

Artificial Intelligence Search Algorithms In Travel Planning

A study performed on search algorithms and its use in travel planning with the Cooperation of YEStravel, travel planner. This Master thesis is 20 weeks work done from January 06 to June 06.

By
Kamran Zaheer
kzr05001@student.mdh.se
Department of Computer sciences and Electronics
Mälardalen University Västerås Sweden.

Examiner : Peter Funk
Supervisors : 1st Mobyen Ahmed, 2nd Peter Funk
Industrial Supervisor :Sandy Li Shenzhi

Abstract

Problem solving in Artificial Intelligence (AI) may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. In AI problem solving by search algorithms is quite common technique. In the coming age of AI it will have big impact on the technologies of the robotics and path finding. It is also widely used in the travel planning. This thesis deals with the different kinds of search algorithms of Artificial intelligence used in travel planning. It describes their structures and their use in travel planning. Heuristic search algorithms and their uses in AI are focused. Travel planning plays major role in human's activities but is complex part. AI has also decreased that complexity. HTN planning technique which is better step for the advancement of travel planning. Heracles which is interactive data driven constraint based travel planners has played a significant role in the field of travel planning. Both Heracles are discussed with their properties and functionality. The information agents and their working used in this travel planner are also discussed. Future work being done in this field and the benefits of that constraint based travel planner are also highlighted in the thesis.

Table of Contents

1 Introduction	6
1.1 Background	7
1.1.1 Problem Solving and Search	7
2 Uniformed Search Strategies.....	10
2.1 Brute force or Blind search methods	10
2.2 Breadth First Search	10
2.3 Depth First Search	11
2.4 DFS Iterative Deepening	12
2.5 Greedy Search	12
2.6 Bidirectional Search	13
2.7 Iterative deepening	14
3 Heuristic Search	14
3.1 A * Search	16
3.2 Hill Climbing Search.....	16
3.2.1 Stochastic Hill climbing	17
3.2.2 First choice Hill climbing.....	18
3.3.3 Random Restart Hill Climbing.....	18
3.3 Simulated Annealing	18
3.4 Generate and Test search.....	19
3.5 Back Tracking	20
3.6 Beam Search.....	21
3.7 Best First Search.....	22
3.8 Branch and Bound.....	22
3.9 Constraint Search	23
3.9.1 Constraint Satisfaction search	23
3.10 Means End Analysis.....	24
4 Asynchronous Search Algorithm.....	24
4.1 Asynchronous Dynamic Programming	25
4.2 Learning Real time A*	25
4.3 Real time A*	25
4.4 Real time multi agent	25
5 Other Search Techniques	26
5.1 Tabu Search.....	26
5.2 Island Search	28
5.3 Semantic Search Algorithm	28
6 Comparison of Different Search Algorithms.....	28
6.1 Measuring Problem Solving Performance	28
7 Implementation of AI Search Algorithms.....	30
7.1 HTN Planning	30

7.2.1 Monitoring Travel Status	35
7.2.2 Constraint Network Representation	38
7.3 Information Agents	39
7.3.1 Defining Information Agent.....	40
7.3.2 Accessing Web sources	40
7.3.3 Monitoring Web sources	40
7.4 Programming of Search Algorithms	41
8 Evaluation of Constraint Network by Heracles	43
8.1 Advantages of Heracles.....	43
8.2 Comparison with other Travel Planners.....	44
8.3 Challenges for Intelligent system in tourism.....	44
8.4 Limitations of Heracles	45
8.5 Opportunities for research.....	45
8.6 Future of AI planning.....	46
9 Discussion.....	46
10 Summary and Conclusion	47
11 References	48

Acknowledgement

I am extremely thankful to my programme supervisor professor Peter Funk and his assistant Mobyen Ahmad who guided me completely about my programme and thesis completion. They gave me full support when I needed. I am also thankful to founder and CEO of YESTravel (YESTravel.com) Sandy Li Shenzhi who gave us ideas to do research on travel planning. I express my gratitude to also my family and friends who encouraged me every time for the completion of my studies.

List f Figures

Figure 1 The Road map of Romania [14]	9
Figure 2 Breadth First Search [22]	11
Figure 3 Depth First Search [11]	11
Figure 4 Greedy Search [4]	13
Figure 5 Bidirectional Search [2]	14
Figure 6 Hill climbing search [9]	18
Figure 7 Working of Generate and test search algorithm	20
Figure 8 Expansion of nodes in beam search [21]	21
Figure 9 Means end analysis states	24
Figure 10 include dependency graph for tabu search	27
Figure 11 methods for travelling from one location to another and how they might be the task of travelling from the Maryland to MIT	31
Figure 12 status information monitoring agent [16]	41

List of Tables

Table 1 Comparison of different search algorithms	29
Table 2 A comparison of travel planning approaches	44

1 Introduction

An algorithm is a set of instructions which describes how to solve the problem. As far as search algorithm is concerned it is a global problem solving mechanism in artificial intelligence. Search algorithms are used for a multitude of AI tasks one of them is path finding. AI area of search is very much connected to problem solving. AI has investigated search methods that allow one to solve path planning problems in large domains. Having formulated problems we need to solve them and it is done by searching through the state space during this process search tree is generated by taking initial state and applying the successive function to it. Most of the researches on search methods have studied how to solve one-shot path-planning problems. Search is mostly a repetitive process therefore many AI systems replan from scratch and it solves the path planning problem independently. The problem becomes more severe when changes occur during planning. Fortunately the changes to the path planning problems are usually small. The problems that have been addressed by AI search algorithms are fall into three general classes: single-agent path finding problems, two player games and constraint-satisfaction problem [1]. Heuristic search methods promise to find the shortest path for path planning problems that are faster than uniformed search methods. On the other hand incremental search methods promise to find shortest path for series of similar path planning problem faster that is possible by solving each path planning problem from scratch.

Travel planning is now becoming a vast field and for economy of any country it plays vital role. Artificial intelligence is playing major role in this field. Many travel planners using heuristic and uniformed algorithms are giving better results than the ordinary travel planners. Both search and graph AI algorithms are used in the travel planning which make any travel planner user friendly and time saving.

In this thesis main subject has been the search algorithms used in the travel planning search for better routes, destination and transport system giving enabling the customer to build his best travel plan for his desired trip. First chapter contains background of the thesis and illustrates it with example. Second chapter contains different uniformed search algorithms with diagrams. Similarly third chapter describes heuristic search strategies and their use in travel planning with different kind of heuristic search algorithms. Other kinds of search algorithms are discussed in chapter 5. Problem solving algorithms are rated by their performance and this performance is

measured by some characteristics which are mentioned in chapter 6. HTN planning which is good example of search algorithms in travel planning is discussed in better way with two different examples in 7th chapter Constraints are important elements for problem solving and in future they will be widely used in the travel planning. Its better example is Heracles which is constraint based travel planner and different kinds of intelligent are used in 8th chapter further more the advantages and comparison of Heracles with other travel planner is also discussed. Report is summarized and concluded in 9th chapter.

Finally good references which helped me for my research on travel planning, both web and book references are mentioned in last chapter with their authors.

1.1 Background

Newell and Simon in 1976 defined that intelligent behaviours come from the manipulation of symbol entities that represent other entities and that process by which intelligence arises is called heuristic search.

1.1.1 Problem Solving and Search

A search algorithm takes a problem as input and returns the solution in the form of an action sequence. Once the solution is found, the actions it recommends can be carried out. This phase is called the execution phase. After formulating a goal and problem to solve the agent calls a search procedure to solve it. A problem can be defined formally by four components which are

- The Initial State

The state in which agent starts for example In (Arad)

- Successor function

Description of possible actions and their outcomes e.g., $S(\text{Arad}) = \{(\text{GO}(\text{Sibiu}), \text{In}(\text{Sibiu})) \dots\}$

- Goal Test

It determines that if the given state is the goal state. For example in chess the goal is to reach a state called checkmate where the opponent's king is under attack and can't escape.

- Path Cost

A path cost is a function that assigns a numeric cost to each path. The problem solving agents choose a cost that reflects its own performance measure, e.g. path distance between the cities.

1.1.2 Problem Solving Agents

Problem solving agents decide what to do by finding sequence of action that lead to desirable states. The simplest agents which have been described so far are reflex agents. They use direct mapping from states to actions and are unsuitable for very large mappings. These are one kind of goal based agents. Problem solving agents find action sequences that lead to desirable state.

Example

In Romania

One holiday in Romania, we are currently in Arad and flight leaves tomorrow from Bucharest. Formulation of goal is that be in Bucharest in time and in formulation of problem is that

States: Being in various cities of Romania

Initial state of agent =In(Arad)

Actions: Drive between cities

Finding solution by sequence of cities for example Arad, Fagaras, Sibiu and Bucharest and in the last solution is executed. In first step we will look at the process of problem formulation then take a look how to search. The basic idea is to explore search space by generating successors of already explored states.

successor (In(Arad))returns
{<Go(Sibu),In(Sibu)>,
<Go(Timisoara),In(Timisoara)>,
<Go(Zerind), In(Zerind)>
}

Goal Test

Determines if the state is in the goal

In(Bucharest)

Path Cost

Assigns numeric cost to each path which allows determining best .Possible metrics are distance, time etc.

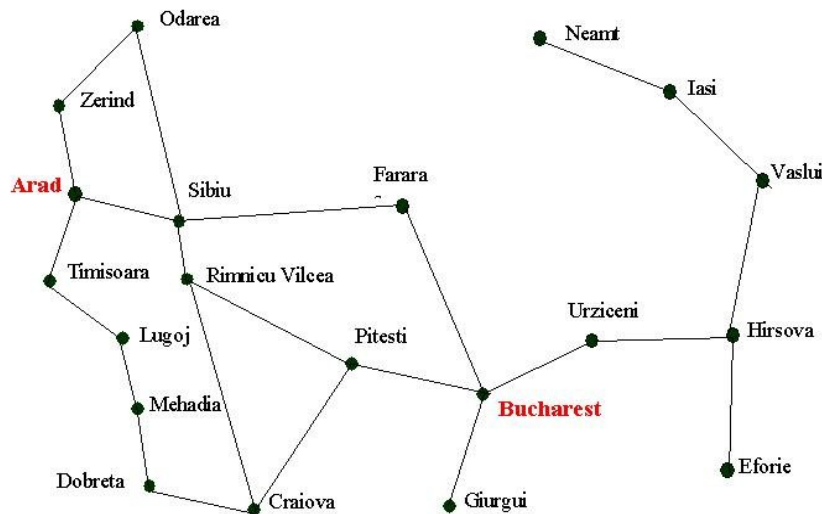


Figure 1 The Road map of Romania [14]

There are four basic kind of agent program that embody the principles underlying almost all intelligent systems:

- Simple reflex agents
- Model based reflex agents
- Goal based agents
- Utility based agents

1.1.3 Problem Formulation.

A PF is a process of deciding what actions or states to consider given a goal. We again take the example of Romanian trip, for the formulation of the problem of getting to Bucharest in terms of initial state, successor function, goal test and path cost. This formulation seems reasonable yet it omits a great many aspects of the real world. Compare the simple state description we have chosen, *In (Arad)* to an actual cross-country trip, where the state of the world includes so many things: the travelling companions, which programme is on the radio, the scenery of the window whether there are any laws of enforcement officers nearby, how far it is to the next rest shop, the condition of the road, the weather and so on. All these considerations are left out of our state descriptions because they are irrelevant to the problem of finding a route to Bucharest. The process of removing detail from a representation is called abstraction [4]. A driving action has

many effects. Besides changing the location of vehicle and its occupants, it takes up time, generates pollution and consumes fuel

2 Uniformed Search Strategies

Uniformed search algorithms for problem solving are a central topic of classical computer science by Horowitz and Sahni 1978, Operations research by Drefus by 1969. Uniformed search strategies use only that information which is available in the problem definition. Followings are important types of uniformed search strategies.

