CA435 MODERN ARTIFICIAL INTELLIGENCE

MODULE-2

SANJAY KUMAR

s.kumar@bitmesra.ac.in

Department Of Computer Science & Engineering

Birla Institute of Technology Mesra

Off-Campus Noida

1

Module-2

Heuristic Search Strategies

2

2

Table of Contents

3

3

Heuristic Search Strategies

Greedy Best-First Search

4

4

Heuristic Search Strategies

- Uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space.
- Informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.
- A heuristic is a method that improves the efficiency of the search process.
- Informed (Heuristic) search methods may have access to a heuristic function $h(n)$ that estimates the cost of a solution from n.

5

5

Heuristic Search Strategies (cont.)

- Heuristic is a function h(n), which is used in Informed Search, and it finds the most promising path.
- Conditions for Optimality – Admissibility and Consistency
- First condition for optimality, h(n) be an admissible heuristic. i.e. it never overestimates the cost to reach the goal.
- i.e. a search algorithm that is guaranteed to find an optimal path to a goal, if one exists, is called admissible.
- Admissibility of the heuristic function h(n) can be defined as:
- h(n) <= h'(n). Here h(n) is heuristic cost, and h'(n) is the estimated cost.

6

6

Heuristic Search Strategies (cont.)

- Second slightly strong condition for optimality called consistency (or monotonicity).
- A heuristic h(n) is consistent if, for every node n and every successor n' of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n' plus the estimated cost of reaching the goal from n':

  $h(n) \leq c(n, a, n') + h(n')$

- This is a form of the general triangle inequality, which stipulates that each side of a triangle cannot be longer than the sum of the other two sides. Here, the triangle is formed by n, n', and the goal G_n closest to n.

7

7

Heuristic Search Strategies (cont.)

- Heuristic search refers to a search strategy that attempts to optimize a problem by iteratively improving the solution based on a given heuristic function or a cost measure.
- It expands nodes based on their heuristic value h(n).
- It maintains two lists, *OPEN* and *CLOSED* list.
- In the *CLOSED* list, it places those nodes which have already expanded and
- in the *OPEN* list, it places nodes which have yet not been expanded.

8

8

Characteristics of Heuristic Search

- Heuristics are knowledge about domain, which help search and reasoning in its domain.
- Heuristic search incorporates domain knowledge to improve efficiency over blind search.
- Heuristic is a function that, when applied to a state, returns value as estimated merit of state, with respect to goal.
- Heuristic evaluation function estimates likelihood of given state leading to goal state.
- Heuristic search function estimates cost from current state to goal, presuming function is efficient.

9

9

Heuristic Search Strategies

Greedy Best-First Search

10

10

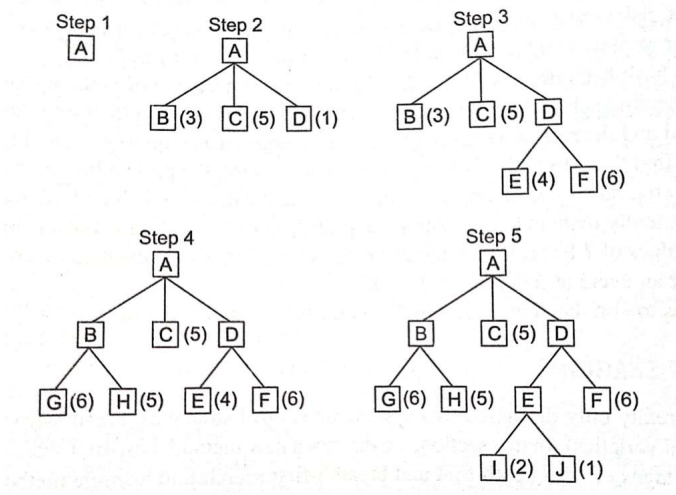
Greedy Best-First Search

- Greedy best-first search algorithm always selects the path which appears best at that moment. With the help of best-first search, at each step, we can choose the most promising node.
- In this algorithm, we expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly.
- Thus, it evaluates nodes by using just the heuristic function; that is, f(n) = h(n). Where h(n)= estimated cost from node n to the goal.
- The greedy best first algorithm is implemented by the priority queue.

