IS/SE/SS ZC444: Artificial Intelligence

08 Genetic Algorithms

Dr. Kamlesh Tiwari
Associate Professor, Department of CSIS,
BITS Pilani, Pilani Campus, Rajasthan-333031 INDIA

Oct 01, 2023 ONLINE WILP @ BITS-Pilani [July-Dec 2023]

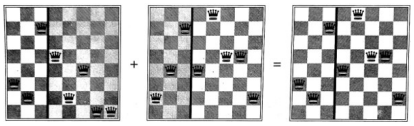
http://ktiwari.in/ai

Genetic Algorithms

Evolution

- Recall Darwin's theory of evolution: "*Survival of the fittest*" [1]

Is it true? Let's formulate and try..



[1] Images taken from various sources on Internet

Genetic Algorithms

Learning approach of Genetic algorithms is based on simulated evolution (appeared in 1975)

- State are represented using fixed length bit strings (chromosome)
- Search for a goal state begins with a population of initial states
  - Members of the current population give rise to the next generation population using random **mutation** and **crossover**
  - States are evaluated using some **fitness** measure
  - Most fit state act as a seeds for producing next generation
- Applied a variety of learning tasks and optimization problems (like robot control and learning parameters for ANN)
- Search can move abruptly. Crowding can happen

Without **guarantee**, GA often finds an object of high fitness

Genetic Algorithms

Algorithm 1: GA ($\texttt{Fitness}$, F_{th}, p, r, m)

```
P ← generate p states at random
while max(Fitness(h1), Fitness(h2), ..., Fitness(hp)) < Fth do
    Select: (1 − r)p members of P
    Crossover: on (r × p)/2 pairs to produce two offspring
    Mutation: randomly invert a bit of m percentage of population
return state hi having maximum Fitness(hi)
```

- Fitness function is typically a heuristic
- The fitness function is a criterion for ranking states to select states probabilistically for inclusion in the next generation population

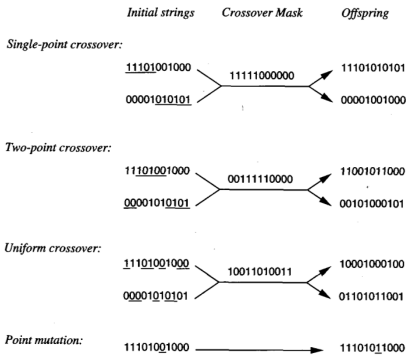
Selection

- **Fitness proportionate**. Probability of selecting a state in next generation is

$$Pr(h_i) = \frac{\texttt{Fitness}(h_i)}{\sum_{j=1}^{p} \texttt{Fitness}(h_j)}$$

- **Tournament selection**. randomly pick two states and then with some predefined probability p the more fit of these two is then selected, and with probability $(1 - p)$ the less fit state is selected
- **Rank selection**. states are sorted by fitness and the probability of selection of a state is proportional to its rank in this sorted list, rather than its fitness
- **Elitist Model**. select a small proportion of the fittest candidates in current population intact into the next generation

Genetic Operators (crossover and mutation)



Does GA works?

- Can we mathematically characterize the evolution
- **Schema:** string of 0, 1 or $*$ like $0*1$ denoting set $\{001, 011\}$
- String 1011 is **representative** of 2^4 schema
- Let $m(s,t)$ be number of instances of schema s in generation t
- Consider *selection*, let fitness of individual h be $f(h)$ and average fitness of whole n size population at time t be $\bar{f}(t)$
- $h \in s \cup p_t$ means 1) h is representative of s and 2) it is present in the population at time t
- Let $\hat{u}(s,t)$ be average fitness of instances of s at time t

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cup p_t} f(h)}{m(s,t)}$$

- We know $Pr(h) = f(h)/(\sum f(h)) = f(h)/(n\bar{f}(t))$

What is artificial intelligence?

Does GA works? (contd..)

