Programowanie gier logicznych

Piotr Beling

Instytut Informatyki
Politechnika Łódzka

25 marca 2007

Rozważane gry

- logiczne
- dwuosobowe - konflikt interesów występuje pomiędzy dwoma uczestnikami
- deterministyczne - bez czynnika losowego
- z pełną informacją - gracze posiadają pełną informację na temat stanu w jakim znajduje się gra
- o sumie zerowej - suma wypłat (miar zwycięstwa) graczy wynosi zero

Przykłady

szachy, warcaby, GO, othello, kółko i krzyżyk

Rozważane gry

- logiczne
- dwuosobowe - konflikt interesów występuje pomiędzy dwoma uczestnikami
- deterministyczne - bez czynnika losowego
- z pełną informacją - gracze posiadają pełną informację na temat stanu w jakim znajduje się gra
- o sumie zerowej - suma wypłat (miar zwycięstwa) graczy wynosi zero

Przykłady

szachy, warcaby, GO, othello, kółko i krzyżyk

Rozważane gry

- logiczne
- dwuosobowe - konflikt interesów występuje pomiędzy dwoma uczestnikami
- deterministyczne - bez czynnika losowego
- z pełną informacją - gracze posiadają pełną informację na temat stanu w jakim znajduje się gra
- o sumie zerowej - suma wypłat (miar zwycięstwa) graczy wynosi zero

Przykłady

szachy, warcaby, GO, othello, kółko i krzyżyk

Rozważane gry

- logiczne
- dwuosobowe - konflikt interesów występuje pomiędzy dwoma uczestnikami
- deterministyczne - bez czynnika losowego
- z pełną informacją - gracze posiadają pełną informację na temat stanu w jakim znajduje się gra
- o sumie zerowej - suma wypłat (miar zwycięstwa) graczy wynosi zero

Przykłady

szachy, warcaby, GO, othello, kółko i krzyżyk

Rozważane gry

- logiczne
- dwuosobowe - konflikt interesów występuje pomiędzy dwoma uczestnikami
- deterministyczne - bez czynnika losowego
- z pełną informacją - gracze posiadają pełną informację na temat stanu w jakim znajduje się gra
- o sumie zerowej - suma wypłat (miar zwycięstwa) graczy wynosi zero

Przykłady

szachy, warcaby, GO, othello, kółko i krzyżyk

Rozważane gry

- logiczne
- dwuosobowe - konflikt interesów występuje pomiędzy dwoma uczestnikami
- deterministyczne - bez czynnika losowego
- z pełną informacją - gracze posiadają pełną informację na temat stanu w jakim znajduje się gra
- o sumie zerowej - suma wypłat (miar zwycięstwa) graczy wynosi zero

Przykłady

szachy, warcaby, GO, othello, kółko i krzyżyk

Gra jako graf

- przepisy gry charakteryzują pewien skierowany graf (dalej nazywany grafem gry)
- węzły grafu gry utożsamiamy z stanami gry
- następnikami każdego węzła (stanu) w grafie są węzły (stany) osiągalne z niego poprzez wykonanie jednego ruchu
- gracze wykonują ruchy na przemian (następnikami węzła są stany w których ruch należy do przeciwnika)
- stany końcowe (dla których przepisy definiują wypłatę) charakteryzuje brak następników

Gra jako graf

- przepisy gry charakteryzują pewien skierowany graf (dalej nazywany grafem gry)
- węzły grafu gry utożsamiamy z stanami gry
- następnikami każdego węzła (stanu) w grafie są węzły (stany) osiągalne z niego poprzez wykonanie jednego ruchu
- gracze wykonują ruchy na przemian (następnikami węzła są stany w których ruch należy do przeciwnika)
- stany końcowe (dla których przepisy definiują wypłatę) charakteryzuje brak następników

Gra jako graf

- przepisy gry charakteryzują pewien skierowany graf (dalej nazywany grafem gry)
- węzły grafu gry utożsamiamy z stanami gry
- następnikami każdego węzła (stanu) w grafie są węzły (stany) osiągalne z niego poprzez wykonanie jednego ruchu
- gracze wykonują ruchy na przemian (następnikami węzła są stany w których ruch należy do przeciwnika)
- stany końcowe (dla których przepisy definiują wypłatę) charakteryzuje brak następników