2.1 Brute force or Blind search methods

Brute force or blind search is a uniformed exploration of the search space and it does not explicitly take into account either planning efficiency or execution efficiency. Blind search is also called uniform search. It is the search which has no information about its domain [3]. The only thing that a blind search can do is to differentiate between a non goal state and a goal state. These methods do not need domain knowledge but they are less efficient in result. All brute force search algorithms must take $O(b^d)$ time and use $O(d)$ space [4]. The most important brute force techniques are breadth first, depth first, uniform cost, depth first iterative deepening and bidirectional search. Uniform strategies don't use any information about how close a node might be to a goal. They differ in the order that the nodes are expanded.

2.2 Breadth First Search

Breadth first search is a general technique of traversing a graph [4]. Breadth first search may use more memory but will always find the shortest path first [3]. In this search a queue data structure is used and it is level by level traversal. Breadth first search expands nodes in order of their distance from the root. It is a path finding algorithm that is capable of always finding the solution, if one exists [4]. The solution which is found is always the optimal solution. This task is completed in a very memory intensive manner. Each node in the search tree is expanded in a breadth wise at each level. Thus all expanded nodes are retained till the search is completed.

It can be implemented most easily by maintaining the queue of nodes. The number of node at level d is (b^d) , the total number of nodes produced in the worst case is $b+b^2+b^3+\dots+b^d$ which

is $O(b^d)$ the asymptotic time complexity of breadth first search. [2]. Breadth first search is a complete algorithm with exponential time and space complexity.

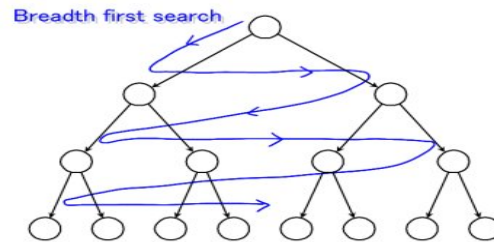


Figure 2 Breadth First Search [22]

2.3 Depth First Search

Depth first search is also important type of uniform or blind search. DFS visits all the vertices in the graph, this type of algorithm always chooses to go deeper into the graph.[6]. After DFS visited all the reachable vertices from a particular source vertices it chooses one of the remaining undiscovered vertices and continues the search. DFS reminds the space limitation of breadth first search by always generating next a child of the deepest unexpanded node.

One interesting property of depth first search is that, the discover and finish time of each vertex form a parenthesis structure. If we use one open parenthesis when a vertex is finished then the result is properly nested set of parenthesis.[6]

The disadvantage of depth first search is that, it may not terminate on an infinite tree, but simply go down the left most paths forever. Even a finite graph can generate an infinite tree. [4]

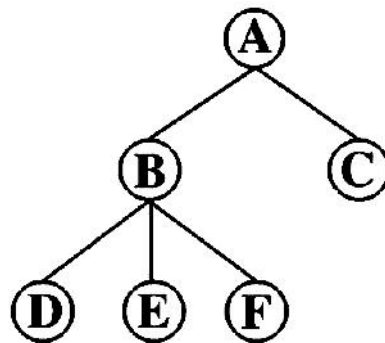


Figure 3 Depth First Search [11]

2.4 DFS Iterative Deepening

This kind of search performs depth first search to bounded depth d , starting $d=1$, and on each iteration it increases by 1[2]. DFID is asymptotically optimal in terms of time and space among all brute force search algorithms that finds optimal solution on a tree. It was created as an attempt to combine the ability of BFS to always find an optimal solution. With the lower memory overhead of the DFS, we can say it combines the best features of breadth first and depth first search[5]. It performs the DFS search to depth one, then starts over, executing a complete DFS to depth two, and continues to run depth first searches to successfully greater depths until a solution is found. This algorithm is equivalent to common backtracking algorithm in that a path is expanded until a solution is found. DFID does better because other nodes at depth d are not expanded [5]. It never generates a node until all shallower nodes have been generated. The first solution found by DFID is quite guaranteed to be along the shortest path and depth is increased one by one. Its main function is that it returns a solution or failure.

This search is faster than BFS because latter also generates nodes at depth $d+1$ even if the solution is at depth d . It is liked often because it is affective compromise between two other methods of search [4]. It terminates if there is a solution. It can produce the same solution as produced by depth first search produces but it does not use the same memory. Its main properties are that it is memory efficient and always find best solution if one exists.

2.5 Greedy Search

This algorithm uses an approach which is quite similar to the best first search algorithm. It is a simple best first search which reduces the estimated cost to reach the goal. Basically it takes the closest node the goal state and continues its searching from there. It expands the node that appears to be closest to the goal [3]. This search starts with the initial matrix and makes very single possible change then looks at the change it made to the score. This search then applies the change till the greatest improvement. The search continues until no further improvement can be made. The GS never makes a lateral or uphill move. It uses minimal estimated cost $h(n)$ to the goal state as measure which decreases the search time but the algorithm is neither complete nor optimal. The main advantage of this search is that it is simple and finds solution quickly and as

far as its disadvantages are concerned it is not optimal, susceptible to false start and the time complexity $O(b^m)$ is same for space complexity [3].

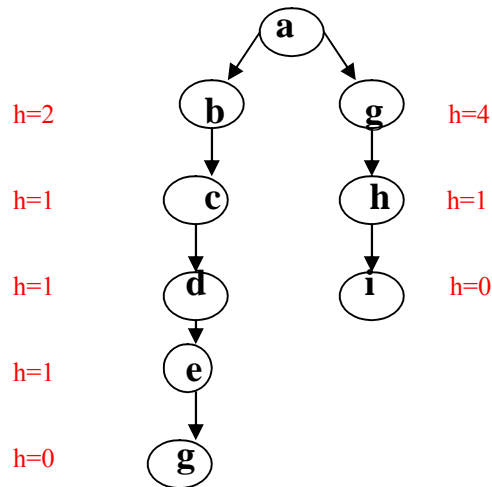


Figure 4 Greedy Search [4]

2.6 Bidirectional Search

The idea behind bidirectional search is to run two searches same time, one forward from the initial state and other backward from the goal stopping when the two searches meet in the middle [4]. Bidirectional search is implemented by having one or both of the searches check each node before it is expanded to see if it is in the fringe of other search tree; if it is so, a solution has been found. For example if a problem has solution depth $d=6$, and each direction runs breadth-first search one node at a time then in the worst case the two searches meet when each has expanded all but one of the nodes at depth 3. For $b=10$, it means a total of 22,200 node generation, compared with 11,111,100 for a standard breadth-first search. This algorithm is complete and optimal, if both searches are breadth first; other combinations may sacrifice completeness, optimally or both [4].

The most difficult case for bidirectional search is when the goal test gives only an implicit description of some possibly large set of goal states.

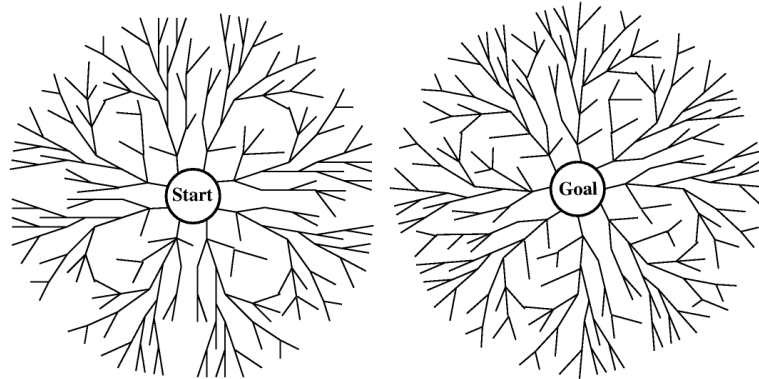


Figure 5 Bidirectional Search [2]

2.7 Iterative deepening

Iterative deepening search or iterative deepening depth-first search is a general strategy which is often used in combination with depth first search and finds the best depth limit. It does this by gradually increasing the limit respectively 0, 1, 2 and so on until a goal is found [4]. This will occur when the depth limit reaches d , the depth of the shallowest goal node. DFID is asymptotically optimal in terms of time and space among all brute force search algorithm that find optimal solutions on a tree.

Since this search never generates a node until all shallower nodes have been generated, the first solution found by DFID is guaranteed to be a shortest path. Furthermore since at any given point it is executing a depth first search, saving only a stack of nodes, and the algorithm terminates when it finds a solution at depth d , the space complexity of DFID is only $O(d)$. [6]

Cost is: $O(b^1 + b^2 + b^3 + b^4 + \dots + b^L) = O(b^L)$.

Iterative deepening has advantages of both depth and breadth first search. Iterative deepening

Gets its memory saved in the same way as depth first search does at the expense of redoing some computations again and again.

3 Heuristic Search

Heuristic is a problem specific knowledge that decreases expected search efforts. It is a technique which some times work but not always. Heuristic search algorithms use information about the problem to help directing the path through the search space. These searches use some functions that estimate the cost from the current state to the goal presuming that such function is efficient. Generally heuristic incorporates domain knowledge to improve efficiency over blind search

[6]. In AI heuristic has a general meaning and also a more specialized technical meaning. Generally a term heuristic is used for any advice that is effective but is not guaranteed to work in every case [5].

Example

In this example of heuristic we consider the problem of driving across the country if you think each of each city as “state” in this problem domain, and each highway connecting cities as an operator that can be applied, we can treat this problem in terms of model of problem space.

Suppose for some reason we are in Okloham, Nebraska and driving towards Boston, Massachusetts. We have to choose which highway to take out of town. If we want to apply Breadth first technique to this problem we would consider all of the highways out of Omaha and then consider all of the highways from the place they lead to. So we would consider 80 going east towards Illions and west towards the 1-76 junction we will consider 1-29 going north towards Sioux city, as well as going south towards Kansas city. From Kansas City we would consider 1-70 towards St Louis, as well as east, heading into Kansas. At each city or junction we would consider all the possible ways to go from there. Clearly we would spend a lot of time considering stupid routes. Clearly what we want to do in this case is to consider roads that at least head north or east that go roughly towards Boston. This is the heuristic because it might be the case that we will have to choose roads some times that don’t go directly towards our goal. For example when we go to Chicago we would have to head south for a while to get around the Lake Michigan. Still it is much better than heading into South Dakota [12] .

We suppose that instead of knowledge that Boston is north and east of Omaha, we know that distance from each city in the county to Boston. We have the following information

Omaha to Boston	1463 miles
Kansas to Boston	1437 miles
Wichita to Boston	1627 miles
Chicago to Boston	1015 miles
Souix city to Boston	1570 miles

So by this table it can be easy for us to decide which route to choose. Heading to Kansas City gets us a bit closer, but if we take 1-80 to Chicago, we will be much closer. Another heuristic in the driving domain is to choose the highway that takes us to the city that s closest to our goal but

again it will not work and for the same reasons lakes, mountains, tornadoes can make what might seem to be the shortest route which is less attractive [12].

