JOHDATUS TEKOÄLYYN

TEEMU ROOS

HELSINGIN YLIOPISTO
PELIPUU
PELIPUU
Search, Games and Problem Solving

Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

To the right. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤1. It could even become smaller still, but that is irrelevant since the maximum is already ≥3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥3. This cannot be changed by values ≤2. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in α.
- For every minimum node the current smallest child value is saved in β.
- If at a minimum node k the current value β ≤ α, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.
- If at a maximum node l the current value α ≥ β, then the search under l can end. Here β is the smallest value of a minimum node in the path from the root to l.
Fig. 6.18  A minimax game tree with look-ahead of four half-moves.

Fig. 6.19  An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
PELIPUUU

Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

• 

MAX 

MIN 

MAX 

MIN 

0 7 9 1
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤ 1. It could even become smaller still, but that is irrelevant since the maximum is already ≥ 3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥ 3. This cannot be changed by values ≤ 2. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in α.
• For every minimum node the current smallest child value is saved in β.
• If at a minimum node k the current value β ≤ α, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.
• If at a maximum node l the current value α ≥ β, then the search under l can end. Here β is the smallest value of a minimum node in the path from the root to l.
A minimax game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result. Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned. The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤ 1. It could even become smaller still, but that is irrelevant since the maximum is already ≥ 3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥ 3. This cannot be changed by values ≤ 2. Thus the remaining subtrees of b can be pruned. The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in α.
- For every minimum node the current smallest child value is saved in β.
- If at a minimum node k the current value β ≤ α, then the search under k can end. Here α is the largest value of a maximum node in the path from the root to k.
- If at a maximum node l the current value α ≥ β, then the search under l can end. Here β is the smallest value of a minimum node in the path from the root to l.
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

In Fig. 6.19, this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in \( \alpha \).
• For every minimum node the current smallest child value is saved in \( \beta \).
• If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
• If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node k the current value $\beta \leq \alpha$, then the search under k can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to k.
- If at a maximum node l the current value $\alpha \geq \beta$, then the search under l can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to l.
Fig. 6.18 A minimax game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The process depicted for the tree from Fig. 6.18 is as follows:

- At every leaf node, the evaluation is calculated.
- For every maximum node, the current largest child value is saved in $\alpha$.
- For every minimum node, the current smallest child value is saved in $\beta$.
- If at a minimum node $k$, the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$, the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.
Search, Games and Problem Solving

Fig. 6.18
A minimax game tree with look-ahead of four half-moves

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked $a$, all other successors can be ignored after the first child is evaluated as the value $1$ because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value $3$. Analogously the tree will be trimmed at node $b$. Since the first child of $b$ has the value $2$, the minimum to be generated for $b$ can only be less than or equal to $2$. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of $b$ can be pruned.

The same reasoning applies for the node $c$. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$. 

MAX
MIN
MAX
MIN

0 1 2 3 4 5 6 7 8 9

0 7 1 6 3 4 1 5 8 9

2 4 1 6

6 4
MINIMAX

MAX-ARVO(Solmu)

```plaintext
if LOPPUTILA(Solmu) return(ARVO(Solmu))
v = -\infty
for each Lapsi in LAPSET(Solmu)
    v = MAX(v, MIN-ARVO(Lapsi))
return(v)
```

MIN-ARVO(Solmu)

```plaintext
if LOPPUTILA(Solmu) return(ARVO(Solmu))
v = +\infty
for each Lapsi in LAPSET(Solmu)
    v = MIN(v, MAX-ARVO(Lapsi))
return(v)
```
MINIMAX

MAX-ARVO(Solmu)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = −∞

for each Lapsi in LAPSET(Solmu)

v = MAX(v, MIN-ARVO(Lapsi))

return(v)

MIN-ARVO(Solmu)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = +∞

for each Lapsi in LAPSET(Solmu)

v = MIN(v, MAX-ARVO(Lapsi))

return(v)
MINIMAX

Game Demonstration

The purpose of this demonstration is to help you develop intuition for how minimax and alpha-beta search methods perform. The particular problem solved is that of finding the best move in a game.

The search type menu item on the menu bar enables you to see either the minimax method working alone or together with the alpha-beta method.


**Fig. 6.18**
A minimax game tree with look-ahead of four half-moves.

**Fig. 6.19**
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked `a`, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node `b`. Since the first child of `b` has the value 2, the minimum to be generated for `b` can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of `b` can be pruned.