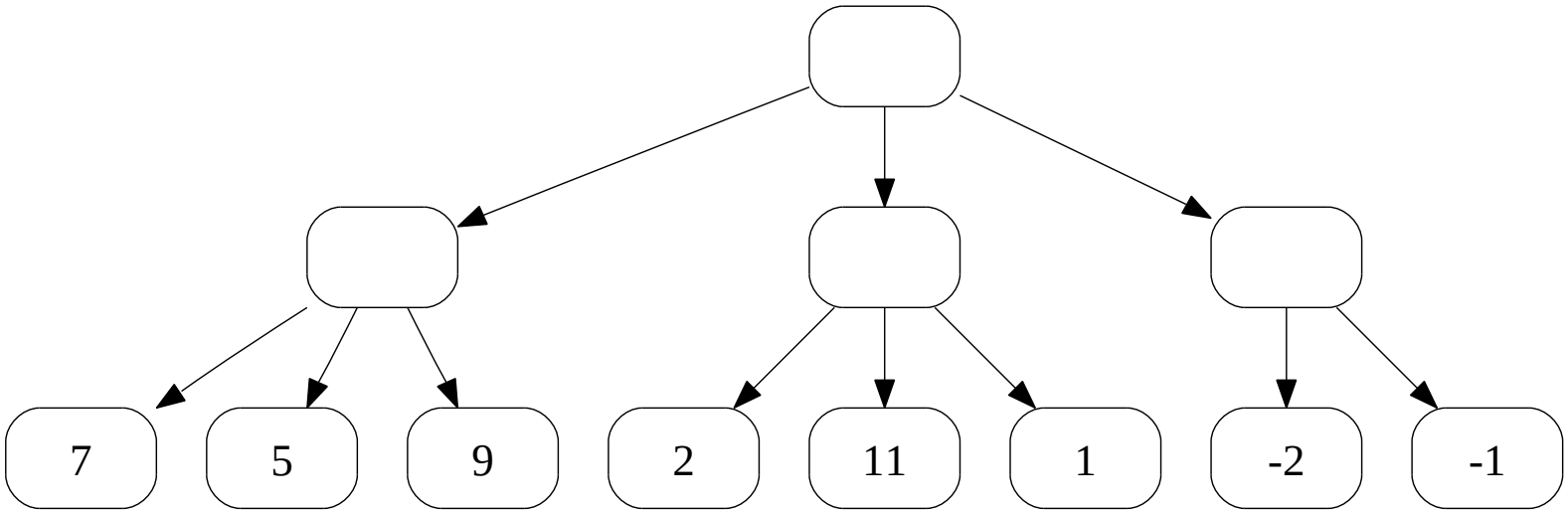
Gra jako graf

- przepisy gry charakteryzują pewien skierowany graf (dalej nazywany grafem gry)
- węzły grafu gry utożsamiamy z stanami gry
- następnikami każdego węzła (stanu) w grafie są węzły (stany) osiągalne z niego poprzez wykonanie jednego ruchu
- gracze wykonują ruchy na przemian (następnikami węzła są stany w których ruch należy do przeciwnika)
- stany końcowe (dla których przepisy definiują wypłatę) charakteryzuje brak następników