Following are main kinds of heuristic search.

3.1 A * Search

A* is a cornerstone name of many AI systems and has been used since it was developed in 1968[1] by Peter Hart, Nils Nilsson and Betram Rapahel. It is combination of Dijkstra's algorithm and best first search. It can be used to solve much kinds of problems. A* search finds the shortest path through a search space to goal state using heuristic function. This technique finds minimal cost solutions and is also directed to a goal state called A* search. The A* algorithm also finds the lowest cost path between the start and goal state, where changing from one state to another requires some cost. A* requires a heuristic function to evaluate the cost path that passes through the particular state [2]. It is very good search method but with complexity problems. This algorithm is complete if the branching factor is finite and every action has fixed cost. A* requires heuristic function to evaluate the cost of path that passes through the particular state. It is defined by the following formula [2]

$$f(n) = g(n) + h(n) \quad [2]$$

Where $g(n)$ is the cost of the path from the start state to node n and $h(n)$ is the cost of path from node n to the goal state. The speed of execution of A* search is highly dependant on the accuracy of the heuristic algorithm that is used to compute $h(n)$. A* search is both complete and optimal. Thus if we are trying to find the cheapest solution a reasonable thing to try first is the node with the lowest value of $g(n) + h(n)$. It turns out that this strategy is more than just reasonable which provides that the heuristic function $h(n)$.

3.2 Hill Climbing Search

Hill climbing search algorithm is simply a loop that continuously moves in the direction of increasing value, which is uphill. It stops when it reaches a "peak" where no neighbour has a higher value. The hill climbing comes from that idea that if you trying to find the top of the hill and you go up direction from where ever you are. The question that remains is whether this hill is indeed the highest hill possible. Unfortunately, without further extensive exploration, that

question cannot be answered[20]. This technique works but as it uses local information that's why it can be fooled. The algorithm does not maintain a search tree, so the current node data structure need only record the state and its objective function value [4]. In this algorithm only a local state is considered when making a decision of which node is to expand next. When a node is entered all of its successors nodes have a heuristic function applied to them. The successor node with the most desirable result is chosen for traversal [3]. Hill climbing sometimes called greedy local search because it catches a good neighbour state without thinking ahead about where to go next. Hill climbing often makes very rapid progress towards a solution because it is usually quite easy to improve a bad state. Hill climbing is best suited to the problems where the heuristic gradually improves the closer it gets to the solution it works badly where there are sharp drop-offs. It assumes that local improvement will lead to global improvement. There are some reasons by which hill climbing often gets stuck which are stated below.

Local Maxima

A local maximum is the peak that is higher than each of its neighbouring states, but lower than the global maximum. Hill climbing algorithms that reach the vicinity of local maximum will be drawn upwards towards the peak, but then will be stuck with nowhere else to go.

Ridges

Steps of east, north south and west may go down but the step to north west may go up. Ridges result in a sequence of local maxima that is very difficult for greedy algorithm to navigate [4].

Plateaus

The space has a broad flat area that gives the search algorithm no direction (random walk).

Many variants of hill climbing have also been invented which are described below.

3.2.1 Stochastic Hill climbing

This variant chooses at random from among the uphill moves and the probability of selection can vary with the steepness of the uphill move. This usually converges more slowly than steepest ascent but in some state landscapes it finds better solution.

3.2.2 First choice Hill climbing

First choice Hill climbing variant implements stochastic hill climbing by generating successors randomly until one is generated that is better than current state. This is a good strategy when a state has thousands of successors [4].

3.3.3 Random Restart Hill Climbing

This Variant adopts the well known adage, if at first you don't succeed try again and again. It conducts a series of hill climbing searches from randomly generated initial states, stopping when a goal is found.

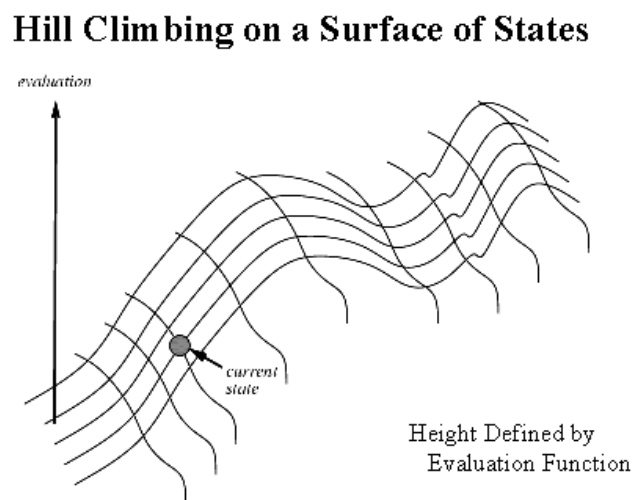


Figure 6 Hill climbing search [9]

3.3 Simulated Annealing

In the early 1980's, Kirkpatrick, Gelatt & Vecchi (1982, 1983) and independently Cerny (1985) introduced the concept of the physical annealing process in combinatorial optimization problem. The reason originates from the analogy between the solid annealing process and the problem of solving large scale combinatorial optimization problem[7]. Simulated annealing exploits an analogy between the way in which metal cools and freezes into a minimum energy, crystalline structure and the search for a minimum in a more general system. Simulated annealing is a probabilistic search algorithm and can avoid becoming trapped at local minima. Simulated annealing uses a control parameter T , which by analogy with the original application is known as

the system temperature. It escapes local maxima by allowing some bad moves but gradually decrease their frequency.

Properties

If T decreases slowly enough, then simulated annealing search will find a local optimum with probability approaching 1.

It is also widely used in VLSI layout, airline scheduling etc.

3.4 Generate and Test search

This is the simplified form which contains the following steps.

- 1- It generates a possible solution
- 2-Compares the possible solution to the goal state.
- 3-If the solution is found it returns the success otherwise it again goes to the step 1.

These are brute force algorithms that simply generate a possible solution and test to see if it is correct if the solution is not correct then they repeat. The main benefit of this algorithm is that it is easy to implement but its time complexity is higher than other search algorithms [3].It takes very long time before the solution is found. This algorithm is improved to hill climbing algorithm. In such algorithms heuristic function is used to estimate the distance from the goal state. Thus only that solution is generated that will minimize the distance.

Disadvantages

The generate and test approach is not very efficient because it also generates many wrong assignments of values of variables which are rejected in the testing phase. Furthermore the generator leaves out the conflicting instantiations and it generates other assignments independently of the conflict. Visibly one can get far better efficiency if the validity of the constraint is tested as soon as its respective variables are instantiated.

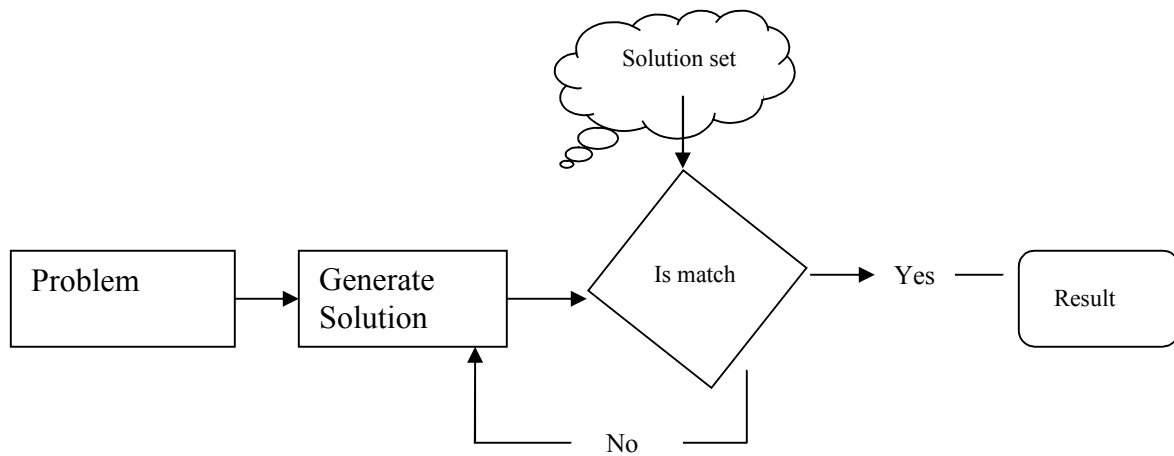


Figure 7 Working of Generate and test search algorithm

3.5 Back Tracking

A variant of depth-first search is called backtracking search which uses still less memory. In this search only one successor is generated at a time rather than all successors. Each partially expanded node remembers which successor to generate next. In this way only $O(m)$ memory is needed rather than $O(bm)$ [4]. It is the most common algorithm for solving constraint satisfaction problem (CSP).

Disadvantages

There are three major disadvantages of the standard backtracking scheme which are

- Thrashing
- Redundant work
- Detects the conflict too late.

Thrashing

Thrashing occurs because the standard BT algorithm does not identify the main reason of the conflict or problem i.e., conflicting variables. That is why search in different parts of the space keeps failing for the same reason. Intelligent back tracking can reduce or finish thrashing. It is

done by the scheme on which backtracking is done directly to the variable that cause the failure[17].

Redundant work

Second disadvantage of backtracking is to perform redundant work. Even if the conflicting values of the variables are identified during the intelligent backtracking they are not recalled for the immediate detection of the same problem in a subsequent computation. There is one method which is backtracking based and that eliminates both above disadvantages that method is called dependency directed backtracking and it is used truth maintenance system.

Conflict Detection

The last drawback of backtracking algorithm is that it detects the conflict too late it means it does not detect the conflict too fast before its occurrence i.e., it detects after assigning the values to all the variables of the conflicting constraint. It can be avoid by applying consistency techniques to forward check the possible conflicts.

3.6 Beam Search

The beam search is another modification of A* search where the list of nodes under consideration is limited to best n. The meaning of beam is to imply the beam of a flashlight wandering around the search space. The local beam search algorithm keeps track of k states rather than just one. It starts with k randomly generated states. At each step, all the successors of all k states are generated. If any one is a goal, the algorithm halts otherwise it selects the k best successors from the complete list and repeat the process [4]. It keeps only best candidates ate ach level. The main idea of beam search is to only keep fraction of the total queue so it means that you just go around those states that are relatively good and forget the rest. This could avoid some local optima but not always. It is most useful when the local optima are not much common and search space is too big [12].

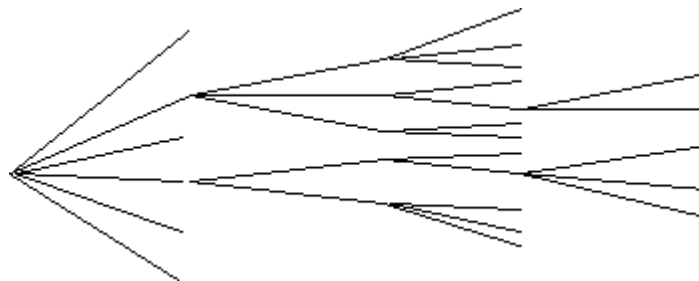


Figure 8 Expansion of nodes in beam search [21]

3.7 Best First Search

Best first search is an instance of the general Tree search or Graph search algorithm in which a node is selected for expansion based on evaluation function $f(n)$ [4]. Traditionally the node with the lowest evaluation is selected for the expansion because the evaluation measures distance to the goal. Best first search can be implemented with in general search frame work via a priority queue, a data structure that will maintain the fringe in ascending order of f values. This search algorithm serves as combination of depth first and breadth first search algorithm. Best first search algorithm is often referred to as greedy algorithms this is because they quickly attack the most desirable path as soon as its heuristic weight becomes the most desirable [3].

