الجامعة التكنولوجية

قسم علوم الحاسوب

فرع الذكاء الاصطناعي

المرحلة الثالثة

الكورس الأول

مادة طرق البحث الموجه

2025 - 2024

أ.د. حسنين سمير عبدالله

1- Intelligent Search Methods and Strategies

Search is inherent to the problem and methods of artificial intelligence (AI). This is because AI problems are intrinsically complex. Efforts to solve problems with computers which human can routinely innate cognitive abilities, pattern recognition, perception and experience, invariably must turn to considerations of search. All search methods essentially fall into one of two categories, exhaustive (blind) methods and heuristic or informed methods.

2 -State Space Search

The state space search is a collection of several states with appropriate connections (links) between them. Any problem can be represented as such space search to be solved by applying some rules with technical strategy according to suitable intelligent search algorithm.

What we have just said, in order to provide a formal description of a problem, we must do the following:

1- Define a state space that contains all the possible configurations of the relevant objects (and perhaps some impossible ones). It is, of course, possible to define this space without explicitly enumerating all of the states it contains.

2- Specify one or more states within that space that describe possible situations from which the problem-solving process may start. These states are called the initial states.

3- Specify one or more states that would be acceptable as solutions to the problem. These states are called goal states.

4- Specify a set of rules that describe the actions (operators) available. Doing this will require giving thought to the following issues:

• What unstated assumptions are present in the informal problem description?

• How general should the rules be?

• How much of the work required to solve the problem should be precomputed and represented in the rules?

The problem can then be solved by using rules, in combination with an appropriate control strategy, to move through the problem space until a path from an initial state to a goal state is found. Thus the process of search is fundamental to the problem-solving process. The fact that search provides the basis for the process of problem solving does not, however, mean that other, more direct approaches cannot also be exploited. Whenever possible, they can be included as steps in the search by encoding them rules. Of course, for complex problems, more sophisticated computations will be needed. Search is a general mechanism that can be used when no more direct methods is known. At the same time, it provide the framework into which more direct methods for solving subparts of a problem can be embedded.

To successfully design and implement search algorithms, a programmer must be able to analyze and predict their behavior. Questions that need to be answered include:

• Is the problem solver guaranteed to find a solution?

• Will the problem solver always terminate, or can it become caught in an infinite loop?

• When a solution is found, is it guaranteed to be optimal?

• What is the complexity of the search process in terms of time usage? Memory usage?

• How can the interpreter most effectively reduce search complexity?

• How can an interpreter be designed to most effectively utilize a representation language?

To get a suitable answer for these questions search can be structured into three parts. A first part presents a set of definitions and concepts that lay the foundations for the search procedure into which induction is mapped. The second part presents an alternative approaches that have been taken to induction as a search procedure and finally the third part present the version space as a general methodology to implement induction as a search procedure. If the search procedure contains the principles of the above three requirement parts, then the search algorithm can give a guarantee to get an optimal solution for the current problem.

3 -General Problem Solving Approaches

There exist quite a large number of problem solving techniques in AI that rely on search. The simplest among them is the generate-and-test method. The algorithm for the generate-and-test method can be fom1ally stated in figure (1).

It is clear from the above algorithm that the algorithm continues the possibility of exploring a new state in each iteration of the repeat-until loop and exits only when the current state is equal to the goal. Most important part in the algorithm is to generate a new state. This is not an easy task.

```
Procedure Generate & Test
Begin
    Repeat
      Generate a new state and call it current-state;
    Until current-state = Goal;
End.
```

Figure (1), Generate and Test Algorithm

If generation of new states is not feasible, the algorithm should be terminated. In simple algorithm, we, however, did not include this intentionally to keep it simplified. But how does one generate the states of a problem? To formalize this, we define a four tuple, called state space, denoted by

{nodes, arc, goal, current },

where

Nodes represent the set of existing states in the search space;

an arc denotes an operator applied to an existing state to cause transition to another state;

Goal denotes the desired state to be identified in the nodes;

and current represents the state, now generated for matching with the goal.

The state space for most of the search problems takes the form of a tree or a graph. Graph may contain more than one path between two distinct nodes, while for a tree it has maximum value of one.

To build a system to solve a particular problem, we need to do four things:

1. Define the problem precisely. This definition must include precise specifications of what the initial situation(s) will be as well as what final situations constitute acceptable solutions to the problem.

2. Analyze the problem. A few very important features can have an immense impact on the appropriateness of various possible techniques for solving the problem.

3. Isolate and represent the task knowledge that is necessary to solve the problem.

4. Choose the best problem-solving technique(s) and apply it (them) to the particular problem.

Measuring problem-solving performance is an essential matter in term of any problem solving approach. The output of a problem-solving algorithm is either failure or a solution. (Some algorithm might get stuck in an infinite loop and never return an output.) We will evaluate an algorithm's performance in four ways:

• **Completeness:** Is the algorithm guaranteed to find a solution when there is one?

• **Optimality:** Does the strategy find the optimal solution?

• **Time complexity:** How long does it take to find a solution?

• **Space complexity:** How much memory is needed to perform the search?

4 - Search Technique

Having formulated some problems, we now need to solve them. This is done by a search through the state space. The root of the search tree is a search node corresponding to the initial state. The first step is to test whether this is a goal state. Because this is not a goal state, we need to consider some other states. This is done by expanding the current state; that is, applying the successor function to the current state, thereby generating a new set of states. Now we must choose which of these possibilities to consider further. We continue choosing, testing and expanding either a solution is found or there are no more states to be expanded. The choice of which state to expand is determined by the search strategy. It is important to distinguish between the state space and the search tree. For the route finding problem, there are only N states in the state space, one for each city. But there are an infinite number of nodes.

There are many ways to represent nodes, but we will assume that a node is a data structure with five components:

• **STATE:** the state in the state space to which the node corresponds؛

• **PARENT-NODE:** the node in the search tree that generated this node؛

• **ACTION:** the action that was applied to the parent to generate the node؛

• **PATH-COST:** the cost, traditionally denoted by g(n), of the path from the initial state to the node, as indicated by the parent pointers; and

• **DEPTH:** the number of steps along the path from the initial state.

As usual, we differentiate between two main families of search strategies: systematic search and local search. Systematic search visits each state that could be a solution, or skips only states that are shown to be dominated by others, so it is always able to find an optimal solution.

5. Heuristic Search Algorithms

In this section, we can saw that many of the problems that fall within the purview of artificial intelligence are too complex to be solved by direct techniques; rather they must be attacked by appropriate search methods armed with whatever direct techniques are available to guide the search. These methods are all varieties of heuristic search.

They can be described independently any particular task or problem domain. But when applied to Particular problems, their efficacy is highly dependent on the way they exploit domain-specific knowledge since in and of themselves they are unable to overcome the combinatorial explosion to which search processes are so vulnerable. For this reason, these techniques are often called weak methods. Although a realization of the limited effectiveness of these weak methods to solve hard problems by themselves has been an important result that emerged from the last decades of AI research, these techniques continue to provide the framework into which domain-specific knowledge can be placed, either by hand or as a result of automatic learning.

Hill climbing is a variant of generate-and-test in which feedback from the test procedure is used to help the generator decide which direction to move in the search space. In a pure generate-and-test procedure, the test function responds with only a yes or no. but if the test function is augmented with a heuristic function that provide an estimate of how close a given is to a goal state. This is particularly nice because often the computation of the heuristic function can be done at almost no cost at the same time that the test for a solution is being performed. Hill climbing is often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available. For example, suppose you are in an unfamiliar city without a map and you want to get downtown. You simply aim for the tall buildings. The heuristic function is just distance between the current location and the location of the tall buildings and the desirable states are those in which this distance is minimized.

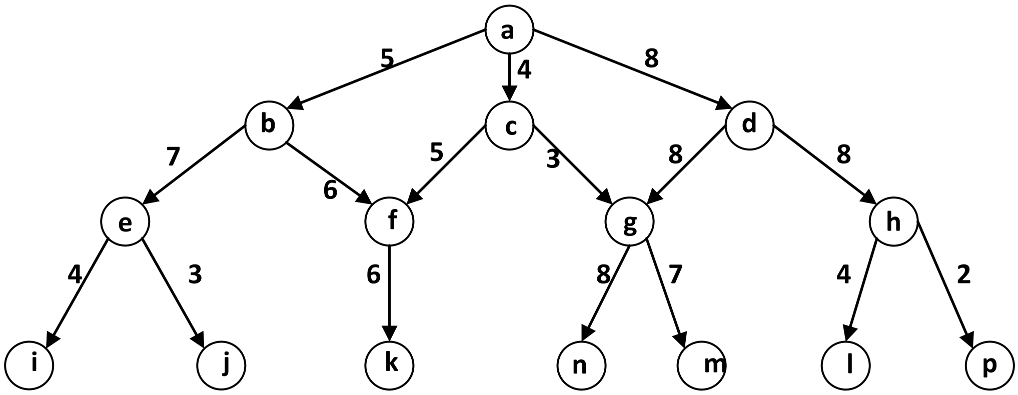
For each state f(n) = h(n) where h(n) is the heuristic function that computes the heuristic value for each state n.

Function Hill Climbing Search

```
Begin
 Open: = [Initial state];                              %initialize
 Closed: = [ ];
 CS= initial state;
 Path= [initial state];
 Stop= FALSE;
 While open <> [ ] do                                  %states remain
  Begin
   If CS=goal then return path
   Generate all children of CS and put them into open;
    If open= [ ] then
    Stop= TRUE
     Else
      Begin
       X= CS;
       For each state Y in open do
        Begin
         Compute the heuristic value of y (h(Y));
         If Y is better than X then
         X=Y
        End;
        If X is better than CS then
        CS=X
         Else
```

```
            Stop= TRUE;
        End;
    End;
    Return (FAIL);                          %open is empty
End.
```

Consider the following problem state space then:



Find the path from **a** to **m** using Hill Climbing search algorithm.