The same reasoning applies for the node `c`. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
MAX-ARVO(Solmu, $\alpha$, $\beta$)

if LOPPUTILA(Solmu) return (ARVO(Solmu))

$v = -\infty$
for each Lapsi in LAPSET(Solmu)
    $v = \text{MAX}(v, \text{MIN-ARVO}(\text{Lapsi}, \alpha, \beta))$
    if $v \geq \beta$ return $v$
    $\alpha = \text{MAX}(\alpha, v)$
return($v$)
MAX-ARVO(Solmu, $\alpha$, $\beta$)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

$v = -\infty$

for each Lapsi in LAPSET(Solmu)
  $v = \text{MAX}(v, \text{MIN-ARVO}(\text{Lapsi, } \alpha, \beta))$
  if $v \geq \beta$ return $v$

$\alpha = \text{MAX}(\alpha, v)$

return($v$)

MIN-ARVO(Solmu, $\alpha$, $\beta$)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

$v = +\infty$

for each Lapsi in LAPSET(Solmu)
  $v = \text{MIN}(v, \text{MAX-ARVO}(\text{Lapsi, } \alpha, \beta))$
  if $v \leq \alpha$ return $v$

$\beta = \text{MIN}(\beta, v)$

return($v$)
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.

$\alpha = 3$

$\beta = 6$
-search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be ≤ 1. It could even become smaller still, but that is irrelevant since the maximum is already ≥ 3 one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be ≥ 3. This cannot be changed by values ≤ 2. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in α.
- For every minimum node the current smallest child value is saved in β.
- If at a minimum node \( k \) the current value \( β \leq α \), then the search under \( k \) can end. Here \( α \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( α \geq β \), then the search under \( l \) can end. Here \( β \) is the smallest value of a minimum node in the path from the root to \( l \).

\[
\text{MIN-ARVO(Solmu, } α, β) \]

\[
\text{if LOPPUTILA(Solmu) return } \text{ARVO(Solmu)}
\]

\[
v = +\infty
\]

\[
\text{for each Lapsi in LAPSET(Solmu)}
\]

\[
v = \text{MIN}(v, \text{MAX-ARVO(Lapsi, } α, β))
\]

\[
\text{if } v \leq α \text{ return } v
\]

\[
β = \text{MIN}(β, v)
\]

\[
\text{return}(v)
\]
Fig. 6.18
A minimax game tree with look-ahead of four half-moves.

Fig. 6.19
An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum.

In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \(a\), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \(\leq 1\). It could even become smaller still, but that is irrelevant since the maximum is already \(\geq 3\) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \(b\).

Since the first child of \(b\) has the value 2, the minimum to be generated for \(b\) can only be less than or equal to 2. But the maximum at the root node is already sure to be \(\geq 3\). This cannot be changed by values \(\leq 2\). Thus the remaining subtrees of \(b\) can be pruned.

The same reasoning applies for the node \(c\). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \(\alpha\).
- For every minimum node the current smallest child value is saved in \(\beta\).
- If at a minimum node \(k\) the current value \(\beta \leq \alpha\), then the search under \(k\) can end. Here \(\alpha\) is the largest value of a maximum node in the path from the root to \(k\).
- If at a maximum node \(l\) the current value \(\alpha \geq \beta\), then the search under \(l\) can end. Here \(\beta\) is the smallest value of a minimum node in the path from the root to \(l\).

```
MIN-ARVO(Solmu, \(\alpha\), \(\beta\))

if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = +\(\infty\)
for each Lapsi in LAPSET(Solmu)
    v = MIN(v, MAX-ARVO(Lapsi, \#, \$))
    if 1 \(\leq 3\) return v
\(\beta\) = MIN(\(\beta\), v)
return(v)
```

\[
\begin{align*}
\text{MAX} & \quad \text{MIN} \\
\text{MAX} & \quad \text{MIN} \\
\end{align*}
\]

\[
\begin{align*}
6 & \quad \alpha = 3 \\
0 & \quad 1 & \quad 6 & \quad 3 & \quad 4 & \quad 1 & \quad 5 & \quad 8 & \quad 9 & \quad 2 & \quad 2 & \quad 3 & \quad 4 & \quad 5 & \quad 1 & \quad 2 & \quad 7 & \quad 6 & \quad 9 & \quad 4
\end{align*}
\]
Fig. 6.18 A minimax game tree with look-ahead of four half-moves

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19, this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).

```
MIN-ARVO(Solmu, \( \alpha \), \( \beta \))

if LOPPUTILA(Solmu) return (ARVO(Solmu))

\( v = +\infty \)

for each Lapsi in LAPSET(Solmu)

\( v = 1 \)
if \( 1 \leq 3 \) return \( v \)
\( \beta = \text{MIN}(\beta, v) \)

return(v)
```
ALPHA-BETA-KARSINTA

At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in $\alpha$.
- For every minimum node the current smallest child value is saved in $\beta$.
- If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
- If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.
104 6 Search, Games and Problem Solving

Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked \( a \), all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be \( \leq 1 \). It could even become smaller still, but that is irrelevant since the maximum is already \( \geq 3 \) one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node \( b \). Since the first child of \( b \) has the value 2, the minimum to be generated for \( b \) can only be less than or equal to 2. But the maximum at the root node is already sure to be \( \geq 3 \). This cannot be changed by values \( \leq 2 \). Thus the remaining subtrees of \( b \) can be pruned.

