CS3243 **AY20/21 Sem1**

Lecture 5: September 9

Lecturer: MEEL, Kuldeep S. *Scribes: Song Qifeng*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

5.1 $\alpha - \beta$ *Search*

5.1.1 Algorithm

Algorithm 1 – Pruning Strategy

```
function MaxValue(state, α, β)
    if Terminal(state) == True then return Utility(state)
    v ← -∞
    for each a in Actions(state) do
        result ← RESULT(state, a)
        v ← max(v, MinValue(result, α, β))
        if v ≥ β then return v
        α ← max(α, v)
    return v

function MinValue(state, α, β)
    if Terminal(state) == True then return Utility(state)
    v ← +∞
    for each a in Actions(state) do
        result ← RESULT(state, a)
        v ← min(v, MaxValue(result, α, β))
        if v ≤ β then return v
        α ← min(α, v)
    return v
```

5.1.2 Analysis

For a game tree of depth d and branch factor b,

Minimax: $O(b^d)$

$\alpha - \beta$: $O(b^{d/2})$ (best case)

5.2 Real-time decisions

5.2.1 Evaluation functions

Although $\alpha - \beta$ search reduces workload by pruning some branches, it still needs to reach terminals, which can be very consuming when the game tree has large depth.

To solve this problem, we modify the $\alpha - \beta$ search.

$$H - Minimax(s, d) = \begin{cases} EVAL(s), & \text{if } CUTOFF - TEST(s, d) \\ max_{a \in Actions(s)} \ H - Minimax(result(s, a), d + 1), & \text{if } player(s) = MAX \\ min_{a \in Actions(s)} \ H - Minimax(result(s, a), d + 1), & \text{if } player(s) = MIN \end{cases}$$

This method modifies $\alpha - \beta$ search in two ways:

- replace the utility function by a heuristic evaluation function **EVAL**, which estimates the position's utility
- replace the terminal test by a **cutoff test** that decides when to apply **EVAL**

Here we introduce one kind of evaluation function called **weighted linear function**

$EVAL(s) \ = \ w_1 * f_1(s) \ + \ w_2 * f_2(s) \ + \ \ldots \ + \ w_n * f_n(s) \ = \ \Sigma w_i * f_i(s)$

where each w_i is a weight and each f_i is a feature of the position.

5.3 Local search

*"**Local search** algorithms operate using a single **current node** and generally move only to neighbors of that node."* - AIMA

5.3.1 Hill-climbing search

Algorithm 2 Hill-climbing search

```
function HILL-CLIMBING(state s) returns a state that is a local best
    minVal ← Val(s)
    minState ←
    for u in N(s) do
        if Val(u) < minVal then minState = u, minVal = Val(u)
    return minState
```

Drawback of this approach:

This algorithm may get stuck at local maximum.

5.3.2 N-Queens Puzzle

On a N*N chess board, place N queens in such a way that none of them attack each other.

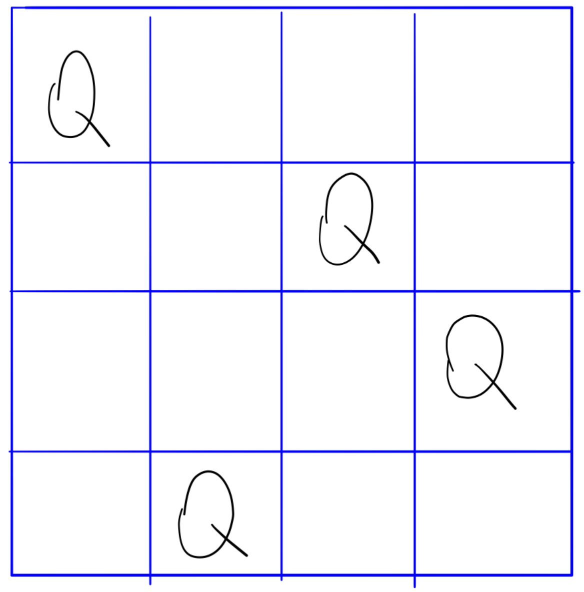

Figure 5.1: Hill-climbing search gets stuck in this non-optimal state

However, if we allow the algorithm to tolerate some regressions, then it is likely to get out of local maximum, as shown below.

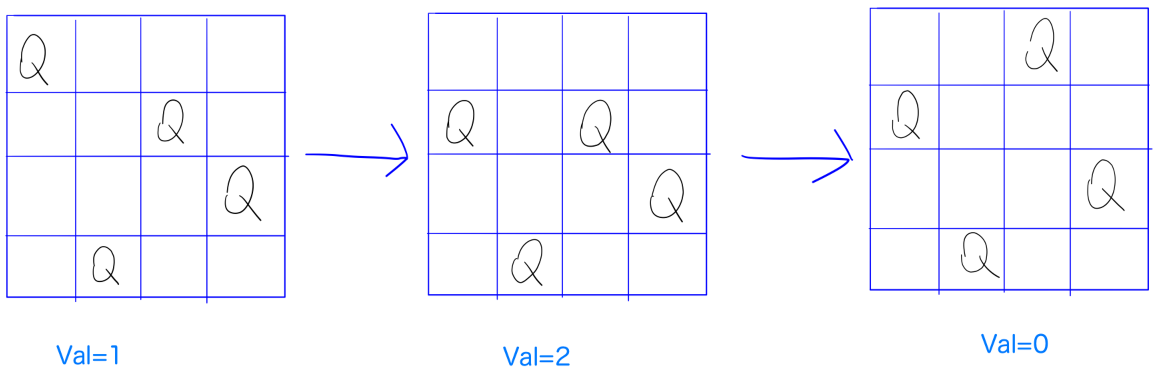

Figure 5.2: A tolerable algorithm can jump out of local maximum