Open	**Closed**
[a]	[]
[b5, c4, d8]	[a]
[c4, b5, d8]	[a]
[f5, g3, b5, d8]	[a, c4]
[g3, f5, b5, d8]	[a, c4]
[n8, m7, f5, b5, d8]	[a, c4, g3]
[m7, n8, f5, b5, d8]	[a, c4, g3]

Stop the goal (m) is found

Now let us discuss a new heuristic method called "Best First Search", which is a way of combining the advantages of both depth-first and breadth-first search into a single method.

The actual operation of the algorithm is very simple. It proceeds in steps, expanding one node at each step, until it generates a node that corresponds to a goal state. At each step, it picks the most promising of the nodes that have so far been generated but not expanded. It generates the successors of the chosen node, applies the heuristic function to them, and adds them to the list of open nodes, after checking to see if any of them have been generated before. By doing this check, we can guarantee that each node only appears once in the graph, although many nodes may point to it as a successors. Then the next step begins.

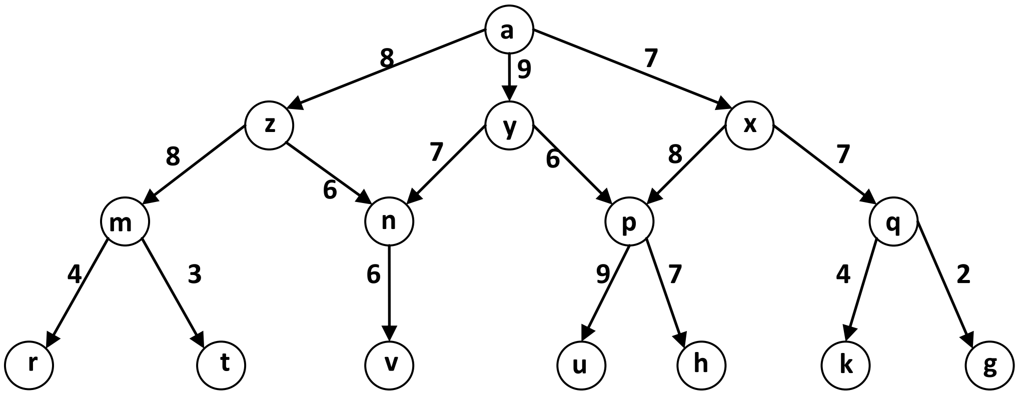
For each state f(n) = h(n) where h(n) is the heuristic function that computes the heuristic value for each state n.

Function Best-First Search

```
Begin
 Open: = [Initial state];                          %initialize
 Closed: = [ ];
 While open <> [ ] do                              %states remain
  Begin
   Remove leftmost state from open, call it X;
   If X = goal then return the path from initial to X
   Else
   Begin
    Generate children of X;
    For each child of X do
    Case
```

```
     The child is not on open or closed;
       Begin
         Assign the child a heuristic value;
         Add the child to open
       End;
     The child is already on open;
       If the child was reached by a shorter path
       Then give the state on open the shorter path
     The child is already on closed;
       If the child was reached by a shorter path then
         Begin
           Remove the state from closed;
           Add the child to open
         End;
   End;                                          %case
   Put X on closed;
   Re-order states on open by heuristic merit (best leftmost)
 End;
Return FAIL                                      %open is empty
End.
```

Consider the following problem state space then:



Find the path from **a** to **k** using Best first Search algorithm.

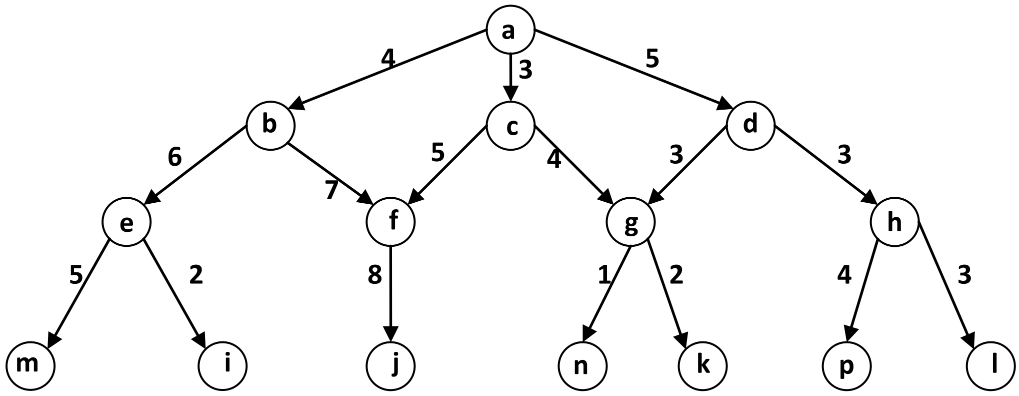
Open	**Closed**
[a]	[]
[z8, y9, x7]	[a]
[x7, z8, y9]	[a]
[p8, q7, z8, y9]	[a, x7]
[q7, p8, z8, y9]	[a, x7]
[k4, g2, p8, z8, y9]	[a, x7, q7]
[g2, k4, p8, z8, y9]	[a, x7, q7]
[k4, p8, z8, y9]	[a, x7, q7, g2]

The goal (k) is found

The first advance approach to the best first search is known as A-search algorithm. A algorithm is simply define as a best first search plus specific function. This specific function represent the actual distance (levels) between the initial state and the current state and is denoted by g(n). A notice will be mentioned here that the same steps that are used in the best first search are used in an A algorithm but in addition to the g(n) as follow;

<u>f(n) = h(n) + g(n) where h(n) is the heuristic function that computes the heuristic value for each state n, and g(n) is the generation function that computes the actual distance (levels) between initial state to current state n.</u>

Example:



Find the path from **a** to **k** using A-search algorithm

Open	**Closed**
[a]	[]
[b4, c3, d5]	[a]
[b4+1, c3+1, d5+1]	[a]
[c4, b5, d6]	[a]
[f5, g4, b5, d6]	[a, c4]
[f5+2, g4+2, b5, d6]	[a, c4]
[f7, g6, b5, d6]	[a, c4]
[b5, g6, d6,, f7]	[a, c4]
[e6, f7, g6, d6, f7]	[a, c4, b5]
[e6+2, f7+2, g6, d6, f7]	[a, c4, b5]
[g6, d6, f7, e8, f9]	[a, c4, b5]
[n1, k2, d6, f7, e8, f9]	[a, c4, b5, g6]
[n1+3, k2+3, d6, f7, e8, f9]	[a, c4, b5, g6]
[n4, k5, d6, f7, e8, f9]	[a, c4, b5, g6]
[k5, d6, f7, e8, f9]	[a, c4, b5, g6, n4]

Stop the goal (k) is found

The second advance approach to the best first search is known as A*-search algorithm. A* algorithm is simply define as a best first search plus specific function. This specific function represent the actual distance (levels) between the current state and the goal state and is denoted by g(n).

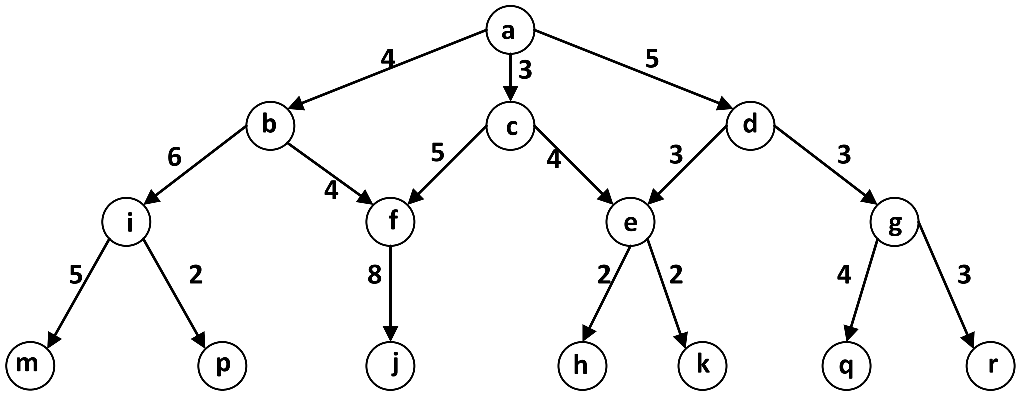
<u>f(n) = h(n) + g(n) where h(n) is the heuristic function that computes the heuristic value for each state n, and g(n) is the generation function that computes the actual distance (levels) between current state n to goal state.</u>

<u>Function A* Search Algorithm</u>

```
Begin
 Open: = [Initial state];                              %initialize
 Closed: = [ ];
 While open <> [ ] do                                  %states remain
  Begin
   Remove leftmost state from open, call it X;
   If X = goal then return the path from initial to X
   Else
   Begin
    Generate children of X;
    For each child of X do
     Begin
      Add the distance between current state to goal state to the heuristic
value for each child                                    %make the g(n)
    Case
      The child is not on open or closed;
```

```
      Begin
        Assign the child a heuristic value;
        Add the child to open
      End;
     The child is already on open;
      If the child was reached by a shorter path
      Then give the state on open the shorter path
     The child is already on closed;
      If the child was reached by a shorter path then
        Begin
          Remove the state from closed;
          Add the child to open
        End;
    End;                                            %case
    Put X on closed;
    Re-order states on open by heuristic merit (best leftmost)
  End;
Return FAIL                                       %open is empty
End.
```

Example:



Find the path from **a** to **k** using A*-search algorithm

Open	Closed
[a]	[ ]
[b4, c3, d5]	[a]
[b4+4, c3+2, d5+2]	[a]
[c5, d7, b8]	[a]
[f5, e4, d7, b8]	[a, c5]
[f5+3, e4+1, d7, b8]	[a, c5]
[e5, d7, f8, b8]	[a, c5]
[h2, k2, d7, f8, b8]	[a, c5, e5]
[h2+2, k2+0, d7, f8, b8]	[a, c5, e5]
[k2, h4, d7, f8, b8]	[a, c5, e5]

Stop, the goal (k) is found

Heuristic Search Methods with Heuristic Function

Hill climbing

For each state f(n) = h(n) where h(n) is the heuristic function that computes the heuristic value for each state n.

