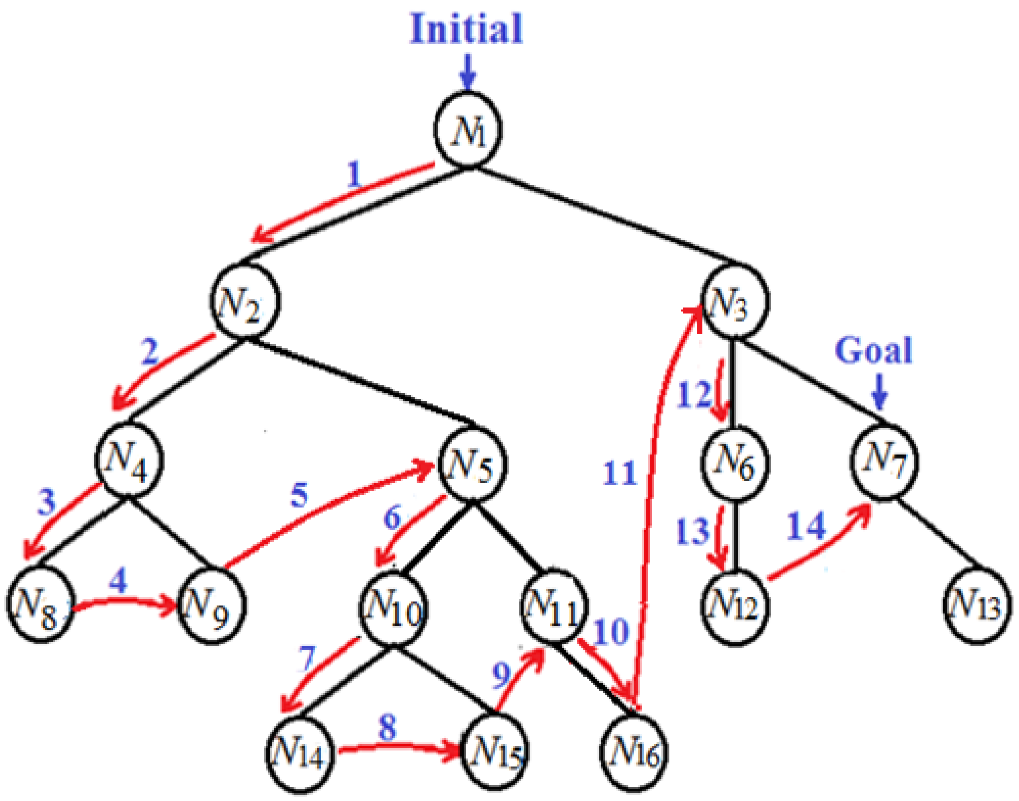


Figure 5: Graphical representation of order of node expansion in depth-first search from the initial node N_1 to the goal node N_7

Depth-first search strategy doesn't require so much memory as the breadth-first search strategy requires. Actually, the space complexity is linear. This search strategy is a time limited, but not space limited. Therefore there is always enough space to save the generated nodes although these may require a lot of time to be processed. The less shallow is the goal depth the less is space complexity and time complexity.

The branching factor can be infinitely deep in this strategy, and a situation may occur that the algorithm may follow a path if it has infinitely many nodes. The goal is not guaranteed to be always found by this strategy if even it exists. This strategy gives no guarantee to always find the best solution of the problem.

3.3 Depth-limited search

The depth-limited search is closer to the depth-first search. The only and not the least difference between these two search methods is that before performing the depth-limited search, a limit length of the depth is defined, whereas in depth-first search, the search limit is not defined; this limit can sometimes be infinity. In order to set the depth of the search, the problem definition must be well known; otherwise this might lead to an inconsistent method.

Figure 6 illustrates the graphical representation of order of node expansion in depth-limited search from the initial node N_1 to the goal node N_7 for the graph represented in Figure 2, and the depth limit is 2.

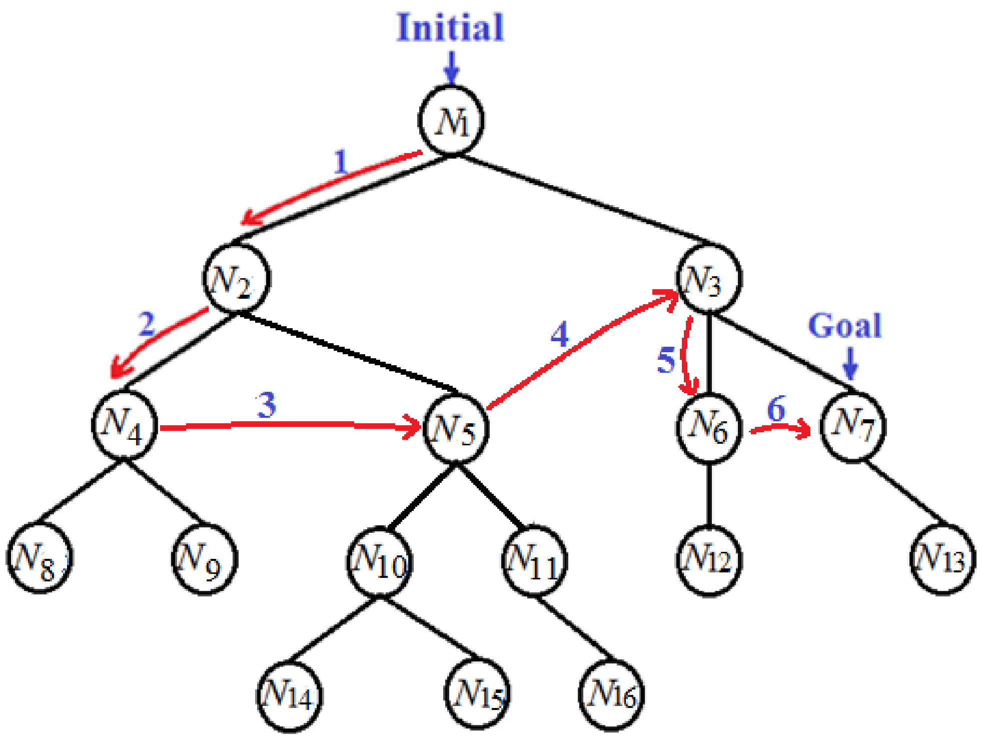


Figure 6: Graphical representation of order of node expansion in depth-limited search from the initial node N_1 to the goal node N_7

Depth-limited search has the following properties:

Complete: No;

Optimal: No;

Time Complexity: $O(b^l)$; where b is branching factor, and l is depth limit;

Space Complexity: $O(bl)$.

3.4 Iterative deepening search

This search method is also known as iterative deepening depth-first search. The method is a combination of two search techniques: the breadth-first search and the depth-first search. The limit of the depth is changed iteratively to best fit the problem definition and the goal sought.

Figure 7 illustrates the order of the node expansion in iterative deepening search from the initial node N_1 to the goal node N_9 for the graph represented in Figure 2 (four iterations are required to reach the goal).

Iterative deepening search has the following properties:

Complete: Yes;

Optimal: Yes;

Time Complexity: $O(b^d)$; where b is branching factor and d is depth of the solution;

Space Complexity: $O(bd)$.

