# Introduction to Artificial Intelligence
# Homework 4

Wang Rongqing
2015K8009929046

September 25, 2017

## Answer Sheet

Prove that

**Theorem 1** *NegaScout is correct.*

Before proving, we must define what **correctness** is.

We take the definition from the zero-sum quality of the game, and the assumption that each player tries its best to win more. Therefore we can define the correct value function $\textsc{Correct}(s)$ as follow.

**Definition 1** *The correct value function* $\textsc{Correct}(s)$ *is defined as:*

$$\textsc{Correct}(s) = \begin{cases} \textsc{Utility}(s), & \textsc{TerminalTest}(s) \quad (1a) \\ \max\{-\textsc{Correct}(\textsc{Result}(s,a)) | a \in \textsc{Action}(s)\}, & \textit{otherwise.} \quad (1b) \end{cases}$$

Notice that the searching tree has no loop. Otherwise "correctness" itself is not well-defined. Also "correctness" can be defined on a looped search tree by replacing the loop nodes into terminal nodes upon second entering, effectively removing loops in the searching tree. Therefore, **no loop** is one of the assumptions.

And the theorem we need to prove is:

**Theorem 2** *For any* $s$ *in the searching tree:*

$$\textsc{NegaScout}(s,\alpha,\beta) \begin{cases} \in [\textsc{Correct}(s), \alpha], & \textsc{Correct}(s) \le \alpha \quad (2a) \\ = \textsc{Correct}(s), & \alpha < \textsc{Correct}(s) < \beta \quad (2b) \\ \in [\beta, \textsc{Correct}(s)], & \textsc{Correct}(s) \ge \beta \quad (2c) \end{cases}$$

Once we have proved the theorem, we can get the correctness of NegaScout algorithm as a corollary:

**Corollary 1**

$$\textsc{NegaScout}(s_0, -\infty, +\infty) = \textsc{Correct}(s_0) \tag{3}$$

*which is that: using NegaScout to search from the starting node* $s_0$ *will yield the correct result.*

For the ease of description, below is the algorithm (rephrased, equivalent):

NegaScout$(s, \alpha, \beta)$, $d$ is null-window width, typically no larger than any $\beta - \alpha$

```
              if TerminalTest(s)
(leaf)            return Utility(s)
              v = −∞
(loop)        for a ∈ Action(s)
                  s′ = Result(s, a)
                  if a is the first action or β − α < d
(first-search)        result = −NegaScout(s′, −β, −α)
                  else
(scout)               result = −NegaScout(s′, −α − d, −α)
                      if α < result < β
(full-research)           result = −NegaScout(s′, −β, −α)
(too-good)        if result ≥ β return result
                  v = max(v, result)
(update-α)        α = max(α, v)
(all-searched) return v
```

Now we prove theorem 2.

**Proof 1** *We use induction on rooted searching tree $T(V, E)$. We can use induction since the tree has no loop.*

- *For all leaves $l \in L = \{s \in V | \textsc{TerminalTest}(s)\}$ on the tree, $\textsc{NegaScout}(l, \alpha, \beta)$ will go to (leaf) line directly. Then for $l$, we have, from* (1a)*, that*

$$\textsc{NegaScout}(l, \alpha, \beta) = \textsc{Utility}(l) = \textsc{Correct}(l). \tag{4}$$

- *For all other nodes on the tree, suppose we have known that, for all of its sub-nodes, the aforementioned theorem holds. Then, the algorithm will stop at* (too-good) *or* (all-searched)*.*

  *We denote the value of $\alpha$ at the entry $\alpha_0$, and the value after the $i$th loop in* (loop) *to be $\alpha_i$, and we denote that $m := |\textsc{Action}(s)|$. Notice that* (update-$\alpha$) *in the algorithm guarantees that the series $\{\alpha_0, \alpha_1, \cdots, \alpha_m\}$ is non-descending.*

  - *If the algorithm stopped at* (too-good)*, then either* (first-search)*,* (scout)*, or* (full-research) *returned a result larger than $\beta$. Let $s_i$ be the offending result state of $s$.*

    * *If result is from* (first-search) *or* (full-research)*, then we can have directly*

    $$\textsc{NegaScout}(s_i, -\beta, -\alpha_{i-1}) \le -\beta, \tag{5}$$

    *by* (2a) *in induction hypothesis, we have*

    $$\textsc{Correct}(s_i) \le \textsc{NegaScout}(s_i, -\beta, -\alpha_{i-1}) \le -\beta. \tag{6}$$

    * *If result is from* (scout)*, then we will have*

    $$\textsc{NegaScout}(s_i, -\alpha_{i-1} - d, -\alpha_{i-1}) \le -\beta. \tag{7}$$

    *However, by induction hypothesis* (2a) *we can have that*

    $$\textsc{Correct}(s_i) \le \textsc{NegaScout}(s_i, -\alpha_{i-1} - d, -\alpha_{i-1}) \le -\beta, \tag{8}$$

*i.e.*

$$\text{Correct}(s_i) \le -\beta. \tag{9}$$

*Therefore, by induction hypothesis* (2a) *we can have*

$$\text{Correct}(s_i) \le \text{NegaScout}(s_i, -\beta, -\alpha_{i-1}) \le -\beta. \tag{6}$$

*In consequence, in both cases,* $\text{Correct}(s_i) \le \text{NegaScout}(s_i, -\beta, -\alpha_{i-1}) \le -\beta$. *Therefore,*

$$\begin{aligned} \text{Correct}(s) &\ge -\text{Correct}(s_i) \\ &\ge -\text{NegaScout}(s_i, -\beta, -\alpha_{i-1}) \\ &= \text{NegaScout}(s, \alpha, \beta) \\ &\ge \beta, \end{aligned} \tag{10}$$

*which is* (2c).

