

Recitation 3: Optimal Search and Search in Games (Notes)¹

September 30, 2005

1. Optimal Search

(a) Methods

To obtain the pseudocode for the optimal search algorithms based on *branch-and-bound* we just need to make minor modifications to that for basic search.

Also, just as in informed/heuristic basic search, we can use a heuristic function $h(n)$ over the nodes n of the graph as an estimate of the cost to the goal from n .

There are different versions of branch-and-bound for optimal search depending on whether or not one uses a heuristic and whether one uses an extended list. The A^* algorithm is simply *branch-and-bound with a heuristic and an extended list*. To implement branch-and-bound using the pseudocode for basic search, we

- i. add the extended paths anywhere in the queue, and
- ii. pick the partial path N with
 - A. the minimum cost of N , if we are *not using a heuristic*, or
 - B. the minimum sum of the cost of N and $h(\text{tail}(N))$, if we are *using a heuristic*.

To *guarantee* that branch-and-bound using just a heuristic (without an extended list) outputs an optimal path, it is *sufficient* that the heuristic values of the nodes in at least one optimal path do not overestimate their cost of the optimal path from the node to the goal. (Show that as an exercise.) The basic intuition behind this condition on the heuristic values is that they do not allow us to miss a shorter path to the goal because we have overestimated its costs due to an overestimate in one of the nodes in the optimal path. Since we do not know *a priori* what the optimal path is (that is precisely what we want to compute), we need to check that every node in a path to the goal satisfies that condition. A heuristic is *admissible* if for *every* node n in the graph, $h(n)$ never overestimates the cost of the optimal path from n to the goal.

However, using an admissible heuristic is not enough to guarantee that A^* will work; we actually need a stronger condition. To *guarantee* that A^* outputs an optimal path, it is *sufficient* that there exists at least one optimal path to the goal with nodes whose heuristic values satisfy the following condition: (*) if n and m are two consecutive nodes in the path and $c(n, m)$ is the cost of the (directed) edge from n to m , then $h(n) \leq c(n, m) + h(m)$. (See Part 3 of this notes for a proof.) Once again, since we do not know the optimal path, we must check that every path to the goal satisfies that condition. A heuristic is *consistent* if condition (*) holds for every directed edge $n \rightarrow m$ in the graph.²

A consistent heuristic is admissible, but the opposite is not always true. (Show that as an exercise). Consistency is also easier to check than admissibility, since we only need to

¹These notes are primarily based on the recitation notes of Kimberle Koile and the slides of Tomás Lozano-Perez from previous terms, as well as the books *Artificial Intelligence (Third Edition)* by Patrick Henry Winston and *Artificial Intelligence: A Modern Approach (Second Edition)* by Stuart Russell and Peter Norvig. Last updated: October 3, 2005.

²Note that an *undirected* edge in a graph is actually formed by two directed edges. Therefore, to check consistency, we need to check condition (*) in *both* directions.

perform *local* checks for consistency (every directed edge), while admissibility generally requires a more *global* check (every path from each node to the goal).

2. Search in Games

(a) Minimax Search

The pseudocode for the Minimax procedure follows. Given a node n in the game tree, it returns the value of n (*i.e.*, the value that the player who has the choice to move at n can achieve assuming the player and its opponent play optimally thereafter). The algorithm is initially called with n as some starting node in the game tree for which a player needs to make a choice. Recall that it usually runs only up to a certain depth of the full game tree. A terminal node of the subtree rooted at the starting node is either a leaf node of the game tree or a node at the given depth used.

```
MINIMAX( $n$ )
if  $n$  is a terminal node then
    return value( $n$ )
end if
if  $n$  is a maximizer node then
     $v \leftarrow -\infty$ 
    for all children  $s$  of  $n$  do
         $v \leftarrow \max(v, \text{MINIMAX}(s))$ 
    end for
    return  $v$ 
end if
if  $n$  is a minimizer node then
     $v \leftarrow \infty$ 
    for all children  $s$  of  $n$  do
         $v \leftarrow \min(v, \text{MINIMAX}(s))$ 
    end for
    return  $v$ 
end if
```

(b) Alpha-Beta Search

The pseudocode for the Alpha-Beta procedure follows. Given as inputs a node n and the upper and lower bound α and β , respectively, it returns the value assigned to n conditioned on α and β . That is, if n is a maximizer node and the maximum value the algorithm has found so far for its children is greater than or equal to β , it returns the maximum value found so far; otherwise, if all of its children values are less than β , it returns the maximum of those values. Similarly, if n is a minimizer node and the minimum value the algorithm has found so far for its children is less than or equal to α , it returns the minimum value found so far; otherwise, if all of its children values are greater than α , it returns the minimum of those values.