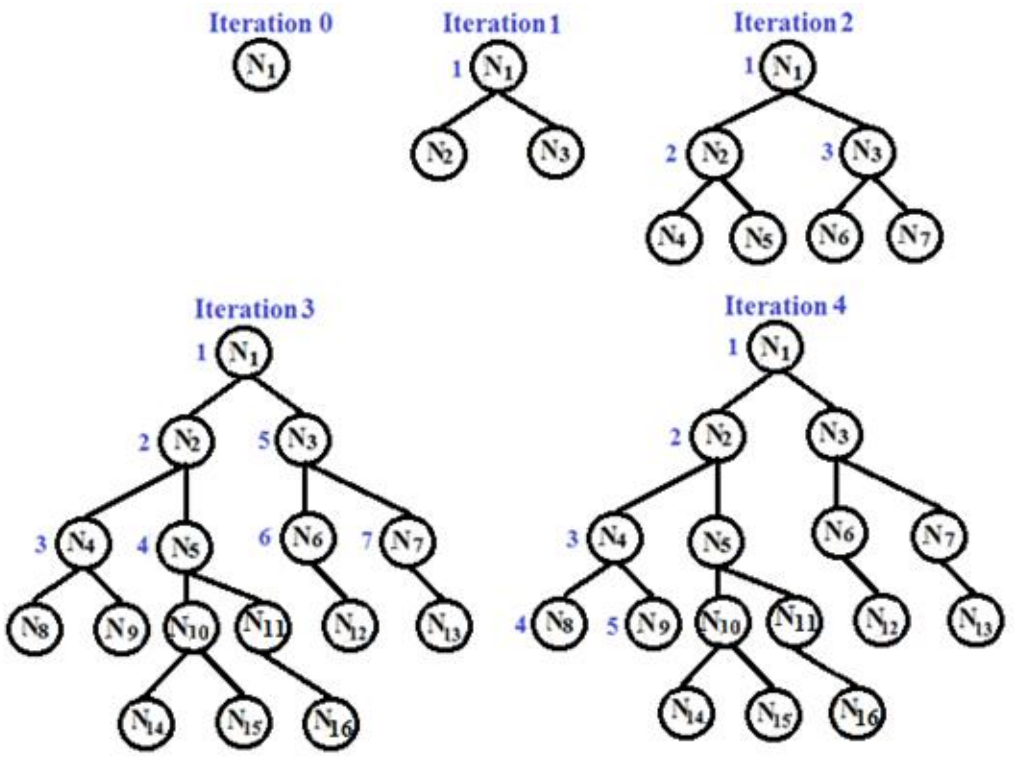


Figure 7: Graphical representation of order of node expansion in iterative deepening search from the initial node N_1 to the goal node N_9

3.5 Uniform cost search

The uniform cost search (UCS) is another uninformed search strategy. The specificity of this search technique is that it is optimal whenever the function evaluating the path length is defined. The uniform cost search implementation has the advantage that it can be done relaying to any generic or common search function.

The following example is a clear illustration of how the uniform cost search is used on weighted digraph to find the optimal path from the initial node N_1 to the goal node N_5 (Figure 8). The prospectus paths are sought beside each other until the shortest one is selected. Actually, there are three possible paths to be followed. However, only one of them appears to be the shortest at the end.

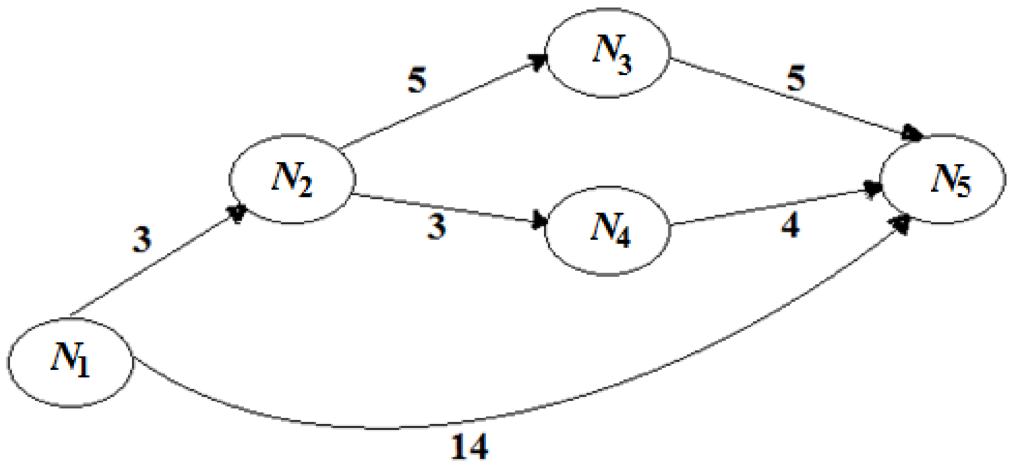


Figure 8: Weighted digraph used for the uniform cost search

The uniform cost search steps are used as follows:

Initial step: {[N_1, 0]};

Step 1: {[$N_1 \rightarrow N_2$, 3], [$N_1 \rightarrow N_5$, 14]};

Step 2: {[$N_1 \rightarrow N_2 \rightarrow N_3$, 8], [$N_1 \rightarrow N_2 \rightarrow N_4$, 6], [$N_1 \rightarrow N_5$, 14]};

Step 3: {[$N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_5$, 13], [$N_1 \rightarrow N_2 \rightarrow N_4 \rightarrow N_5$, 10], [$N_1 \rightarrow N_5$, 14]};

Step 4: Final result: The shortest path from the node N_1 to the node N_5 is defined as

$$N_1 \rightarrow N_2 \rightarrow N_4 \rightarrow N_5$$

Uniform cost search has the following properties:

Complete: Yes;

Optimal: Yes;

Time Complexity: $O(b^m)$; where b is branching factor and m is maximum depth of the tree;

Space Complexity: $O(b^d)$; where d is depth of the solution.

3.6 Bidirectional Search

In this method the search has a running technique in two directions. Indeed, two searches run simultaneously. One of them is done in a forward direction from the initial state, and another one is done backward from the goal state. The aim is that these two searches should join in the midway. The motivation of implementing such a search method is the following: if a search algorithm requires a complexity of "b^d" to reach the goal, splitting it into two sub-algorithms will require "$b^{d/2} + b^{d/2}$" of complexity which is significantly less than "b^d", especially when the values of b and d increase. The goal is used as the starting state of the backward search. Therefore the goal test is replaced by a flag with the rule to check whether two searches have expanded up to their intersection point. Although the bidirectional search seems to be good in terms of complexity, difficulties occur in implementing the backward search.

Bidirectional search has the following properties:

Complete: Yes;

Optimal: Yes;

Time Complexity: $b^{d/2}$; where b is branching factor and d is depth of the solution;

Space Complexity: $b^{d/2}$.

For example, for the branching factor b = 8 and depth d = 4, the breadth-first search needs time which is proportional to 8^4 = 4096 whereas the bidirectional search is more effective because of time which is proportional to $2*8^2$ = 128 to reach the goal.

It is also to note that bidirectional search is not practical for using if the value of d is large.

Chapter 4

INFORMED SEARCH STRATEGIES

This chapter focuses on a search family known as informed or heuristic search. The main difference between informed (heuristic) and uninformed search is that beyond a well-defined problem, the informed search method uses some problem specification whereas the uninformed search strategy performs the search in a blind mode, without any prior knowledge or specification of the problem. The problem specification in heuristic search helps efficiently find a solution of the problem.

4.1 Best-first search

Best-first search is created by combining breadth-first and depth-first search techniques. Because of this reason the best-first search gets the advantages of both breadth-first and depth-first search techniques. The best-first search does not consider all the possible paths in graph, and this strategy finds the path which has no loop.

In best-first search the evaluation function $f(n)$ is used for each node to find out the desirability of a node; so the most promising and desirable unexpanded node is expanded. So the node to be chosen should have lowest value of the heuristic function $f(n)$.