There is a whole family of Best first search algorithms with different evaluation functions. A key component of these algorithms is a heuristic function denoted $h(n)$:

$h(n)$ = estimated cost of the cheapest path from node n to a goal node [4] .

The main steps of this search algorithm are; add the initial node (starting point) to the queue and Secondly it compares the front node to the goal state, if they match then the solution is found and If they don't match then expand the front node by adding all the nodes from the links

If all the nodes in the queue are expanded then the goal state is not found i.e., there is no solution and it stops.

Apply the heuristic function to evaluate and reorder the nodes I n the queue [11].

3.8 Branch and Bound

The branch and bound method was first used for parsimony by Hendy and Penny [8]. In 1975 La Blanc presented a branch and bound algorithm solution methodology for the discrete equilibrium transportation network design problem [13]. These search methods basically rely on that idea that we can divide our choice into sets using same domain knowledge and ignore a set when we can determine that the optimal element can't be in it. In 1991 Chen proposed a branch and bound with a stochastic incremental traffic assignment approach for the single class network design problem It is an algorithmic technique which finds the optimal solution by keeping the best solution found so far. If partial solution can't improve on the best it is abandoned [7]. By this method the number of nodes which are explored can also be reduced. It also deals with the

optimization problems over a search that can be presented as the leaves of search tree. The usual technique for eliminating the sub trees from the search tree is called pruning. The load balancing aspects for branch and bound algorithms make it parallelization difficult. The primary difficulty being that usual assumption requires no prior information about the likely location of the search target.

For example travelling sales man problem we decompose our set of choice into set of sets, in each set of which we have taken different route of the current city. We continue to decompose until we have complete paths in the graph. In the mean while if we are decomposing the sets, we find two paths that lead to the same node. We can eliminate the more expensive one [5].

The reason of success of the branch and bound approach is limited to small size networks. In the problems of large networks, where the solution search space grows exponentially with the scale of the network, the approach becomes relatively prohibitive. In the worst case, given a network design problem with n binary variables, it would require the solution of 2^n number of solutions of the traffic assignment routine [13].

3.9 Constraint Search

A constraint search does not refer to any specific search algorithm but to a layer of complexity added to existing algorithms that limit the possible solution set [3]. Heuristic and acquired knowledge can be combined to produce the desired result.

3.9.1 Constraint Satisfaction search

A constraint satisfaction problem is a special kind of search problem in which states are defined by the values of a set of variables and the goal test specifies a set of constraints that the value must obey. Constraint may be higher order, Unary or binary.

Unary: These constraint concern the value of single variable

Binary: These kind of constraints relate pairs of variables

Higher order: In this order constraints are between all variable in a row.

3.10 Means End Analysis

Mean end analysis is also an important kind of search algorithm and it is used in AI applications when a complex search is needed to be done. It is a different approach to find the solution and they are a common form of heuristic algorithm that has stood the test of time [3].

Early implementations included the general problem solver (GPS), FDS and STRIPS. Now a day Means-End analysis is still used to create effective searches in the field of distributed computed artificial intelligence. It also focuses the search on actions which decrease the distance between current and target.

There are three main kind of goals used in mean end analysis search algorithm which are

- 1- Transform a state into a set of states
- 2- Decrease a distance possessed by a state
- 3- Apply an operator to the state to reduce the difference.[3]

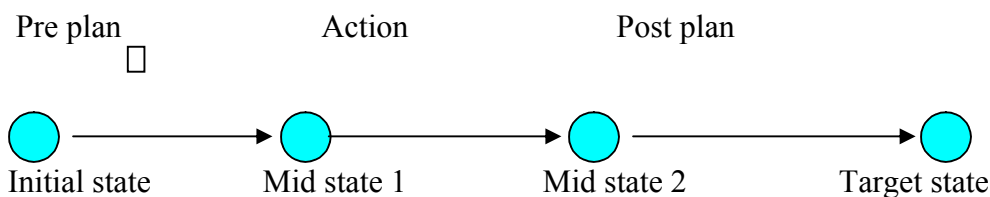


Figure 9 Means end analysis states

4 Asynchronous Search Algorithm

In recent times there have been a lot of attempts to rectify and create a set of algorithms which can perform well in the parallel asynchronous fashion. This process allow for each individual programme unit to operate locally without knowledge of the global problem, yet still benefit the search as whole. This algorithm include

- Asynchronous dynamic programming
- Learning real time A*
- Real Time A*
- Real time multi agent [3].

4.1 Asynchronous Dynamic Programming

Asynchronous dynamic programming is most basic and common form of asynchronous path finding algorithm. Due to its simplicity it is applicable in very small and controlled environment. It serves as a starting point for other asynchronous path finding solutions [3].

The main idea behind this model is to asynchronously determine the minimum distance from each node to goal state and this distance is represented by $h^*(n)$ is calculated by using that function [3].

$f^*(j) = k(j, n) + h^*(j)$ for each successor j

$h^*(n) = \min f^*(j)$ [3]

4.2 Learning Real time A*

Learning real time A* is a variation, LRTA* is a variation of Dynamic programming and it does not require a search process to be running on every node in the problem space. If the problem is to be solved repeatedly with the same goal state but different initial states, one would like that algorithm that improves its performance over time and LRTA* is such algorithm [5].

4.3 Real time A*

Real time A* algorithm produces an action every k second, where k depends upon the depth of the search horizon. RTA* will find a path to solution if there exist one and the search graph can be accessed easily [6]. It is not necessary that the path will be optimal. Though RTA* will make those decisions which are locally optimal.

4.4 Real time multi agent

Real time multi agent require for inter-agent cooperation. When the real time asynchronous search is performed with multiple agents it is named Real time multi agent search. This is mostly done by having all agents operate in parallel in the same problem space while they also attempting to obtain the same goal. Each agent runs its own version of the search, while all agents share the information which is useful for them. When the first agents search is complete then all other stop searching since the current task is completed [3].

An intelligent web hunting agent is a reality and can be explored to find out the applications of semantic searching of real life applications.

5 Other Search Techniques

5.1 Tabu Search

Tabu search was developed by Glover (1982, 1989, 1990, 1993) [13]. As a heuristic search strategy approach for relatively hard optimization problems has a wide range of applications from graph theory and matroid settings to general pure and mixed integer programming problems. In order to proof the efficiency of the exploration process one needs to keep the track not only of local information but also of some information related to the exploration process. This systematic use of memory is an important quality of Tabu search [10]. Tabu search could simply be viewed as the as an extremely general heuristic procedure. In the search process it is some time fruitful to intensify the search in some region of S because it may contain some acceptable solutions. The results from different tabu search applications are very encouraging. These applications include a job schedule problem, traveling salesman, mixed integer programming, also the transportation equilibrium network design problem and a variety of other discrete optimization problems Tabu search has been applied successfully to many combinatorial problems some of applications are describes as follows.

- The graph colouring problems
- The maximum independent set problems
- The course scheduling problems.

To describe the working of Tabu search a combinational optimization problem is presented in the following form.

Minimize $c(x)$ $x \in X$ in R^n

$c(x)$ is the objective function $x \in X$ is the constraint

Tabu search elements are listed below

- Tabu List T
- Aspiration Level
- Strategic Oscillation
- Intermediate and long term memory

Tabu List T

The tabu list encloses the most recent moves. Its main function is to stop main move in the period of time to prevent cycling and avoid local optimum [13].

Aspiration Level

Aspiration level use is one of the main functions of Tabu search. If the objective value $c(s(y))$ of move $s(y)$ is less than of pre specified level $A(s,y)$ then the Tabu status of the move may be overridden

Strategic Oscillation

SO is also an important feature of Tabu search, referring to the search state where the moves are allowed to enter the infeasible region. The search oscillates back and forth between the feasible and infeasible region.

Intermediate and long term memory

The intermediate memory function records features that are common to a set of best trial solutions during a specific period of the search. The search then continues using these common features as a heuristic to identify new solution. Secondly the long term memory function diversifies the solution from the current search stage by using a heuristic which is usually generated from the search [13].

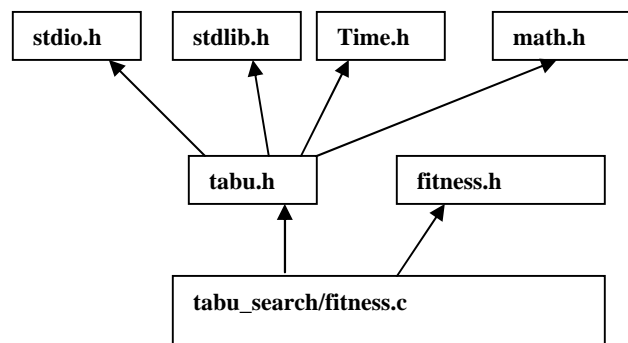


Figure 10 include dependency graph for tabu search

5.2 Island Search

The island search is also used to solve the scientific problem. Island search first of all locate intermediate states that will be the part of final solution and then search between those states.

If there exist many such islands the search between them relatively will be short and easy. The island might be found by another search process in a much space of abstract. For example we might find the solution first that involves the steps like remove muffler bearing and install generator. Once we find the solution we can then figure out how to do the task. This approach is better when the operators can be separated and when doing one does not affect the result of many others [12].

5.3 Semantic Search Algorithm

In semantic world of intelligent agents used for searching, the stress will be more on the information semantics or in other words the meaningfulness of the information. Implementing an intelligent web hunting agent is a reality and can be explored to find out the applications of semantic searching of real life applications.

6 Comparison of Different Search Algorithms

6.1 Measuring Problem Solving Performance

The output of problem solving algorithm is either failure or solution. Some algorithms might get stuck in an infinite loop and never return an output. When determining which search algorithm is appropriate for a problem space, it is necessary to derive and compare general attributes of each algorithm [3]. We will evaluate algorithm's performance in four ways

1- Completeness

Is that algorithm guaranteed to find a solution when there is one? This column is a Boolean indicator of whether or not the search algorithm is exhaustive.

2- Optimality

Does that strategy find the optimal solution? This column indicates that whether or not the solution found will always be the optimal solution.

3- Time Complexity

How long does it take to find a solution? It is the order of complexity search time used by algorithm expressed as a function.

4- Space Complexity

How much memory is needed to perform the search? This column is the order of complexity memory requirements of algorithm also expressed as a function.

The comparison among different search algorithms by these factors is shown by the table below.

Algorithm	Time	Memory	Complete	Optimal
Breadth First	$O(b^d)$	$O(b^d)$	Yes	Yes
Depth First	$O(b^d)$	$O(d)$	No	No
DF iterative deepening	$O(bd)$	$O(d)$	Yes	Yes
Bidirectional	$O(b^{d/2})$	$O(b^{d/2})$	Yes	Yes
Hill climbing	$O(b^d)$	$O(1)-O(b^d)$	No	No
Best First	$O(b^d)$	$O(b^d)$	Yes	No
A*	$O(b^d)$	$O(b^d)$	Yes	Yes
ID A*	$O(b^d)$	$O(d)$	Yes	Yes
Beam Search	$O(n^d)$	$O(n^d)$	No	No
Means End	$O(b^d)$	$O(b^d)$	No	No
Generate & Test	$O(((bd!)/((bd-d)!))/2)$	$O(d)$	Yes	No
Asynchronous Dynamic	$O(b^d)$	$O(1)$	Yes	Yes
Learning Real Time A*	$O(b^d)$	$O(b^d)$	Yes	Yes
Real Time A*	$O(b^d)$	$O(b^d)$	Yes	No

Where

d = depth of solution with in search tree

b = branching factor of search tree

n = subset of b for which algorithm will actually process.