The same reasoning applies for the node \( c \). However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

- At every leaf node the evaluation is calculated.
- For every maximum node the current largest child value is saved in \( \alpha \).
- For every minimum node the current smallest child value is saved in \( \beta \).
- If at a minimum node \( k \) the current value \( \beta \leq \alpha \), then the search under \( k \) can end. Here \( \alpha \) is the largest value of a maximum node in the path from the root to \( k \).
- If at a maximum node \( l \) the current value \( \alpha \geq \beta \), then the search under \( l \) can end. Here \( \beta \) is the smallest value of a minimum node in the path from the root to \( l \).
MIN-ARVO(Solmu, α, β)

if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = +∞

for each Lapsi in LAPSET(Solmu)

v = 2

if 2 ≤ 3 return v

β = MIN(β, v)

return(v)
**MIN-ARVO(Solmu, α, β)**

if LOPPUTILA(Solmu) return(ARVO(Solmu))

v = +∞

for each Lapsi in LAPSET(Solmu)

v = 2

if 2 ≤ 3 return v

β = MIN(β, v)

return(v)
SHAKKI

Deep Blue beat G. Kasparov in 1997
SHAKKI
1769 Wolfgang von Kempelen rakentaa “Turkin”
1912 L. Torres y Quevedo rakentaa koneen kuningas-torni vs kuningas -loppupeleihin
1948 Norbert Wiener esittää syvyysrajoitetun minimax-algoritmin heuristisella arviontifunktiolla
1950 Claude Shannon julkaisee artikkelin “Programming a Computer for Playing Chess”
1951 Alan Turing kehittää ensimmäisen algoritmin, joka pystyy pelaamaan kokonaisen shakkiohjelun
1956 Los Alamos chess: ensimmäinen tietokoneohjelma, joka pelaan (yksinkertaistettua) shakkia
1956 John McCarthy keksii alpha-beta-karsinnan
1957 Ensimmäiset oikeaa shakkia pelaavat ohjelmat
1966-67 Ensimmäiset tietokoneohjelmien väliset ottelut (Moskova voittaa.)
1967 Ensimmäinen tietokoneohjelman voitto turnauksessa.
1981 Cray Blitz voittaa Mississippin osavaltion mestaruuden ja saa ensimmäisenä tietokoneena mestarin statuksen.
1989 Garry Kasparov voittaa kaksi näytösottelua Deep Thoughtia vastaan.
1997 Deep Blue voittaa Garry Kasparovin kuuden pelin ottelussa.
2006 Deep Fritz voittaa maailmanmestari Vladimir Kramnikin.
SHAKKI

- TILA: (LAUDAN TILANNE)
- SIIRTYMÄT: (SALLITUT SIIRROT)
- MENETELMÄ: SYVYYSSRAJOITETTU ALPHA-BETA-KARSINTA
Fig. 6.18 A minimax game tree with look-ahead of four half-moves.

Fig. 6.19 An alpha-beta game tree with look-ahead of four half-moves. The dotted portions of the tree are not traversed because they have no effect on the end result.

Like in minimax search, in the minimum nodes the minimum is generated from the minimum value of the successor nodes and in the maximum nodes likewise the maximum. In Fig. 6.19 this process is depicted for the tree from Fig. 6.18. At the node marked a, all other successors can be ignored after the first child is evaluated as the value 1 because the minimum is sure to be $\leq 1$. It could even become smaller still, but that is irrelevant since the maximum is already $\geq 3$ one level above. Regardless of how the evaluation of the remaining successors turns out, the maximum will keep the value 3. Analogously the tree will be trimmed at node b. Since the first child of b has the value 2, the minimum to be generated for b can only be less than or equal to 2. But the maximum at the root node is already sure to be $\geq 3$. This cannot be changed by values $\leq 2$. Thus the remaining subtrees of b can be pruned.

The same reasoning applies for the node c. However, the relevant maximum node is not the direct parent, but the root node. This can be generalized.

• At every leaf node the evaluation is calculated.
• For every maximum node the current largest child value is saved in $\alpha$.
• For every minimum node the current smallest child value is saved in $\beta$.
• If at a minimum node $k$ the current value $\beta \leq \alpha$, then the search under $k$ can end. Here $\alpha$ is the largest value of a maximum node in the path from the root to $k$.
• If at a maximum node $l$ the current value $\alpha \geq \beta$, then the search under $l$ can end. Here $\beta$ is the smallest value of a minimum node in the path from the root to $l$.