UNIT II

2. SEARCHING TECHNIQUES

Informed Search and Exploration, Informed (Heuristic) Search Strategies, Heuristic Function, Local Search Algorithms and Optimistic Problems, Local Search in Continuous Spaces, Online Search Agents and Unknown Environments, Constraint Satisfaction Problems (CPS), Backtracking Search and Local Search for CSP, Structure of Problems, Adversarial Search, Games – Optimal Decisions in Games, Alpha-Beta Pruning, Imperfect Real Time Decision, Games that Include an Element of Chance

2.1. Heuristic (or) Informed Search Strategies

2.2. Heuristic Search (or) Informed Search

The path-cost from the present state to target state is computed, to choose the least path-cost for the further state.

Additional information can be added as assumption to solve the problem.

E.g.

a. Greedy search
b. Best first search
c. A*search

Best First Search (BFS)

BFS is an example of the common GRAPH_SEARCH or TREE_SEARCH algorithm in which a node is chosen for extension based on a valuation function “f (n)”.

A Key element of these algorithms is a heuristic_function “h (n)”

h (n)=expected cost of the least-path from node, ‘n’ to a target node.

Two types of valuation functions are:

Extend the node nearby to the goal state using expected cost as the evaluation is known as **Greedy Best-First-Search**.

Extend node on the minimum cost solution path by using expected cost and actual cost as the evaluation function is called **A* search**.

A* search: reducing the total expected solution cost.

Extend the node on the minimum cost solution path by using expected cost and actual cost as the evaluation function is called **A*search**. It valuates nodes by joining the cost to arrive at the node, h(n), and g(n), the cost to obtain from the node to the target:

$$f(n) = h(n)+g(n).$$

h(n) is the expected cost of the least-path from n to the target and g(n) provides the path-cost from the begin node to n node, and, we contain

f (n)=expected cost of the least result throughout ‘n’.

A*search is both complete and optimal.

Greedy Best First Search

It attempts to extend the node, which is nearby to target, on the positions this is similarly to direct to a result rapidly. Therefore, it valuates nodes applying the **heuristic_function:**

$$f(n) = h(n).$$

It look like DFS in a way it wish to track an individual path each of the way to the target, but can backup when it knocks a corner. It is **not complete** and also **not optimal**. The bad case **space-complexity** and **time** is ‘O (bm)’, here ‘m’ is the highest depth of the search-space. It may be compressed with best heuristic function.

Monotonicity (consistency): In search tree every path from the root, the f-cost not at all reduces. This state is true for approximately every permissible heuristics. A heuristic which satisfies the property is called monotonicity.

Optimality: It is derived with two approaches. They are a) A* used with Tree-search b) A* used with Graph-search.

Memory Bounded Heuristic Search

The memory necessities of A* is decreased by joining the heuristic function with iterative deepening resulting an “IDA* algorithm”. The major dissimilarity between principle repetitive deepening and IDA* is the cut-off utilized is the fcost(g+h) alternatively the depth; at every repetition, the cut-off value is the minimum f-cost of each node, which go beyond the cut-off on the before repetition. The major difficulty is, it will require more storage space in complex domains.

The two recent memory bounded algorithms are:

1. Memory bounded A* search (MA*)
2. Recursive Best First Search(RBFS)

Recursive Best-first Search (RBFS)

RBFS is an easy recursive-algorithm utilizes only linear space. This algorithm is as follows:

```
function RECURSIVE BEST FIRST SEARCH (problem ) returns a solution, or failure
    RBFS (problem,MAKE NODE( INITIAL STATE[problem ] ),8 )

function RBFS ( problem,node,f_limit ) returns a result, or failure and a latest f-cost limit
    if GOAL TEST [ problem ] ( state )then return node
    Successors ← EXPAND( node, problem )
    if successors is blank then return failure, 8

    foreach 's' in successors do
        f[ s ] ← max ( g ( s )+ h ( s ).f [ node ] )
    repeat
        best ← the minimum f-value node in successors
        if f [ best ]>f_limit then return failure,f [ best ]
        alternative ← the 2nd minimum f-value among successors result,
        f[ best ] ← RBFS ( problem,best,min (f_limit, alternative ) )
        if result ? failure then return result
```

Figure 2.1: RBFS Algorithm

Its **time complexity** depends on both the on how often the better path modifies as nodes are extended and correctness of the heuristic function. Its **space complexity** is O (bd), even though more memory is available.

Search techniques which use every accessible memory are:

1) SMA* (Simplified MA*)
2) MA* (Memory BundedA*)

SMA*

It will apply of all existing storage to reveal the search.

Properties: i) It sidesteps repetitive states as distant as its memory permit.

ii) It can use whatever memory is exists to it.

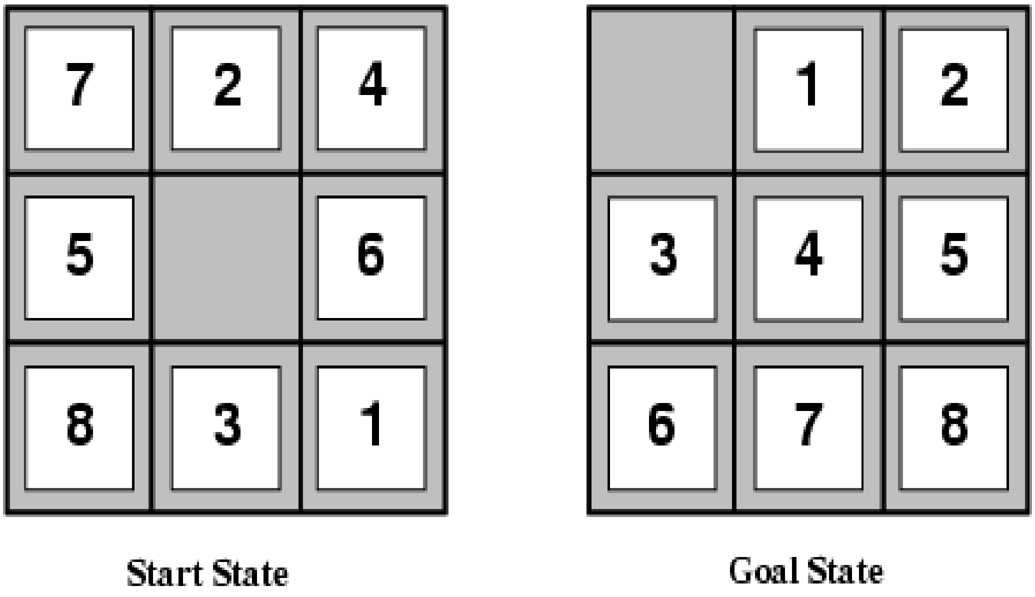
It is **complete,** if the existing storage is enough to backup the deepest resultant path.