Best First Search

For each state f(n) = h(n) where h(n) is the heuristic function that computes the heuristic value for each state n.

A-search algorithm

f(n) = h(n) + g(n) where h(n) is the heuristic function that computes the heuristic value for each state n, and g(n) is the generation function that computes the actual distance (levels) between initial state to current state n.

A*-search algorithm

f(n) = h(n) + g(n) where h(n) is the heuristic function that computes the heuristic value for each state n, and g(n) is the generation function that computes the actual distance (levels) between current state n to goal state.

Problems with Hill Climbing Search Procedure

1- Fost Hill (Local Minima)

This problem causes stopping search procedure.

The algorithm not found the goal state although it is existed in the search space, this is because of the algorithm search performance and behavior which depends on a determined strategy without backing path from dead end state which causes algorithm termination, this problem can be solved by using backtracking process in the algorithm strategy.

2- Plateau Problem

This problem causes stopping search procedure.

When the search procedure reach to a state has an equivalent heuristic values (choices), the algorithm stops searching for the goal and not get the solution path although it is existed in the search space, in other words, there is a state has two or more children with the same heuristic value (Plateau partial search space), this problem can be solved by some kind of search procedures such as continuing search with the most left side.

3- Ridge Problem

This problem does not cause stopping search procedure.

The search procedure gets the solution path with some cost measurements which is not considered the best, since the best path is existed in dominate partial search space; this problem can be solved by applying more than one rule in each search procedure stage.

A Comparison between Heuristic Search and Blind Search

	Blind Search	**Heuristic Search**
1	In term of complexity: it is less complex.	In term of complexity: it is more complex.
2	In term of memory capacity: usually need more memory capacity.	In term of memory capacity: usually need less memory capacity.
3	In term of run time consuming: usually consumes more run time.	In term of run time consuming: usually consumes less run time.
4	Guarantee for solution.	Guarantee for solution, except Hill Climbing (not always).
5	Usually does not find the optimal solution path.	Usually finds the optimal solution path or nearly the optimal solution path.
6	It does not have a guider in search behavior.	It has a guider in search behavior (Heuristic Function).
7	It is not efficient in game playing.	It is efficient in game playing such as Minmax or Alpha-Beta procedures.

Using Heuristic in Games

The sliding-tile puzzle consists of three black tiles, three white tiles, and an empty space in the configuration shown in Figure (1). The puzzle has two legal moves with associated costs:

- A tile may move into an adjacent empty location. This has a cost of 1.
- A tile can hop over one or two other tiles into the empty' position, this has a cost equal to the number of tiles jumped over.

The goal is to have all the white tiles to the left of all the black tiles. The position of the blank is not important.

a. Analyze the state space with respect to complexity and looping.

b. Propose a heuristic for solving this problem and analyze it.

B	B	B		W	W	W

Figure (5), the sliding block puzzle

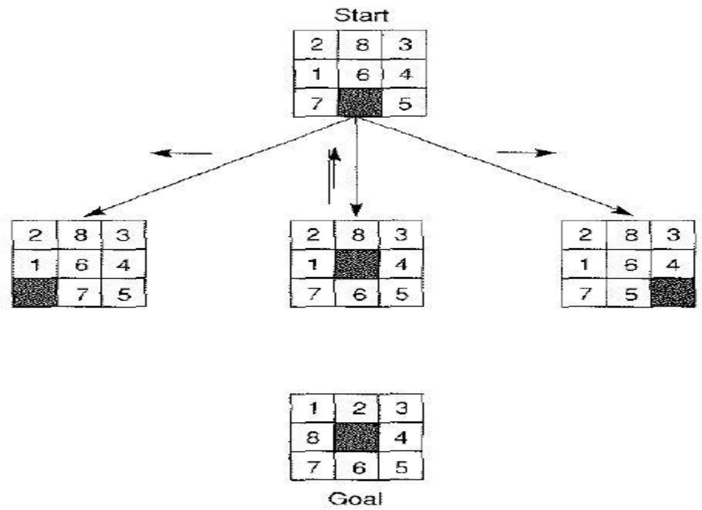
The 8-puzzle Problem

We now evaluate the performance of several different heuristics for solving the 8-puzzle. Figure (6), shows a start and goal state for the 8-puzzle, along with the first three states generated in the search.

The simplest heuristic counts the tiles out of place in each state when it is compared with the goal. This is intuitively appealing, because it would seem that, all else being equal; the state that had fewest tiles out of place is probably closer to the desired goal and would be the best to examine next.

However, this heuristic does not use all of the information available in a board configuration, because it does not take into account the distance the tiles must be moved.

A "better" heuristic would sum all the distances by which the tiles are out of place, one for each square a tile must be moved to reach its position in the goal state. Both of these heuristics can be criticized for failing to acknowledge the difficulty of tile reversals. That is, if two tiles are next to each other and the goal requires their being in apposite locations, it takes (many) more than two moves to put them back in place, as the tiles must "go around" each other (Figure 7).



Figure(6), The start state, first moves, and goal state for an example 8-puzzle.

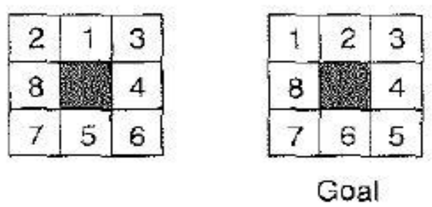


Figure (7) An 8-puzzle state with a goal and two reversals: 1 and 2, 5 and 6.

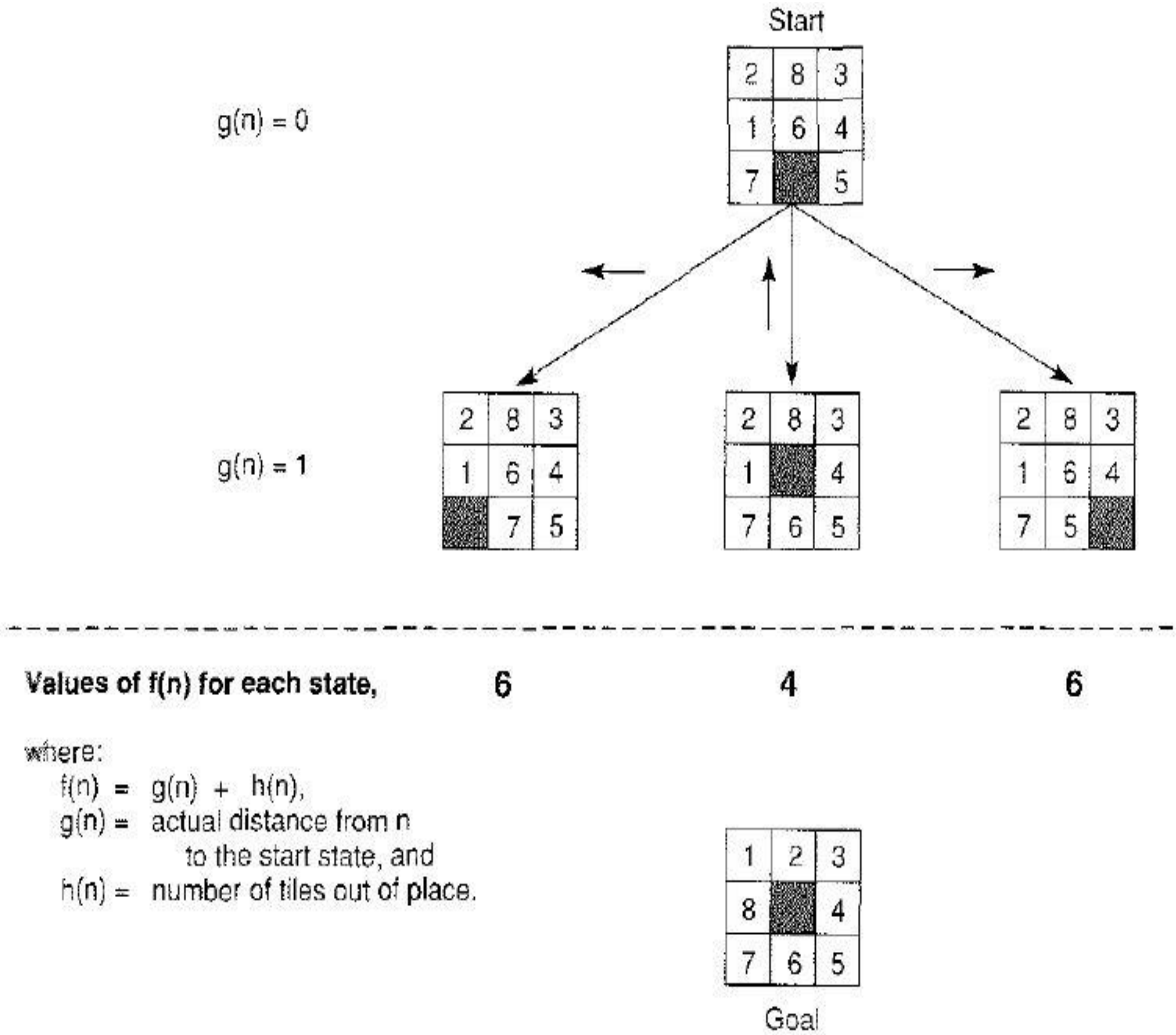


Figure (8), the 8-puzzle problem solving with heuristic values

<u>For the 8-puzzle Grid</u>

There is one center location.

There are four corners location.

There are four sides location.