- Probability that we will select a representative of schema s is

$$\begin{aligned} Pr(h \in s) &= \sum_{h \in s \cup p_t} Pr(h) = \sum_{h \in s \cup p_t} f(h)/(n\bar{f}(t)) \\ &= \frac{\hat{u}(s,t)}{n\bar{f}(t)} m(s,t) \end{aligned}$$

- Expected number of instances of s resulting from the n independent selection steps that create the entire new generation is just n times this probability. Therefore,

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Does GA works? (contd..)

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)$$

Expected number of representative instances of a schema s in the generation at time $t+1$ is

1. Proportional to the average fitness $\hat{u}(s,t)$ of instances of this schema at time t , and
2. Inversely proportional to the average fitness $\bar{f}(t)$ of all members of the population at time t

Thus, we can expect schema with above average fitness to be represented with increasing frequency on successive generations

Does GA works? (contd..)

Also consider negative effects of single point crossover and mutation

- Let p_c represents the probability of **crossover** on an individual. $d(s)$ be the distance between left most and right most defined bit of s and l be the length of individual bit string
- Let p_m represents the probability of **mutation** on an individual and $o(s)$ be number of defined bits in s

Full schema theorem thus provides a lower bound on the expected frequency of schema s, as follows

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} m(s,t)(1 - p_c \frac{d(s)}{l-1})(1 - p_m)^{o(s)}$$

Similar expression. More fit schemas will tend to grow in influence.

Local Search in Continuous State

Issue: Number of next states (branching factor) becomes infinite

Example: Induct three new airports in Romania

- Let at (x_1, y_1), (x_2, y_2) and (x_3, y_3) on the map
- Minimize sum of distances of all the cities from its nearest airport

$$f(x_1, y_1, x_2, y_2, x_3, y_3) = \sum_{i=1}^{3} \sum_{c \in C_i} ((x_i - x_c)^2 + (y_i - y_c)^2)$$

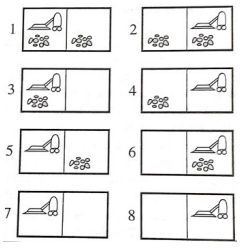
- If you discretize the neighborhood then there are only 12 next state (move only $\pm\delta$ in one step). One can apply hill climbing.
- If you attempt to use gradient $\nabla f = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial y_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial y_2}, \frac{\partial f}{\partial x_3}, \frac{\partial f}{\partial y_3})$ it cannot be solved as globally finding ∇f is not possible.
- Given locally correct values of $\frac{\partial f}{\partial x_1} = 2\sum_{c \in C_1}(x_i - x_c)$ one can perform steepest-ascent using $x \leftarrow x + \alpha \nabla f$

Search with Non-Deterministic Actions

Non-Deterministic: not sure what would be the next state[2]

Consider erratic vacuum world

sometime 1) also cleans neighboring room 2) deposit dirt



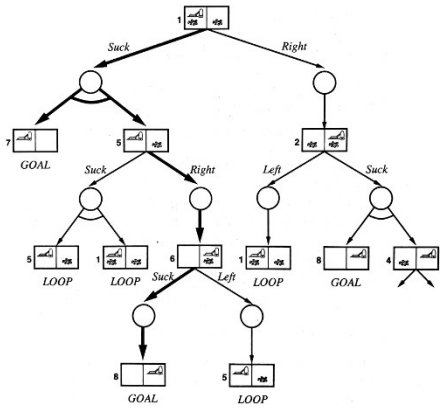
- Transition would lead use to more than one state
- *suck* in 1, would lead {5,7}
- Solution would have nested if-else

[*suck*, **if** state=5 **then** [*right*,*suck* **else** []]

- Search tree would contain some OR nodes and some AND nodes

[2] Percepts would tell where have we reached.