It is **optimal,** if the sufficient storage is exists to backup the deepest resultant path or else, it comes back the better result that will be arrived at with the existing storage.

Advantage: SMA* uses only the available memory.

Disadvantage: If enough memory is not available it leads to suboptimal solution.

Time and Space complexity: depends on the available nodes count.



2.3. Heuristic Functions

The 8-puzzle Problem

Given:

Task: Discover the least solution by applying heuristic-function, which not at all approximate the steps count to the target.

Solution: To perform the given task two candidates are required, which are named as 'h1' and 'h2'.

h1=the quantity of misplaced tiles. From figure, each of the 8-tiles are not in the location, so the begin state could contain h1 = 8. ‘h1’ is a permissible heuristic, for the reason that it is understandable, which every tile that is not in a position can be changed not less than one time.

h2 is the addition of the distances of the tiles from their target places is known as City Block Distance or Manhattan-distance:

$$h2=1+3+2+3+2+2+3+2=18$$

The result of heuristic correctness on performance:

Effectual Branching Factor (b*)

In the search tree, if the total nodes count extended by A * for an exact problem is ‘N’, the resultant depth is ‘d’, next ‘b *’ is the branching-factor, which is an identical tree of depth-d should have to contain in array to consist of N + 1 nodes. therefore

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

For example: depth = 5; N = 52;

Effective branching factor = 1.91

Relaxed Problem

A Problem with lesser limitations on the operations is known as **relaxed problem**. A*n optimal result cost to be relaxed-problem is a permissible heuristic for the genuine problem*. If the specified problem is a relaxed-problem then it is available to create best heuristic function.

2.4. Optimization Problems and Local Search Algorithms

Local-search algorithms function by applying an individual present state and in general shift just to neighbors of that state. Local-search algorithms are unmethodical, they contains **2 key benefits**:

a) It utilizes small storage – regularly a stable quantity.

b) It may usually discover logical results in maximum or unlimited state-spaces for which methodical algorithms are not fitting.

To know the local-search, we can discover it extremely helpful to believe the state-space landscape as shown in the bellow figure.

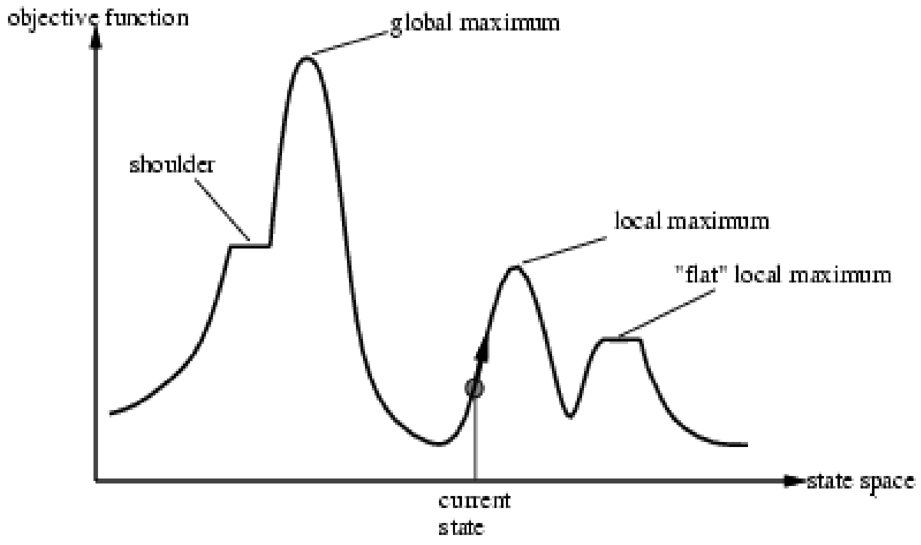


Figure: One-dimensional State Space Landscape

Applications

- Telecommunications network development
- Factory – floor layout
- Integrated –circuit plan
- Automatic programming
- Job-shop scheduling
- Vehicle routing

Advantages

- Constant search space. It is fit for online and also offline search.
- The search cost is low when matching to informed search strategies.

Some of the regional search algorithms are:

a) Genetic algorithm (GA)
b) Simulated annealing (SA)
c) Hill-climbing search (HCS)
d) Local beam search (LBS)

Hill-climbing Search (HCS)

A search technique that move in the direction of growing value to arrive at a peak state. It ends when it arrive at a dead-end where other neighbor does not have a maximum value. The Hill- climbing search algorithm as displaying in the bellow figure.

```
function HILL-CLIMBING-SEARCH ( problem ) returns a state that is local highest
    inputs:problem,a problem
    local variables:present,a node
                    neighbor,a node
    present? MAKE –NODE ( INITIAL – STATE [ problem ] )
    loop do
        neighbor ? a maximum - valued successor of present
        if VALUE [ neighbor ]=VALUE [ present ] then return STATE [ present ]
        present ? neighbor
```