11

11

Greedy Best-First Search (cont.)



12

12

Greedy Best-First Search Algorithm

1. Start with *OPEN* containing just the initial state.
2. Until a goal state is found or there are no nodes left on *OPEN* do:
   a) Pick the best node on *OPEN*.
   b) Generate its successors.
   c) For each successor do:
      i. If it has not been generated before, evaluate it, add it to *OPEN*, and record its parent.
      ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

13

13

Greedy Best-First Search (cont.)

- Advantages
  - It can switch between BFS and DFS by gaining the advantages of both the algorithms.
  - This algorithm is more efficient than BFS and DFS algorithms.
- Disadvantages
  - It can behave as an unguided depth-first search in the worst case scenario.
  - It can get stuck in a loop as DFS.
  - This algorithm is not optimal.

14

14

Heuristic Search Strategies

A* Search

15

15

A* Search

- The most widely known form of best-first search is called A* search (pronounced “A-star search”).
- For A*, we need a heuristic function f(n) that estimates the merits of each node it generate. For each node, it can be define as the sum of two components g(n) and h’(n).

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

- g(n) gives the path cost from the start node to current node n,
- h’(n) is the estimated of additional cost of getting from the current node n to a goal node,
- f(n) approximation to a function that gives the true evaluation of the node .

16

16

A* Search (cont.)

- A* always chooses the path on the frontier with the lowest estimated distance from the start to a goal node constrained to go via that path.
- Role of g function: This lets us choose which node to expand next on the basis of not only of how good the node itself looks, but also on the basis of how good the path to the node was.
- Role of h' function: h', the distance of a node to the goal. If h' is a perfect estimator of h, then A* will converge immediately to the goal with no search.

17

17

A* Search Algorithm

1. Place the starting node in the *OPEN* list.
2. Check if the *OPEN* list is empty or not, if the list is empty then return failure and stops.
3. Select the node from the *OPEN* list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise
4. Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the *OPEN* or *CLOSED* list, if not then compute evaluation function for n' and place into Open list.
5. Else if node n' is already in *OPEN* and *CLOSED*, then it should be attached to the back pointer which reflects the lowest g(n') value.
6. Return to Step 2.

18

18

A* Search (cont.)

- Advantages
  - A* search algorithm is the best algorithm than other search algorithms. It can solve very complex problems.
  - It is optimal and complete.
- Disadvantages
  - It does not always produce the shortest path as it mostly based on heuristics and approximation.
  - A* search algorithm has some complexity issues.
  - The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

19

19

Characteristics of A* Search

- A* is complete (finds a solution, if one exists), if the graph it is searching, is locally finite (that is, it has a finite branching factor) and
- if every arc between two nodes in the graph has a non-zero cost.

- A* is admissible as A* is the name given to the algorithm where the h(node) function is admissible.
- In other words, it is guaranteed to provide an underestimate of the true cost to the goal.

20

20

Characteristics of A* Search (cont.)

- A* is consistence as it is using admissible heuristic function h(n) and optimal GRAPH-SEARCH algorithm which keeps all checked nodes in memory to avoid repeated states.

- A* satisfies a form of the general triangle inequality if admissible heuristic function h(n) of A* is consistence. Here, the triangle is formed by n, n', and the goal G_n closest to n.
- The general triangle inequality stipulates that each side of a triangle cannot be longer than the sum of the other two sides.

21

21

Characteristics of A* Search (cont.)

- A* search algorithm is optimal (finds the optimal path to a goal) as it satisfies both conditions of optimality – admissibility and consistency.
- i.e., it always finds the solution with the lowest total cost if the heuristic h(n) is admissible (means if h(node) is always an underestimate of the distance of a node to a goal node).

22

22

Characteristics of A* Search (cont.)

- Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.
- Space Complexity: The space complexity of A* search algorithm is O(b^d)

23

23

Heuristic Search Strategies

Memory Bounded Heuristic Search

24

24

Memory Bounded Heuristic Search

- The problem with A⋆ as presented is that it needs to keep track of all fringe and closed nodes.
- Thus, it tends to run out of space before time.
- We now look at a couple of ways to solve this problem by using memory bounded heuristic search algorithms.

25

25

Iterative Deepening A* (IDA*)

- In IDA⋆ we fix a constant μ and
- we modify the expand-node function so that it only adds nodes to the fringe that are of cost less than the current threshold value.
- IDA* is then:
  - Do A⋆ search for thresholds <μ, <2μ, <3μ, ... until you find a solution.
- The main difference between IDA* and standard iterative deepening is that the cutoff used is the f-cost (g + h) rather than the depth; at each iteration, the cutoff value is the smallest f-cost of any node that exceeded the cutoff on the previous iteration.

26

26

Recursive Best-First Search (RBFS)

- It is a simple recursive algorithm that attempts to mimic the operation of standard best-first search, but using only linear space.
- It replaces the f-value of each node along the path with best f-value of its children.
- It suffers from using too little memory, even if more memory were available, RBFS has no way to make use of it.

27

27

Simplified Memory Bounded A* (SMA*)

- Do A* until we run out of memory.
- When we don't have enough memory to add a new node to the fringe, discard from closed or fringe node of worst cost.
- SMA* is complete, if any reachable solution.

28

28

Heuristic Search Strategies

Local Search Algorithms & Optimization Problems

Local Search Algorithms & Optimization Problems

- Local search algorithms operate using a single current node (rather than multiple paths).
- Generally move only to neighbors of that node.
- Typically, the paths followed by the search are not retained.

Local Search Algorithms & Optimization Problems (cont.)

- Advantages of Using Local Search Algorithms
  - Use of little memory – usually a constant amount of memory
  - We can find reasonable solution in case of large state spaces of the problem. More useful to solve pure optimization problems.

31

31

Heuristic Search Strategies

Hill Climbing Search

32

32

Hill Climbing Search

- It is simply a loop that continually moves in the direction of increasing value—that is, uphill. It terminates when it reaches a "peak" where no neighbor has a higher value.
- It does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.

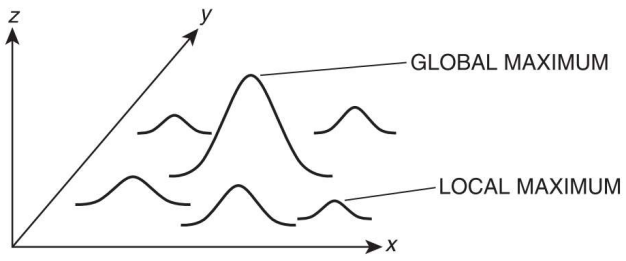
```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    current ← problem.INITIAL
    while true do
        neighbor ← a highest-valued successor state of current
        if VALUE(neighbor) ≤ VALUE(current) then return current
        current ← neighbor
```

33

33

Drawbacks of Hill Climbing Search

- Local Maxima: A local maximum is a part of the search space that appears to be preferable to the parts around it, but which is in fact just a foothill of a larger hill.



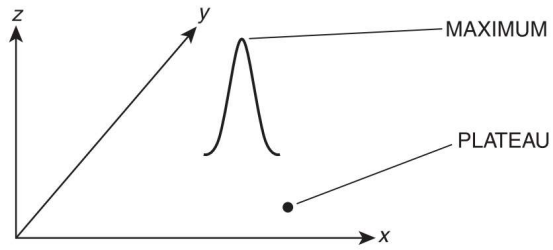
- Backtrack to some earlier node and try going in a different direction for dealing with local maxima.

34

34

Drawbacks of Hill Climbing Search (cont.)

- Plateau: A plateau is a region in a search space where all the values are the same.