AND-OR Search Tree



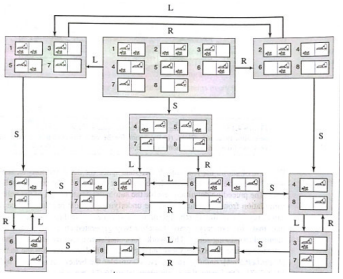
Solution

1. has goal node at every leaf
2. takes one action at each OR node
3. includes every outcome branch at each AND node

Searching with Partial Observations

When percepts do not suffice to pin down the exact state

- **Sensor less**. consider [*right,suck,left,suck*] guarantees to reach in state 8 that is a goal state (traverses through belief states)
- All possible belief states may not be reachable (only 12 out of 2^8)

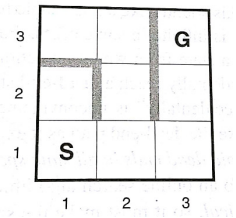


Online Search and Unknown Environment

Agent interleaves computation and action

Take action → observe environment → compute next action

- Online Search is necessary for unknown environment



1. Consider following maze problem
2. A robot need to go from S to G
3. Knows nothing about the environment

Random-walk?

No algorithm can avoid dead-end in all state space

Adversarial Search (game)

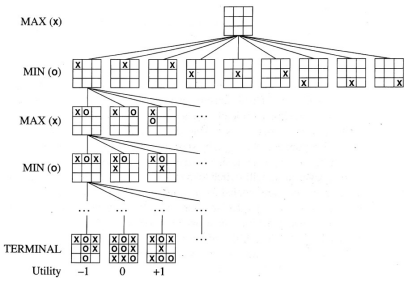
Agents having conflicting goals in competitive multiagent environment

- Deterministic, fully-observable, turn-taking, two-player, zero-sum
- Chess has roughly branching factor 35, moves 50 so tree search space is $35^{100} = 10^{154}$ however, graph has 10^{40} nodes
- Finding optimal move is infeasible but, needs an ability to decide

Game is between MAX and MIN (MAX moves first)

- S_0: the initial state
- PLAYER(s): defines which player has move to start
- ACTIONS(s): returns set of legal moves in a state
- RESULT(s, a):termination model defining result of a move
- TERMINAL_TEST(s): is true when game is over
- UTILITY(s, p): utility function defining reward (for chess +1,0,1/2)

Game Tree for tic-tac-toe

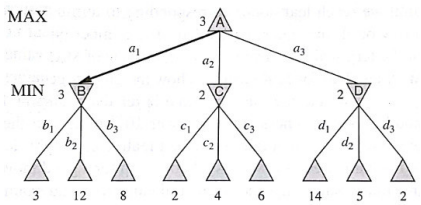


The search tree of the game has less than 9! = 362880 nodes.

MAX must find a contingent **strategy**.

Analogous to AND-OR search (MAX plays OR and MIN plays AND)

Two half moves is one **ply**



Given the game tree, optimal strategy can be determined from **minimax value** of each node.

$$MINIMAX(s) = \begin{cases} UTILITY(s) & \text{if } TERMINAL_TEST(s) \\ argmax_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MAX \\ argmin_{a \in Actions(s)} MINIMAX(RESULT(s,a)) & \text{if } PLAYER(s) = MIN \end{cases}$$

Action a_1 is the optimal choice [3]

(essentially optimizing worst-case outcome for MAX)

[3] utility value for MAX of being in corresponding state (assuming then onwards both player play optimally)

MINIMAX Algorithm

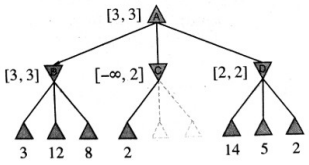
Returns the action corresponding to best move

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

Recursion proceeds all the way down to the leaves. Time complexity $O(b^m)$ that is impractical but provides a basis of solution.

ALPHA-BETA Pruning

- Number of nodes to examine in minimax search is exponential in the depth of tree $O(b^m)$.
- Sometime we can make it $O(b^{m/2})$ using **alpha-beta pruning**
- When applied to standard minimax tree, it returns the same move as minimax but, prunes away branches that cannot possibly influence the decision.