Figure: The Hill - climbing – Search (HCS) Algorithm

Drawbacks

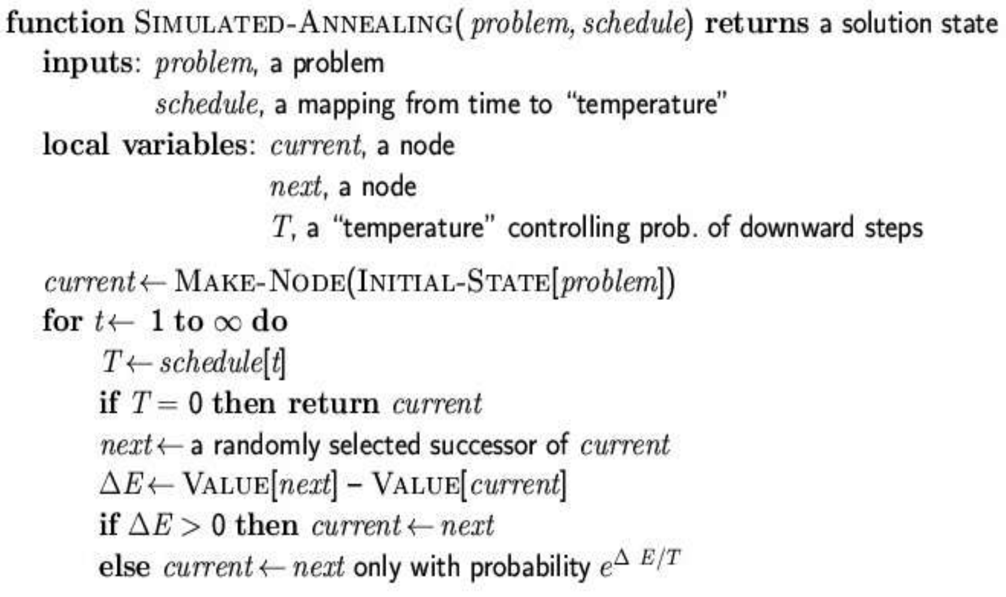
- **Local maxima** (Foot hills): a local maximum is a dead-end, which is maximum than every one of its neighboring states, but smaller than the overall highest.
- **Plateaux** (shoulder): It is a part of the state-space landscape wherever the valuation task is flat. This may be a flat-local highest.
- **Ridges**: a series of local maxima, which had a slope that gently moves to a peak.

Some of the variations of hill-climbing are:

- **First-choice hill-climbing**: applies statistic hill climbing by producing incomers at random up to one is produced, which is improved than the present state.
- **Stochastic-hill-climbing**: selects at random from between the uphill shifts.
- **Random-restart hill-climbing**: overcomes local-maximum slightly complete.

Simulated-Annealing Search

The algorithm which joins hill-climbing with unplanned walk to give up both completeness and effectiveness. This algorithm was developed using annealing process the procedure of slowly cooling a liquid up to it freezes.



```
function SIMULATED-ANNEALING( problem, schedule) returns a solution state
   inputs: problem, a problem
           schedule, a mapping from time to "temperature"
   local variables: current, a node
                    next, a node
                    T, a "temperature" controlling prob. of downward steps

   current ← MAKE-NODE(INITIAL-STATE[problem])
   for t ← 1 to ∞ do
       T ← schedule[t]
       if T = 0 then return current
       next ← a randomly selected successor of current
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability $e^{\Delta E/T}$
```

Figure 2.4: The algorithm for simulated annealing search

Applications

- VLSI layout
- Airline scheduling

Local Beam Search (LBS)

The LBS algorithm remains tracking of k-states alternatively only a single state. It starts with 'k' unplanned produced states. At every step, overall incomers of all k-states are produced. If anyone is a target, the algorithm stops or else, it chooses the 'k' greatest incomers from the entire index and iterates. *In LBS, helpful data is travelled between the k related search-threads*.

This search should undergo from short of variety between k-states. Accordingly a variant labeled as statistic-beam search selects 'k' incomers at unplanned manner, with the possibility of selecting a specified incomer being a growing task of its value.

Genetic Algorithm (GA)

A GA is a variation of statistic beam-search wherein incomer states are produced by joining 2 parent-states, alternatively by changing a single state. The bellow figure explains a GA algorithm.

```
function GENETIC ALGORITHM ( population,FITNESS-FN ) returns an entity
    inputs:population, a set of entities
        FITNESS-FN,a function that evaluates the condition of an entity
    repeat
        new-population ← blank set
        loop for 'i' from one to SIZE ( population ) do
            x ← RANDOMSELECTION ( population, FITNESS - FN )
            y ←RANDOMSELECTION ( population,FITNESS –FN )
            child ← REPRODUCE ( x, y )
            if (small-random-probability ) then child ?  MUTATE ( child )


            add child to new-population
        population?  new-population
    until a few entity is fit sufficient, or sufficient time has expired
    return the greatest entity in population, according to FITNESS - FN

function REPRODUCE ( x, y ) returns an entity
    inputs: parent entities, x, y
    n?  LENGTH ( x )
    c?  random number from one to n
    return APPEND ( SUBSTRING ( x, 1, c ), SUBSTRING ( y, c+1, n ) )
```

Figure: The Genetic Algorithm

2.5. Local Search (LS) in Continuous Spaces

LS in continuous space are the one that deals with the real world problems.

- One method to resolve permanent problems is to distinct the nearer of every state.
- Simulated Annealing and Statistic hill-climbing are executed openly in the continuous space.
- Empirical gradient, line search, Newton-raphson method can be applied in this domain t find the successor state.

- It may also lead to local maxima, ridges and plateau. This situation is avoided by using random restart method.

2.6. Online Search Agents and Unknown Environments

Offline search: They calculate the whole result earlier configuration of the foot in the actual universe next apply the result with no remedy to their rules.