The algorithm is initially called with n as the starting node in the tree for which a player needs to make a choice, $\alpha = -\infty$ and $\beta = \infty$.

```
ALPHA-BETA( $n, \alpha, \beta$ )
if  $n$  is a terminal node then
    return value( $n$ )
```

```

end if
if  $n$  is a maximizer node then
     $v \leftarrow -\infty$ 
    for all children  $s$  of  $n$  do
         $v \leftarrow \max(v, \text{ALPHA-BETA}(s, \alpha, \beta))$ 
        if  $v \geq \beta$  then
            return  $v$ 
        end if
         $\alpha \leftarrow \max(v, \alpha)$ 
    end for
    return  $v$ 
end if
if  $n$  is a minimizer node then
     $v \leftarrow \infty$ 
    for all children  $s$  of  $n$  do
         $v \leftarrow \min(v, \text{ALPHA-BETA}(s, \alpha, \beta))$ 
        if  $v \leq \alpha$  then
            return  $v$ 
        end if
         $\beta \leftarrow \max(v, \beta)$ 
    end for
    return  $v$ 
end if

```

The parameter α can be interpreted as the maximum value that the *maximizer* player can guarantee itself from any choice point (*i.e.*, node) in the path from the root of the search tree to the node n . Similarly, β corresponds to the minimum value that the *minimizer* player can guarantee itself from any choice point in the path to n .

The effectiveness of the Alpha-Beta procedure is largely dependent on the order of the nodes in the game tree. If we consider nodes from left to right in the game tree, for Alpha-Beta to be most effective, the children of the *maximizer* nodes need to be ordered from *the highest to the lowest* valued, while the children of the *minimizer* nodes need to be ordered from *the lowest to the highest* valued.

(c) **Progressive deepening**

Progressive deepening, also known as iterative deepening, solves search problems by systematically increasing the search depth or horizon in the search or game tree. Progressive deepening is a very useful search technique in general and can be used along with any of the traditional search methods. It allows us to have an answer ready if one is needed by first solving the problem up to some small depth. In the context of games, for instance, the longer we run the algorithm, the deeper we go into the game tree and the better the decision about which of the choices for the player is best. Therefore, applications of progressive deepening lead to what are called *anytime algorithms*.

We can use progressive deepening along with the Alpha-Beta procedure by using the solutions from applying Alpha-Beta up to a certain depth to reorder the nodes in the game tree so as to hopefully make later runs of Alpha-Beta that go deeper into the tree more efficient. Another heuristic that can be used during progressive deepening is to use

the bounds Alpha-Beta obtained at each node in previous steps of progressive deepening to initialize the bounds for further applications of progressive deepening (instead of just using $[-\infty, \infty]$ every time we restart from the starting node of the game tree).

3. On the Sufficiency of Consistency for A^*

First, let me introduce some notation. Let s be the starting node and g be the goal. If X is a path such that $s \rightarrow \dots \rightarrow y \rightarrow \dots \rightarrow z \rightarrow \dots \rightarrow g$ (i.e., the path goes through y before z), let $c_X(y, z)$ be the cost of going from y to z along the path X . Naturally, if X is optimal, then $c_X(y, z) = c(y, z)$, where $c(y, z)$ is the minimum cost of any path from y to z . If X goes through y before going through z , we denote it by $z >_X y$. Similarly, for $y \leq_X z$. Finally, I say that a *path is consistent* if it satisfies condition (*) above. We denote the queue used by A^* by Q .

The result follows from the following two claims.

Claim 1 *For every node m in the search graph, if there is a consistent optimal path P to m , then for every node n in P , A^* would pick an optimal path to n from Q before a non-optimal path N to m .*

The claim follows because for every n in P ,

$$\begin{aligned} c(s, n) + h(n) &\leq c(s, n) + c(n, m) + h(m) && \text{[since } P \text{ is consistent]} \\ &= c(s, m) + h(m) && \text{[by the definition of } c] \\ &< c_N(s, m) + h(m). && \text{[since } N \text{ is non-optimal]} \end{aligned}$$

The result now follows by Claim 1 if we can guarantee that an optimal path to a node in a consistent optimal complete path is always in Q . The following claim helps to provide such a guarantee.

Claim 2 *If there is a consistent optimal complete path O in the search graph, then the following invariant holds throughout the execution of A^* : if A^* picks an optimal path to some node $m \neq g$ in O from Q at round T , and for all rounds prior to T , there is a node $o \leq_O m$ such that Q has an optimal path to o , then Q has an optimal path to a node $n >_O m$ at round $T + 1$.*

The proof of this claim is as follows. If m has not been extended at round T , then the invariant holds. Otherwise, for the sake of obtaining a contradiction, assume Q does not have an optimal path to any $n >_O m$ at round T . Since, for all rounds prior to T , there is a node $o \leq_O m$ such that Q has an optimal path to o , then, by Claim 1, m must have been extended by some optimal path to an optimal path R_1 that ends at the node m_1 in O immediately following m . But since, by assumption, R_1 is not in Q at round T , we have that by Claim 1 again, m_1 must have been extended by an optimal path to an optimal path R_2 that ends at the node m_2 in O immediately following m_1 . Continuing the same argument, we can conclude that Q had an optimal path to g at some round prior to T , but that optimal complete path is not in Q at round T . But this is a contradiction, since in that case A^* would have stopped prior to round T (that is, it could not pick R). Hence, the claim holds.

Since the condition in the invariant holds initially (i.e., $m = s$, $T = 1$), applying Claim 2 recursively gives the result.