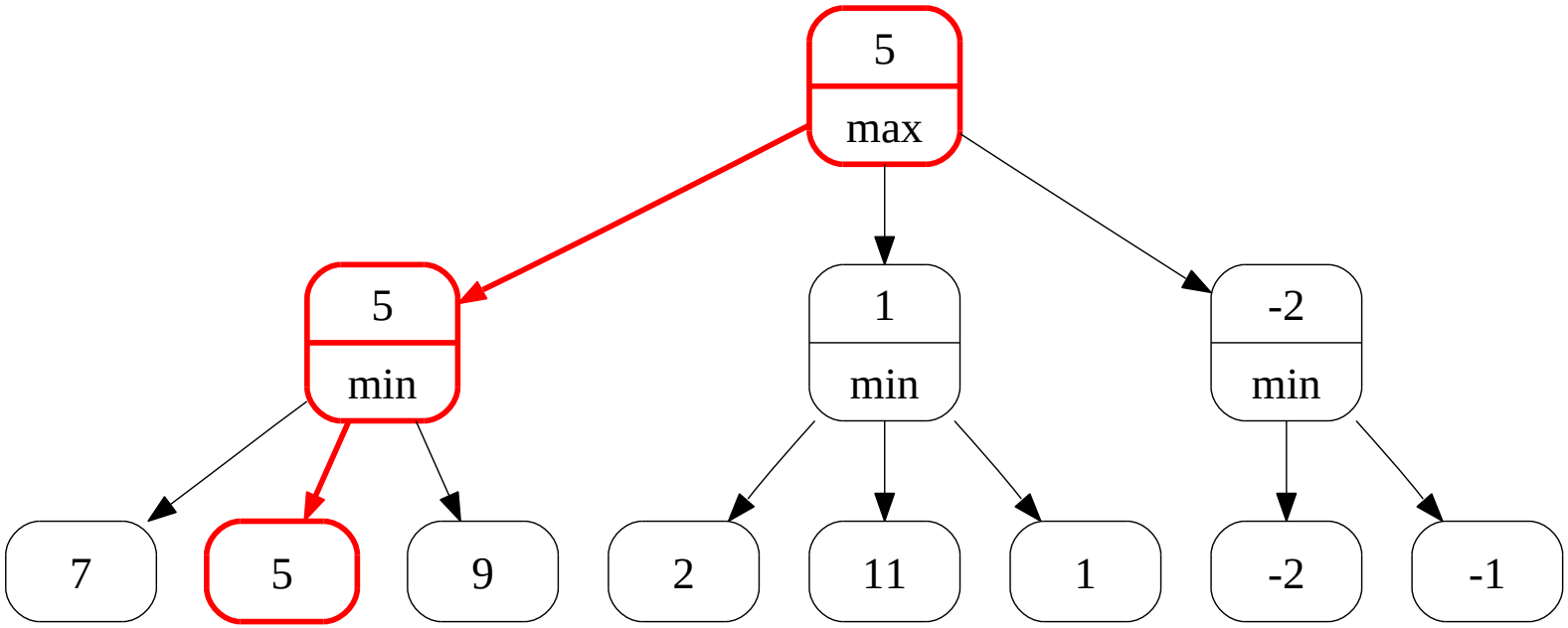
Gra jako graf

- przepisy gry charakteryzują pewien skierowany graf (dalej nazywany grafem gry)
- węzły grafu gry utożsamiamy z stanami gry
- następnikami każdego węzła (stanu) w grafie są węzły (stany) osiągalne z niego poprzez wykonanie jednego ruchu
- gracze wykonują ruchy na przemian (następnikami węzła są stany w których ruch należy do przeciwnika)
- stany końcowe (dla których przepisy definiują wypłatę) charakteryzuje brak następników

Przykładowy graf gry. Który ruch należy wykonać?



Przykładowe drzewo poszukiwań algorytmu Min-Max



Algorytm Min-Max. Czy warunek stopu jest wystarczający?

```
1  int MinMax(Stan S) {
2      vector<Stan> N = nast(S);     //następniki S
3      if (N == ∅)
4          return wyplata(S, G);     //wypłata gracza G
5      int result = MinMax(N[0]);
6      if (S.czyj_ruch == G) { //szukamy maksimum:
7          for (int i = 1; i < |N|; i++) {
8              int val = MinMax(N[i]);
9              if (val > result) result = val;
10         }
11     } else {                      //szukamy minimum:
12         for (int i = 1; i < |N|; i++) {
13             int val = MinMax(N[i]);
14             if (val < result) result = val;
15         }
16     }
17     return result;
18 }
```

Algorytm Min-Max z ograniczeniem głębokości

```
1  int MinMax(Stan S, int Depth) {
2      if (Depth == 0) return ocena_G(S);
3      vector<Stan> N = nast(S);
4      if (N == ∅) return wyplata(S, G);
5      int result = MinMax(N[0], Depth-1);
6      if (S.czyj_ruch == G) { //szukamy maksimum:
7          for (int i = 1; i < |N|; i++) {
8              int val = MinMax(N[i], Depth-1);
9              if (val > result) result = val;
10         }
11     } else {                    //szukamy minimum:
12         for (int i = 1; i < |N|; i++) {
13             int val = MinMax(N[i], Depth-1);
14             if (val < result) result = val;
15         }
16     }
17     return result;
18 }
```