<u>Possible Moves</u>

- *When the space position is in the center of the grid, possible moves = 4.*
- *When the space position is in the corner of the grid, possible moves = 2*
- *When the space position is in the side of the grid, possible moves = 3.*

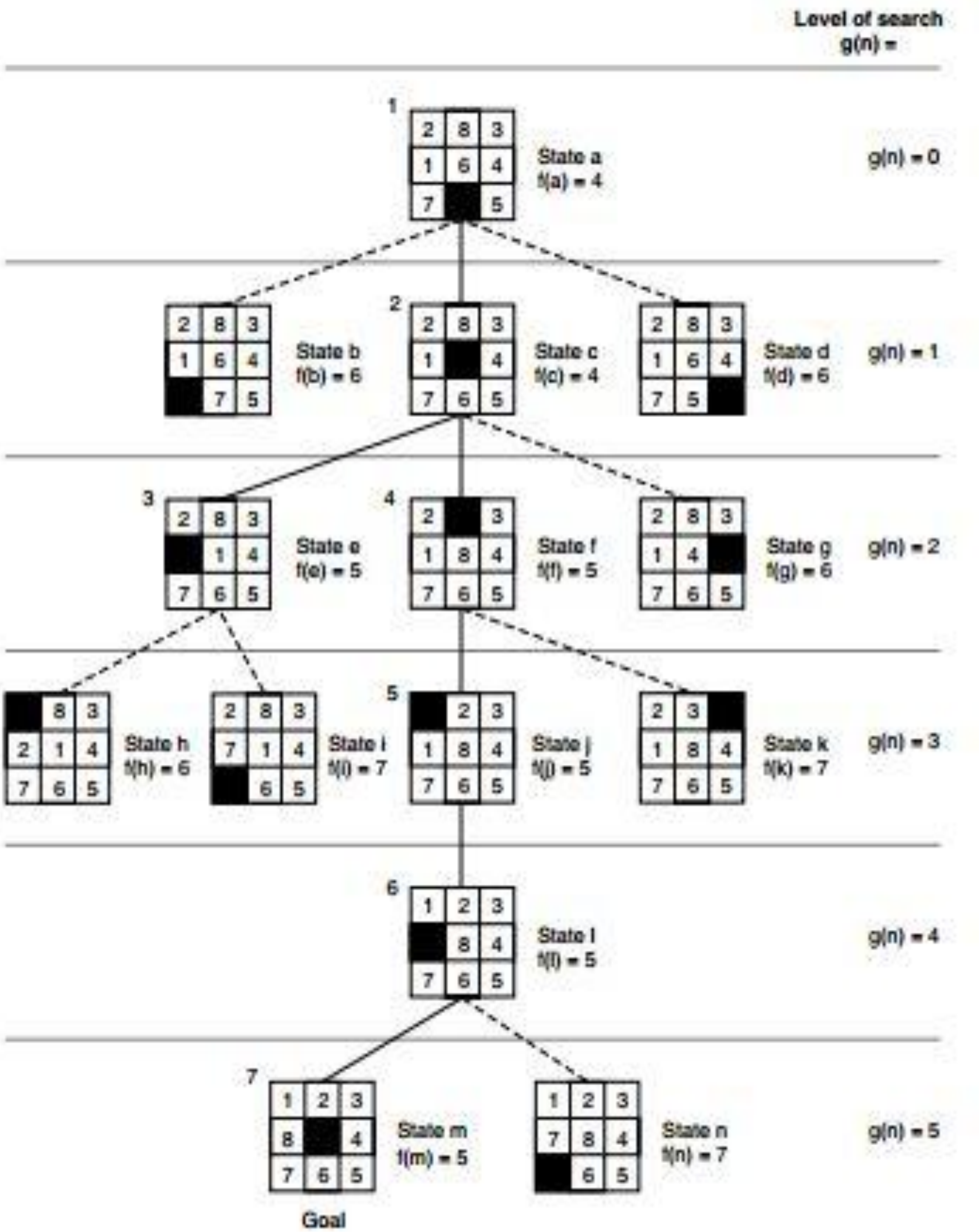
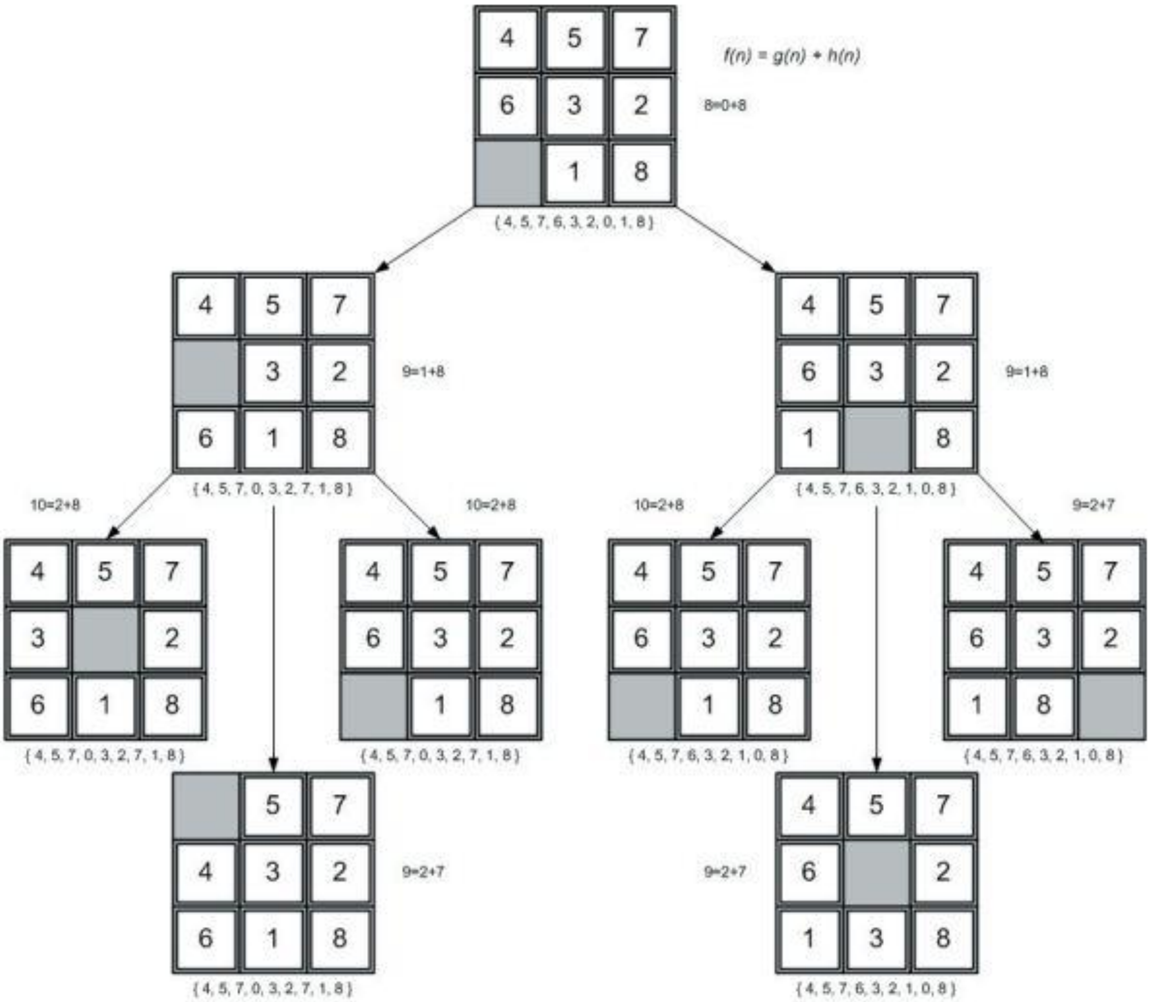


Figure (9), the 8-puzzle problem solved by A-search algorithm

Another Examples of 8-Puzzle Problem



Consider the following **8-puzzle** problem then draw the problem state space to find the goal using A-search algorithm (or Best first or Hill climbing) then list the *solution path.*

	R	I
A	O	O
P	L	G

I.S

A		I
P	R	O
L	O	G

G.S

Consider the following 8-tiles problem then draw the problem state space to give the following requirements:

A	B	
B	C	A
C	C	B

I.S

A		A
B	B	B
C	C	C

G.S

Find the goal using Best first search (or A-algorithm or Hill climbing) algorithm

---

The Minmax and Alpha-Beta search Algorithms

The idea for alpha-beta search is simple: rather than searching the entire space to the ply depth, alpha-beta search proceeds in a depth-first fashion. Two values, called alpha and beta, are created during the search. The alpha value associated with MAX nodes, can never decrease, and the beta value associated with MIN nodes, can never increase. Two rules for terminating search, based on alpha and beta values, are:

1. Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors.

2. Search can be stopped below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN node ancestors.

Alpha-beta pruning thus expresses a relation between nodes at ply n and nodes at ply n + 2 under which entire sub-trees rooted at level n + 1 can be eliminated from consideration. Note that the resulting backed-up value is identical to the minimax result and the search saving over minimax is considerable. With fortuitous ordering states in the search space, alpha-beta can effectively double the depth of the search considered with a fixed space/time computer commitment. If there is a particular unfortunate ordering, alpha-beta searches no more of the space than normal minimax; however, the search is done in only one pass.

c(n) = M(n) – O(n) Where M(n) = number of my possible winning lines.

Now, we will discuss a new type of algorithm, which does not require expansion of the entire space exhaustively. This algorithm is referred to as alpha-beta cutoff algorithm. In this algorithm, two extra ply of movements are considered to select the current move from alternatives. Alpha and beta denote two cutoff levels associated with MAX and MIN nodes. As it is mentioned before the alpha value of MAX node cannot decrease, whereas the beta value of the MIN nodes cannot increase. But how can we compute the alpha and beta values? They are the backed up values up to the root like MINIMAX. There are a few interesting points that may be explored at this stage. Prior to the process of computing MAX / MIN of the backed up values of the children, the alpha-beta cutoff algorithm estimates e(n) at' all fringe nodes n. Now, the values are estimated following the MINIMAX algorithm. Now, to prune the unnecessary paths below a node, check whether:

• The beta value of any MIN node below a MAX node is less than or equal to its alpha value. If yes. prune that path below the MIN node.

• The alpha value of any MAX node below a MIN node exceeds the beta value of the MIN node. if yes prune the nodes below the MAX node.

Based on the above discussion, we now present the main steps in the α-β search algorithm.