Table 1 Comparison of different search algorithms

In the table 1, the attributes create a basis for decision making. Each of the algorithms examined in the paper contains weak and strong attributes. It is the function of the problem space to weight the trade-offs between the algorithm and determine which algorithms provides the best solution

[3]. It can be seen from the table that the time estimate from all the searches are similar. The three exceptions are the Bidirectional, Beam and Generate and test series. The main reason that the Bidirectional search has a lesser time estimate is because it is simultaneously working from both ends of the problem looking for a common intermediate node. The beam search has a time estimate of $O(n^d)$ as opposed to the more common $O(b^d)$. It is because the Beam search is modified A* search that examines on the best n branches at any node. It speeds up processing, but at the cost of assuming that a suboptimal node will never need to be travelled to reach the goal state. If that is the case the solution to the search will never be found. The memory requirement of the search algorithms are more distributed than the time estimates. In many cases a search algorithm will approach a problem breadth first or depth first.

7 Implementation of AI Search Algorithms

The problem solving approach has been applied to a vast array of task environments. A toy problem and 8-puzzle problem are main examples of it. In the highway system a heuristic search approach combined simulated annealing and tabu search strategy and it was developed to provide a fast and efficient methodology for the problem.. The heuristic search strategy is used to find the good solution with the use of the historical information and properties of the problem and then it provides attractive alternative for the large scale applications. So we are going to implement the heuristic search methodology approach to solve the problem of travel planning by Hierarchical task network (HTN) planning and AI based travel planner Heracles.

7.1 HTN Planning

HTN planning (Tate 1977, Sacerdoti 1977, Currie and Tate 1985, Wilkins 1988) is an AI planning methodology that creates plans by task decomposition. In this process the planning system decomposes task into smaller and smaller subtasks, until primitive tasks are found that can be performed directly [15]. HTN planning system has methods which are knowledge based. Each method has a prescription for how to decompose some task into a set of subtasks, with different restrictions that must be satisfied in order for the method to be applicable and also various constraints of the subtask and relationship among them. Given a task to complete, the planner chooses an applicable method, instantiate it to decompose the task to subtasks and chooses and instantiates other method to decompose it to subtasks and it continues. If the constraint in the

subtask or the interactions among them prevents the plan from being feasible the planning system will backtrack and try other methods.

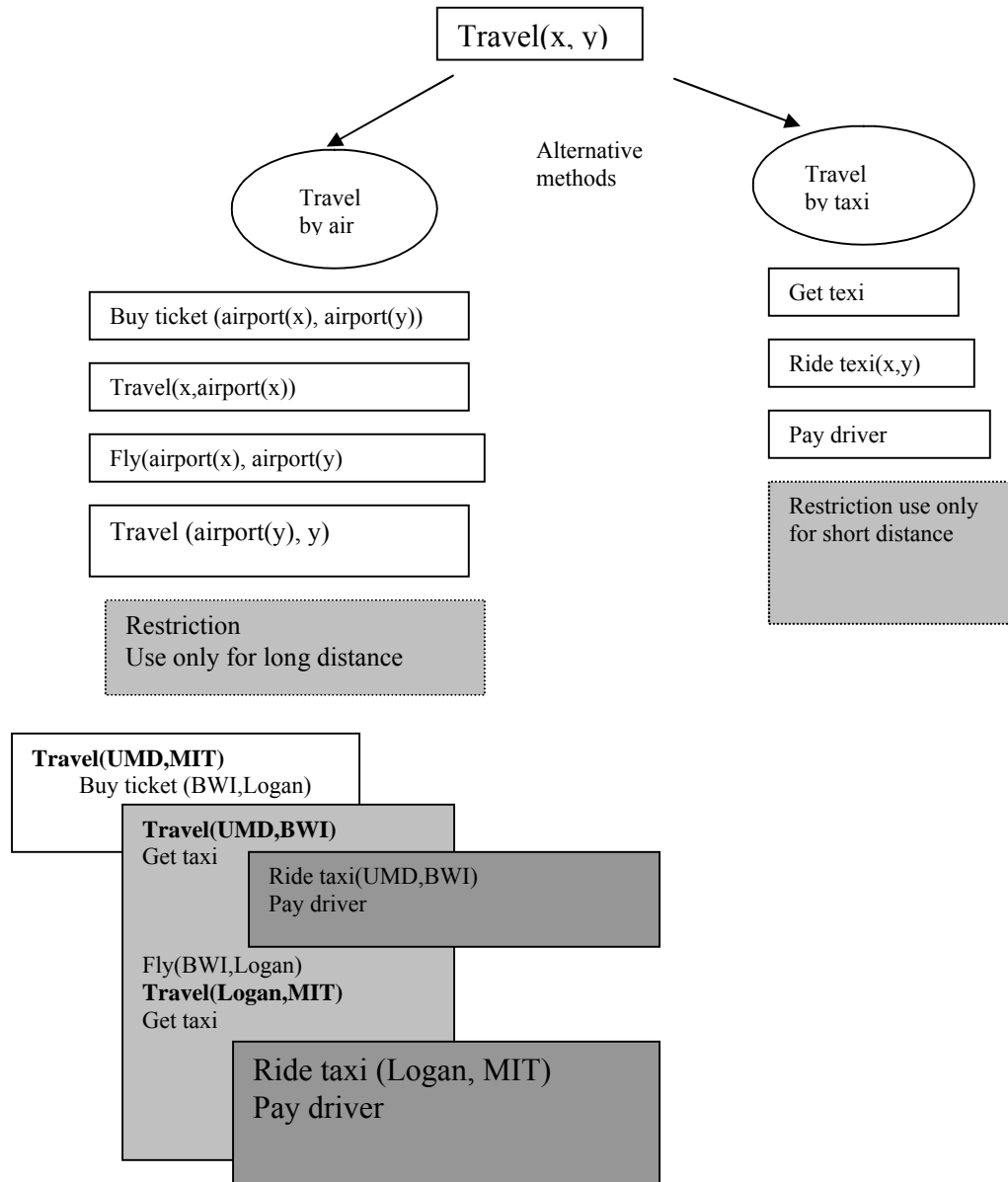


Figure 11 methods for travelling from one location to another and how they might be the task of travelling from the Maryland to MIT.

Fig 11 shows two methods for the task of travelling from one location to another: which are travelling by air and travelling by taxi. Travelling by air involves different subtasks which are purchasing a plane ticket, travelling to the local airport, flying to an airport close to our

destination and travelling from there to our destination. Travelling by taxi involves the subtasks of calling a taxi, riding it in to the final destination and paying the driver. Each method has restrictions on when it can be used i.e., air travel is used only for long distances and travel by taxi is only applicable for short distances.

Now we will consider the task of travelling from the University of Maryland to MIT. Since this is along distance so we cant apply the travel by taxi so we will choose the travel by air method .As it is shown in figure it decomposes the task in the following subtasks purchase a ticket from Baltimore Washington International airport to Logan airport, travel from the university of Maryland to BWI, fly from BWI to Logan and then travel from Logan to MIT. For the subtask of travelling from the University of Maryland to BWI and travelling from Logan to MIT we can use the travel by Taxi method to produce additional subtasks as show in fig. Solving a planning problem using HTN is not as easy as it is shown in this simple problem. It is more complicated. We will discuss some complications which arise generally during solving the travel planning problem.

The planner may need to recognize and resolve interactions among the subtasks i.e., in planning how to get to the airport one needs to make sure one will arrive at time to the airport to catch the plane.

In the above example it is always clear that which method will be used but in general more than one method may be applicable to a task. If it is not possible to solve the subtasks produced by one method, it may be necessary to backtrack and try another method rather than that [15].

7.2 Travel Planning By Heracles an AI based travel planner

Planning a trip is common but complicated human activity. Once we planned it is affected by many possible events such as schedule changing and flight cancellations and checking for these possible events it requires lot of time and efforts. A travel planner must integrate this information with user preference such as for airline flying times, cost constraint and company policies. To support planning the system must:

- Collect and integrate the information in a coherent structure that catches the task need for application domain

- Evaluate trade offs and select among alternative courses of action
- Let the user search and override system suggestions.

For providing flexible and easy interaction the system must

- Let the user input data and change his choices any time during planning and
- Handle information source that returns result back to user asynchronously.

So now we are going to describe an AI based travel planner named Heracles II which is generally information gathering and monitoring tool and its working for simplifying these possible complicated events. Heracles divides the network hierarchically corresponding to the task structure of the application domain in the same manner which is similar to hierarchical task network planning. The application designer group's variable and constraints related to the distinct task into a package which is called a template [16]

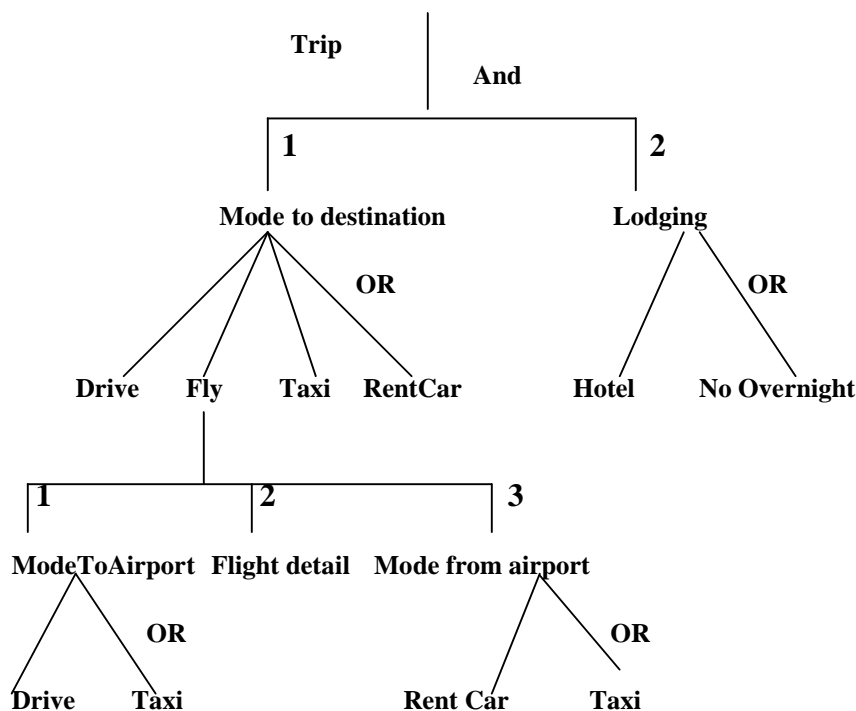
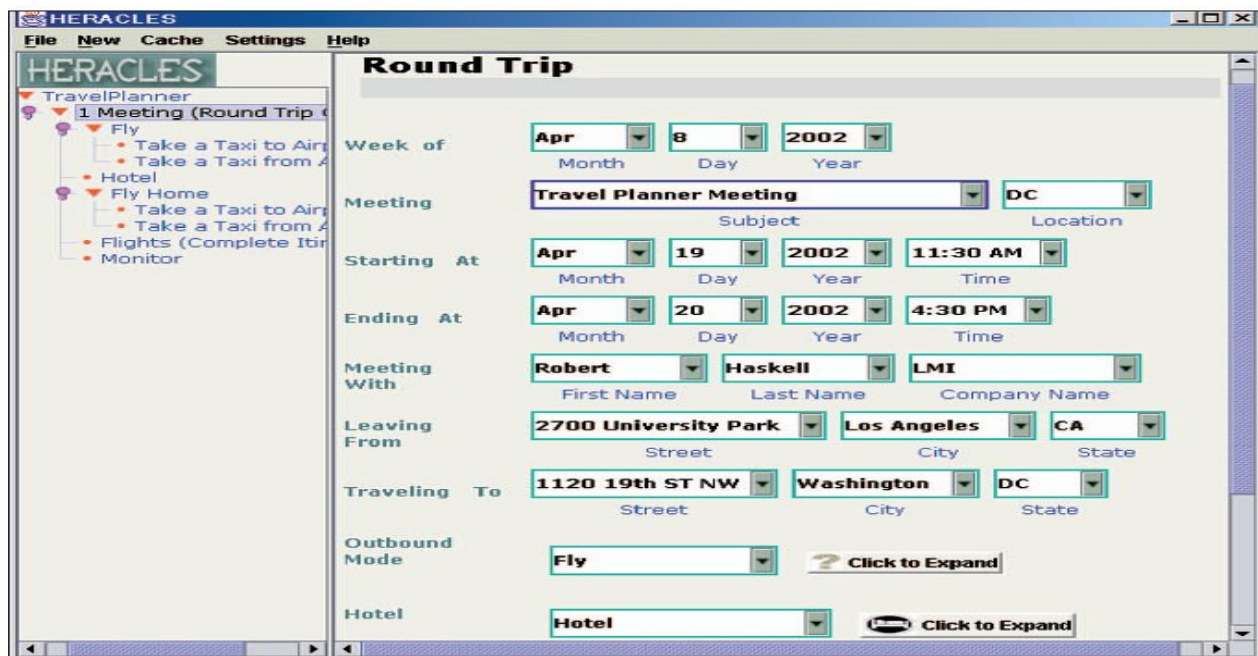