- *If the algorithm stopped at* (all-searched)*, then all results yielded in* (first-search)*,* (scout)*, or* (full-research) *is less than* $\beta$.
  - *If* $v = \alpha_m > \alpha$, *then there must exist* $s_i$ *in* (first-search) *and* (full-search) *that yields the* $v$. *By induction hypothesis the* $v$ *is correct itself, i.e.*

    $$\text{NegaScout}(s_i, -\beta, -\alpha_{i-1}) = \text{Correct}(s_i). \tag{11}$$

    *(*$\text{NegaScout}(s_i, -\beta, -\alpha_{i-1}) < -\alpha_{i-1}$, *since otherwise* $s_i$ *is not the best state found)*
    *For other* $s_j$, *either its calculated value is no higher than* $-\alpha_{j-1}$, *in which case*

    $$\begin{aligned} \text{Correct}(s_j) &= \text{NegaScout}(s_j, -\beta, -\alpha_{j-1}) \\ &\ge \text{NegaScout}(s_i, -\beta, -\alpha_{i-1}) = \text{Correct}(s_i), \end{aligned} \tag{12}$$

    *or it is higher than* $-\alpha_{j-1}$, *in which case*

    $$\begin{aligned} \text{Correct}(s_j) &\ge \text{NegaScout}(s_j, -\beta, -\alpha_{j-1}) \ge -\alpha_{j-1} \\ &\ge -\alpha_m = \text{NegaScout}(s_i, -\beta, -\alpha_{i-1}) = \text{Correct}(s_i). \end{aligned} \tag{13}$$

    *Considering both cases, hence, by* (1b)*,*

    $$\text{NegaScout}(s, \alpha, \beta) = -\text{Correct}(s_i) = \max_{1 \le j \le m} \{-\text{Correct}(s_j)\} = \text{Correct}(s), \tag{14}$$

    *which is* (2b).
  - *If* $v \le \alpha$, *that is,* $\alpha$ *is not updated in the whole for loop, i.e.* $\alpha = \alpha_0 = \alpha_1 = \cdots = \alpha_m$, *then by definition, for all* $s_j$ *occurred, either*

    $$\text{NegaScout}(s_j, -\alpha - d, -\alpha) \ge -\alpha, \tag{15}$$

    *or*

    $$\text{NegaScout}(s_j, -\beta, -\alpha) \ge -\alpha, \tag{16}$$

    *and both are equivalent and can deduce to each other (although* $\text{NegaScout}(s_j, -\alpha - d, -\alpha)$ *does not necessarily equal* $\text{NegaScout}(s_j, -\beta, -\alpha)$*), since both can be deduced to and from* $\text{Correct}(s_j) \ge -\alpha$ *by induction hypothesis* (2c)*. Therefore, we have*

    $$\text{Correct}(s_j) \ge \text{NegaScout}(s_j, -\beta, -\alpha) \ge -\alpha, \tag{17}$$

    *i.e. for all* $s_j$ *occurred,*

    $$-\text{Correct}(s_j) \le -\text{NegaScout}(s_j, -\beta, -\alpha) \le \alpha. \tag{18}$$

*Suppose*

$$\textsc{NegaScout}(s, \alpha, \beta) = -\textsc{NegaScout}(s_{i_1}, *, -\alpha), \tag{19}$$

*and*

$$\textsc{Correct}(s) = \max_{1 \le j \le m} \{-\textsc{Correct}(s_j)\} = -\textsc{Correct}(s_{i_2}). \tag{20}$$

*Then, we will have*

$$\begin{aligned}
\textsc{Correct}(s) &= -\textsc{Correct}(s_{i_2}) \\
&\le -\textsc{NegaScout}(s_{i_2}, *, -\alpha) \\
&\le -\textsc{NegaScout}(s_{i_1}, *, -\alpha) \\
&= \textsc{NegaScout}(s, \alpha, \beta) \\
&\le \alpha,
\end{aligned} \tag{21}$$

*which is* (2a).

□

Notice that the proof itself shows that, even if you *use* null-window in the first action, the correctness is ensured. However, the design of the algorithm assumes that the first action is the best, so it does not waste time to do scout and directly goes to (first-search). Searching after the first node is only used to verify whether the first action is actually the best, therefore only when (scout) finds out $\alpha < result < \beta$, namely that the action is actually better than the previously found best (i.e. the current $\alpha$ value) will the algorithm search more.