Online search: Agents operate by interlinking execution and activity: initially, it receives an operation next it monitors computes and the environment of the subsequent operation.

Online search is the best thought in **semi-dynamic** (or) **dynamic** domains and **statistic** domains.

Online search is a required idea for an **exploration-problem**, wherever the actions and states are not known to the agent.

Online Search Problems (OSP)

An OSP should be resolved just by an agent applying operations, alternately by a execution process. We can guess that the agent notices the bellow statements:

- The step_cost_function c(s,a,s‘) recognized the agent when it arrives at s‘.
- ACTIONS (s), whichever gives back a list of operations permitted in state ‘s’.
- GOAL-TEST(s).

Here the agent can’t access state successors, not including by trying all available actions in that state. This drawback of an agent can be avoided by:

- Possible to find Manhattan distance heuristic.
- Actions are deterministic.
- Visited states are noted to the agent.

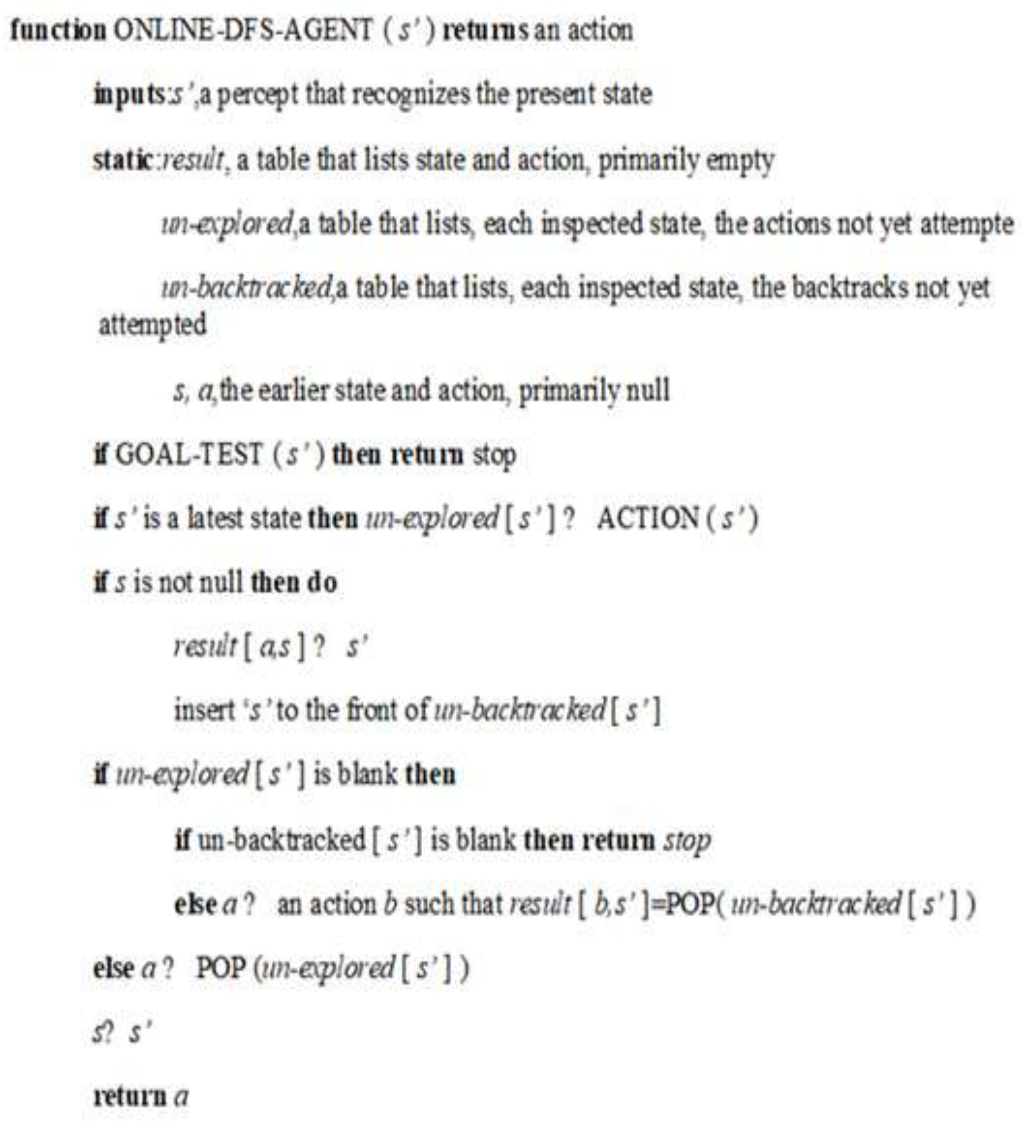
E.g. A simple maze problem

Competitive ratio: This defines the comparison between the total path costs with the computationally shortest complete exploration path cost. This ratio, should be as small as possible for better results.

Online Search Agent

An online algorithm, increase only a node, which it physically fills. After each and every operation, an online-search agent collects a rule to identify the state. It had arrived and it may augment its map of the environment, to decide where to go next.

An online DFS agent is exposed in figure:



```
function ONLINE-DFS-AGENT ( s' ) returns an action
    inputs:s',a percept that recognizes the present state
    static:result, a table that lists state and action, primarily empty
        un-explored,a table that lists, each inspected state, the actions not yet attempte
        un-backtracked,a table that lists, each inspected state, the backtracks not yet
    attempted
        s, a,the earlier state and action, primarily null
    if GOAL-TEST ( s' ) then return stop
    if s' is a latest state then un-explored [ s' ] ?   ACTION ( s')
    if s is not null then do
        result [ a,s ] ?   s'
        insert 's' to the front of un-backtracked [ s' ]
    if un-explored [ s' ] is blank then
        if un-backtracked [ s' ] is blank then return stop
        else a ?   an action b such that result [ b,s' ]=POP( un-backtracked [ s' ] )
    else a ?   POP (un-explored [ s' ] )
    s? s'
    return a
```

Figure: An Online Search Agent that Uses Depth First Exploration

Online Local Search (OLS)

Hill-climbing search technique can be utilized to perform online local search because it maintains one present state in storage. To avoid the drawback of local maxima, random walk is chosen to explore the environment instead of random restart method. The concept of hill climbing with data stores a present greatest guess H(s) of the cost to arrive at the target from all states, which have been visited is executed as Learning Real Time A*(LRTA *) algorithm.