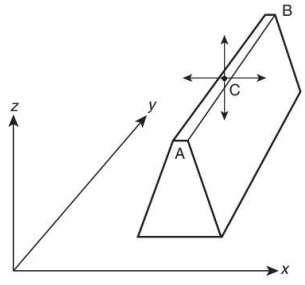
- Make a big jump in some direction to try to get to a new section of the search space for dealing with plateau.

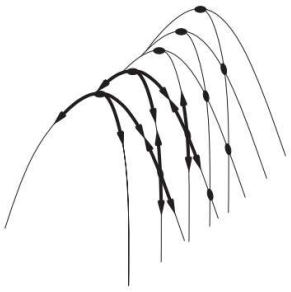
35

35

Drawbacks of Hill Climbing Search (cont.)

- Ridge: A ridge is a long, thin region of high land with low land on either side.





- Apply two or more rules before doing the test for dealing with ridge.

36

36

Heuristic Search Strategies

Simulated Annealing Search

Simulated Annealing Search

- Annealing refers to an analogy with thermodynamics, specifically with the way that metals cool and anneal.
- Simulated annealing uses the objective function of an optimization problem instead of the energy of a material.
- It is useful in finding global optima in the presence of large numbers of local optima.

Simulated Annealing Search (cont.)

- This search algorithm is basically hill-climbing except instead of picking the best move, it picks a random move.
- If the selected move improves the solution, then it is always accepted.
- Otherwise, the algorithm makes the move anyway *with some probability* less than 1.
- The probability decreases exponentially with the "badness" of the move, which is the amount ΔE by which the solution is worsened (i.e., energy is increased.)

39

39

Simulated Annealing Search Algorithm

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  current ← *problem*.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    next ← a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E > 0$ **then** *current* ← *next*
    else *current* ← *next* only with probability $e^{-\Delta E/T}$

40

40

Simulated Annealing Search (cont.)

- Applications of Simulated Annealing Search
  - Simulated annealing is typically used in discrete, but very large, configuration spaces, such as
  - the set of possible orders of cities in the Traveling Salesman problem and
  - in VLSI routing.
  - It has a broad range of application that is still being explored.

41

41

Heuristic Search Strategies

Local Beam Search

42

42

Local Beam Search

- Keeping just one node in memory might seem to be an extreme reaction to the problem of memory limitations.
- The local beam search algorithm keeps track of k states rather than just one.
- It begins with *k* randomly generated states. At each step, all the successors of all *k* states are generated.
- If anyone is a goal, the algorithm halts. Otherwise, it selects the *k* best successors from the complete list and repeats.
- It quickly abandons unfruitful searches and moves its resources to where the most progress is being made.

43

43

Heuristic Search Strategies

Genetic Algorithm

44

44

Genetic Algorithm

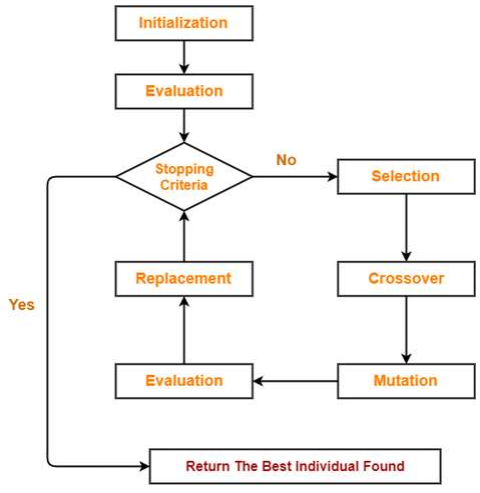
- Genetic algorithm (GA) is a search heuristic that is inspired by Charles Darwin's theory of natural evolution.
- This algorithm reflects the process of natural selection, where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.
- In a genetic algorithm successor states are generated by mutation and by crossover, which combines pairs of states from the population.

45

45

Genetic Algorithm (cont.)

- How genetic algorithm works?