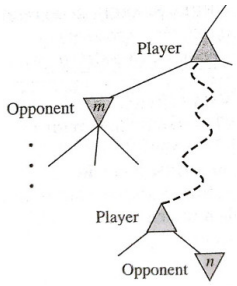


Consider two unevaluated successor of node C have value x and y

MINIMAX(root)
= max(min(3,12,8), min(2,x,y), min(14,5,2))
= max(3, min(2,x,y), 2)
= max(3, z, 2) where z=min(2,x,y)≤ 2
= 3

ALPHA-BETA Pruning

- Alpha-beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtree rather than just leaves.



If m is better than n for player then we would never go to n in play

α	=	value of best choice (highest) found so far for MAX
β	=	value of best choice (lowest) found so far for MIN

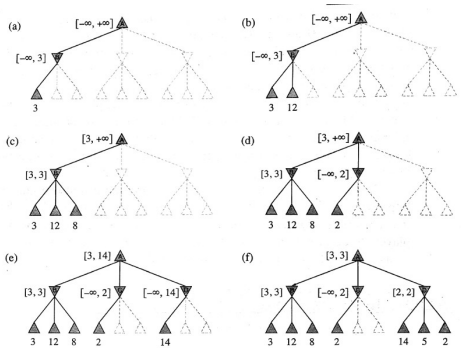
ALPHA-BETA Pruning

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v

function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Order matters. So, examine likely to be best successor first.

Is it possible?
No

In-action: ALPHA-BETA Pruning



Move Ordering

- With perfect ordering we need to examine only $O(b^{m/2})$ nodes
- When successors are examined in random order it needs to examine $O(b^{3m/4})$ nodes
- In chess, a strategy that capture → threat → forward → backward, gets you to within about a factor of 2 of the best case $O(b^{m/2})$
- Try first the move that were found useful in past (killer move)
- Iterative deepening could help (adds constant fraction time)
- Hash table of previously seen positions (**transposition table**) can restrict re-computation of states

Imperfect Real-time Decision

- Generating entire search space is overkill
- Covering till large depth is not possible
- Idea is to cutoff the search earlier

$$H\text{-}MINIMAX(s,d) = \begin{cases} Eval(s) & \text{if } CUTOFF_TEST(s,d) \\ argmax_{a \in Actions(s)} H\text{-}MINIMAX(RESULT(s,a), d+1) & \text{if } PLAYER(s) = MAX \\ argmin_{a \in Actions(s)} H\text{-}MINIMAX(RESULT(s,a), d+1) & \text{if } PLAYER(s) = MIN \end{cases}$$

Evaluation Function

- Returns the estimate of expected utility
- Performance would deeply depend on it
- Eval(win) $\geq$ Eval(draw) $\geq$ Eval(lose)
- Computation should be quick
- Value should relate to actual chance of winning
- Can use features (Number of pawns, Number of queens, ...)
- Can have various categories (all pawn vs one pawn, etc) Suppose experience suggests 72% of states encountered in the two-pawn vs. one pawn category lead to win, 20% lose, and 8% in draw. then eval could be

$$0.72 \times 1 + 0.20 \times 0 + 0.08 \times 0.5 = 0.76$$

- **Weighted linear function** are also possible

$$Eval(s) = w_1 \times f_1(s) + w_2 \times f_2(s) + w_3 \times f_3(s) + ...$$

Important Considerations

- **Cutting off search**: having fixed depth may not be a good idea. **Quiescent** states are unlikely to have wild swing so expand till it. **Horizon effect** is difficult to eliminate so **singular extension**[4] could be used
- **Forward Pruning**: cutting off the some search without further consideration. (we do not evaluate all possibilities **beam search**, we only use some in our mind) This may be fatal.
- **Search Vs Lookup** We have some policy on how to start and finish well. Use them as lookup

[4] can I take a previous good move

Thank You!

Thank you very much for your attention!

Queries ?

(Reference[5])

[5] 1) Book - *AIMA*, ch-03+04, Russell and Norvig. 2) Book - Machine Learning, ch-09, *Tom Mitchell*