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Funkcja oceniająca (ocena$_G$(S))

- aproksymuje MinMax(S)
- jest obliczana na podstawie statycznych cech stanu S (bez „przewidywania" ruchów), np. w warcabach może ona uwzględniać przewagę w ilości posiadanych pionów lub damek, odległość pionów od linii promocji, itd.
- może być postaci (przykładowo):
  - (w większości programów) $ocena(S) = \sum_{i=0}^{|C|-1} w_i C_i(S)$ gdzie $w_i \in \Re$ dla $i = 0, 1, \ldots, |C| - 1$ to waga i-tej cechy (C_i)
  - sieci neuronowej
- cechy i współczynniki mogą być dobierane:
  - (w większości programów) ręcznie
  - automatycznie (np. metody: Samuela, różnic w czasie TD(λ), Generalized Linear Evaluation Model (GLEM), wstecznej propagacji błędu dla sieci neuronowych)

Algorytm Nega-Max - zasada działania

- Zapis algorytmu Min-Max można ujednolić korzystając ze spostrzeżenia, że:

$$\forall a_1, \ldots, a_N \in \Re\colon\ \min(a_1, \ldots, a_N) = -\max(-a_1, \ldots, -a_N)$$

- We wszystkich węzłach typu minimum można obliczać maksimum z wartości przeciwnych do tych zwróconych przez podrzędne wywołania rekurencyjne. Wywołanie nadrzędne (które jest dla węzła typu maksimum) dołoży minus do zwróconego wyniku.
- Nega-Max ocenia węzeł z punktu widzenia gracza idącego (analogicznie muszą być też zdefiniowane funkcje *wyplata* i *ocena*).

Algorytm Nega-Max - zasada działania

- Zapis algorytmu Min-Max można ujednolić korzystając ze spostrzeżenia, że:

$$\forall a_1, \ldots, a_N \in \Re\colon\ \min(a_1, \ldots, a_N) = -\max(-a_1, \ldots, -a_N)$$

- We wszystkich węzłach typu minimum można obliczać maksimum z wartości przeciwnych do tych zwróconych przez podrzędne wywołania rekurencyjne. Wywołanie nadrzędne (które jest dla węzła typu maksimum) dołoży minus do zwróconego wyniku.
- Nega-Max ocenia węzeł z punktu widzenia gracza idącego (analogicznie muszą być też zdefiniowane funkcje *wyplata* i *ocena*).

Algorytm Nega-Max - zasada działania

- Zapis algorytmu Min-Max można ujednolić korzystając ze spostrzeżenia, że:

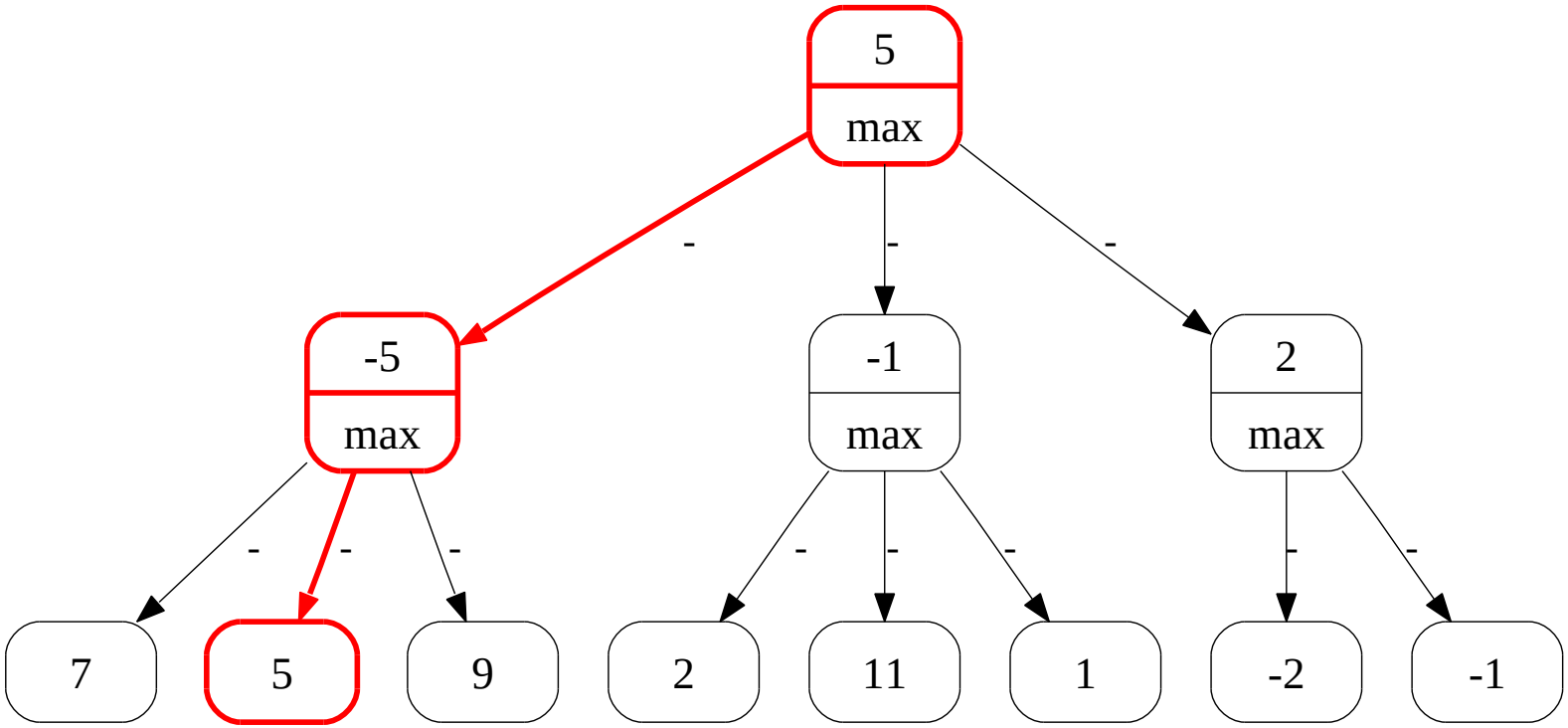
$$\forall a_1, \ldots, a_N \in \Re\colon\ \min(a_1, \ldots, a_N) = -\max(-a_1, \ldots, -a_N)$$

- We wszystkich węzłach typu minimum można obliczać maksimum z wartości przeciwnych do tych zwróconych przez podrzędne wywołania rekurencyjne. Wywołanie nadrzędne (które jest dla węzła typu maksimum) dołoży minus do zwróconego wyniku.
- Nega-Max ocenia węzeł z punktu widzenia gracza idącego (analogicznie muszą być też zdefiniowane funkcje *wyplata* i *ocena*).

