Artificial Intelligence

CS482, CS682, MW 1 – 2:15, SEM 201, MS 227
Prerequisites: 302, 365
Instructor: Sushil Louis, sushil@cse.unr.edu, http://www.cse.unr.edu/~sushil
Informed Search

• Best First Search
  • A*
  • Heuristics

• Basic idea
  • Order nodes for expansion using a specific search strategy
    • Remember uniform cost search?
      • Nodes ordered by path length = path cost and we expand least cost
      • This function was called $g(n)$
  • Order nodes, $n$, using an evaluation function $f(n)$
  • Most evaluation functions include a heuristic $h(n)$
    • For example: Estimated cost of the cheapest path from the state at node $n$ to a goal state
    • Heuristics provide domain information to guide informed search
Romania with straight line distance heuristic

h(n) = straight line distance to Bucharest
Greedy search

- $F(n) = h(n) = \text{straight line distance to goal}$
- Draw the search tree and list nodes in order of expansion (5 minutes)

Time?
Space?
Complete?
Optimal?
Greedy search

(a) The initial state
(b) After expanding Arad
(c) After expanding Sibiu
(d) After expanding Fagaras
Greedy analysis

- Optimal?
  - Path through Rimniu Velcea is shorter

- Complete?
  - Consider Iasi to Fagaras
  - Tree search no, but graph search with no repeated states version → yes
    - In finite spaces

- Time and Space
  - Worst case $b^m$ where m is the maximum depth of the search space
  - Good heuristic can reduce complexity
A*

- $f(n) = g(n) + h(n)$
- = cost to state + estimated cost to goal
- = estimated cost of cheapest solution through $n$
Draw the search tree and list the nodes and their associated cities in order of expansion for going from Arad to Bucharest
5 minutes
A*
\( A^* \)

- \( f(n) = g(n) + h(n) \)
  - = cost to state + estimated cost to goal
  - = estimated cost of cheapest solution through \( n \)
- Seem reasonable?
  - If heuristic is \textit{admissible}, \( A^* \) is optimal and complete for Tree search
    - Admissible heuristics underestimate cost to goal
  - If heuristic is \textit{consistent}, \( A^* \) is optimal and complete for graph search
    - Consistent heuristics follow the triangle inequality
    - If \( n' \) is successor of \( n \), then \( h(n) \leq c(n, a, n') + h(n') \)
      - Is less than cost of going from \( n \) to \( n' \) + estimated cost from \( n' \) to goal
    - Otherwise you should have expanded \( n' \) before \( n \) and you need a different heuristic
  - \( f \) costs are always non-decreasing along any path
**A* contours**

- Non decreasing \( f \) implies
  - We can draw contours
  - Inside the 400 contour
    - All nodes have \( f(n) \leq 400 \)
  - Contour shape
    - Circular if \( h(n) = 0 \)
    - Elliptical towards goal for \( h(n) \)
- If \( C^* \) is optimal path cost
  - \( A^* \) expands **all** nodes with \( f(n) < C^* \)
  - \( A^* \) may expand some nodes with \( f(n) = C^* \) before getting to a goal state
  - If \( b \) is finite and all step costs > \( e \), then \( A^* \) is complete since
    - There will only be a finite number of nodes with \( f(n) < C^* \)
      - Because \( b \) is finite and all step costs > \( e \)
Pruning, IDA*, RBFS, MA/SMA

- A* does not expand nodes with $f(n) > C^*$
  - The sub-tree rooted at Timisoara is pruned
- A* may need too much memory
- **Iterative Deepening A* (IDA*)**
  - Iterative deepening using $f(n)$ to limit depth of search
  - Much less memory
  - Depth cutoff used: min $f(n)$ from prior step
- **Recursive Best First Search (RBFS)**
  - Best first search
  - Again uses $f(n)$ to limit depth
  - Whenever current $f(n) >$ next best alternative, explore alternative
  - Keep track of best alternative
- **Memory Bounded A* (MA) or Simple Memory Bounded A* (SMA)**
  - A* with memory limit
  - When memory limit exceeded drop worst leaf, and back up $f$-value to parent
  - Drops **oldest** worst leaf, and expands **newest** best leaf
Heuristic functions

• Some consistent heuristics are better than others

• Analysis
  • Consider the **effective** branching factor, $b^*$
  • The better the heuristic, the closer that $b^*$ is to 1
  • $N + 1 = 1 + b^* + (b^*)^2 + \ldots + (b^*)^d$
  • If $d = 5$, and $N = 52$, then $b^* = 1.92$

• There are techniques for generating admissible heuristics
  • Relax a problem
  • Learn from pattern database
Non-classical search

- Path does not matter, just the final state
- Maximize objective function
We have a black box “evaluate” function that returns an objective function value.
Local Hill Climbing

```python
function HILL-CLIMBING(problem) returns a state that is a local maximum

    current ← MAKE-NODE(problem.INITIAL-STATE)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.VALUE ≤ current.VALUE then return current.STATE
        current ← neighbor
```

• Move in the direction of increasing value
• Very greedy
• Subject to
  • Local maxima
  • Ridges
  • Plateaux
• 8-queens: 86% failure, but only needs 4 steps to succeed, 3 to fail
Hill climbing