In the first stage of the best-first search the OPEN list with starting node and CLOSED list with empty set are defined. If the OPEN list has no successor then there is no solution. If there are successors of the starting node, the node with best (minimum) heuristic value N is selected. If the node N is the goal node then the path from starting node to the goal node is defined. Otherwise each direct successor node of N, if it is in neither OPEN nor CLOSED list, is added to OPEN list and the node N is set as a parent node. Afterwards the node N is deleted from the OPEN list, and added to the CLOSED list. This process will continue until goal node is found or OPEN list is empty.

The most important property of best-first search is that after the most promising successor node is selected and the move is done towards this successor, other nodes can be taken into account to be visited later if required.

It should be also noted that the heuristic function can sometimes lead to the costly path.

In Figure 9 the graphical representation of steps of best-first search example is illustrated. The starting (initial) and goal nodes of undirected graph are N_1 and N_{10}, respectively.

The best-first search has the following properties:

Complete: No;

Optimal: No;

Time Complexity: $O(b^m)$; where b is branching factor, and m is maximum depth of the tree;

Space Complexity: $O(b^m)$.

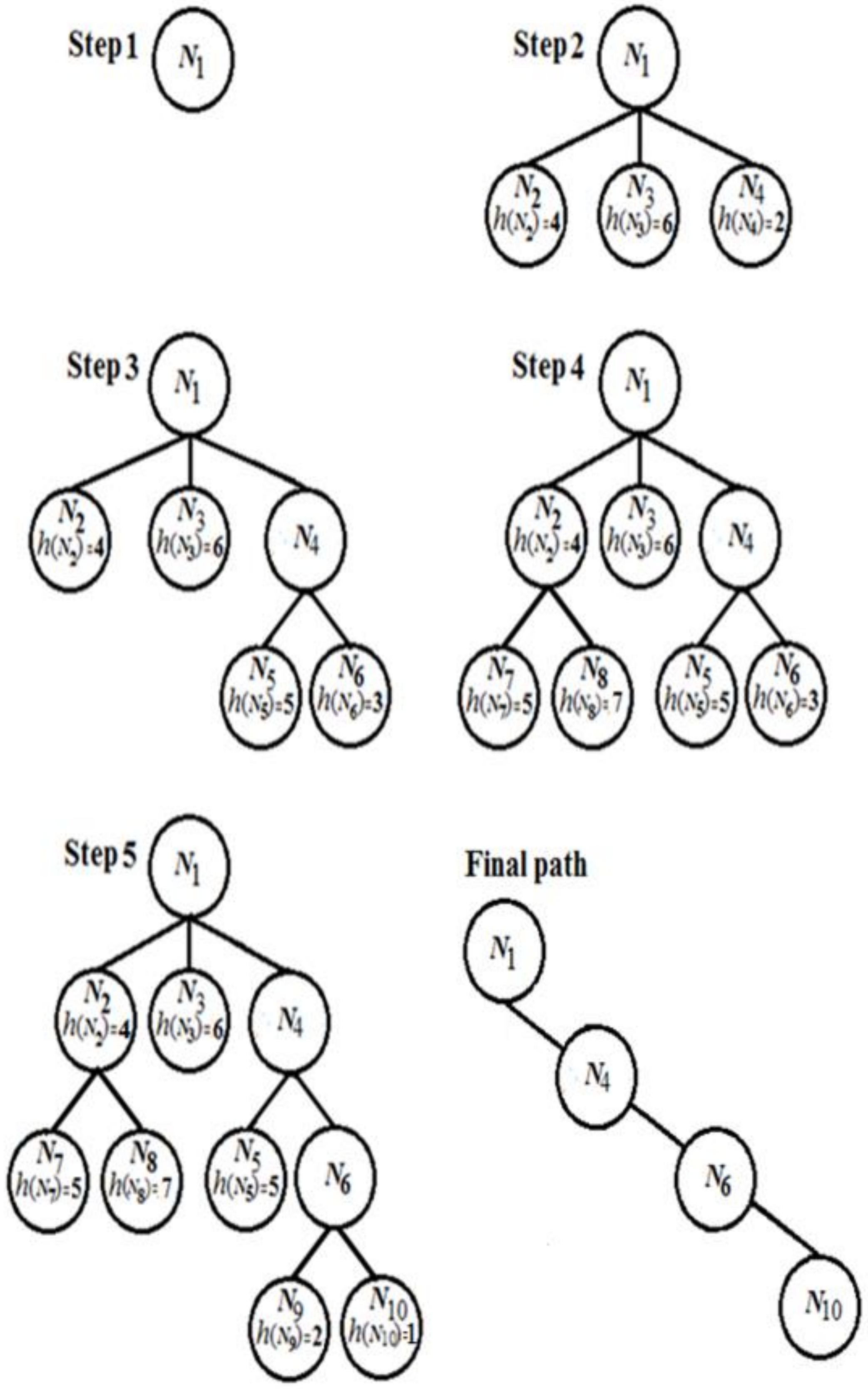


Figure 9: Graphical representation of steps of best-first search example

4.2 Greedy best-first search

This search strategy is a bit similar to breadth-first search method, and targets the goal by trying to expand the closest node to the goal. The heuristic function $f(n) = h(n)$ is used for the node evaluation. The heuristic describes the estimated cost from the particular node to the goal.

Greedy best-first search algorithm stands as a special case of the best-first strategy. In comparison with the best-first search where the evaluation function $f(n)$ is used, the greedy best-first search uses the heuristic function $h(n)$ and heuristic values to try to obtain the best (optimal) solution of the problem, but unfortunately the best solution is not guaranteed. This algorithm is recursive and therefore must be implemented with care to avoid the infinite loop.

Greedy-best first search has the following properties:

Complete: No;

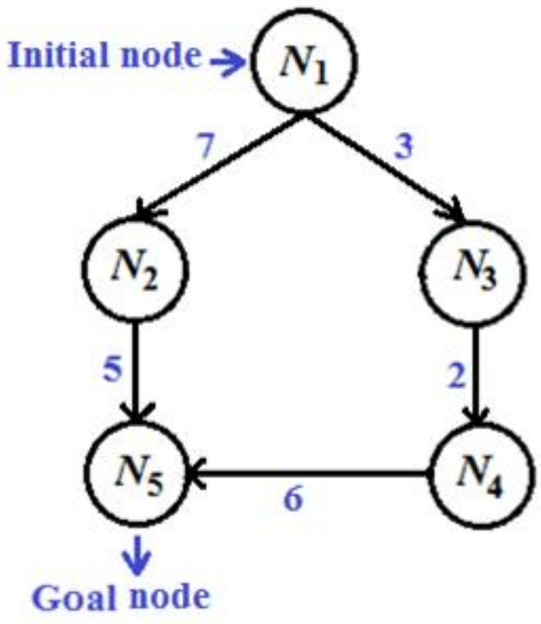
Optimal: No;

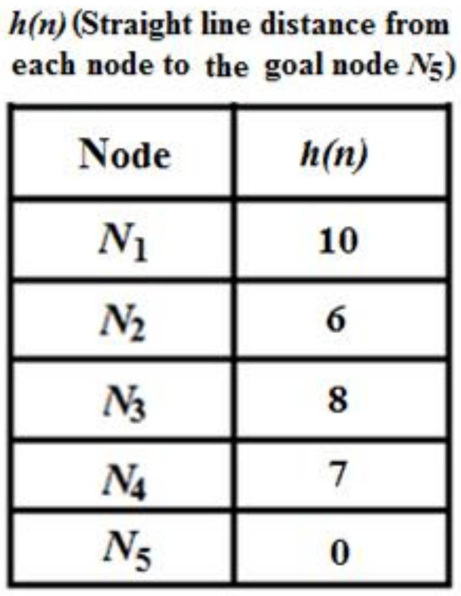
Time Complexity: $O(b^m)$; where b is branching factor, and m is maximum depth of the tree;

Space Complexity: $O(b^m)$.

