Department of Electronics, Information, and Bioengeneering
Politecnico di Milano

Artificial Intelligence 2019-20

Marco Colombetti

4. Adversarial search

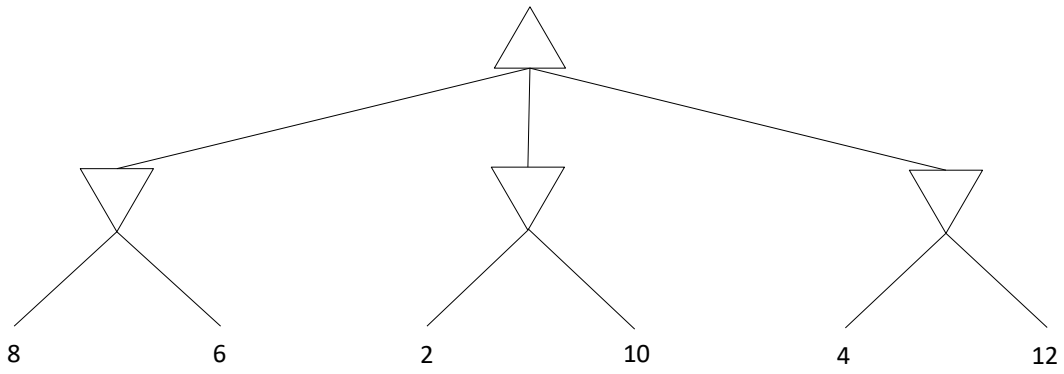
4.1 α-β search

There are different ways of specifying how to perform α-β pruning in a Minimax search. Here is a simple formulation:

- the search below a Min node is pruned if the node's upper bound is less than, or equal to, the lower bound of some Max ancestor;
- the search below a Max node is pruned if the node's lower bound is greater than, or equal to, the upper bound of some Min ancestor.

Example 1

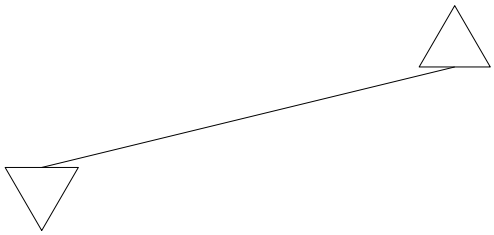
Here is an example of a complete game tree:



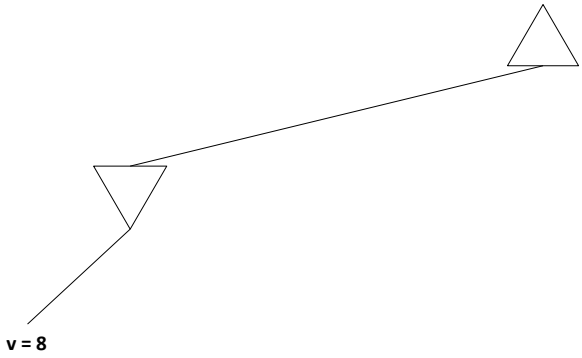
Below is an execution trace of α-β search, on the assumption that no bound for the values of the terminal nodes is known from the start. New values or bounds established during the execution are shown in bold. The parts of the game tree that are implicitly deleted after backtracking are dashed.

Step 1. Start from the root node (a Max node):

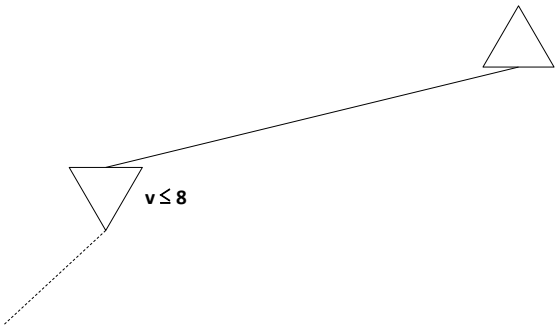
Step 2. Max performs an action and a Min node is generated:



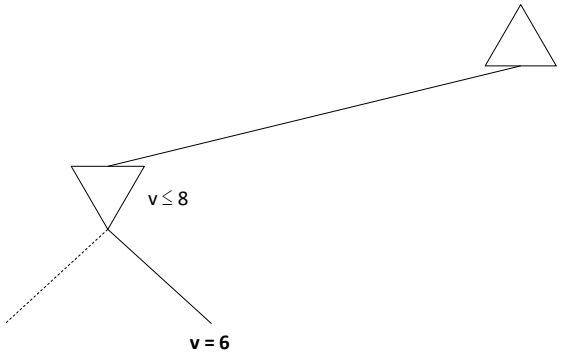
Step 3. Min performs an action and a terminal node is generated:



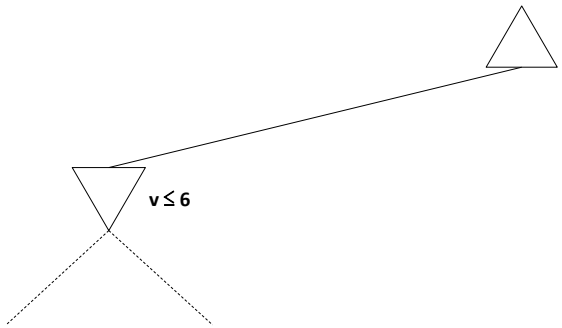
Step 4. After backtracking, the value of the terminal node becomes an upper bound for the current Min node:



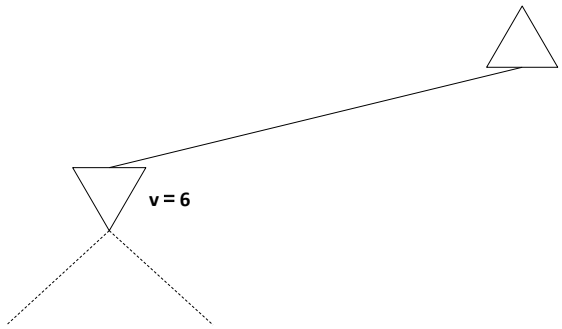
Step 5. Min performs a second action and another terminal node is generated:



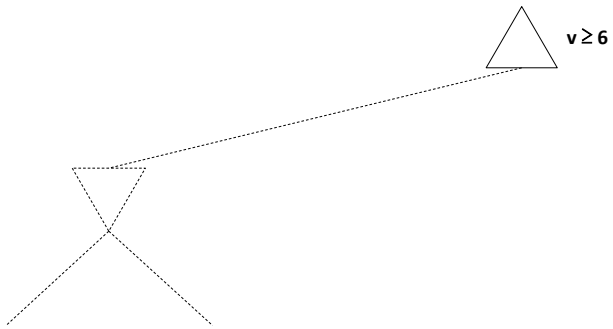
Step 6. After backtracking, the upper bound of the current Min node is updated:



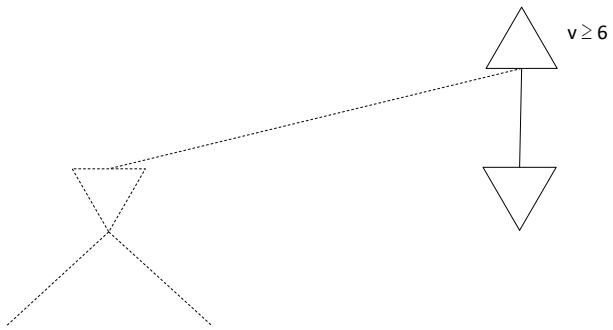
Step 7. As no further action can be performed, the value of the current Min node is set to its current upper bound:



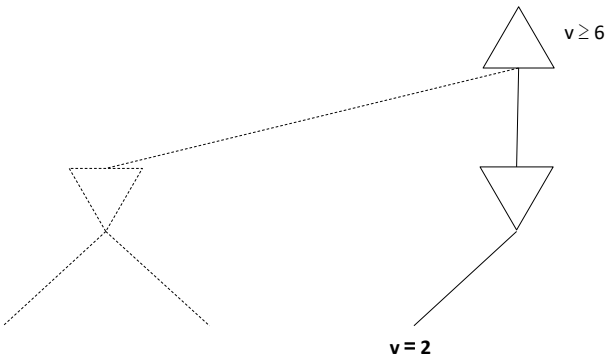
Step 8. After backtracking, the lower bound of the current Max node is updated:



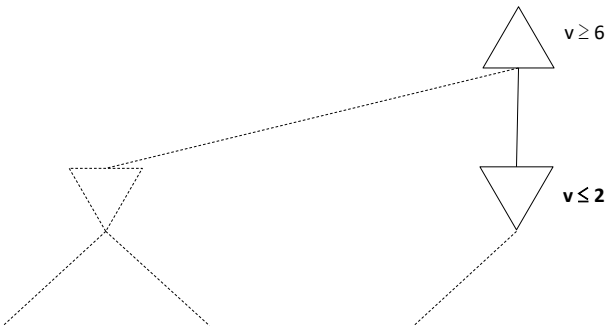
Step 9. Max performs a second action and another Min node is generated:



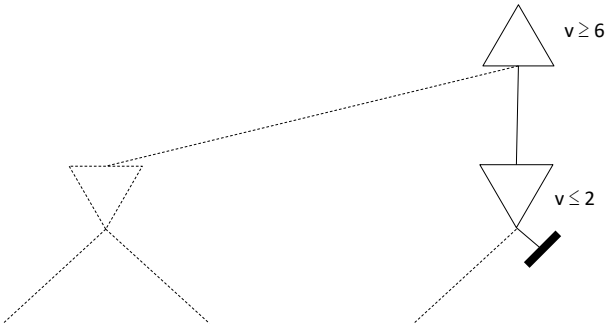
Step 10. Min performs an action and a terminal node is generated:



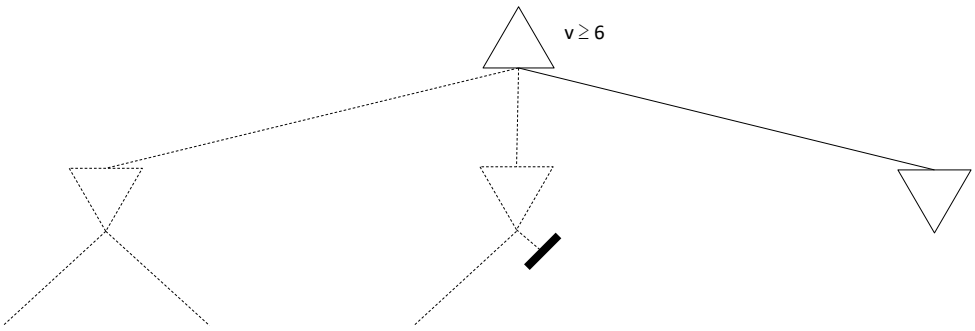
Step 11. After backtracking, the value of the terminal node becomes an upper bound for the current Min node:



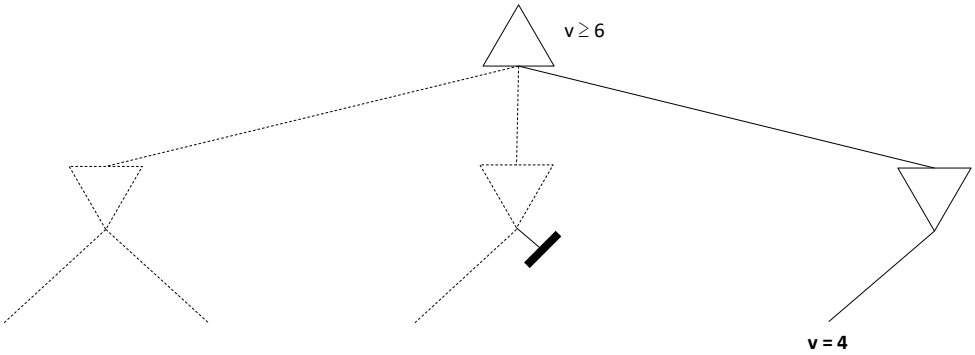
Step 12. The node's upper bound is $\leq$ than the lower bound of a Max ancestor. This means that it would be useless to proceed with the analysis of the current subtree, because Max has already discovered a better alternative. The game tree is therefore pruned immediately below the current node and no further Min action is executed:



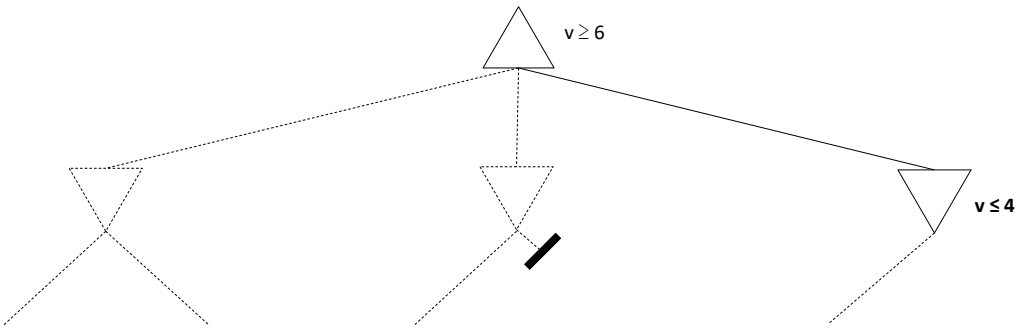
Step 13. After backtracking, Max performs a further action and a new Min node is generated:



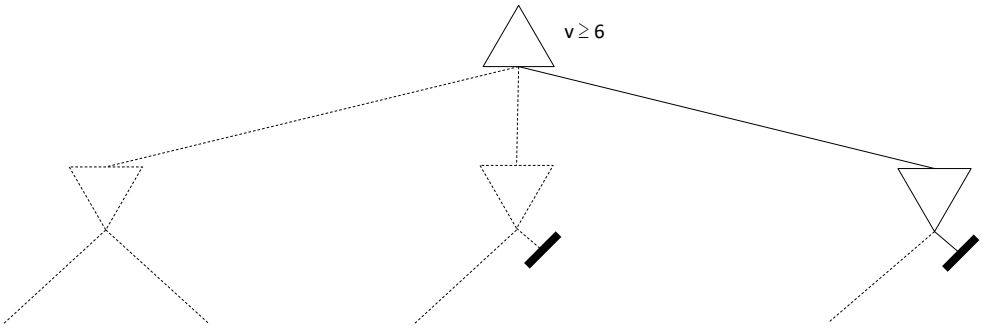
Step 14. Min performs an action and a terminal node is generated:



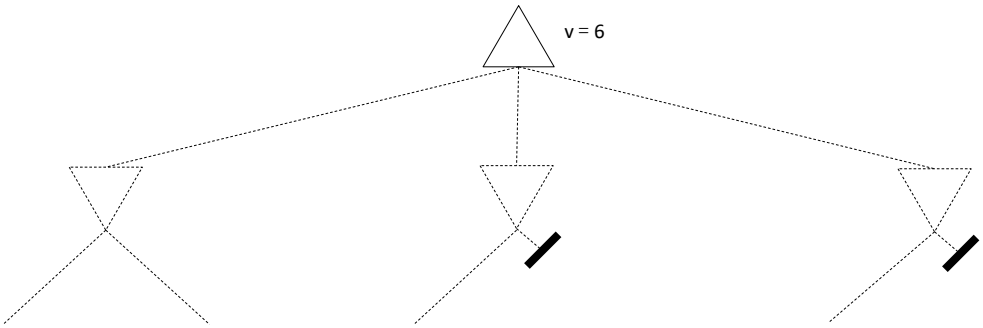
Step 15. After backtracking, the value of the terminal node becomes an upper bound for the current Min node:



Step 16. Again, the node's upper bound is ≤ than the lower bound of a Max ancestor. The game tree is pruned immediately below the current node and no further Min action is executed:



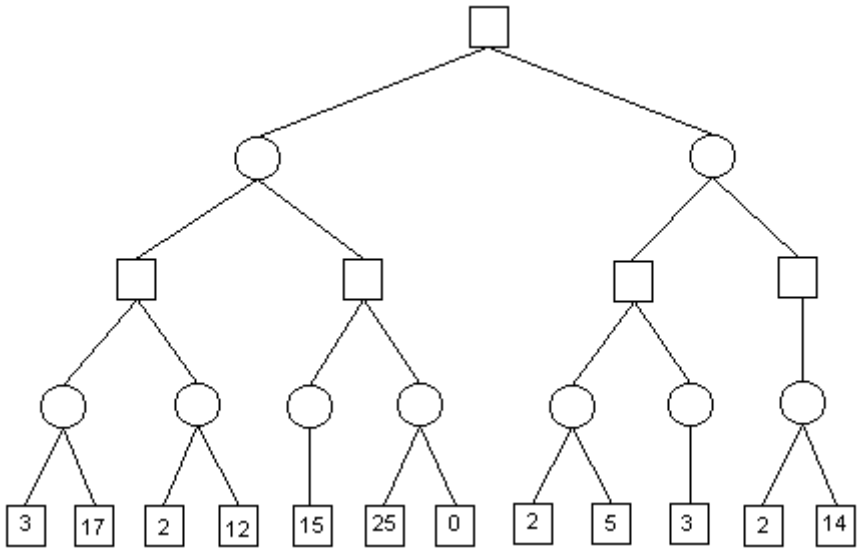
Step 17. As no further action can be performed, the value of the current Max node is set to its lower bound:



The value of the root node has now been established. Note that the search algorithm must remember which action Max has to perform to earn this value (in our example, the action corresponding to the leftmost branch of the tree).