1. Create a new node, if it is the beginning move, c1seexpand the existing tree by depth first manner. To make a decision about the selection of a move at depth d, the tree should be expanded at least up to a depth (d+ 2).

2. Compute e(n) for all leave (fringe) nodes n in the tree.

3. Computer αmin (for max nodes) and βmax values (for min nodes) at the ancestors of the fringe nodes by the following guidelines. Estimate the minimum of the values (e or α) possessed by the children of a MINIMIZER node N and assign it its βmax value. Similarly, estimate the maximum of the values (e or β) possessed by the children of a MAXIMIZER node N and assign it its αmin value.

4. If the MAXIMIZER nodes already possess αmin values, then their current αmin value = Max (αmin value, αmin,); on tile other hand, if the MANIMIZER nodes already possess βmax values, then their current βmax value = MIN (βmax value, βmax).

5. If the estimated βmax value of a MINIMIZER node N is less than the αmin value of its parent MAXIMIZER node N' then there is no need to search below the node MINIMIZER node N. Similarly, if the αmin value of a MAXIMIZER node N is more than the βmax value of its parent node N then there is no need to search below node N.

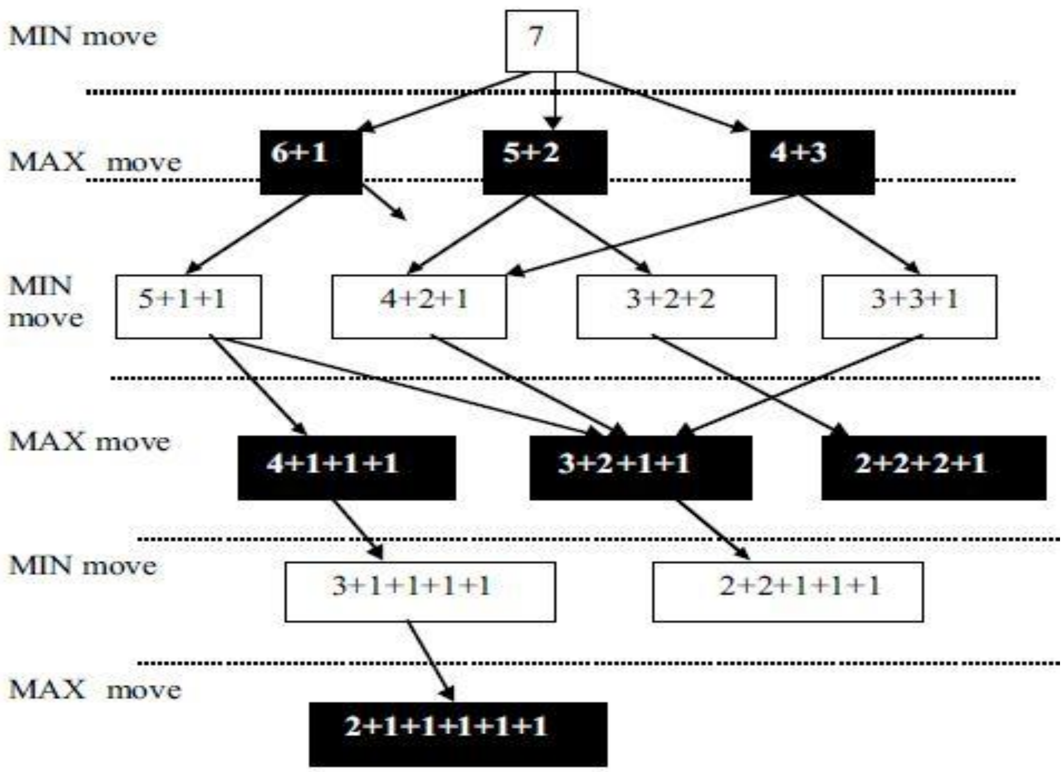


Figure (10), state space for the minmax game

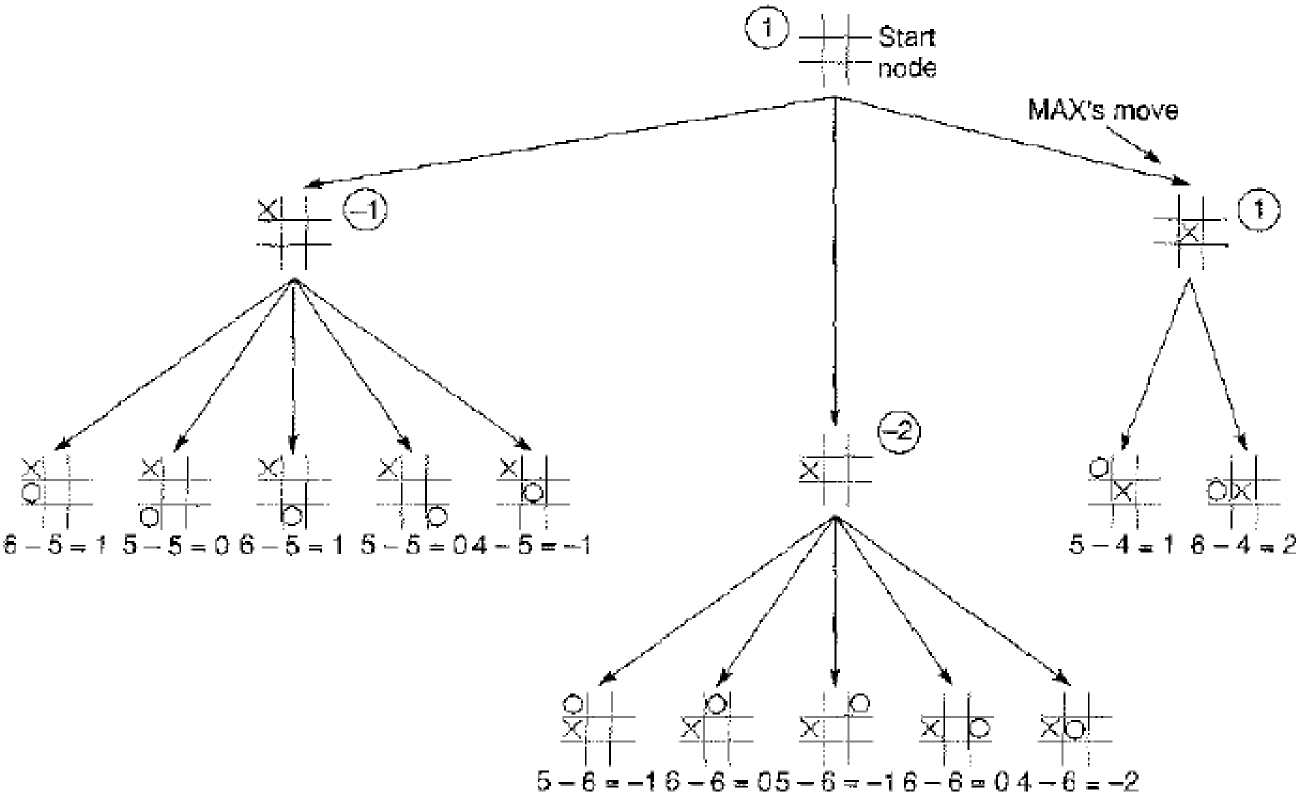


Figure (11), the Tic Tac Toe state space for the Alpha-Beta procedure

In heuristic Search, two aims must be achieved to overcome the limitations in other search methods which are:

1- Problem Reduction

2- Guarantee of Solution

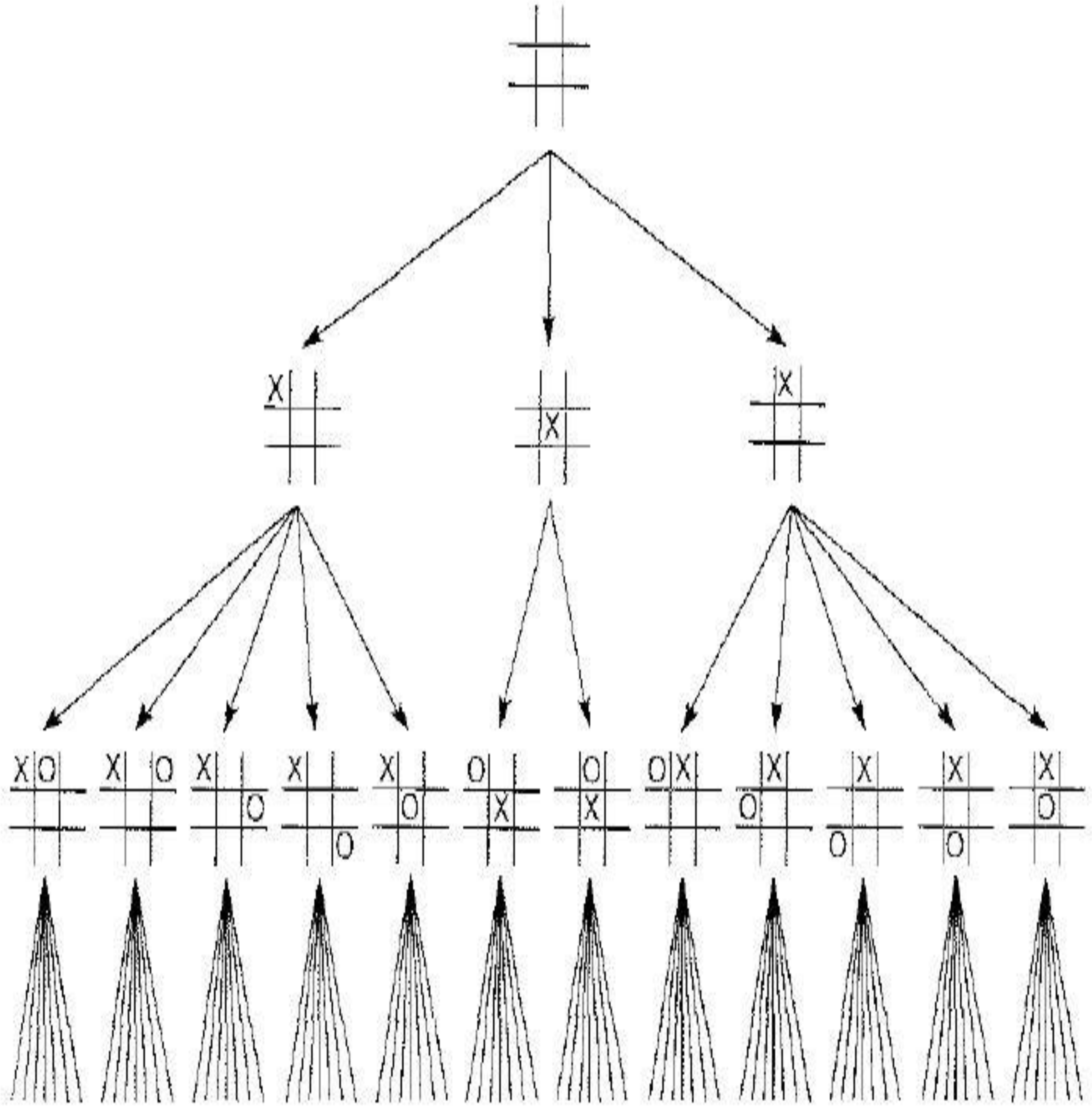


Figure (12), first three levels of the tic-rae-toe state space reduced by symmetry