Why is the greedy-best first search not optimal? To understand it, let's consider the following example. The digraph has five nodes in which the initial node is N_1 and the goal node is N_5. The aim is to apply the greedy best-first search to find the shortest (best) path from the initial node N_1 to the goal node N_5, and the heuristic function

$h(n)$ describes the straight line distance from each node to the goal node N_5 (Figure 10).




$h(n)$ (Straight line distance from each node to the goal node N_5)

Node	$h(n)$
N_1	10
N_2	6
N_3	8
N_4	7
N_5	0

Figure 10: Digraph for the greedy best-first search with straight line distance from each node to the goal node

The heuristic function here gives the estimated cost from each node to the goal node as follows: N_1=10, N_2=6, N_3=8, N_4 =7, N_5=0 (Note: N_1=10 means that the estimated straight line cost from N_1 to N_5 is 10, similarly, N_5=0 means that the estimated straight line cost from N_5 to N_5 is 0 which is quite logical).

The edge weight shows the distance between two adjacent nodes in a graph; so these nodes are endpoints of an edge.

The figure 11 shows the steps of nodes' expansion in digraph for the greedy best-first search (represented in Figure 10) until the goal is reached.

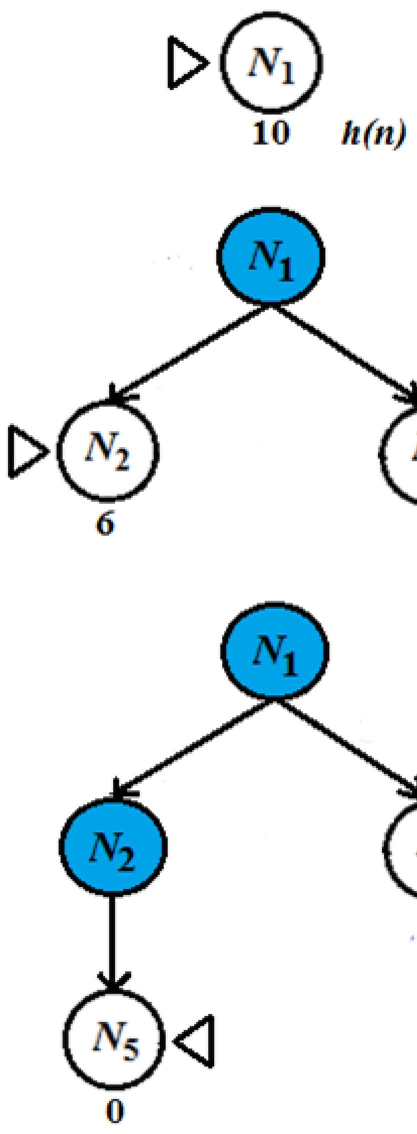


Figure 11: Steps of nodes' expansion in digraph for the greedy best-first search

According to the evaluation function $f(n) = h(n)$, the best path is found as $N_1 \to N_2 \to N_5$ with total heuristic cost 16, but this path is not admissible (optimal), because the solution cost which is a sum of edge costs of the path $N_1 \to N_2 \to N_5$ = 7+5=12 is longer than a sum of edge costs of another available path $N_1 \to N_3 \to N_4 \to N_5$ = 3+2+6=11.

4.3 A* search strategy

A* is another informed search strategy. A* search is actually the most popular best-first search strategy adopted by scientists and researchers.

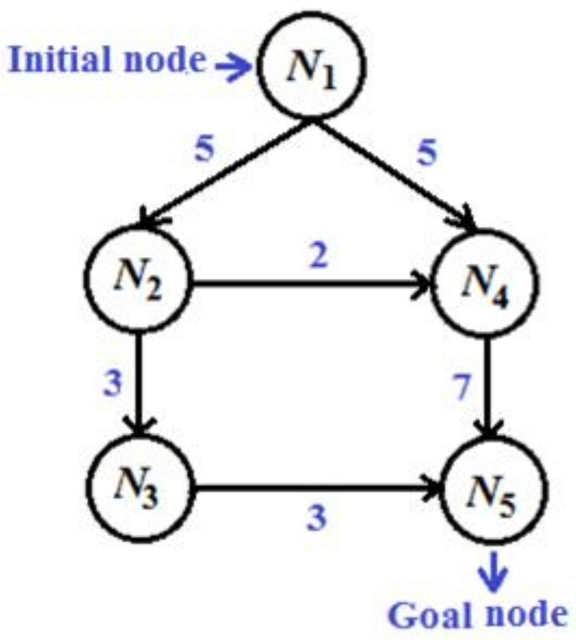
A* search is a best-first search algorithm, but not greedy best-first algorithm. The evaluation function $f(n)$ in A* search strategy is represented as $f(n) = g(n) + h(n)$, where n is the last node of the path, $g(n)$ is the cost of the path from the starting node to the node n, and $h(n)$ is heuristic used to estimate the cost from the node n to the goal node.

Combining $g(n)$ and $h(n)$ leads to the estimation of $f(n)$ which is actually the optimal cost function used to reach a solution.

The uniform cost search and best-first search are the special cases of A* search. If heuristic $h(n)$ is equal to 0 in $f(n) = g(n) + h(n)$, then we get $f(n) = g(n)$ which is used in uniform cost search. If $g(n)$ is equal to 0 in $f(n) = g(n) + h(n)$, then we get $f(n) = h(n)$ which is used in best-first search.

The big advantage of A* search strategy that doesn't exist in a greedy best-first search strategy is that A* search is complete. Moreover it is optimal. Therefore one can say that the A* search technique satisfies the goal of informed search strategy.

The figure 12 represents the digraph for A* search with straight line distance from each node to the goal node.



h(n) **(Straight line distance from each node to the goal node N_5)**

Node	*h(n)*
N_1	10
N_2	9
N_3	4
N_4	8
N_5	0

Figure 12: Digraph for A* search with straight line distance from each node to the goal node

Let's find the shortest path from the initial (starting) node N_1 to the goal node N_5.

$N_{1\,,}\, g + h = 0 + 10 = 10$

$N_1 \rightarrow N_2\,, g + h = (0 + 5) + 9 = 5 + 9 = 14$

$N_1 \rightarrow N_4\,, g + h = (0 + 5) + 8 = 5 + 8 = 13$

$N_1 \rightarrow N_2 \rightarrow N_3\,, g + h = (5 + 3) + 4 = 8 + 4 = 12$

$N_1 \rightarrow N_2 \rightarrow N_4\,, g + h = (5 + 2) + 8 = 7 + 8 = 15$

$N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_5\,, g + h = (5 + 3 + 3) + 0 = \underline{11}$

$N_1 \rightarrow N_2 \rightarrow N_4 \rightarrow N_5\,, g + h = (5 + 2 + 7) + 0 = \underline{14}$

$N_1 \rightarrow N_4 \rightarrow N_5, g + h = (5 + 7) + 0 = \underline{12}$

Since $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_5$ is the shortest path among available paths from the initial node N_1 to the goal node N_5, this path is optimal.