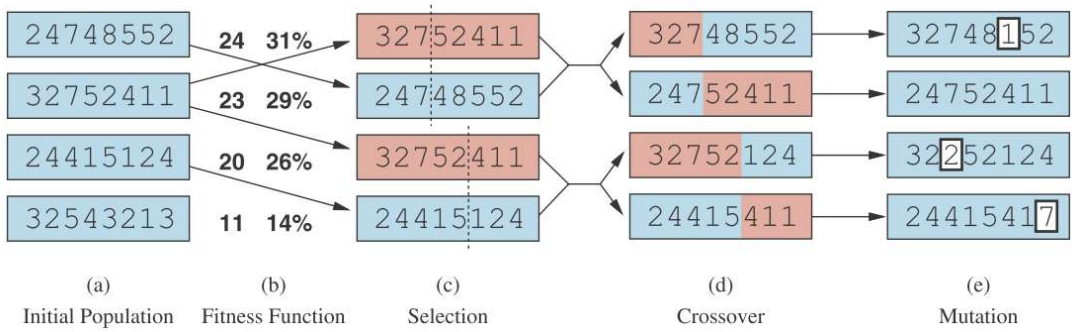


46

46

Genetic Algorithm (cont.)

- A genetic algorithm, illustrated for digit strings representing 8-queens states.

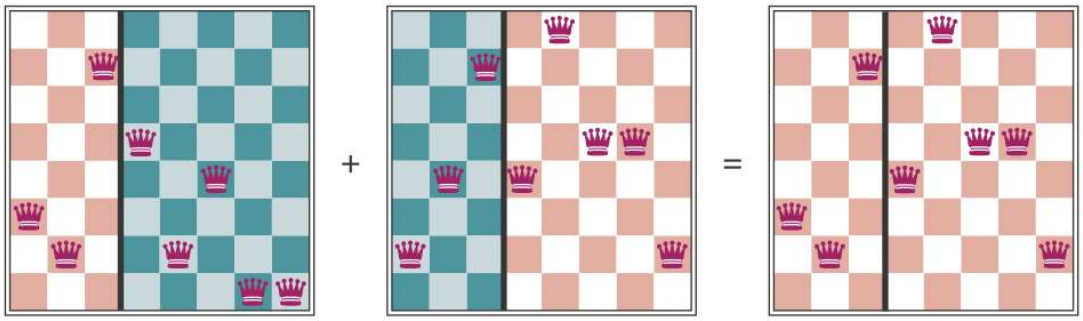


47

47

Genetic Algorithm (cont.)

- A genetic algorithm, illustrated for the 8-queens states corresponding to the first two parents.



48

48

Genetic Algorithm (cont.)

```
function GENETIC-ALGORITHM(population, fitness) returns an individual
  repeat
    weights ← WEIGHTED-BY(population, fitness)
    population2 ← empty list
    for i = 1 to SIZE(population) do
      parent1, parent2 ← WEIGHTED-RANDOM-CHOICES(population, weights, 2)
      child ← REPRODUCE(parent1, parent2)
      if (small random probability) then child ← MUTATE(child)
      add child to population2
    population ← population2
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to fitness

function REPRODUCE(parent1, parent2) returns an individual
  n ← LENGTH(parent1)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(parent1, 1, c), SUBSTRING(parent2, c + 1, n))
```

49

49

Heuristic Search Strategies

Constraint Satisfaction Problems

50

50

Constraint Satisfaction Problems (CSPs)

- So far when discussing search, we have looked at environments in the states were indivisible, that is, atomic.
- We now consider states which are allowed to have field variables. i.e., we consider environments with factored representations.
- For factored representation, we say a state solves the problem when each field variable satisfies all the constraints on that variable.
- A problem described in this way is called a constraint satisfaction problem, or CSP.
- Many problems in AI can be viewed as problems of constraints satisfaction in which the goal is to discover some problem state that satisfies a given set of constraints.

51

51

Constraint Satisfaction Problems (cont.)

- A constraint satisfaction problem consists of three components, X, D, and C:
  - X is a set of variables, $\{ X_1, X_2, \ldots, X_n \}$.
  - D is a set of domains, $\{ D_1, D_2 \ldots, D_n \}$, one for each variable.
  - C is a set of constraints that specify allowable combinations of values.
- Each domain D consists of a set of allowable values, $\{v_1, \ldots, v_k\}$ for X_i.
- Each constraint in C consists of a pair $\langle scope, rel \rangle$, where *scope* is a tuple of variables that participate in the constraint and *rel* is a relation that those variables can take on.