The LRTA* agent algorithm is shown in figure:

```
function LRTA*-AGENT (s') returns an action
    input:s',a percept that finds the present state
    static:result,a table that lists action and state, primarily empty

    'H',a table of cost estimation list by state, primarily empty

    a, s, the earlier state and action, primarily null
    if GOAL TEST ( s' ) then return stop
    if s' is a latest state ( not in 'H' ) then H [ s' ] ?  h ( s' )

    unless 's' is null
        result [ a,s ] ?  s'
        a?  an action b in ACTIONS ( s' ) that decreases
        LRTA*  COSTS ( s',b,result [ b.s' ], H )
        s?  s'
     return a


    function LRTA* COST ( s,a,s',H ) returns a cost estimate
    if s' is unspecified then return h ( s )
          else return c ( s,a,s' )+H [ s' ]
```

Figure: LRTA*-Agent algorithm

2.7. Constraint Satisfaction Problems(CSP)

CSP is described by a group of **constraints**, C1, C2,….., Cm and a group of **variables**, X1,X2,X3,……., Xn. Every variable 'Xi' not having empty **domain** 'Di' of all available **values.** A whole task is a thing in which each variable is declared and a **result** to a CSP is a total task that gratifies each constraint. A few CSPs in addition need a result that increases an **objective function.**

Some examples of CSP's are:

- Then-queens problem
- A crossword problem
- A map coloring problem

Constraint graph: A CSP is generally shown as not a directed graph it is known as constraint graph wherever the corners are the binary-constraints and the nodes are the variables.

CSP should be considered as an additional formulation as an ideal inspection problem as the bellow points:

- **Primary state**: the blank set {}, in that no variables are assigned.
- **Successor function**: give an element to a not allocated variable, given that this is not dispute with earlier allocated variables.
- **Goal test**: the present set is completed.
- **Path cost**: an invariable cost for each step.

Discrete Variables

a) **Infinite domains**: For n variables with infinite domain size such as strings, integers etc. **E.g.** set of strings and set of integers.

b) **Finite domains:** For n number of variables with a limited size domain d, the complexity is O(dn). Entire assignment is available.

e.g. Map-coloring problems.

Continuous variables: Linear restrictions solvable in a polynomial point in time by linear programming. **E.g:** begin / end instants for Hubble-space telescope considerations.

Types of Constraints

a. **Unary constraints**: It limits the value of a variable.

b. **Binary constraints**: It associates with the pair of variables.

c. **Greater order constraints:** It involves more than or equal to three variables.

2.8. Backtracking Search for CSP's

Backtracking search, a form of DFS that selects values for only one variable individually and backtracks whenever a variable does not having permissible values to allocate. This algorithm is exposed in the bellow figure:

```
function BACKTRACKING_SEARCH ( csp ) returns a result, or failure
    return RECURSIVE_BACKTRACKING ( {},csp )
function RECURSIVE_BACKTRACKING ( assignmen, csp ) returns a result, or failure
    if assignment is completed then return assignment
    var?  SELECT_UNASSIGNED_VARIABLE ( VARIABLE [ csp ], assignment,csp )
    foreach value in ORDER_DOMAIN_VALUES ( var,assignment,csp ) do
    if value is reliable with assignment giving to CONSTRAINT [ csp ] then
            insert{ var = value } to assignment
            result?  RECURSIVE_BACKTRACKING( assignment,csp )
            if result? failure then return result
            delete{var = value } from assignment
    return failure
```

Figure: An Easy Backtracking Algorithm for CSP

Propagating Information through Constraints

Forward checking: The key steps of forward checking process are:

- Be track of left over permissible values for not assigned variables.
- End search when every variable does not having permissible values.

This method propagates the information from allocate to unallocated variables, but does not give early discovery for all non-success.

Constraint Propagation

Constraint propagation frequently implements constraints locally to notice inconsistencies. This propagation may be complete with dissimilar kinds of consistency methods.

1. **Node consistency**(one consistency): The node showing a variable 'V' in constraint-graph is node-consistent, if the value 'X' in the present domain of 'V', every single constraint on 'V' is fulfilled. The node inconsistency may be removed by eliminating the

values from the domain ‘D’ of every variable, which does not gratify single constraint on ‘V’.

2. **Arc consistency** (two consistency): Arc specifies to a ordered arc in the control graph. The different versions of Arc-consistency algorithms are exist such as AC- 1, upto AC-7, but frequently used are AC-3 or AC-4.
3. **Path consistency** (K-consistency): The algorithm for creating a constraint-graph powerfully three constant that is regularly referred as path-consistency make sure that problem may be resolved without back tracking.

Handling Special Constraints

1. **Alldiff constraint**: All the variables concerned must have different values.
2. **Resource constraint**: Higher order or atmost constraint, in which consistency is achieved by removing the highest value of every domain, if it is inconsistent with least values of the further domains.

Intelligent Backtracking

Chronological backtracking: when a branch of the search be unsuccessful, backup to the previous variable and attempt a distinct value for it. At that point, the latest decision point is updated.

Conflict-directed back-jumping: It backtracks straight to the basis of the problem.

2.9. Local Search for CSP

Local search by means of the minimum variances heuristic have been tested to constraint fulfillment problems with big achievement. They apply a whole state formulation: the primary state allots a value to each variable and the successor-function regularly performs by modifying the value of 1 variable individually.