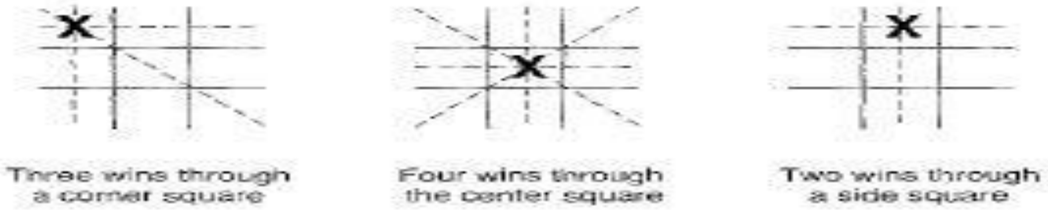


Figure The "most wins" heuristic applied to the first children in tic-tac-toe.

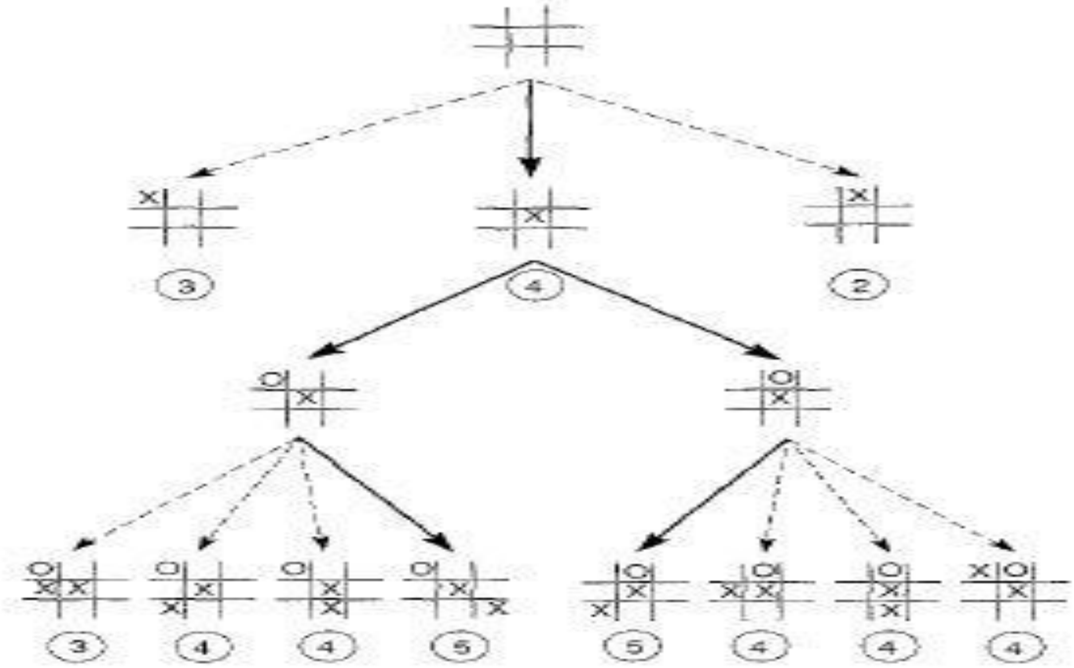


Figure (13), heuristically reduced state space for tic-tao-toe.

Systems Based on Simple Search and Pattern Recognition (1)

Text Recognition System (An AI Program for Psychology Counseling)

Pattern in Sentence

From human	Machine (Computer) Responds
…..I fell…..	How long have you felt that way?
….. father …..	Tell me more about your family.
…..friendly…..	Better friends help you when you need.
…..hell…..	Please do not use words like that.
…..yes or no….	Please be more explicit.
…..high performance…..	This gives you additional respecting.
…..sadness or anger…..	How are you feeling right now?
…..complex or fixation…..	Too many mind games
……..father……….	Earlier you mentioned your father.
……………………..	……………………………………

```
clauses
helpme :-
        write("Speak up, what is your problem?"), nl,
        repeat,
        getclause(L),
        makeans(L),
        fail.

getclause(L) :-
                readln(S),
                str_to_list(S,L).
```

```
makeans(L) :-
        recognize(L,1),
        write ("How long have you felt that way?"), nl, !.
makeans(L) :-
        recognize(L,2),
        write ("Tell me more about your family"), nl, !.
makeans(L) :-
        recognize(L,3),
        write ("better friends help you when you need."), nl, !.
makeans(L) :-
        recognize(L,4),
        write ("Please do not use words like that."), nl, !.
makeans(L) :-
        recognize(L,5),
        write ("Please be more explicit."), nl, !.
makeans(L) :-
        recognize(L,6),
        write ("This gives you additional respecting."), nl, !.
makeans(L) :-
        recognize(L, 7),
        write ("How are you felling right now?"), nl, !.
makeans(L) :-
        recognize(L,8),
        write ("Too many mind games."), nl, !.
makeans(L) :-
        recognize(L,9),
        write ("Earlier you mentioned your father."), nl, !.
```

```
makeans(L) :-
        recognize(L,10), write ("Tell me more."), nl, !.

recognize(L, 1) :- contains([i, feel], L).
recognize(L, 2) :- contains([father], L) assert(father).
recognize(L, 3) :- contains([friendly], L)
recognize(L, 4) :- contains([hell], L).
recognize(L, 5) :- L= [yes]; L=[no].
recognize(L, 6) :- contains([high, performance], L).
recognize(L, 7) :- contains([sadness], L); contains([anger], L).
recognize(L, 8) :- contains([complex], L) ; contains([fixation], L).
recognize(L, 9) :- father.
recognize(_, 10).
```

Systems Based on Heuristic Search and Pattern Recognition (2)

The Chemical Synthesis System

```
domains
rxnlist = reactions*.
reactions = rxn(symbol, ls, integer, integer).
ls = symbol*.
chemicalList= chemicalForm*.
chemicalForm= chemical(symbol, rxnList, integer, integer).
Li= integer*.

predicates
rxn(symbol, ls, integer, integer).
rawmaterial(symbol, integer, integer).
chemical(symbol, rxnlist, integer, integer).
all_chemical(symbol, chemicalList).
best_chemical(symbol, chemicalForm).
one_chemical(symbol, chemicalForm).
append(rxnlist, rxnlist, rxnlist).
min(chemicalList, chemicalForm).
run(symbol).

clauses
rxn(a, [b1, c1], 12, 60).
rxn(b1, [d1, e1], 5, 45).
rxn(c1, [f1, g1], 3, 15).
```

```
rxn(a, [b2, c2], 10, 50).
rxn(b2, [d2, e2], 2, 20).
rxn(c2, [f2, g2], 6, 30).

rawmaterial(d1, 2, 0).
rawmaterial(e1, 0, 0).
rawmaterial(f1, 2, 0).
rawmaterial(g1, 0, 0).

rawmaterial(d2, 0, 0).
rawmaterial(e2, 1, 0).
rawmaterial(f2, 1, 0).
rawmaterial(g2, 0, 0).

chemical(Y, [], Cost, Time):- rawmaterial(Y, Cost, Time).
chemical(Y, L, Ct, T):-
        rxn(Y, [X1, X2], C, T1),
        chemical(X1, L1, C1, T2),
        chemical(X2, L2, C2, T3),
        append(L1, L2, Q),
        Ct = C+C1+C2,
        T = T+T2+T3,
        append([rxn(Y, [X1, X2], C, T1)], Q, L).

best_chemical(Y, M):- all_chemical(Y, X), min(X, M).


all_ chemical(Y, X):- findall(S, one_chemical(Y, S), X).


one_chemical(Y, chemical(Y, L, Ct, T)):- chemical(Y, L, Ct, T).
```

```
append([], L, L):-!.
append([H|T], L, [H|T1]):- append(T, L, T1).

min([chemical(Y, L, Ct, T)], chemical(Y, L, Ct, T)).
min([chemical(Y, L, Ct, Time)|T], chemical(Y, L, Ct, Time)):-
                        min(T, chemical(Y1, L1, C1, Time1)), Ct <= C1.
min([chemical(Y, L, Ct, Time)|T], chemical(Y, L2, Ct2, Time2)):-
                        min(T, chemical(Y, L2, Ct2, Time2)), Ct2 <= Ct.

run(X):- write(" chemical synthesis is:"), nl, chemical(X, L, Cost, Time),
        write(L, "\n with total cost =", Cost, " Time =", Time), nl, fail.
run(X):- write("\n Best chemical synthesis:"), nl, best_chemical(X, Y),
write(Y), nl.
```

Goal: run(a).
chemical synthesis:
[rxn("a", ["b1", "c1"], 12, 60), rxn("b1", ["d1", "e1"], 5, 45), rxn("c1", ["f1", "g1"], 3, 15)]
with total cost = 24 time = 120
[rxn("a", ["b2", "c2"], 10, 50), rxn("b2", ["d2", "e2"], 2, 20), rxn("c2", ["f2", "g2"], 6, 30)]
with total cost = 20 time = 100
best chemical synthesis :
chemical("a", [rxn("a", ["b2", "c2"], 10, 50) rxn("b2", ["d2", "e2"], 2, 20), rxn("c2", ["f2", "g2"], 6, 30)], 20, 100

Search with Heuristic Embedded in Rules

Student Advisor System

/* Set of Facts */

given_now(logic design).

given_now(Mathematics).

given_now(prolog language).

given_now(computation theory).

given_now(data structure).

given_now(artificial intelligence).

given_now(expert systems).

given_now(computation theory).

given_now(computer architecture).