Exercise 1

Apply α-β search to the following complete game tree, in which squares represent max nodes and circles represent min nodes. Then compare your solution to the one given at the web page from which the example is taken (last visit 24th October 2019).



From http://web.cs.ucla.edu/~rosen/161/notes/alphabeta.html.

4.2 Exploiting knowledge of minVal and maxVal

The pruning rules specified at the beginning of the previous subsection do not allow for pruning below the root node, because this is a Max node that does not have a Min ancestor.

In certain applications of α-β search, finite values of minVal and maxVal (i.e., the minimum and maximum possible evaluations) are known from the beginning. For example, if a game can be resolved by generating the whole game tree, and the possible values of terminal nodes are only −1 and +1 (or −1, 0, and +1), then it is known from the start that minVal = −1 and maxVal = +1. Being

predefined, these bounds can be considered as associated to virtual higher-level Min or Max nodes (respectively $\leq +1$ and ≥ -1).

Exercise 2

In a game of Nim, Max and Min take turns removing objects from N distinct heaps. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap. The player who removes the last object wins (in a different version of Nim, the player who removes the last object loses, but here we follow the convention stated before).

As a concrete example, assume that there are exactly $N = 2$ heaps, respectively containing 3 objects and 1 object. Specify a representation of the game states. Define the set of all allowable actions and put them in a well-defined order. Specify a utility function (from Max's point of view). Then build the game tree using α-β search with a backtracking strategy, always applying the actions in the order that you have previously specified. Is there a winning strategy for Max?

Solution

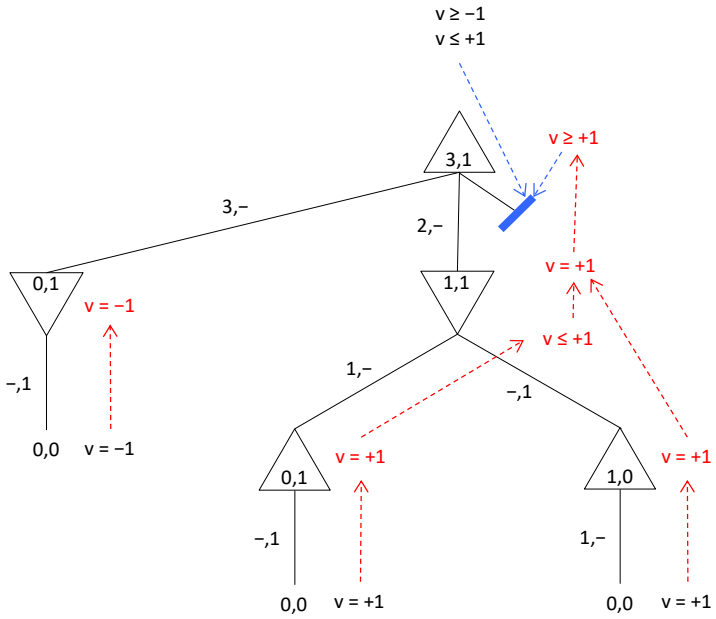
State representation: n,m (the numbers of objects in heaps 1 and 2, respectively).

Ordered list of actions: $3,-$; $2,-$; $1,-$; $-,1$ (where $k,-$ means "remove k objects from heap 1", and $-,k$ means "remove k objects from heap 2").

Utility function:

Max wins = +1 (maxVal)
Min wins = –1 (minVal)

Game tree:



4.3 MCTS (Monte Carlo Tree Search)

For this topic, refer to:

- the website `http://mcts.ai`
- in particular, Section 3 of the article "A survey of Monte Carlo Tree Search methods", available at: `http://mcts.ai/pubs/mcts-survey-master.pdf`
- the Wikipedia entry `https://en.wikipedia.org/wiki/Monte_Carlo_tree_search`
- other web pages on the same topic, like for example `http://jeffbradberry.com/posts/2015/09/intro-to-monte-carlo-tree-search/`