In selecting a latest value for a variable, the better noticeable heuristic is to choose the value that solution in the lower conflicts count with further variables, the **least conflicts** heuristic.

Minimum conflicts is surprisingly valuable for several CSPs, mainly when given a sensible primary state. Surprisingly, on the n-queens problem, if we do not calculate the number of primary assignment of queens, the continuation of minimum variances is approximately *autonomous of problem size*.

```
function MIN_CONFLICTS ( csp,max_steps ) returns a result or failure
    input:csp,a constraint gratification problem
        max_steps, the number of steps permitted before allowing up
    present an primary complete assignment for csp
    for i=1 to maximum steps do
        if present is a result for csp then return present
        var 'a' randomly selected, conflicted variable from VARIABLES [
        csp ] the value 'v' for var that decreases CONFLICTS (
        var,v,present,csp )
        set var= value in present
    return failure
```

Figure: Minimum Conflicts Algorithm for Solving CSP

2.10. The Structure of Problems

The difficulty of solving CSP is powerfully correlated to the structure of its constraint-graph. If the CSP may be separated into independent sub-problems, then every sub-problem is resolved independently then the results are joined. When 'n' variables are divided as 'n/c' sub-problems, each will take dc operation to resolve. Therefore the whole operation is O(dcn /c).

Every tree structured CSP may be resolved in time limited in the variables count. The algorithm contains the bellow statements:

1. Select one variable as the origin of the tree and arrange the variables from the origin to the leafs just as all parent nodes in the tree predates it in the arranging label the variables X1,X2,…,Xn in order, all variables excluding the root has only 1 variable's parent.
2. For 'j' from one to n, allocate one value for 'Xj' consistent with the value allocated for 'Xi', wherever 'Xj' is the child of 'Xi'
3. For j from 'n' downward to two, use arc-consistency to the arc(Xi,Xj), wherever 'Xj' is the child of 'Xi', deleting values from DOMAIN[Xi] as essential.

The complete algorithm runs in time O (nd2).

General constraint graphs may be decreased to trees on 2 ways. They are:

1. Removing nodes – Cutest conditioning
2. Collapsing nodes together – Tree decomposition

2.11. Adversarial Search

Games

Game may be described by the primary state, the permissible operations in all states, a utility-function and a closing test that uses to closing states.

In game playing to select the next state, search technique is required. The **pruning technique** permits us to exclude location of the search-tree, which makes no dissimilarity to the last selection, and **heuristic-evaluation function** permit us to discover the utility of a state is not performing a total search.

2.12. Optimal Decisions in Games

Game may be officially described as a type of exploration problem with the bellow elements:

- **Primary state**: This contains the panel place and finds the performer to shift.
- A **successor_function** (operators) that arrivals a listing of pairs (move, state), all indicting a permissible shift and the resultant state.
- A **terminal-test** that decides whenever the game is completed.
- A **utility function** (payoff function or objective function) that provides a numeral values for the closing states.

The Mini-max Algorithm

This algorithm executes the mini-max selection from the present state. It executes an entire DFS of the game-tree. If the highest depth of the tree is 'm', and it has 'b' permissible shifts at all positions, next the complexity time of the mini-max algorithm is O(bm).

Minimax is for deterministic games with perfect information. The **mini-max** algorithm creates the complete game tree and uses the utility function to every closing state. Next it transmits the utility value up single level and carry on to perform so until arriving the begining node.

The mini-max algorithm is as follows:

```
function MINIMAX_DECISION ( state ) returns an action
        input:state, present state in
        game v?  MAX_VALUE (
        state )
        return the action in SUCCESSORS ( state ) with value 'v'

function MAX_VALUE ( state ) returns a utility value
        if TERMINAL_TEST( state ) then return UTILITY ( state )
        V?  - 8
        For s,a, in SUCCESSORS ( state ) do
                v?  MAX (v,MIN_VALUE ( s ) )
return v

function MIN_VALUE ( state ) returns a utility value
        if TERMINAL_TEST ( state ) then return UTILITY ( state )

        v?  8
        for s,a in SUCCESSORS ( state ) do
                v?  MIN (v,MAX_VALUE ( s ) )
return v.
```

Figure: An Algorithm for Calculating Minimax Decisions

2.13. Alpha Beta Pruning

Pruning: The procedure of removing a branch of the search tree from discussion without investigation is called pruning. The 2 specifications of pruning methods are mentioned bellow:

a. **Alpha(α):** Better selection for the value of MAX(maximum) along the path or lesser bound on the value, which on increasing node can be finally allocated.
b. **Beta(β)**: Better selection for the value of MIN(minimum) along the path or higher bound on the value, which a decreasing node can be finally allocated.

Alpha-Beta Pruning: The values of alpha and beta are applied on a mini-max tree, it gives back the similar shift as mini-max, but prunes left the branches that may not control the last choice is called **Cutoff or Alpha Beta pruning**.

Consider the 2 play game tree from the bellow figure:

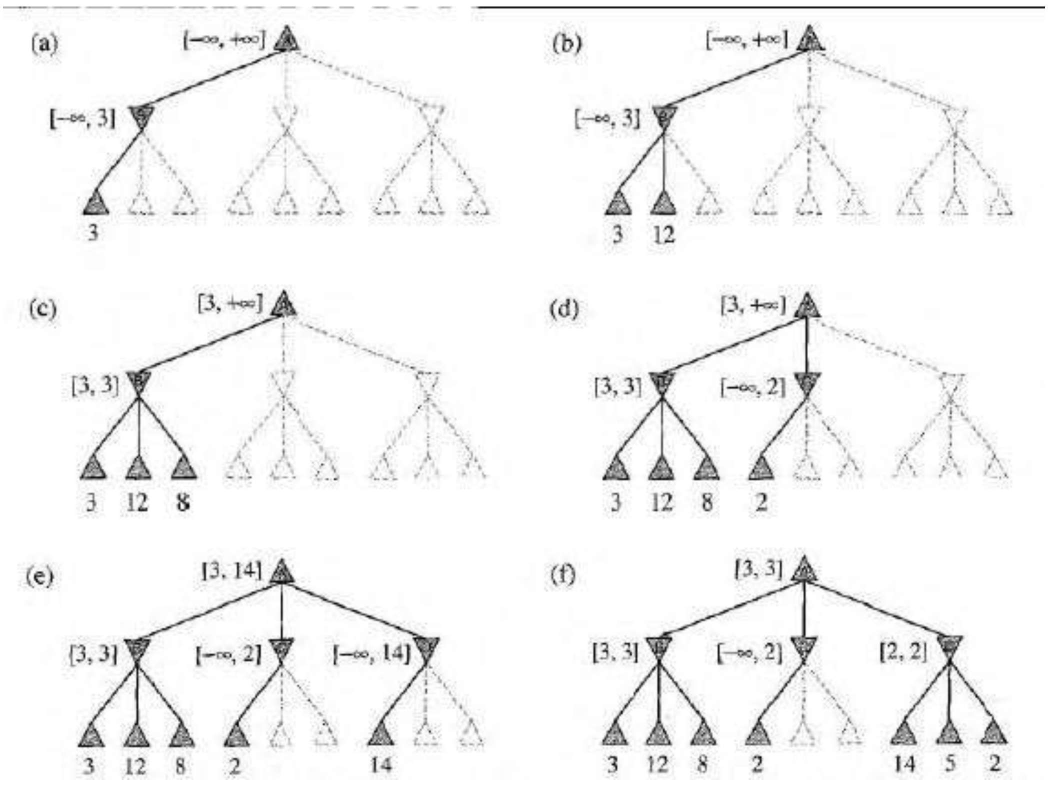


Figure 6.5 Stages in the calculation of the optimal decision for the game tree in Figure 6.2. At each point, we show the range of possible values for each node. (a) The first leaf below B has the value 3. Hence, B, which is a MIN node, has a value of *at most* 3. (b) The second leaf below B has a value of 12; MIN would avoid this move, so the value of B is still at most 3. (c) The third leaf below B has a value of 8; we have seen all B's successors, so the value of *B* is exactly 3. Now, we can infer that the value of the root is *at least* 3, because MAX has a choice worth 3 at the root. (d) The first leaf below C has the value 2. Hence, *C*, which is a MIN node, has a value of *at most* 2. But we know that *B* is worth 3, so MAX would never choose C. Therefore, there is no point in looking at the other successors of C. This is an example of alpha–beta pruning. (e) The first leaf below D has the value 14, so D is worth *at most* 14. This is still higher than MAX'S best alternative(i.e., 3), so we need to keep exploring D's successors. Notice also that we now have bounds on all of the successors of the root, so the root's value is also at most 14. (f) The second successor of D is worth 5, so again we need to keep exploring. The third successor is worth 2, so now D is worth exactly 2. MAX's decision at the root is to move to B, giving a value of 3.

Alpha-Beta pruning may be practiced to trees of all depths and this is frequently probable to prune complete sub trees in place of leaves.

