JOHDATUS TEKOÄLYYN

TEEMU ROOS
PELIPUU
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 \( l \).
MINIMAX

MAX-ARVO(Solmu)

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

v = -\infty

for each Lapsi in LAPSET(Solmu)

\hspace{1em} v = \text{MAX}(v, \text{MIN-ARVO}(Lapsi))

return(v)
MINIMAX

MAX-ARVO(Solmu)

    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)

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

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 \(l\).
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}(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}(Lapsi, $\alpha$, $\beta$))$
    if $v \leq \alpha$ return $v$

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

return($v$)
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 $l$. 
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. 6.19: 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-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.

The same reasoning applies for the node 𝑑. 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 \( l \).
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 \( l \).
Figure 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 Figure 6.19, this process is depicted for the tree from Figure 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 \( l \).
SHAKKI
<table>
<thead>
<tr>
<th>vuosi</th>
<th>tapahtuma</th>
<th>detallit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1769</td>
<td>Wolfgang von Kempelen rakensa &quot;Turkin&quot;</td>
<td></td>
</tr>
<tr>
<td>1912</td>
<td>L. Torres y Quevedo rakensaa koneen kuningas &amp; torni vs kuningas - loppupeleihin</td>
<td></td>
</tr>
<tr>
<td>1948</td>
<td>Norbert Wiener esittää syvyysrajoitetun minimax-algoritmin heuristisella arviontiufunktioilla</td>
<td></td>
</tr>
<tr>
<td>1950</td>
<td>Claude Shannon julkaisee artikkelin &quot;Programming a Computer for Playing Chess&quot;</td>
<td></td>
</tr>
<tr>
<td>1951</td>
<td>Alan Turing kehittää ensimmäisen algoritmin, joka pystyy pelaamaan kokonaisen shakkiottelun</td>
<td></td>
</tr>
<tr>
<td>1956</td>
<td>Los Alamos chess: ensimmäinen tietokoneohjelma, joka pelaaa (yksinkertaistettua) shakkia</td>
<td></td>
</tr>
<tr>
<td>1956</td>
<td>John McCarthy keksii alpha-beta-karsinnan</td>
<td></td>
</tr>
<tr>
<td>1957</td>
<td>Ensimmäiset oikeaa shakkaia pelaavat ohjelmat</td>
<td></td>
</tr>
<tr>
<td>1966-67</td>
<td>Ensimmäiset tietokoneohjelmien väliset ottelut (Moskova voittaa.)</td>
<td></td>
</tr>
</tbody>
</table>
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 Kramnikinin.
SHAKKI

- **TILA:** *(LAUDAN TILANNE)*
- **SIIRTYMÄT:** *(SALLITUT SIIRROT)*
- **MENETELMÄ:** *SYVYYSRÄJOITETTU ALPHA-BETA-KARSINTA*
<table>
<thead>
<tr>
<th>MAX</th>
<th>MIN</th>
<th>MAX</th>
<th>MIN</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

**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 \( l \).

**ARVIOITA TILANTEEN HYVYYDESTÄ**
TILA: (LAUDAN TILANNE)

SIIRTYMÄT: (SALLITUT SIIRROT)

MENETELMÄ: SYVYYRSRAJOITETTU ALPHA-BETA-KARSINTA

TEHTÄVÄ: SUUNNITTELE HEURISTINEN ARVIOINTIFUNKTIO
ARPAPELIT

MIN

ARPA

MAX

0.5

0.5

ARVO = 0.5 MAX₁ + 0.5 MAX₂