Figure 12 Hierarchical Organization of templates for the travel planner

The above figure shows the hierarchical organization templates for the travel planner. This is the top level template of the Heracles travel planner which includes the most important information about the trip such as the origin place, destination and dates of travel. The next layer of decision

includes the alternative means of transportation such as flying, taking the train, renting the car, driving the users own car or taking a taxi to destination. Secondly choice of accommodation on reaching at destination. Below figure a shows that system suggests flying the variable and constraints constitute another template in figure b Heracles further decomposes each template into more specific sub templates. For example one user has chosen fly as the main transportation mode, the system must evaluate then how to get to the airport i.e., if it is by taxi, by driving a car and parking it to the airport.



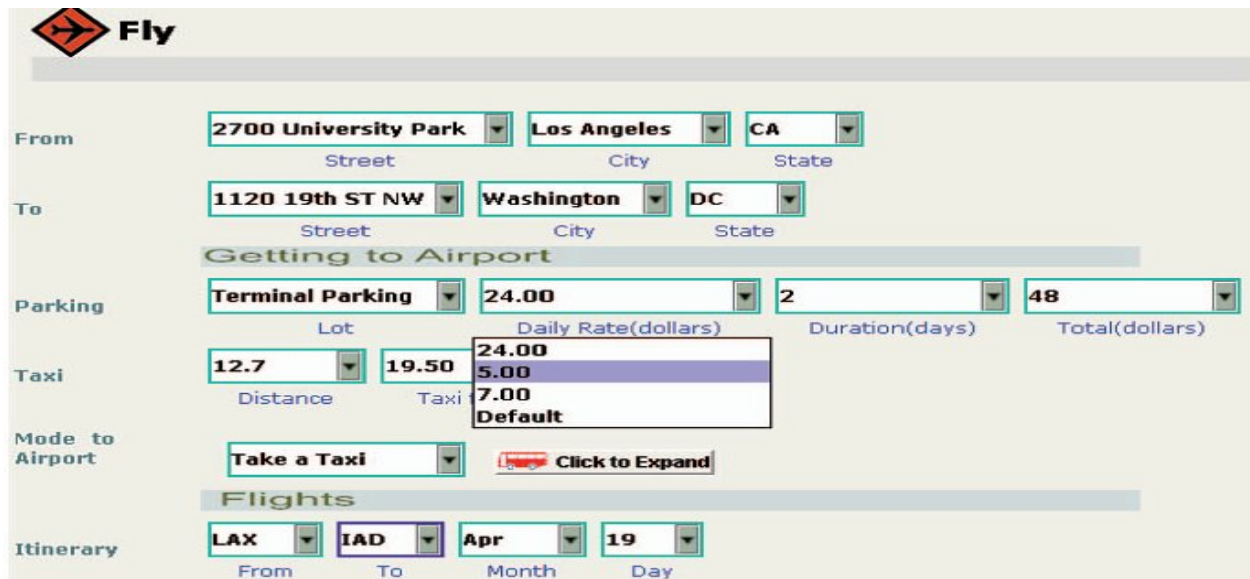
The screenshot shows the HERACLES Travel Planner application window. The title bar reads 'HERACLES'. The menu bar includes 'File', 'New', 'Cache', 'Settings', and 'Help'. On the left is a tree view under 'TravelPlanner' with the following structure:

- TravelPlanner
 - 1 Meeting (Round Trip)
 - Fly
 - Take a Taxi to Airport
 - Take a Taxi from Airport
 - Hotel
 - Fly Home
 - Take a Taxi to Airport
 - Take a Taxi from Airport
 - Flights (Complete Itinerary)
 - Monitor

The main area is titled 'Round Trip' and contains the following fields:

- Week of:** Apr (Month), 8 (Day), 2002 (Year)
- Meeting:** Travel Planner Meeting (Subject), DC (Location)
- Starting At:** Apr (Month), 19 (Day), 2002 (Year), 11:30 AM (Time)
- Ending At:** Apr (Month), 20 (Day), 2002 (Year), 4:30 PM (Time)
- Meeting With:** Robert (First Name), Haskell (Last Name), LMI (Company Name)
- Leaving From:** 2700 University Park (Street), Los Angeles (City), CA (State)
- Traveling To:** 1120 19th ST NW (Street), Washington (City), DC (State)
- Outbound Mode:** Fly (dropdown), ? Click to Expand
- Hotel:** Hotel (dropdown), ? Click to Expand

(a)



Fly

From: 2700 University Park (Street), Los Angeles (City), CA (State)

To: 1120 19th ST NW (Street), Washington (City), DC (State)

Getting to Airport

Parking: Terminal Parking (Lot), 24.00 (Daily Rate(dollars)), 2 (Duration(days)), 48 (Total(dollars))

Taxi: 12.7 (Distance), 19.50 (Taxi), 24.00 (Default), 5.00, 7.00

Mode to Airport: Take a Taxi (Click to Expand)

Flights

Itinerary: LAX (From), IAD (To), Apr (Month), 19 (Day)

(b)

Figure 13 Travel planner template (a) a top level and (b) fly template [16]

7.2.1 Monitoring Travel Status

There are various dynamic events that can affect the travel plan for example flight delay, cancellations, fare reduction etc. Many of these events can be detected in advance by monitoring information source. The travel assistant is always aware of that some of the information it got is subject to change, so it delegates the task of following the evolution of such information to a set of monitoring agents. For example a flight schedule change is a critical event since it can have affect no only on the user schedule at the destination but also on the reservation on the hotel and a care rental agency. Behind that there are also monitoring agents who aim is to make a trip more convenient or cost-effective. For example tracking airfares or finding restaurants near the current location of the user. These main agents are described as under

1- Flight status monitoring agent

These agents use the ITN flight tracker sight to get the current status of a given flight. If the flight is on time the agent sends the use a message to that effect two hours before departure. If the user flight is delayed or cancelled and sends the message to the user preferred device i.e. cellular

phone of pager. If the flight is delayed more than one hour the agent sends a fax to the car rental counter to confirm the user's reservation

2- The Air fare monitoring agent

The air fare monitoring agents keep track of current prices for the user's itinerary. The airlines change price unpredictably, but the traveller can claim for refund if the price drop below the purchase price. Mostly agent gets air fares from Orbitz (www.orbitz.com)

3- The flight schedule monitoring agent

The flight schedule monitoring agent keeps record of the changes to the schedule departure time of a flight and notifies the user if there is any change. Without this type of agent traveller mostly knows this type of schedule after reaching at the airport.

4- The earlier flight monitoring agent


The earlier flight monitoring agent also used Orbitz to find that flight which leave earlier than the scheduled flight. It shows the alternative earlier flights and their status. This information becomes very important when the scheduled flight is significantly delayed or cancelled.

5- The flight connection agent

These agents track the user's current flight and a few minute before its landing, it sends the user gate and status of the connecting flight.

6- The restaurant finder agent

The restaurant finder agent allocates the restaurant for the user either by global positioning system (GPS) or by his own location according to the plan. If requested it provides 5 closest restaurants providing cuisine type, price, address, phone number, longitude, latitude and distance from user's location.


Taxi

Leaving From

2700 University Park

Los Angeles

CA

St.
City
State

Driving To

LAX

Los Angeles

CA

St.
City
State

Suggested Departure

Apr

18

2002

10:31 PM

Month
Day
Year
Time

Predicted Arrival

Apr

18

2002

10:55 PM

Month
Day
Year
Time

Taxi fare

19.50

Total Drive


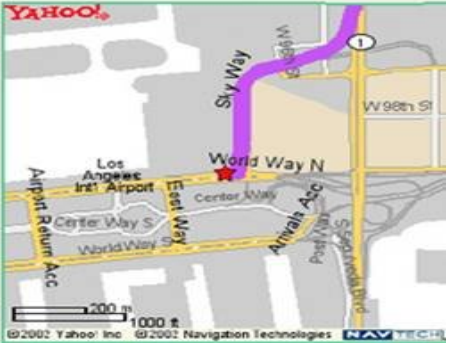
12.7

0

24

DistYahoo
Hrs4
Mins

Maps


Round Trip Flights

Preference

Lowest Price

Departs

LAX

Los Angeles, CA

Apr

19

Code
City, State
Month
Day

Returns

LGB

Wash DC/Dulles, DC

Apr

20

City, State
Month
Day

Price

SNA

ONT

BFL

SAN

Default

Outbound Flight 1

Lines

192

LAX

IAD

Thu

Apr

18

Airline
Flight #
From
To
Day
Month
Date

11:55 PM

7:36 AM

Depart
Arrive

a (the Taxi template)

b(user changes values in Round trip flights)

Figure 14 user interaction and constraint propagation for the travel planner[16]

7.2.2 Constraint Network Representation

A constraint network is usually set of variables and constraints that are related to each other and define the true values for the variables. In Heracles the set of possible values for each variable is determined by domain expression. The primitive constraints that operate on the same variable are combined to generate a domain expression. The grammar of domain expression is stated as under.

```
Exp =(AND Exp AND)|
    (OR Exp Exp) |
    (DLIST Exp Exp...)|
    Primitive Constraint
Primitive Constraint =
    PREDICATE Var 1 Var 2.....VarN
```

A primitive constraint is a computable component which may be implemented by a local table lookup by retrieving a set of tuples from a remote wrapper, by the computation of a local function or by calling an arbitrary external program.