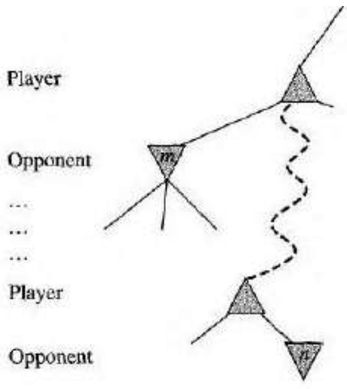


Figure Alpha Beta pruning, the common case. If 'm' is improved than 'n' for player, we can never obtain to 'n' in play.

```
function ALPHA-BETA-SEARCH(state) returns an action
  inputs: state, current state in game

  v ← MAX-VALUE(state, −∞, +∞)
  return the action in SUCCESSORS(state) with value v
```

```
function MAX-VALUE(state, a, β) returns a utility value
  inputs: state, current state in game
          a , the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for a, s in SUCCESSORS(state) do
    v ← MAX(v, MIN-VALUE(s, a , β))
    if v ≥ β then return v
    a ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, a , β) returns a utility value
  inputs: state, current state in game
          a, the value of the best alternative for MAX along the path to state
          β, the value of the best alternative for MIN along the path to state

  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a, s in SUCCESSORS(state) do
    v ← MIN(v, MAX-VALUE(%a, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure: An Alpha-beta Search Algorithm

2.14. Imperfect, Real Time Decisions

Effectiveness of Alpha Beta Pruning

It requires to observe only $O(b^{m/2})$ nodes to select the better move.

Futility cut-off

Finishing the searching of a sub tree, which attempt slight possibility for enhancement over other recognized path, is called futility cut off.

2.15. Games that Contain an Element of Possibility

Backgammon problem is a common game that joins the skill and luck. Dice are rotated at the starting of a player turn to decide the group of permissible shifts that is accessible to the player.

The bellow figure, white has rotated a 5-6, and has 4 probable shifts.

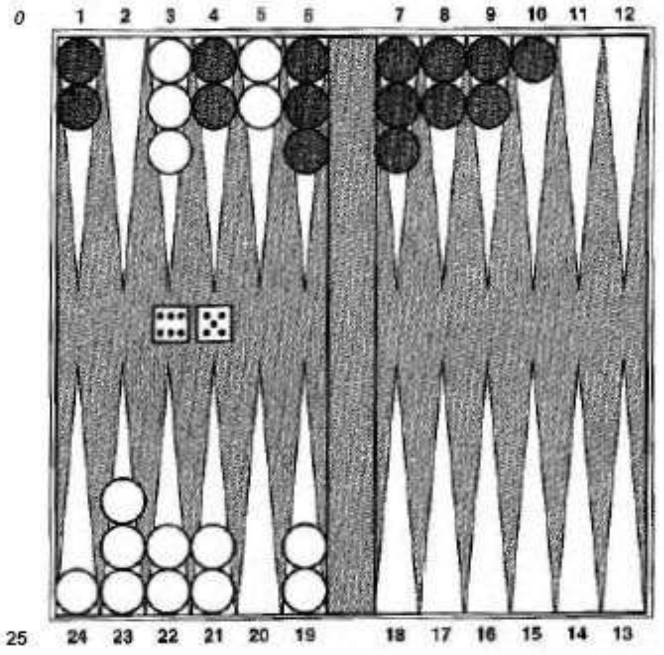


Figure A typical backgammon position. The goal of the game is to move all one's pieces off the board. White moves clockwise toward 25, and black moves counterclockwise toward 0. A piece can move to any position unless there are multiple opponent pieces there; if there is one opponent, it is captured and must start over. In the position shown, White has rolled 6–5 and must choose among four legal moves: (5–10,5–11), (5–11,19–24), (5–10,10–16), and (5–11,11–16).

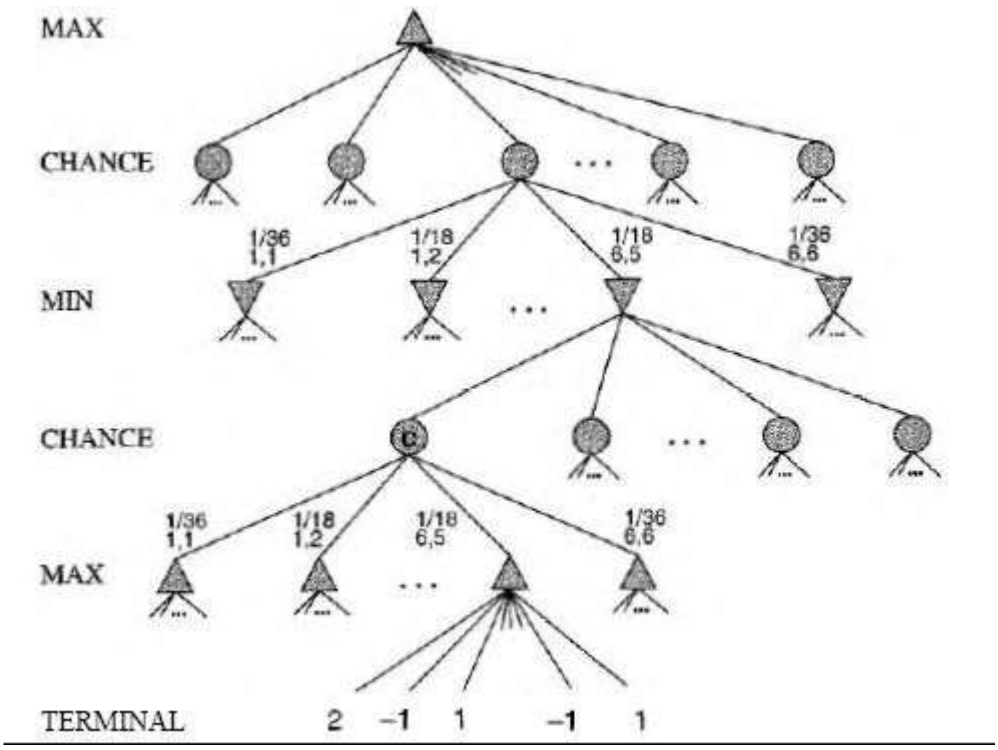


Figure: The Schematic Game Trees for a Backgammon Location

Question Bank

Unit - II

Part - A

1. Explain the local-minima problem.
2. How to increase the efficiency of a search-based problem-solving method?
3. Identify the complexity of mini-max.
4. What is the purpose of online search agents in unidentified environments?
5. Explain the difference between A*search with greedy search.
6. Describe relaxed problem.
7. Explain the 3 benefits of hill-climbing.
8. Define pruning.
9. How the search technique is perform powerful in continuous space.
10. Explain in detail about chance nodes.
11. Describe cycle cut set.
12. What are constraint-satisfaction problems? How can we prepare them as search problems?

13. Explain the effective branching factor with an example.
14. Write the step by step process of mini-max algorithm.
15. State the purpose to avoid the drawback of mini-max algorithm.
16. Explain the alpha and beta cutoff with example.
17. What is the requirement of memory delimited in heuristic search?
18. Describe CSP.
19. Define constraint graph.
20. List the different types of constraints.
21. Define backtracking search.
22. Define constraint propagation
23. What is a constraint-satisfaction problem?
24. Explain the game playing problems with an example.
25. What is local-search algorithm and explain the different types and applications.
26. Explain the memory bounded search with two examples.
27. How free-decomposition is attained?
28. Differentiate online search with offline search.
29. Describe the horizon problem with an example.
30. Explain the optimality of A* search and monotonocity.
31. What is the requirement of arc-consistency?
32. Explain the different kinds of consistency methods.
33. How does alpha beta pruning method works?
34. Describe the conflict-direct back jumping.

Part – B

1. Explain back-tracking search of CSP along with algorithm.
2. Explain in brief regarding memory-bounded search algorithms with one example for all searches.
3. How does hill-climbing make sure greedy-local-search?
4. Discuss the different issues related with the backtracking search from CSPs. How they addressed?
5. Explain in brief about heuristic-search methods with one example for every search.
6. What is genetic algorithm? Explain with one example.
7. Explain mini-max algorithm with its process with one example.
8. Explain the performance of A* search algorithm with one example.
9. Explain wheatear the problem structure affects on the solving method?

10. Describe the structure of problem using CSP?
11. What is the requirement of online search agent? Discuss with algorithm.
12. Describe the alpha beta pruning and provide the order changes to the mini-max process to increase its performance.
13. Explain the alpha-beta pruning with its process and with an example.