- Keep going on a plateau?
  - Advantage: Might find another hill
  - Disadvantage: infinite loops → limit number of moves on plateau
    - 8 queens: 94% success!!
- Stochastic hill climbing
  - randomly choose from among better successors (proportional to obj?)
- First-choice hill climbing
  - keep generating successors till a better one is generated
- Random-restarts
  - If probability of success is $p$, then we will need $1/p$ restarts
  - 8-queens: $p = 0.14 \sim 1/7$ so 7 starts
  - 6 failures (3 steps), 1 success (4 steps) = 22 steps
  - In general: Cost of success + $(1-p)/p \times$ cost of failure
  - 8-queens sideways: 0.94 success in 21 steps, 64 steps for failure
    - Under a minute
Simulated annealing

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
    inputs: problem, a problem
             schedule, a mapping from time to “temperature”

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

- Gradient descent (not ascent)
- Accept bad moves with probability $e^{ΔE/T}$
- $T$ decreases every iteration
- If $schedule(t)$ is slow enough we approach finding global optimum with probability 1
Genetic Algorithms

- Stochastic hill-climbing with information exchange
- A population of stochastic hill-climbers

```plaintext
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

repeat
  new_population ← empty set
  for i = 1 to SIZE(population) do
    x ← SELECTION(population, FITNESS-FN)
    y ← SELECTION(population, FITNESS-FN)
    child ← REPRODUCE(x, y)
    if (small random probability) then child ← MUTATE(child)
    add child to new_population
  population ← new_population
until some individual is fit enough, or enough time has elapsed
return the best individual in population, according to FITNESS-FN

function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals
  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```
More detailed GA

- Generate pop(0)
- Evaluate pop(0)
- T=0
- While (not converged) do
  - Select pop(T+1) from pop(T)
  - Recombine pop(T+1)
  - Evaluate pop(T+1)
  - T = T + 1
- Done
Generate pop(0)

Initialize population with randomly generated strings of 1’s and 0’s

```plaintext
for(i = 0 ; i < popSize; i++){
    for(j = 0; j < chromLen; j++){
        Pop[i].chrom[j] = flip(0.5);
    }
}
```
Genetic Algorithm

- Generate pop(0)
- Evaluate pop(0)
- T=0
- While (not converged) do
  - Select pop(T+1) from pop(T)
  - Recombine pop(T+1)
  - Evaluate pop(T+1)
  - T = T + 1
- Done
Evaluate pop(0)

Decoded individual \rightarrow \text{Evaluate} \rightarrow \text{Fitness}

Application dependent fitness function
Genetic Algorithm

- Generate pop(0)
- Evaluate pop(0)
- T=0
- While (T < maxGen) do
  - Select pop(T+1) from pop(T)
  - Recombine pop(T+1)
  - Evaluate pop(T+1)
  - T = T + 1
- Done
Genetic Algorithm

• Generate pop(0)
• Evaluate pop(0)
• T=0
• While (T < maxGen) do
  • Select pop(T+1) from pop(T)
  • Recombine pop(T+1)
  • Evaluate pop(T+1)
  • T = T + 1
• Done
Selection

- Each member of the population gets a share of the pie proportional to fitness relative to other members of the population
- Spin the roulette wheel pie and pick the individual that the ball lands on
- Focuses search in promising areas
int roulette(IPTR pop, double sumFitness, int popsize)
{
    /* select a single individual by roulette wheel selection */
    double rand, partsum;
    int i;

    partsum = 0.0; i = 0;
    rand = f_random() * sumFitness;

    i = -1;
    do{
        i++;
        partsum += pop[i].fitness;
    } while (partsum < rand && i < popsize - 1) ;

    return i;
}
Genetic Algorithm

- Generate pop(0)
- Evaluate pop(0)
- $T=0$
- While ($T < \text{maxGen}$) do
  - Select pop($T+1$) from pop($T$)
  - Recombine pop($T+1$)
  - Evaluate pop($T+1$)
  - $T = T + 1$
- Done
Crossover and mutation

Mutation Probability = 0.001
Insurance

Xover Probability = 0.7
Exploration operator
void crossover(POPULATION *p, IPTR p1, IPTR p2, IPTR c1, IPTR c2) 
{
/* p1,p2,c1,c2,m1,m2,mc1,mc2 */
 int *pi1,*pi2,*ci1,*ci2;
 int xp, i;

 pi1 = p1->chrom;
 pi2 = p2->chrom;
 ci1 = c1->chrom;
 ci2 = c2->chrom;

 if(flip(p->pCross)){
     xp = rnd(0, p->lchrom - 1);
     for(i = 0; i < xp; i++){
         ci1[i] = muteX(p, pi1[i]);
         ci2[i] = muteX(p, pi2[i]);
     }
     for(i = xp; i < p->lchrom; i++){
         ci1[i] = muteX(p, pi2[i]);
         ci2[i] = muteX(p, pi1[i]);
     }
 } else {
     for(i = 0; i < p->lchrom; i++){
         ci1[i] = muteX(p, pi1[i]);
         ci2[i] = muteX(p, pi2[i]);
     }
 }
Mutation code

```c
int muteX(POPULATION *p, int pa)
{
    return (flip(p->pMut) ? 1 - pa : pa);
}
```
Search

- Problem solving by searching for a solution in a space of possible solutions
- Uninformed versus Informed search
- Atomic representation of state
- Solutions are fixed sequences of actions