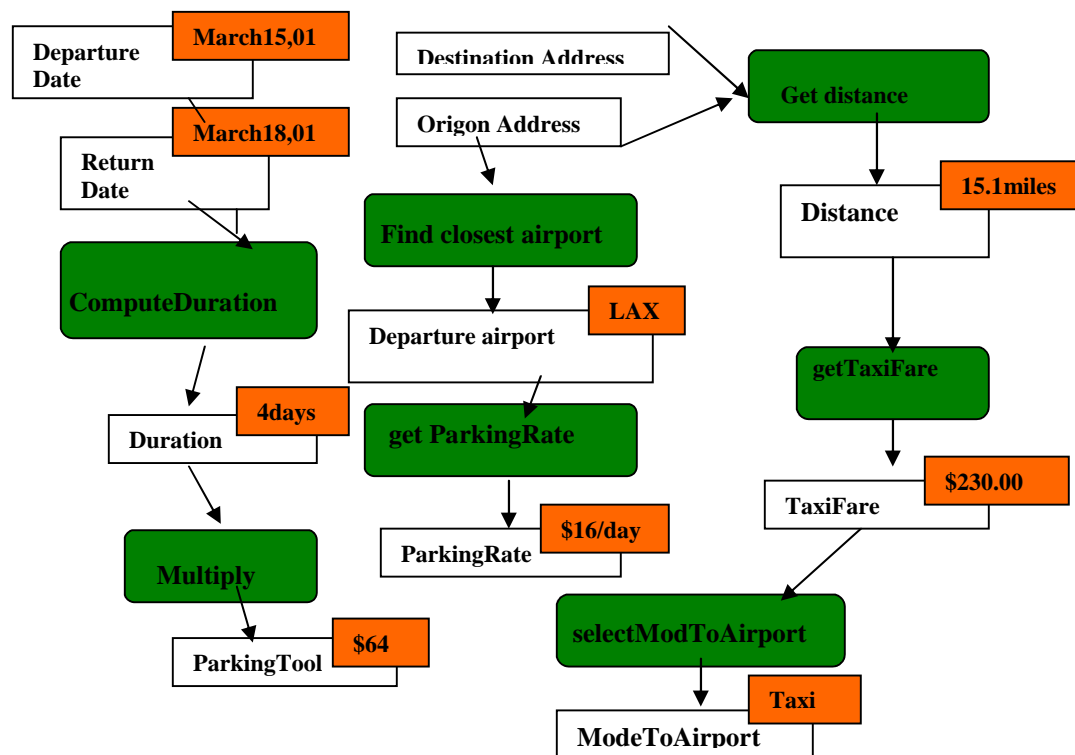


Figure 15 Constraint Network Driving versus Taking Taxi[16]

Fig 15 defines the fragment of the constraint network of the travel assistant that addresses the selection of the method of travel from user's original location to the airport. There are two choices under consideration which are

1-Driving one's car which implies leaving it parked at the parking of the airport for the duration of the trip.

2- Taking a taxi

In the sample network figure the variables are shown with white rectangles and the assigned values as orange rectangles next to them. The variable keep the relevant information for this task in the application domain such as Duration of the trip, DepartureDate, the ParkingTool, the TaxiFare and the ModeToAirport. The Departure Airport has been assigned value of LAX which is assigned by the system since it is the closest airport to the user's address.

A constraint is a n-array which relates a set of n variables by defining values of those variables. In Heracles the constraints are directed [16]. The system evaluates constraint only after it has assigned values to a subset of its variables and the input variables. The constraint evaluation process produces the set of possible values for the output variables.

In the Fig 15 the constraint are shown as rounded green rectangles. In this example the ComputeDuration constraint involves three variables (DepartureDate, ReturnDate and Duration) and it is implemented by function that computes the duration of a trip in the given departure and return dates. The constraint getParkingRate is implemented by calling an information agent that accesses the website that contains parking rate for airports in the country. Each variable can also be related to preference constraint. This constraint is often implemented as function that imposes an ordering of the values of the domain. Preference for business travel is to choose a hotel closest to the meeting is such kind of example.

7.3 Information Agents

Heracles use information agents to support both trip planning and supporting, these information agents are similar to other type of software agents. Their plans are differentiated by a focus on gathering, integrating and monitoring of data from distributed and remote sources. For efficient plan execution Theseus is used which is streaming dataflow architecture for plan execution.

7.3.1 Defining Information Agent

Building an information agent requires defining a plan in Theseus. Each plan contains name, a set of input and output variables and a network of operators connected in a producer consumer fashion. Plans in Theseus are same like operators, they are named and have input and output arguments. Any plan can be called an operator within any other plan. This is a sub plan capability which allows a developer to define new agents by composing existing ones

7.3.2 Accessing Web sources

Information agents also access web sources and with this they help to create a better plan. There is no data stored locally in the system of travel assistant. All the information is taken from the web source. For this process a wrapper are build who enable web sources to be queried as structured data. Once a wrapper for a site is created, Theseus agents can programmatically access data from the site using the wrapper operator in their plans.

It is same in the Yahoo weather when we send a request to get the weather report for a specific region and it returns the corresponding data to us.

7.3.3 Monitoring Web sources

Theseus agents are able to monitoring online sources and performing a set of actions which are based on observed changes. This monitoring is performed by retrieving data from online sources and comparing the returned results against information that was retrieved previously and stored locally in the database. This process provides the capability to not only check current status but also determine how that information has changed overtime.

There are many methods in which plans can react to a monitoring event. First of all a plan can use email or fax operator to asynchronously notify interested parties about important updates to monitored data. Secondly a plan can schedule or unscheduled other agents in response to some condition. For instance, once a flight has departed the flight monitoring agent can schedule the Flight connection agent to run a few minutes before the scheduled arrival time. Theseus agents are managed by a scheduling system which allows agent to be run first or run at fixed interval. As example information agents that monitor a data source consider the pal for the flight status which is shown in fig 16. Initially the agent executes a wrapper operator to retrieve the current flight

status information. The resulting normalized flight status information shows that the flight is arrived, cancelled or departed. If the flight is delayed or cancelled the user is notified via Email operator.

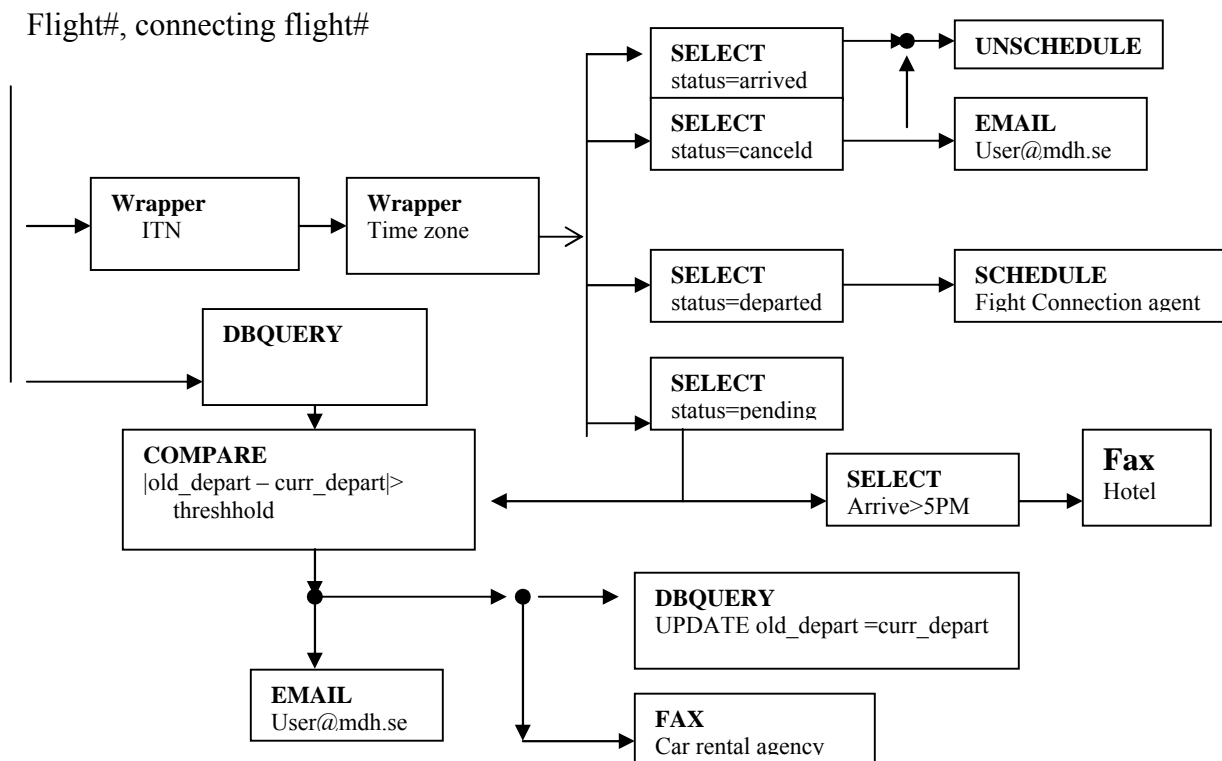


Figure 12 status information monitoring agent [16]

7.4 Programming of Search Algorithms

Four different kind of programmes have been made in which the AI search algorithms have been implemented the explanation of them is described below.

1- In first programme the name of an input file from the command line is accepted. The programme reads the string from the input file into dynamically allocated array of pointers to characters. It prompts the user for a string to search for. After this it searches the array using binary search. If the array is found it reports the row and explain in which target is found. If the target is not found it asks for try it again.

2- The second programme is about binary tree search, in which binary tree search algorithm is used. In the list of alphabetically strings. This search is made with in the limit of first item and to last item using a binary tree search method. We have to determine a pointer to the string and it is found in the pointer string. The first item I the subindex limit of PtrList array of pointers and similarly LastItem which is the high subindexlimit of PtrList array of pointers. The function supposes that pointers in PtrList point to alphabetically ordered strings. The first and lastitem allow the users to find it in the subset of PtrList array of pointers searching by binary tree. If found it returns the index values when not found returns 0 and ask for try again.

3- In the third programme AI search algorithm is utilized. Search is base class which implements the unidirectional search. From this class we can derive other classes which implement the actual search algorithms such as depth first search and breadth first search are easily derived. The behaviour of the search i.e. the type of the search algorithm depends upon the implementation of the virtual function add(). The class search processes objects of class node and classes derived from node(DepthNode, Uninode and BestNode). On the other hand Application level classes that are derived from one the derivatives of the search class must implement the following function. is goal () : determines if the node is goal node

1: yes 0:No . The pseudo code is written below

```
#include <stdio.h>
#include<ainodes.h>
class Search
public:
    Search (int numop , Node *start , Node *goal) ;
    Virtual ~search ();
    Virtual int  is goal( const Node *);
    Int generate ();
Void display () const ;
IntrList <Node> *get_sol();
Node *get_goal() const ;
Void set_startnode(Node *);
Void set_goalnode(Node *);
```

```
Void clear ();  
Private :  
    Node *solve ();  
Virtual int  add (Node *) =0  
Void  print _sol(Node *) const ;  
    Int num_op;  
Node *goalnode,  
    *solgoal;  
Protected  
    SortedIntrList<Node> open ,  
                           closed;  
};  
#endif;
```

8 Evaluation of Constraint Network by Heracles

8.1 Advantages of Heracles

Heracles consists of AI templates and these templates have three main kinds of advantages

1-Complexity Reduction

They reduce complexity .Templates let us exploit, expertise and use default activities rather than reason every time from the first principles .They include all the relevant variables therefore they make clear the requirements for solving a problem even when solution methods are not known .Further more when we encounter novel problems templates make us able to differentiate the familiar parts from the new parts. This makes it possible to have the machine handle routine details while human focus on the novel parts.