The A* search strategy covers some lacks that are found in the best-first search and the greedy best-first search. The differences between these tree strategies are mostly observed in their properties.

A* search has the following properties:

Complete: Yes;

Optimal: Yes;

Time Complexity: $O(b^d)$; where b is branching factor, and d is depth of the solution;

Space Complexity: $O(b^d)$.

A* search technique is also used to solve the 8-puzzle game. This game was invented by Loyd in 1870. The 8-puzzle game is a well-known game used for the evaluation and showcase of the power of Artificial Intelligence in general and search methods in particular.

The game principle is described as follows: there are 8 sliding tiles which are numbered by digits from 1 to 8 to be located in an array with three rows and three columns; the number 0 represents the empty cell (blank space), and any numbered tile can slide into an empty sell. It means that the digit 0 can simply be replaced by the void. The initial state is made by a random positioning of the digits and the goal state is usually defined in a way that the numbers from 1 to 8 can be read in ascending or descending order, following a unique direction. The number of different possible configurations the 8-puzzle game is equal to 9!=362880. The number of accessible state configurations in 8-puzzle game is just half of 9!; in other words 9!/2 is the number of solvable (reachable) configurations in 8-puzzle sliding game.

The following assumptions, notations and operators are used: Opr(Row, Column, Direction) with Row & Column = {1,2,3}; Direction = {U,D,L,R}, where U, D, L, and R represent Up, Down, Left, and Right, respectively. Explicitly Opr stands for operator. The row and column index values range from 1 to 3 (from the set {1,2,3}). The following are the possible directions: u = upper; d = down; l = left; r = right. As summary, the operator 'Opr(Row, Column, Direction)' has three parameters which are the row index, the column index and the direction toward which the located element is moved.

The optimal solution of the 8-puzzle game (with a minimum number of moves) for the given initial and final states is described in Figure 13.

8 Puzzle Solution Finder program was used to find the optimal solution of the problem for the given initial and goal states.

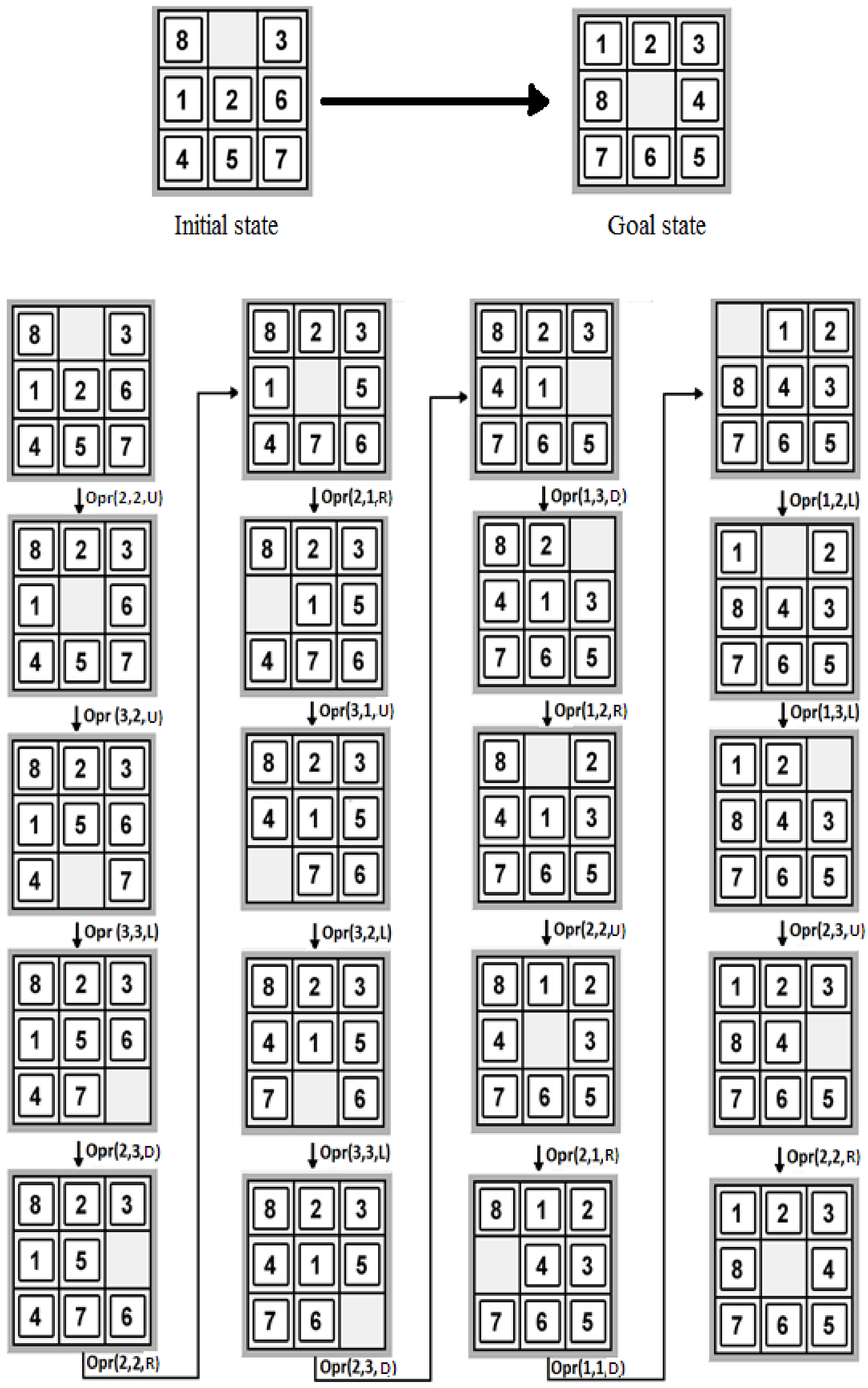


Figure 13: Optimal solution of the 8-puzzle game for the given initial and goal states

4.4 Hill climbing search

Hill climbing is heuristic and local search technique, and in the initial stage we select the starting node, and then we move to the heuristically closest position which leads to the goal. This process continues until no better position remains to move.

The graphical representation of order of node expansion in hill climbing search is represented in Figure 14.