52

52

Constraint Satisfaction Problems (cont.)

- Suppose X={X_1, X_2} and D={{A,B},{A,B}}.
- We would like to have the constraint that X_1 and X_2 take different values.
- To do this we could set C={⟨(X_1, X_2),*rel*⟩} where rel is the relation {(A,B),(B,A)}.
- It is often convenient to use common abbreviations for well-known relations.
- I.e., we could write C as {⟨(X_1, X_2), $X_1 \neq X_2$ ⟩}.
- Notice the variables themselves are obvious from the relation, so we often abbreviate ⟨(X_1, X_2), $X_1 \neq X_2$ ⟩ further as just $X_1 \neq X_2$.

53

53

Constraint Satisfaction Problems (cont.)

- Constraint satisfaction is a search procedure that operates in a space of constraint sets.
- The initial state contains the constraints that are originally given in the problem description.
- The aim is to choose a value for each variable so that the resulting possible world satisfies the constraints. We want a model of the constraints.
- A problem is said to be solved when each variable has a value that satisfies all the constraints on the variable.

54

54

Constraint Satisfaction Problems (cont.)

- Constraint Satisfaction Problem Solution:
  - To solve a CSP we need to define a state space and the notion of a solution.
  - Each state in a CSP is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, ...\}$.
  - An assignment which does not violate any constraints is called a consistent or legal assignment.
  - A complete assignment is one in which every variable is assigned; an assignment which only assigns values to some of the variables is called a partial assignment.
  - A solution to a CSP is a complete, and consistent assignment.

55

55

Constraint Satisfaction Problems (cont.)

- Constraint Satisfaction is a two-step process:
  - First constraints are discovered and propagated as far as possible throughout the system.
  - Then if there is still not a solution, search begins. A guess about something is made and added as a new constraint.

56

56

Constraint Satisfaction Problems (cont.)

Constraint Satisfaction Algorithm

1. Propagate available constraints. To do this, first set *OPEN* to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until *OPEN* is empty:
   (a) Select an object *OB* from *OPEN*. Strengthen as much as possible the set of constraints that apply to *OB*.
   (b) If this set is different from the set that was assigned the last time *OB* was examined or if this is the first time *OB* has been examined, then add to *OPEN* all objects that share any constraints with *OB*.
   (c) Remove *OB* from *OPEN*.
2. If the union of the constraints discovered above defines a solution, then quit and report the solution.
3. If the union of the constraints discovered above defines a contradiction, then return failure.
4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:
   (a) Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.
   (b) Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

57

57

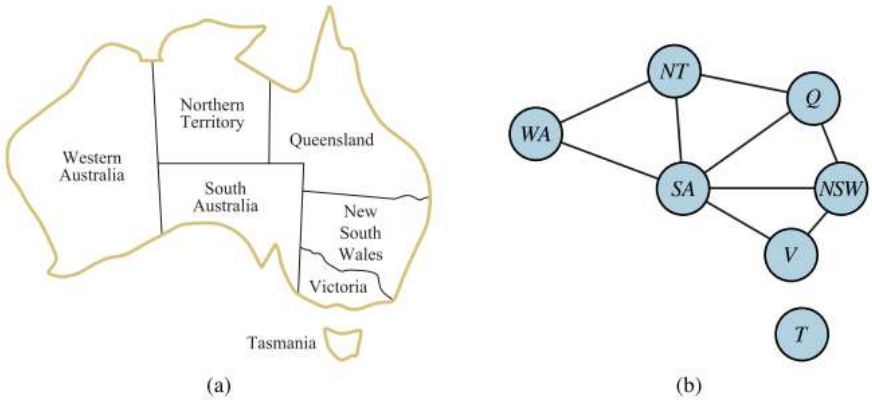
Heuristic Search Strategies

Examples Constraint Satisfaction Problems

58

58

CSP Example: Map-Coloring Problem



(a) The principal states and territories of Australia. Coloring this map can be viewed as a constraint satisfaction problem (CSP). The goal is to assign colors to each region so that no neighboring regions have the same color. (b) The map-coloring problem represented as a constraint graph.