.
.
.

required(prolog).

required(logic design).

required(artificial intelligence).

required(expert systems).

required(machine learning).

required(data structure).

required(c++).

.
.
.

elective(computer graphics).

elective(object oriented programming).

elective(data security).

elective(web programming).

elective(operations researches).

.
.
.

waived(digital signal processing).

waived(image processing).

waived(information systems principles).

waived(software engineering).

waived(data hiding).

.
.
.

impreq(object oriented programming, c++).

impreq(prolog language, logic design).

impreq(artificial intelligence, prolog language).

impreq(expert systems, artificial intelligence).

impreq(computer architecture, logic design).

impreq(data structure, c++).

.
.
.

passed(logic design).

passed(prolog language).

passed(artificial intelligence).

passed(mathematics).

passed(data structure).

passed(c++).

```
passed(computation theory).
passed(computer organization).
.
.
.

pos_req_course(X) :-
                required(X),
                given_now(X),
                not(done_with(X)),
                have_preq_for(X).

pos_elec_course(X) :-
                elective(X),
                given_now(X),
                not(done_with(X)),
                have_preq_for(X).

done_with(X) :- waived(X).
done_with(X) :- passed(X).

all_done_with(L) :- findall(X, done_with(X), L).

have_preq_for(X) :-
                all_preq_for(X, Z),
                all_done_with(Q),
                subset(Z, Q).
```

```
all_preq_for(X, Z):-
            findall(Y, preq(X, Y), Z).

preq(X,Y):- impreq(X, Y).
preq(X, Y):- impreq(X, W), preq(W, Y).
```

Knowledge Engineering, Acquisition and Discovery

1. Domain Expert and Knowledge Engineering

The primary people involved in building an expert system are the *knowledge engineer*, the *domain expert*, and the *end user*. The knowledge engineer is the AI language and representation expert. His or her main task is to select the software and hardware tools for the project, help the domain expert articulate the necessary knowledge, and implement that knowledge in a correct and efficient knowledge base. Often, the knowledge engineer is initially ignorant of the application domain. The domain expert provides the knowledge of the problem area. The domain expert is

generally someone who has worked in the domain area and understands its problem solving techniques, such as shortcuts, handling imprecise data, evaluating partial solutions, and all the other skills that mark a person as an expert problem solver. The domain expert is primarily responsible for spelling out these skills to the knowledge engineer. Once the knowledge engineer has obtained a general overview of the problem domain and gone through several problem-solving sessions with the expert, he or she is ready to begin actual design of the system: selecting a way to represent the knowledge, such as rules or frames, determining the search strategy, forward, backward, depth-first, best-first etc., and designing the user interface. After making these design commitments, the knowledge engineer builds a prototype.

This prototype should be able to solve problems in a small area of the domain and provide a test bed for preliminary design assumptions. Once the prototype has been implemented, the knowledge engineer and domain expert test and refine its knowledge by giving it problems to solve and correcting its shortcomings. Should the assumptions made in designing the prototype prove correct; the prototype can be incrementally extended until it becomes a final system**?**

2. Selecting a Problem and the Knowledge Engineering Process

Expert systems involve a considerable investment of cost and human effort. Attempts to solve a problem that is too complex, too poorly understood, or otherwise unsuited to the available technology can lead to costly and embarrassing failures. Researchers have developed guidelines to determine whether a problem is appropriate for expert system solution:

1. ***The need for the solution justifies the cost and effort of building an expert system***. Many expert systems have been built in domains such as mineral exploration, business, defense, and medicine where a large potential exists for savings in terms of cost, time, and human life.

2. ***Human expertise is not available in all situations where it is needed*****.** In geology, for example, there is a need for expertise at remote mining and drilling sites. Often, geologists and other engineers find themselves traveling large distances to visit sites, with resulting expense and wasted time. By placing expert systems at remote sites, many problems may be solved without needing a visit.

3. ***The problem may be solved using symbolic reasoning***. Problem solutions should not require physical dexterity or perceptual skill. Robots and vision systems currently lack the sophistication and flexibility of humans.

4. ***The problem domain is well structured and does not require common sense reasoning.*** Highly technical fields have the advantage of being well studied and formalized: terms are well defined and domains have clear and specific conceptual models. In contrast, common sense reasoning is difficult to automate.

5. ***The problem may not be solved using traditional computing methods***. Expert system technology should not be used where unnecessary. If a problem can be solved satisfactorily using more traditional techniques, then it is not a candidate.

6. ***Cooperative and articulate experts exist.*** The knowledge used by expert systems comes from the experience and judgment of humans working in the domain. It is important that these experts be both willing and able to share knowledge.

7. ***The problem is of proper size and scope***. For example, a program that attempted to capture all of the expertise of a medical doctor would not be feasible; a program that advised MDs on the use of a particular piece of diagnostic equipment or a particular set of diagnoses would be more appropriate.

3. Knowledge Acquisition in Computing Approach

Knowledge acquisition is a pertinent issue in the process of development of expert systems. A good expert system should contain a well-organized, complete and consistent knowledge base. An incomplete or inconsistent knowledge base may cause instability in reasoning, while a less organized system requires quite a significant time for search and matching of data. The malfunctioning of the above forms originates in an expert system generally due to the imperfections in

i) the input resources of knowledge

ii) their encoding in programs.

The imperfection in the input resources of knowledge can be overcome by consulting proved knowledge-rich sources, such as textbooks and experts of respective domains. The encoding of knowledge could be erroneous due to either incorrect understanding of the pieces of knowledge or their semantic misinterpretation in programs. A knowledge engineer, generally, is responsible for acquiring knowledge and its encoding.

Understanding knowledge from experts or textbooks, therefore, is part of his duties. A clear understanding of the knowledge base, however, requires identification of specific knowledge from a long narration of the experts. The knowledge engineer, who generally puts objective questions to the expert, therefore, should allow the expert to answer them in sufficient detail, explaining the points. The semantic knowledge earned from the experts could be noted point-wise for subsequent encoding in programs. Occasionally, the experts too are not free from bias. One way to make the knowledge base bias-free is to consult a number of experts of the same problem domain and take the view of the majority of the members as the acquired knowledge.

4. Knowledge Discovery

To formalize the knowledge discovery processes (KDPs) within a common framework, we introduce the concept of a **process model**. The model helps organizations to better understand the KDP and provides a roadmap to follow while planning and executing the project. This in turn results in cost and time savings, better understanding, and acceptance of the results of such projects. We need to understand that such processes are non trivial and involve multiple steps, reviews of partial results, possibly several iterations, and interactions with the data owners. There are several reasons to structure a KDP as a **standardized process model**:

1. ***The end product must be useful for the user/owner of the data*.** A blind, unstructured application of DM techniques to input data, called *data dredging*, frequently produces meaningless results/knowledge, i.e., knowledge that, while interesting, does not contribute to solving the user's problem. This result ultimately leads to the failure of the project. Only through the application of well-defined KDP models will the end product be valid, novel, useful, and understandable.

2. ***A well-defined KDP model should have a logical, cohesive, well-thought-out structure and approach that can be presented to decision-makers who may have difficulty understanding the need, value, and mechanics behind a KDP.*** Humans often fail to grasp the potential knowledge available in large amounts of untapped and possibly valuable data. They often do not want to devote significant time and resources to the pursuit of formal methods of knowledge extraction from the data, but rather prefer to rely heavily on the skills and experience of others (domain experts) as their source of information. However, because they are typically ultimately responsible for the decision(s) based on that information, they frequently want to understand the technology applied to that solution. A process model that is well structured and logical will do much to alleviate any misgivings they may have.

3. ***Knowledge discovery projects require a significant project management effort that needs to be grounded in a solid framework.*** Most knowledge discovery projects involve teamwork and thus require careful planning and scheduling. For most project management specialists, KDP and DM are not familiar terms. Therefore, these specialists need a definition of what such projects involve and how to carry them out in order to develop a sound project schedule.

4. ***Knowledge discovery should follow the example of other engineering disciplines that already have established models.*** A good example is the software engineering field, which is new and dynamic discipline that exhibits many characteristics that are pertinent to knowledge discovery.

Software engineering has adopted several development models, including waterfall and spiral models that become well-known standards.

5. *There is a widely recognized need for standardization of the KDP*. The challenge for modern data miners is to come up with widely accepted standards that will stimulate major industry growth. Standardization of the KDP model would enable the development of standardized methods and procedures, thereby enabling end users to deploy their projects more easily. It would lead directly to project performance that is faster, cheaper, more reliable, and more manageable. The standards would promote the development and delivery of solutions that use business terminology rather than the traditional language of algorithms, matrices, criterions, complexities, and the like, resulting in greater exposure and acceptability for the knowledge discovery field.

Below we define the KDP and its relevant terminology. We also provide a description of several key KDP models, discuss their applications, and make comparisons, the reader will know how to structure, plan, and execute a (successful) KD project.

5. What is the Knowledge Discovery Process?

Because there is some confusion about the terms data mining, knowledge discovery, and knowledge discovery in databases, we first define them. Note, however, that many researchers and practitioners use DM as a synonym for knowledge discovery; DM is also just one step of the KDP.