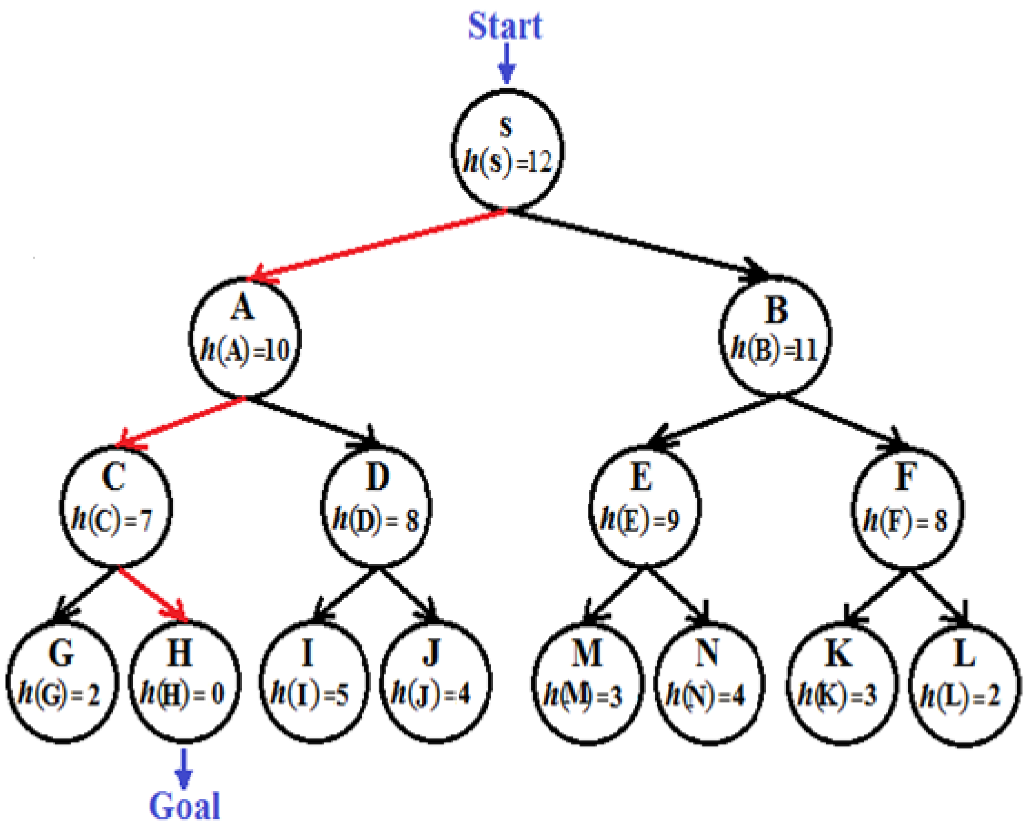


Figure 14: Graphical representation of order of node expansion in hill climbing search

From the Figure 14 it is seen that the search path after using hill climbing strategy is $S \rightarrow A \rightarrow C \rightarrow H$.

Hill climbing search has the following properties:

Complete: No;

Optimal: No;

Time complexity: $O(\infty)$;

Space Complexity: $O(b)$, where b is branching factor.

By using the hill climbing search technique it is possible to expand exactly one successor node at a time regardless the number of successors the node has, and there is no effort to backtrack. In other words, hill climbing technique never tries to consider all rejected nodes. It seems that Hill climbing search puts less efforts to reach the goal state, but there is no guarantee that the quality of the solution of the problem will be complete.

4.5 Dijkstra's Algorithm

There exists a well-known informed search algorithm called Djikstra's algorithm. This algorithm is used in problems where the goal is to find the minimum cost path from the source (initial) node to all other nodes of the graph. We are discussing the search methods, and we can't end without mentioning the properties of this powerful algorithm.

This algorithm was invented and introduced by a Dutch scientist Edsger Dijkstra in 1959. His first idea which is the essence of the algorithm was to find the shortest path in graph with nonnegative values on the edges. The limitation of the Dijkstra's algorithm is that this algorithm doesn't handle with the negative weight edges.

Moreover, it is useful to consider the Dijkstra's algorithm because it appears as an extension of A* search algorithm. So the Dijkstra's algorithm is actually an extended form of A* search algorithm.

The pseudo-code of Dijkstra's algorithm is given below.

```
Dijkstra Algorithm
input:  Graph and weight /*A set of cities to visit*/
output: Minimum distance for a path
Distance [t]🡐0        /* initial distance is set to 0 */
for all  u from U-{t} do
Distance[u]🡐 infinity          /*All other distances are set to infinity.*/
end for
Sity🡐 ∅.                      /*All the cities to be visited */
Queue🡐U;
while Queue≠∅ do
d🡐minimum_distance(Queue, Distance)
Site🡐Site ◡{d}
for all t from neighborhoods[d] do
if Distance [t] > Distance [d]+weight (t,d) then
dist(t)🡐dist(d) + weight (t,d)
end if
end for
end while
return Distance
end.
```

The efficiency of the Dijkstra's algorithm can be evaluated in terms of its complexity. Due to the fact that it is a graph based algorithm, the number of nodes and edges is important for the evaluation of the complexity.

Let's consider a graph with *n* vertices and *k* edges. The time complexity of running of Dijkstra's algorithm on such graph is $O(n^2)$. More usually when we pose the assumption of a graph made of a binary and well-balanced tree, the Fibonacci heap is used to evaluate the complexity. In this case the mentioned complexity is of order $O(k + n\log(n))$.

It is to note that one of the inconveniences of applying the Dijkstra's algorithm is that it is not backtracking, and therefore it may fall into an infinite loop if it finds a forked node.

In Figure 15 the initial form of weighted digraph used for Dijkstra's algorithm is represented.

We consider all the steps of Dijkstra algorithm for the graph represented in Figure 15 to compute the shortest paths from the initial node N_1 to all other nodes N_2, N_3, N_4, N_5 and N_6.

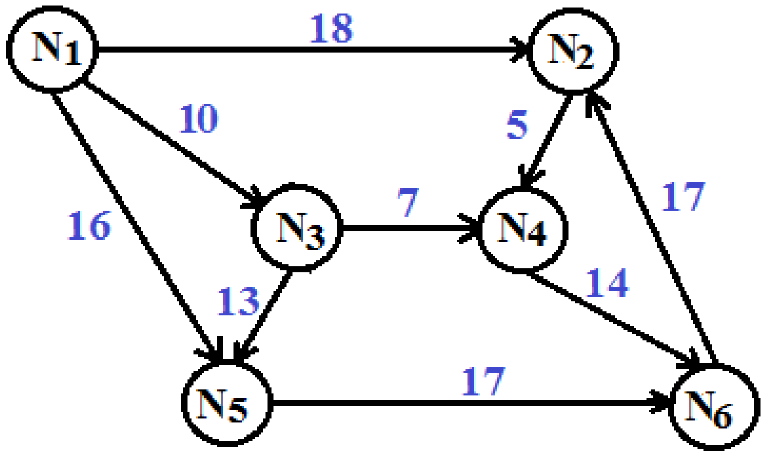


Figure 15: Initial form of weighted digraph used for Dijkstra's algorithm

Initial Step 0:

Distance $[N_1]$←0 /*Distance from the initial state to itself*/

Distance $[N_2]$←∞; Distance $[N_3]$←∞; Distance $[N_4]$←∞; Distance $[N_5]$←∞; Distance $[N_6]$←∞; /*The distances between the nodes which are yet to be processed are set to infinity*/.

The shortest paths after the initial step 0 of Dijkstra's algorithm are given in Table 1:

Table 1: Shortest paths after the initial step 0 of Dijkstra's algorithm

Node	Shortest paths from the node N_1	Previous node
N_1	0	-
N_2	∞	-
N_3	∞	-
N_4	∞	-
N_5	∞	-
N_6	∞	-

Step 1:

The nodes N_2, N_3 and N_5 can be directly reached from the node N_1. The update distances are given below.

Distance $[N_2] \leftarrow 18$; Distance $[N_3] \leftarrow 10$; Distance $[N_5] \leftarrow 16$.

The update procedure is based on the following paradigm: if the calculated distance from the initial node to a given node is smaller than the known distance in the previous step, then the distance is updated.

Figure 16 represents the weighted digraph after the step 1 of Dijkstra's algorithm, and the shortest paths after the step 1 of Dijkstra's algorithm are given in Table 2.