2- Uncertainty Reduction

Templates in Heracles also reduce uncertainty; they provide some enumeration of the probable actions of agents, for whatever reason is out of communication and cannot be observed. For distributed, coordinated planning and execution control under these circumstances knowing what collaborating agents will likely do in a given situation is a good advantage.

3- Facilitating Process Improvement

Successful templates can be learned and reused not only for problem solving but also for other purposes for example learning better strategies, training and selecting appropriate activities in real solutions.

8.2 Comparison with other Travel Planners.

When compared with other travel planners Heracles and Heracles 2 both are superior to others. In the below table we compare them with other travel planners. As far as planning and constraint satisfaction is concerned Heracles has this ability to plan better and its programming technique is based on constraint satisfaction. It also has better user interaction as it provides graphical user interface. Unlike other travel planners it is not time consuming.

It gathers information very smartly and fast. Other travel planner smart client has no better technique for information gathering. On the other hand trip planner travel planner is not better in information gathering. Expedia and other travel planners or websites are also better but have no constraint satisfaction programming technique.

	Planning/Constraint Satisfaction Programming	User interaction	Information Gathering
Heracles, Heracles2	✓	✓	✓
SmartClient	✓	✓	
Trip-Planner	✓		✓
Expedia and other Travel web sites.		✓	✓

Table 2 A comparison of travel planning approaches.

8.3 Challenges for Intelligent system in tourism

The industry's main features are its heterogeneous and worldwide distributed quality and its strong SME base. It's another inherent quality is mobility where the entire tourist life cycle is integrated with the respective supplier process .The frame work of future with better qualities Must possess following features.

- The system must be cooperative, heterogeneous and distributed.
- It must enable full autonomy of the respective participants
- Should support the entire consumer life cycle.
- Allow dynamic network configurations
- Provide intelligence for customers and suppliers as well as in the network
- It must focus on mobile communication enabling multi channel distribution.

8.4 Limitations of Heracles

Heracles showed that constraint based approach was well applied to support mixed initiative planning where user interaction and asynchronous information collection were main requirements but it has also two serious limitations.

First the template selection mechanism was hardcoded into the implementation and not integrated with the constraint network. The template selection in Heracles was performed by procedures that inspected the value of differentiated variables called expansion variables. Heracles adds the corresponding child template to the constraint network and removes the alternative child template. This hardcoded behaviour means that most of the logic for selecting among the templates needs to appear at the parent template even if such information logically belongs to the child template. This diminishes the modularity of templates and tended to create large monolithic templates.

Second limitation is that Heracles could not handle cycles in the constraint network .This required the template designer to specify the constraints some time in an unintuitive way or forego some lines of reasoning altogether[4].

These limitations were removed and new version of Heracles, Heracles 2 was presented.

8.5 Opportunities for research

Planning research has been central to AI since its inception, and papers on planning are a staple of mainstream AI journals and conferences.

The future success of online content provider will mainly depend upon their ability to filter mass information available on the web into the form that is truly useful to the individual user.

Most designers in this field have been too realistic and implemented mostly agent based constraint travel planners. In the future web Heracles will have bright image and will be used most of the travel planners. The tourism domain is an excellent example of the trend towards personalized service and complex market mechanism. It reflects user becoming a part of product creation. Researchers must also study non technical issues related to markets and users such as

- Dynamic market and network structures.
- Pricing and market design.
- Design and experimenting business models
- User decision modelling and usage analysis.

E-tourism market is dynamic and there are lot of opportunities in the research in this field. Future systems mainly emphasize e-tourism importance. A lot of work is being done in constraint satisfaction programming for tourism and travel planning.

8.6 Future of AI planning

AI planning has bright future in the coming age of technology and path finding. Many research works are being done in this field and it is progressing very fast. In the future it will be widely used in the multimedia games and for transportation planning .In future travel planner will be mostly expert system based and AI programming techniques will be widely used in it. It will create better correlation between travel agent and travellers. Constraint based travel planners will be appreciated on both sides.

9 Discussion

The AI search algorithms are common algorithms in AI for finding better routes and also for navigating routes. All AI search algorithms are important but some of them are used quite often due to which they are more usable and mostly implemented than others. The base of all algorithms is A* algorithm which is used for shortest path and fast algorithm for finding better routes. As all the algorithms and their properties are mentioned in the table 1. As far as implementation is concerned I have chosen bidirectional search, breadth and depth first search algorithms. Breath first search is mostly used in AI search techniques to find shortest path and it is not very time consuming. As we measure functionality of all algorithms In terms of four factors which are completeness, optimality, time complexity and space complexity. In the breadth

first search it is complete and is guaranteed to give the solution of the problem and, secondly it also provides the optimal solution. Memory wise it also keeps more record and also better than other algorithms and performs function faster. When we compare this algorithm in aspect of time, it takes $O(b^d)$ which means it does not take much time to give solution of problem. Depth first search is not optimal or complete but memory and space wise is better. Basically these algorithms are checked for finding better or fast solution, if these algorithms are failed to provide the exact solution of problem then other algorithms are tested. One reason is also shortage of time and it requires more research.

10 Summary and Conclusion

In this theoretical thesis of travel planning search algorithms for AI are discussed. It gives general overview of all the AI search algorithms and their importance in artificial intelligence and in the field of travel planning. Artificial intelligence search algorithms are widely used in travel planning and it is illustrated with the examples and diagrams. The comparison between these algorithms and their advantages and disadvantages are also briefly discussed. Hierarchical Task Network(HTN) travel planning which is new progress in AI, travel planning by algorithms is also discussed with examples. This travel planning algorithm captures hierarchical structure of the planning domain. This planning domain contains non primitive actions and schemas for reducing them. These reduction schemas are given by the designer and they also express the preferred ways to complete the tasks. The Heracles will be more user friendly and convenient travel planner. It is constraint based AI travel planner which will reduce both time and money for the travellers.

11 References

- [1] Lecture Notes by RobSaunders from City University London on Introduction to Artificial Intelligence for games .November 2004. <http://www soi.city.ac.uk/~rob/Lecture09-8up.pdf>
Last referred on February 2006.
- [2] Article on Applying Artificial Intelligence Search Algorithms and Neural Networks for Games by Stuart James Kelly who is professional programmer in UK wrote on 13-1-03 <http://www.generation5.org/content/2003/KellyMiniPaper.asp>,Last referred on March 06.
- [3]Hemant M. Joshi, Joshua A. Mcadams, Search Algorithms in Intelligent Agents, Scientific Paper on various search algorithms. Last referred on March 2006.
- [4]Artificial intelligence a Modern Approach a book by Stuart Russell and Peter Norvig
A book on Artificial Intelligence and its algorithms. Last referred February 2006.
- [5] Scientific paper on Artificial Intelligence Search Algorithms on June 1999 By Richard E-Korf who belongs to computer science department University of California Los Angeles. <http://citeseer.ist.psu.edu/cachedpage/388096>, Last referred May 2006.
- [6]Definations by Patrik Owen Doyle who is PhD from Stanford University wrote definations On AI search algorithms and techniques.<http://www.cs.dartmouth.edu/7Ebrd/Teaching/AI/Lectures/summaries/search.html#definitions> ,Last referred on May 2006.
- [7]Brian T Luke, “Evolutionary Programming Applied to the development of Quantative Structure property relationships. <http://members.aol.com/btluke/featur03.htm>.
Last referred on April 2006.
- [8] An article on Phylogenetic trees by Peer Itsik on 1-1-2001.
<http://www.cs.tau.ac.il/~rshamir/algmb/00/scribe00/html/lec08/node10.html>, Last referred On April 2006.
- [9] Lecture notes on AI and search algorithms from the Masey University Institute of Information and mathematic science. [cs-alb-pc3.massey.ac.nz/ notes/59302/103.html](http://cs-alb-pc3.massey.ac.nz/notes/59302/103.html),
Last referred on May 2006.
- [10]A tutorial on Tabu search by Alain Hertz, Eric Taillard, and Dominique de Werra.
<http://www.cs.colostate.edu/~whitley/CS640/hertz92tutorial.pdf>,Last referred March 2005.
- [11] Tony Abou-Assaleh computer science student Brooke University Canada, Article on Intelligent Search Algorithm in 1999.http://www.cosc.brocku.ca/~cspress/Helloworld/1999/02-feb/search_algorithms.html, Last referred May 2006.
- [12] John Betali Associate professor Department of cognitive science university of California Sandiago, lecture notes on cognitive science and search algorithms. Nov 16, 1999.

<http://cogsci.ucsd.edu/%7Ebatali/108b/lectures/heuristic.html>, Last referred March 2006.

- [13] Dr Qiefeng Zeng, Dr Kyriacos C Mouskos, Heuristic search strategies to solve Transportation network design, Final Report to the New Jersey Department of Transportation and the national center for Transportation and Industry productivity in December 1997.
- [14] Graham Kendall, Reader in Computer Science - School of Computer Science and Information technology, the University of Nottingham G5AIAI, notes on Introduction To Artificial Intelligence .<http://www.cs.nott.ac.uk/~gxxk/courses/g5aiai/003blindsearches.htm> on Sep 2001, Last referred May 2006.
- [15] Stephen J.J Smith, Dana Nau, Tom Throp. Computer Bridge: A big win for AI planning AI magazine, 19(2):93-105, June 1998. <http://www.cs.umd.edu/~nau/papers/bridge-ai-mag98.pdf>, Last referred April 2006.
- [16] Jose Luis Ambite, Craig A. Knoblock and Maria Muslea, University of Southern California, Information science Institute, Steven Minton Fetch Technologies. <http://www.isi.edu/integration/papers/ambite05-is.pdf>, Last referred May 2006.
- [17] Introduction to Stochastic Search and Optimization (Estimation, Simulation and Control) By James C. Spall, ISBN 0-471-33052-3, 2003 by Wiley-Interscience series. Last referred February 2006.
- [18] Algorithms By Robert Sedgewick 2d ed. 650p. 24cm ISBN 0-201-06673-4, Addison Wesley Publishing Company, Last referred February 2006.
- [19] Artificial Intelligence(Structures and Strategies for Complex Problem Solving) by George F Luger 4th edition ISBN 0-201-64866-0, Addison Wesley Publishing Company, Last referred March 2006.
- [20] Neural Networks(Algorithms ,Applications and Programming Techniques) By James A A Freeman/David M. Skapura ISBN 0-201-51376-5, Addison Wesley Publishing Company Last referred February 2006.
- [21] The University of New South Wales Australia, The Aq Induction Algorithm, By Claude Sammut Professor of Computer sciences at UNSW Australia, <http://www.cse.unsw.edu.au/~Teaching/AI/notes/ml/06prop/aq/aq.html>. Last accessed 24 May 2006.
- [22] The Artificial Intelligence company website of Japan, Introduction to different search Algorithms, http://ray.sakura.ne.jp/search_problem/depth_breadth.html, last accessed June 2006.