Let us just add here that DM is also known under many other names, including *knowledge extraction*, *information discovery*, *information harvesting*, *data archeology*, and *data pattern processing*.

The **knowledge discovery process** (KDP), also called knowledge discovery in databases, seeks new knowledge in some application domain. It is defined as the nontrivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. The process generalizes to non database sources of data, although it emphasizes databases as a primary source of data. It consists of many steps (one of them is DM), each attempting to complete a particular discovery task and each accomplished by the application of a discovery method. Knowledge discovery concerns the entire knowledge extraction process, including how data are stored and accessed, how to use efficient and scalable algorithms to analyze massive datasets, how to interpret and visualize the results, and how to model and support the interaction between human and machine. It also concerns support for learning and analyzing the application domain.

This book defines the term **knowledge extraction** in a narrow sense. While the authors acknowledge that extracting knowledge from data can be accomplished through a variety of methods — some not even requiring the use of a computer — this book uses the term to refer to knowledge obtained from a database or from textual data via the knowledge discovery process. Uses of the term outside this context will be identified as such.

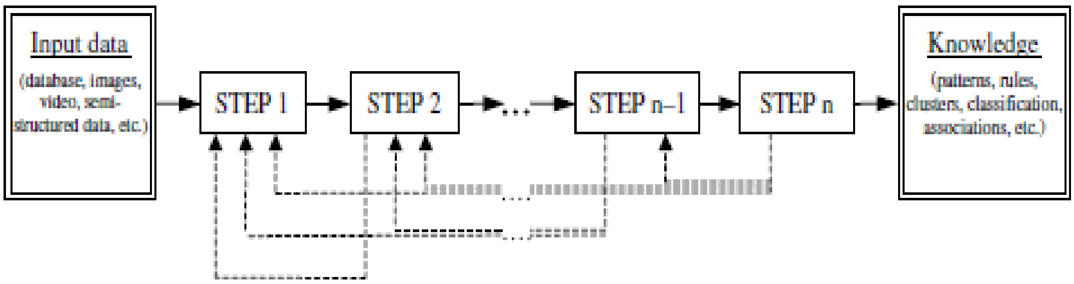


Figure (1), sequential structure of the KDP model.

6. Overview of the Knowledge Discovery Process

The KDP model consists of a set of processing steps to be followed by practitioners when executing a knowledge discovery project. The model describes procedures that are performed in each of its steps. It is primarily used to plan, work through, and reduce the cost of any given project.

Since the 1990s, several different KDPs have been developed. The initial efforts were led by academic research but were quickly followed by industry. The first basic structure of the model was proposed by Fayyad et al. and later improved/modified by others. The process consists of multiple steps, that are executed in a sequence. Each subsequent step is initiated upon successful completion of the previous step, and requires the result generated by the previous step as its input. Another common feature of the proposed models is the range of activities covered, which stretches from the task of understanding the project domain and data, through data preparation and analysis, to evaluation, understanding, and application of the generated results. All the proposed models also emphasize the iterative nature of the model, in terms of many feedback loops that are triggered by a revision process. A schematic diagram is shown in Figure (1).

The main differences between the models described here lie in the number and scope of their specific steps. A common feature of all models is the definition of inputs and outputs. Typical inputs include data in various formats, such as numerical and nominal data stored in databases or flat files; images; video; semi-structured data, such as XML or HTML; etc. The output is the generated new knowledge — usually described in terms of rules, patterns, classification models, associations, trends, statistical analysis, etc.

7. Knowledge Discovery Process Models

Although the models usually emphasize independence from specific applications and tools, they can be broadly divided into those that take into account industrial issues and those that do not.

However, the academic models, which usually are not concerned with industrial issues, can be made applicable relatively easily in the industrial setting and vice versa. We restrict our discussion to those models that have been popularized in the literature and have been used in real knowledge discovery projects.

The Fayyad et al. KDP model consists of nine steps, which are outlined as follows:

1. ***Developing and understanding the application domain*****.** This step includes learning the relevant prior knowledge and the goals of the end user of the discovered knowledge.

2. ***Creating a target data set*****.** Here the data miner selects a subset of variables (attributes) and data points (examples) that will be used to perform discovery tasks. This step usually includes querying the existing data to select the desired subset.

3. ***Data cleaning and preprocessing*****.** This step consists of removing outliers, dealing with noise and missing values in the data, and accounting for time sequence information and known changes.

4. ***Data reduction and projection*****.** This step consists of finding useful attributes by applying dimension reduction and transformation methods, and finding invariant representation of the data.

5. ***Choosing the data mining t*****ask.** Here the data miner matches the goals defined in Step 1 with a particular DM method, such as classification, regression, clustering, etc.

6. ***Choosing the data mining al*****gorithm.** The data miner selects methods to search for patterns in the data and decides which models and parameters of the methods used may be appropriate.

7. ***Data mining*****.** This step generates patterns in a particular representational form, such as classification rules, decision trees, regression models, trends, etc.

8. ***Interpreting mined patterns*****.** Here the analyst performs visualization of the extracted patterns and models, and visualization of the data based on the extracted models.

9. ***Consolidating discovered knowledge*****.** The final step consists of incorporating the discovered knowledge into the performance system, and documenting and reporting it to the interested parties. This step may also include checking and resolving potential conflicts with previously believed knowledge.

8. Machine Learning Approach to Knowledge Acquisition

Manual acquisition of knowledge is difficult for two main reasons. *First* the knowledge engineer has to remain in constant touch with the experts for a significant amount of time, which sometimes may be of the order of years. *Secondly*, the experts themselves in many cases cannot formally present the knowledge. The above difficulties in acquisition of knowledge can, however, be overcome by autonomously encoding knowledge through machine learning. The schematic view for elicitation of knowledge by the machine learning
approach is presented in figure (2).

The database in figure (2) is extracted from experts or other reasoning systems. The machine learning unit grabs these data and attempts to acquire new knowledge out of it. The acquired knowledge is then transferred to the knowledge base for future usage. In some systems, the knowledge base need not be extended, but may be refined with respect to its internal parameters. For instance, certainty factor of the rules in a knowledge base may be refined based on the estimated certainty factors of proven case histories.

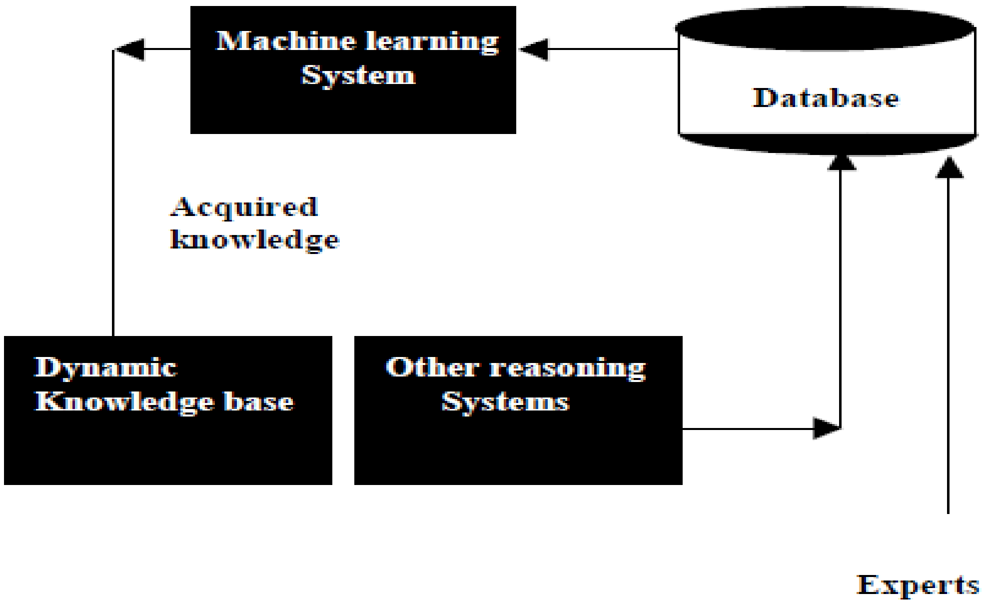


Figure (2), principles of automated knowledge acquisition.

AI Branch 3rd Class 1st Course Study Questions

Q1) What heuristics would you use in solving these problems?

1. You are looking for a parking space in a moderately crowded parking lot.
2. You think a particular radio show you want to hear is on now, but you do not know where it is on the dial, and you have no other guidance such as a newspaper listing.
3. You are in a large office building. You are lost, and you want to find the personal office, but you are embarrassed to ask where it is.

Q2) Write a program as searching with heuristics embedded in rules approach to construct optimal restaurant menus that follows the pattern of Student Advisor System.

Q3) The chemical synthesis program currently works with reactions like this:

X + y --→ z ……………..with cost (c)

1. How would things have to be modified so that reactions like this one could be included in the reaction data base that the program knows about? r --→ s ..…..with cost (c)
   This is anticipating the type of reaction where you treat a chemical in a certain way (heating perhaps) and it turns into something else.
2. How would things have to be modified so that reactions like this one could be included? q + r + s --→ w …….with cost (c)
3. What modification would be necessary for the program to carry along two costs with each synthesis: One might be the reaction cost and the other the length of time the reaction took to complete.
4. What modification would be necessary for the program to include a function that carries the best synthesis among many syntheses?

Q4) Compare between knowledge engineering and knowledge acquisition processes in domain expert environment.

Q5) What are the relations among the Knowledge Discovery, Knowledge Acquisition and Knowledge Engineering?

References:

1. George F. Luger, "***Artificial Intelligence Structures and Strategies for Complex Problem Solving"***, Pearson Education Asia (Singapore), Sixth edition.
2. Stuart J. Russell and Peter Norvig, ***"Artificial Intelligence A Modern Approach"***, Second Edition, Prentice Hall.
3. Amit Konar, ***"Artificial Intelligence and Soft Computing"***, Behavior and Cognitive Modeling of the Human Brian, CRC Press.
4. Daniel H. .Marcellus,***" Expert Systems Programming in Turbo Prolog"***, prentice Hall (New Jersey).