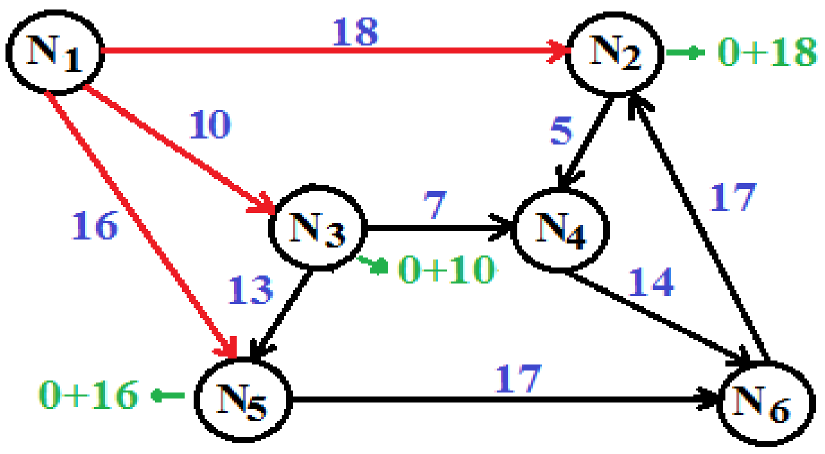


Figure 16: Weighted digraph after the step 1 of Dijkstra's algorithm

Table 2: Shortest paths after the step 1 of Dijkstra's algorithm

Node	Shortest paths from the node N_1	Previous node
N_1	0	-
N_2	18	N_1
N_3	10	N_1
N_4	∞	-
N_5	16	N_1
N_6	∞	-

Step 2:

Choose the node with the shortest distance among the known distances and explore the neighborhood nodes.

From N_3 one can directly visit N_4 and N_5. The path N_3N_4 has a minimum distance 10. Let us find the distance from N_1 to N_4 through N_3. The known distance from N_1 to N_4 in Table 2 was infinity, we then update this distance to 0+10+7=17.

The distance from N_1 to N_5 was 16 in Table 2 which is less than the distance from N_1 to N_5 through N_3 which is 10+13=23, so no need to update the distance between N_1 and N_5.

Figure 17 represents the weighted digraph after the step 2 of Dijkstra's algorithm, and the shortest paths after the step 2 of Dijkstra's algorithm are given in Table 3.

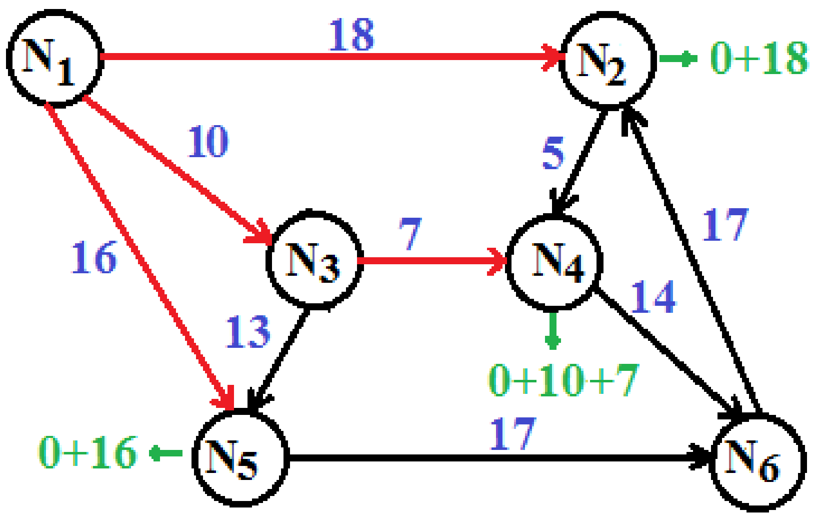


Figure 17: Weighted digraph after the step 2 of Dijkstra's algorithm

Table 3: Shortest paths after the step 2 of Dijkstra's algorithm

Node	Shortest paths from the node N_1	Previous node
N_1	0	-
N_2	18	N_1
N_3	10	N_1
N_4	17	N_3
N_5	16	N_1
N_6	∞	-

Step 3:

We again choose the node with the shortest distance among the known distances and explore the possible neighborhood nodes.

The node N_3 cannot be chosen because this node was already visited. We therefore choose the node N_5. The node that can be visited from N_5 is N_6. The known distance from N_1 to N_6 in Table 3 was infinity; we then update it to the distance 0+16+17=33.

Figure 18 represents the weighted digraph after the step 3 of Dijkstra's algorithm, and the shortest paths after the step 3 of Dijkstra's algorithm are given in Table 4.

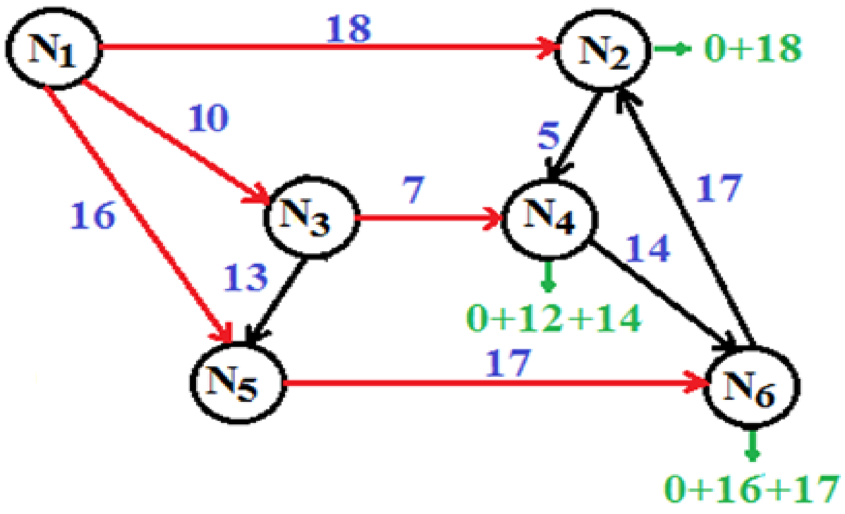


Figure 18: Weighted digraph after the step 3 of Dijkstra's algorithm

Table 4: Shortest paths after the step 3 of Dijkstra's algorithm

Node	Shortest paths from the node N_1	Previous node
N_1	0	-
N_2	18	N_1
N_3	10	N_1
N_4	17	N_3
N_5	16	N_1
N_6	33	N_5

Step 4:

The node with the shortest distance among the known distances is chosen and the possible visited neighbor nodes are explored. The nodes N_3 and N_5 cannot be chosen because these nodes were already visited. We therefore choose the node N_4. The node that can be directly visited from the node N_4 is N_6. The known distance from the node N_1 to the node N_6 in Table 3 was 33. The current distance from the node N_1 to the node N_6 through N_4 is 0+10+7+14=31. Since 31<33, we update the previous distance from the node N_1 to the node N_6 to make it equal to the current distance 31.

So all the shortest paths from the nodes N_1 to all other nodes N_2, N_3, N_4, N_5 and N_6 are defined, and there will be no updates anymore.

Figure 19 represents the weighted digraph after the step 4 of Dijkstra's algorithm, and the shortest paths after completing all the steps of Dijkstra's algorithm are given in Table 5.