59

59

CSP Example: Map-Coloring Problem (cont.)

- Australia consists of seven states and territories. Let's call a state or territory, a region.
- We are given the task of coloring each region red, green, or blue on a map in such a way that no neighboring regions have the same color.
- For this problem, X = {WA, NT, Q, NSW, V, SA, T}
- The domain D_i for each of these variables is {red, green, blue}.
- C = {SA≠WA,SA≠NT,SA≠Q,SA≠NW,SA≠V,WA≠NT,NT≠Q,Q≠NSW,NSW≠V}
- An example solution to the problem might be:

  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=red}.

60

60

CSP Example: Cryptarithmetic Problem

- A cryptarithmetic problem. Each letter stands for a different digit.

```
  SEND
+MORE
MONEY
```

- The aim is to find a substitution of digits for letters such that
- the resulting sum is arithmetically correct,
- with the added restriction that no leading zeros are allowed.

61

61

CSP Example: Cryptarithmetic Problem (cont.)

```
  SEND
+MORE
MONEY
```

- For this problem, X_i = {D, E, M, N, O, R, S, Y}.
- The domain D_i = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.
- (D + E = Y & C_1 = 0) or (D + E = 10 + Y & C_1 = 1)
  (N + R + C_1 = E & C_2=0) or (N + R + C_1 = 10 + E & C_2 = 1)
- The solution proceeds in cycles. At each cycle, two significant things are done:
  i. Constraints are propagated by using rules that correspond to the properties of arithmetic.
  ii. A value is guessed for some letter whose value is not yet determined.

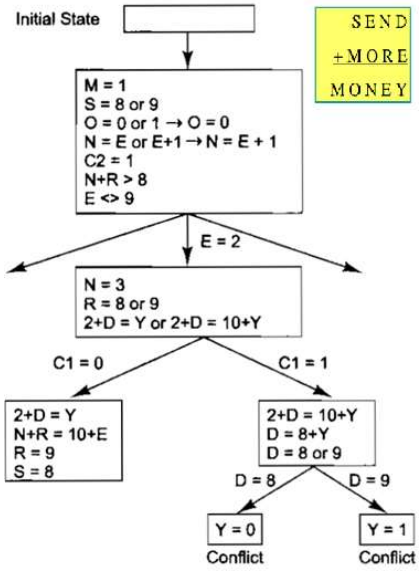
62

62

CSP Example: Cryptarithmetic Problem (cont.)

- An example solution to the cryptarithmetic problem might be:

$$\begin{array}{r} 9567 \\ +\ 1085 \\ \hline 10652 \end{array}$$



63

63

Heuristic Search Strategies

Local Search for Constraint Satisfaction Problems

64

64

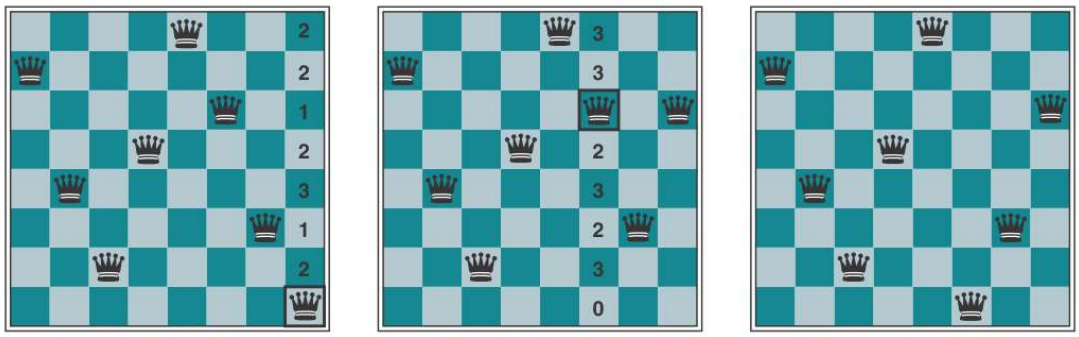
Local Search for Constraint Satisfaction Problems