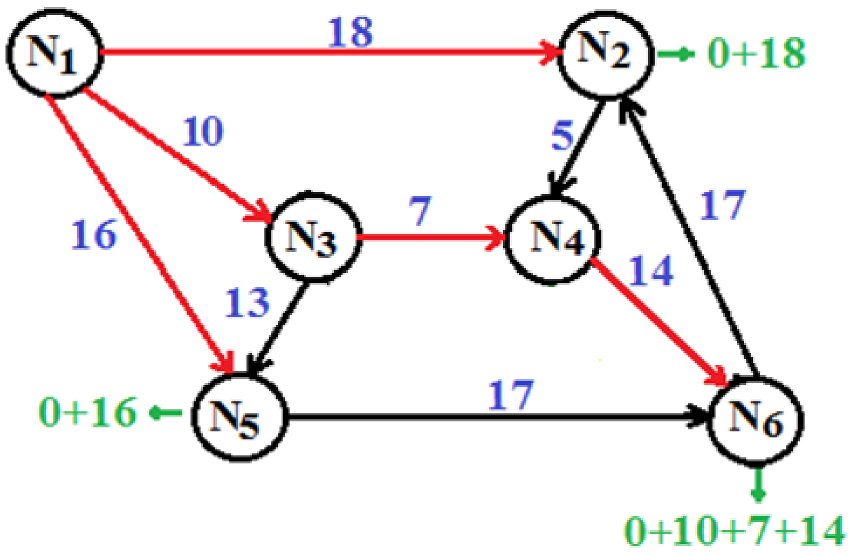


Figure 19: Weighted digraph after the step 4 of Dijkstra's algorithm

Table 5: Shortest paths after completing all the steps of Dijkstra's algorithm

Node	Shortest paths from the node N_1	Previous node
N_1	0	-
N_2	18	N_1
N_3	10	N_1
N_4	17	N_3
N_5	16	N_1
N_6	31	N_4

Dijkstra's algorithm has many real life applications. In the communication area, for instance, networks are made of routers and connected links among them. Considering routers and links as nodes and edges respectively, it appears that one may find the shortest path from the initial node to all other nodes. In communication such path is called a min-delay path.

Dijkstra's algorithm has proven its successful application in the fields of traffic planning, path optimization for a mobile robot; transports and flight scheduling management etc.

Chapter 5

CONCLUSION

Artificial Intelligence aims designing and implementing strategies that can help solve problems efficiently.

Search techniques are the main tools through which Artificial Intelligence acts. The time and the space complexities are the main properties used to evaluate the searching algorithms. The search methods can be uni-objective or multi-objective.

In a uni-objective search, there is only one variable and the goal has less constraints. The uni-objective was the first search method developed.

The multi-objective search is based on many parameters and many constraints. This type of search can be considered as an extension of the uni-objective search. Indeed, the multi-objective search problems' solutions are usually derived from a similar technique used in uni-objective problems.

The search strategies are classified as uninformed search and informed search strategies. The uninformed search strategy is considered as a blind search, because no prior knowledge or assumption which can help easily reach the goal is available. The strategy is given in the following form: all the elements from the possible solution set are compared with the prospective solution (the goal). The search ends when the goal is found or if all the elements have been checked. This search strategy is a bit primitive and might properly work in using the uni-objective search. The strategy is time and space consuming. The time complexity in the worst case is very high. An improvement of this strategy consists of sorting the elements of the search space. However many problems involve multiples goals. This is the geneses of the multi-objective also called multi-agent search strategy.

The informed search strategy is mostly used for problems with multiple parameters. The solution might be difficult to find if an uninformed strategy is used. A prior knowledge to the solution is used to define a heuristic function. The heuristic function acts like a guide. It considerably reduces the time and space complexity of the strategy.

The informed search strategy tries to act like a human brain; this search strategy tries to use all the knowledge and properties of a given problem to reduce the time and space consumption to reach the goal. However, many advances are still required for implementing an intelligence system that might act exactly like a human brain.

REFERENCES

[1] Kuruvilla, M., & Tabassum, M. (2014). Experimental Comparison of Uninformed and Heuristic AI Algorithms for N Puzzle and 8 Queen Puzzle Solution. *International Journal of Digital Information and Wireless Communications* (IJDIWC), 4(1), 143-154.

[2] Zhou, R., & Hansen, E.A. (2008). Combining Breadth-First and Depth-First Strategies in Searching for Treewidth. *First International Symposium on Search Techniques in Artificial Intelligence and Robotics*, Chicago, 640-645.

[3] Akanmu, T. A., Olabiyisi, S. O., Omidiora, E. O., Oyeleye, C. A., Mabayoje, M.A., & Babatunde, A. O. (2010). Comparative Study of Complexities of Breadth-First Search and Depth-First Search Algorithms using Software Complexity Measures. *Proceedings of the World Congress on Engineering (WCE 2010),* Vol I, London, U.K..

[4] Ashwani, C., & Manu, S. (2014). Searching and Optimization Techniques in Artificial Intelligence: A Comparative Study & Complexity Analysis. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET),* Volume 3 Issue 3, 866-871.

[5] Stern, R., Kiesel, S., Puzis, R., Felner, A., & Ruml, W. (2015). Max Is More than Min: Solving Maximization Problems with Heuristic Search. *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015),* 4324-4330.

[6] Sethuraman, J., Mahanti, A., & Saha, D. (2006). An uninformed best first search based wavelength assignment algorithm to minimize the number of SONET ADMs in WDM rings. *Telecommunications Planning: Innovations in Pricing, Network Design and Management, Chapter 12,* Springer, 225-239.

[7] Likhachev, M., & Koenig, S. (2002). Speeding up the Parti-Game Algorithm. *Proceedings of the 15th International Conference on Neural Information Systems (NIPS'02),* 1595-1602.

[8] Weise, T., Bleul, S., Comes, D., & Geihs, K. (2008). Different Approaches to Semantic Web Service Composition. *The Third International Conference on Internet and Web Applications and Service,* 90-96.

[9] Chiong, R., Sutanto, J.H., & Jap, J.H. (2008). A Comparative Study on Informed and Uninformed Search for Intelligent Travel Planning in Borneo Island. *International Symposium on Information Technology,* ITSim 2008, 1-5.

[10] Soltani, A.R., Tawfik, H., Goulermas, J.Y., & Fernando, T. (2002). Path planning in construction sites: performance evaluation of the Dijkstra, A^*, and GA search algorithms. *Advanced Engineering Informatics 16 (2002),* 291–303.

[11] Lee, J. (2007). A novel three-phase trajectory informed search methodology for global optimization. *Journal of Global Optimization,* Volume 38, Issue 1, 61 -77.

[12] Muller, M., & Zhichao, L. (2006). Locally Informed Global Search for Sums of Combinatorial Games. *Lecture Notes in Computer Science,* Volume 3846, Chapter: Computers and Games, 273-284.

[13] Sun, G., Zhang, A., Wang, Z., Yao, Y., Ma, J., & Couples J.D. (2016). Locally informed gravitational search algorithm. *Knowledge-Based Systems,* 104, 134-144.

[14] Manolescu, D.- A., & Morgan, T.E. (1996). Informed Search Using Equivalent-Class Templates. *Proceedings of the 5th ISCA Conference*, USA, 197-201.

[15] Taillandier, P., Duchene, C., & Drogoul A. (2008). Knowledge revision in systems based on an informed tree search strategy: application to cartographic generalisation. *Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology* (CSTST'08), 273-278.

[16] Schaeffer, J., Plaat, A., Junghanns, A. (2001). Unifying single-agent and two-player search. *Information Sciences,* Volume 135, Issues 3-4, 151-175.

[17] Narayanan, V., Aine, S., & Likhachev, M. (2015). Improved Multi-Heuristic A* for Searching with Uncalibrated Heuristics. *Proceedings of the Eighth International Symposium on Combinatorial Search (SoCS-2015),* 78-86.

[18] Bokadia, S., & Valasek, J. (2001). Severe Weather Avoidance Using Informed Heuristic Search. *AIAA Guidance, Navigation, and Control Conference & Exhibit,* 1-9.

[19] Schmidt, K., Kapinski, J., & Krogh, B. H. (2004). Control Input Synthesis for Hybrid Systems Using Informed Search. *IEEE International Symposium on Computer Aided Control Systems Design*, 41-46.