- Local search algorithms turn out to be effective in solving many CSPs.
- They use a complete-state formulation: the initial state assigns a value to every variable, and the search changes the value of one variable at a time.
- For example, in the 8-queens problem, the initial state might be a random configuration of 8 queens in 8 columns, and each step moves a single queen to a new position in its column.
- Typically, the initial guess violates several constraints.
- The point of local search is to eliminate the violated constraints.

65

65

Minimum Conflicts Local Search Technique



A two-step solution using min-conflicts for an 8-queens problem. At each stage, a queen is chosen for reassignment in its column. The number of conflicts (in this case, the number of attacking queens) is shown in each square. The algorithm moves the queen to the min-conflicts square, breaking ties randomly.

66

66

Minimum Conflicts Local Search Technique (cont.)

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
          max_steps, the number of steps allowed before giving up

  current ← an initial complete assignment for csp
  for i = 1 to max_steps do
      if current is a solution for csp then return current
      var ← a randomly chosen conflicted variable from csp.VARIABLES
      value ← the value v for var that minimizes CONFLICTS(csp, var, v, current)
      set var = value in current
  return failure
```

The MIN-CONFLICTS local search algorithm for CSPs. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

67

67

Minimum Conflicts Local Search Technique (cont.)

- Min-conflicts is surprisingly effective for many CSPs.
- Amazingly, on the n-queens problem, if you don't count the initial placement of queens, the run time of min-conflicts is roughly independent of problem size.
- It solves even the million-queens problem in an average of 50 steps (after the initial assignment).
- Min-conflicts also works well for hard problems.
- For example, it has been used to schedule observations for the Hubble Space Telescope, reducing the time taken to schedule a week of observations from three weeks (!) to around 10 minutes.

68

68

Constraint Weighting Local Search Technique

- It can help concentrate the search on the important constraints.
- Each constraint is given a numeric weight, W_i, initially all 1.
- At each step of the search, the algorithm chooses a variable/value pair to change that will result in the lowest total weight of all violated constraints.
- The weights are then adjusted by incrementing the weight of each constraint that is violated by the current assignment.

69

69

Practice Problems

1. Provide a definition of the word "heuristic". In what ways can heuristics be useful in search? Name three ways in which you use heuristics in your everyday life.
2. What does it mean to say that a search method is monotonic? How desirable is this property?
3. Explain the components of the path evaluation function f(node) used by A*. Do you think it is the best evaluation function that could be used? To what kinds of problems might it be best suited? And to what kinds of problems would it be worst suited?
4. What is the drawback of greedy best-first search and how A* search strategy overcome this problem?
5. Describe A* search strategy with the help of following characteristics: admissible, heuristic, consistency, monotonicity, triangular inequality and optimality.
6. Discuss memory bounded heuristic search strategies.
7. Explain and compare hill climbing search algorithm and simulated annealing search algorithm.
8. Write a short note on local beam search and genetic algorithms.
9. What is constraint satisfaction problem? Write an algorithm for the solution of constraint satisfaction problem. Discuss and solved any two CSPs.
10. Justify the use of local search for constraint satisfaction problems.

70

70

Further Readings/References

1. Russel S. and Norvig P., “Artificial Intelligence: A Modern Approach”, 3rd Edition, Pearson Education, 2010.
2. Rich E. & Knight K., “Artificial Intelligence”, 3rd Edition, Tata McGraw-Hill Publishing Company Limited, 2008.
3. Dan W. Patterson, “Introduction to Artificial Intelligence Expert Systems”, Prentice Hall India New Delhi, 2006.
4. D. W. Rolston, “Principles of AI and Expert System Development”, Tata McGraw-Hill Publishing Company Limited, 2015.

71