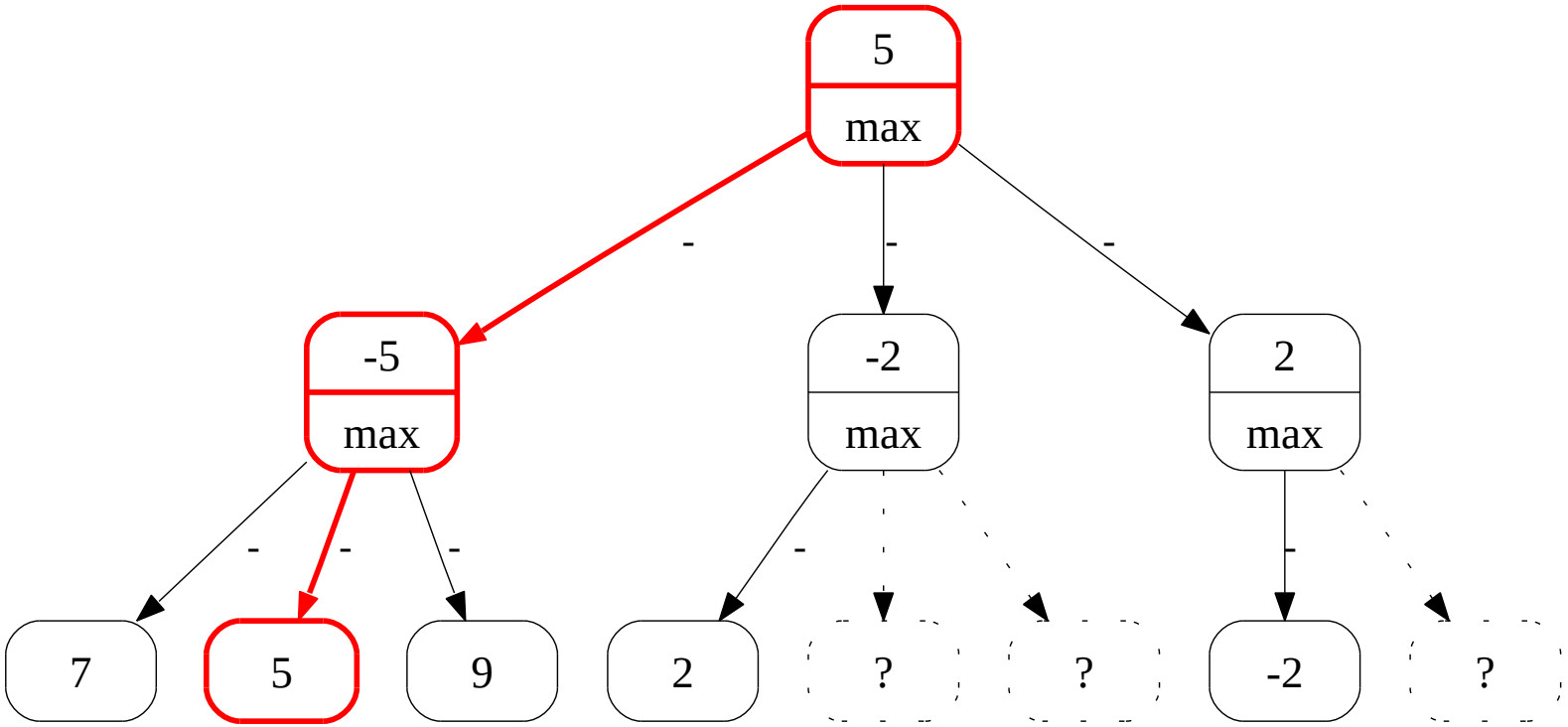
Algorytm Nega-Max - listing

```
1  //Zwraca przewidywaną wypłatę gracza do którego należy ruch w S
2  //S - aktualny stan gry, Depth - głębokość poszukiwań
3  int NegaMax(Stan S, int Depth) {
4      if (Depth == 0)
5          return ocena(S);
6      vector<Stan> N = nast(S);
7      if (N == ∅)
8          return wyplata(S);
9      int result = −∞;
10     //szukamy maksimum z ocen kolejnych pozycji
11     for (int i = 0; i < |N|; i++) {
12         int val = − NegaMax(N[i], Depth−1);
13         if (val > result)
14             result = val;
15     }
16     return result;
17 }
```

Przykładowe drzewo poszukiwań algorytmu Nega-Max



Przykładowe drzewo poszukiwań algorytmu α-β



algorytm α-β - zasada działania

Jeśli oznaczymy przez G zawodnika do którego należy ruch w stanie S to:

- α jest największą wartością jaką osiągną dotychczas G, tzn. mógł on wcześniej wykonać posunięcie, które w efekcie doprowadziłoby grę do stanu o ocenie nie mniejszej niż α
- β jest najmniejszą wartością (z punktu widzenia G) do jakiej mógł doprowadzić rywal gracza G

Jeżeli bieżąca wartość oceny stanu S osiągnie lub przekroczy β, to przeszukanie kolejnych następników S nie ma sensu, gdyż rywal G będzie wolał wykonać wcześniej ruch, który doprowadzi grę do stanu o ocenie β zamiast do S. Może nastąpić odcięcie i jako wartość stanu S można zwrócić β.

algorytm α-β - listing

```
1  //Zwraca przewidywaną wypłatę gracza do którego należy ruch w S
2  //S - aktualny stan gry, Depth - głębokość poszukiwań
3  //α, β - największa i najmniejsza znaleziona wcześniej wartość
4  int AlfaBeta(Stan S, int Depth, int α, int β) {
5      if (Depth == 0) return ocena(S);
6      vector<Stan> N = nast(S);
7      if (N == ∅) return wyplata(S);
8      sort(N); //krok opcjonalny
9      for (int i = 0; i < |N|; i++) { //szukamy maksimum
10         int val = - AlfaBeta(N[i], Depth-1, -β, -α);
11         if (val ⩾ β)
12             return β; //β-cięcie
13         if (val > α)
14             α = val; //poprawa maksimum
15     }
16     return α;
17 }
```

twierdzenie o związku α-β z Nega-Max

Niech $d \geqslant 0$, $\alpha < \beta$, S - stan gry. Oznaczmy ponadto $ab = AlfaBeta(S, d, \alpha, \beta)$ oraz $n = NegaMax(S, d)$. Wtedy możemy wyróżnić trzy sytuacje:

- (succes) $\alpha < ab < \beta \quad \Rightarrow \quad ab = n$
- (failing low) $\alpha \geqslant ab \quad \Rightarrow \quad ab \geqslant n$
- (failing high) $\beta \leqslant ab \quad \Rightarrow \quad ab \leqslant n$

Wniosek z implikacji (succes)

$$\forall S \in \mathbb{S}, d \geqslant 0: \qquad AlfaBeta(S, d, -\infty, \infty) = NegaMax(S, d)$$

co określa możliwość bezpośredniego wykorzystania α-β do znalezienia wartości negamaksowej stanu gry.

twierdzenie o związku α-β z Nega-Max

Niech $d \geqslant 0$, $\alpha < \beta$, S - stan gry. Oznaczmy ponadto $ab = AlfaBeta(S, d, \alpha, \beta)$ oraz $n = NegaMax(S, d)$. Wtedy możemy wyróżnić trzy sytuacje:

- (succes) $\alpha < ab < \beta \quad \Rightarrow \quad ab = n$
- (failing low) $\alpha \geqslant ab \quad \Rightarrow \quad ab \geqslant n$
- (failing high) $\beta \leqslant ab \quad \Rightarrow \quad ab \leqslant n$

Wniosek z implikacji (succes)

$$\forall S \in \mathbb{S}, d \geqslant 0: \qquad AlfaBeta(S, d, -\infty, \infty) = NegaMax(S, d)$$

co określa możliwość bezpośredniego wykorzystania α-β do znalezienia wartości negamaksowej stanu gry.

twierdzenie o związku α-β z Nega-Max

Niech $d \geqslant 0$, $\alpha < \beta$, S - stan gry. Oznaczmy ponadto $ab = AlfaBeta(S, d, \alpha, \beta)$ oraz $n = NegaMax(S, d)$. Wtedy możemy wyróżnić trzy sytuacje:

- (succes) $\alpha < ab < \beta \quad \Rightarrow \quad ab = n$
- (failing low) $\alpha \geqslant ab \quad \Rightarrow \quad ab \geqslant n$
- (failing high) $\beta \leqslant ab \quad \Rightarrow \quad ab \leqslant n$

Wniosek z implikacji (succes)

$$\forall S \in \mathbb{S}, d \geqslant 0: \qquad AlfaBeta(S, d, -\infty, \infty) = NegaMax(S, d)$$

co określa możliwość bezpośredniego wykorzystania α-β do znalezienia wartości negamaksowej stanu gry.

twierdzenie o związku α-β z Nega-Max

Niech $d \geqslant 0$, $\alpha < \beta$, S - stan gry. Oznaczmy ponadto $ab = AlfaBeta(S, d, \alpha, \beta)$ oraz $n = NegaMax(S, d)$. Wtedy możemy wyróżnić trzy sytuacje:

- (succes) $\alpha < ab < \beta \quad \Rightarrow \quad ab = n$
- (failing low) $\alpha \geqslant ab \quad \Rightarrow \quad ab \geqslant n$
- (failing high) $\beta \leqslant ab \quad \Rightarrow \quad ab \leqslant n$

Wniosek z implikacji (succes)

$$\forall S \in \mathbb{S}, d \geqslant 0: \qquad AlfaBeta(S, d, -\infty, \infty) = NegaMax(S, d)$$

co określa możliwość bezpośredniego wykorzystania α-β do znalezienia wartości negamaksowej stanu gry.

twierdzenie o związku α-β z Nega-Max

Niech $d \geqslant 0$, $\alpha < \beta$, S - stan gry. Oznaczmy ponadto $ab = AlfaBeta(S, d, \alpha, \beta)$ oraz $n = NegaMax(S, d)$. Wtedy możemy wyróżnić trzy sytuacje:

- (succes) $\alpha < ab < \beta \quad \Rightarrow \quad ab = n$
- (failing low) $\alpha \geqslant ab \quad \Rightarrow \quad ab \geqslant n$
- (failing high) $\beta \leqslant ab \quad \Rightarrow \quad ab \leqslant n$

Wniosek z implikacji (succes)

$$\forall S \in \mathbb{S}, d \geqslant 0: \qquad AlfaBeta(S, d, -\infty, \infty) = NegaMax(S, d)$$

co określa możliwość bezpośredniego wykorzystania α-β do znalezienia wartości negamaksowej stanu gry.

algorytm α-β - złożoność

- wielkość drzewa poszukiwań (i ilość β-odcięć) ściśle uzależniona od kolejności rozpatrzenia ruchów
- $O(\mathbb{B}^d)$ w najgorszym wypadku (gdy nie nastąpi żadne odcięcie), taka jak dla Nega-Max
- $O(\mathbb{B}^{d/2})$ w najlepszym przypadku (gdy najlepszy z możliwych ruchów zawsze zostanie sprawdzony jako pierwszy)
- $O(\mathbb{B}^{3d/4})$ oczekiwana (dla ruchów wykonanych w losowej kolejności)

Objaśnienia symboli

- d głębokość poszukiwań
- $\mathbb{B}$ średni czynnik rozgałęzienia drzewa poszukiwań

algorytm α-β - złożoność

- wielkość drzewa poszukiwań (i ilość β-odcięć) ściśle uzależniona od kolejności rozpatrzenia ruchów
- $O(\mathbb{B}^d)$ w najgorszym wypadku (gdy nie nastąpi żadne odcięcie), taka jak dla Nega-Max
- $O(\mathbb{B}^{d/2})$ w najlepszym przypadku (gdy najlepszy z możliwych ruchów zawsze zostanie sprawdzony jako pierwszy)
- $O(\mathbb{B}^{3d/4})$ oczekiwana (dla ruchów wykonanych w losowej kolejności)

Objaśnienia symboli

- d głębokość poszukiwań
- $\mathbb{B}$ średni czynnik rozgałęzienia drzewa poszukiwań

algorytm α-β - złożoność

- wielkość drzewa poszukiwań (i ilość β-odcięć) ściśle uzależniona od kolejności rozpatrzenia ruchów
- $O(\mathbb{B}^d)$ w najgorszym wypadku (gdy nie nastąpi żadne odcięcie), taka jak dla Nega-Max
- $O(\mathbb{B}^{d/2})$ w najlepszym przypadku (gdy najlepszy z możliwych ruchów zawsze zostanie sprawdzony jako pierwszy)
- $O(\mathbb{B}^{3d/4})$ oczekiwana (dla ruchów wykonanych w losowej kolejności)

Objaśnienia symboli

- d głębokość poszukiwań
- $\mathbb{B}$ średni czynnik rozgałęzienia drzewa poszukiwań

algorytm α-β - złożoność

- wielkość drzewa poszukiwań (i ilość β-odcięć) ściśle uzależniona od kolejności rozpatrzenia ruchów
- $O(\mathbb{B}^d)$ w najgorszym wypadku (gdy nie nastąpi żadne odcięcie), taka jak dla Nega-Max
- $O(\mathbb{B}^{d/2})$ w najlepszym przypadku (gdy najlepszy z możliwych ruchów zawsze zostanie sprawdzony jako pierwszy)
- $O(\mathbb{B}^{3d/4})$ oczekiwana (dla ruchów wykonanych w losowej kolejności)

Objaśnienia symboli

- d głębokość poszukiwań
- $\mathbb{B}$ średni czynnik rozgałęzienia drzewa poszukiwań

Sposoby redukowania wielkości drzewa poszukiwań

- porządkowanie ruchów (by spowodować większą ilość β-cięć)
- unikajanie redundantnych obliczeń (różne sekwencje ruchów mogą prowadzić do tej samej sytuacji, co nie musi oznaczać konieczności wielokrotnego jej sprawdzania)
- manipulowanie oknem (α, β)
- głębsze przeszukiwanie „ciekawszych” gałęzi (szczędzenie czasu na mniej obiecujących)
- wykonywanie ruchów z bazy

Przykładowe metody

heurystyka historyczna, heurystyka ruchów morderców, iteracyjne pogłębianie, tablica transpozycji

Sposoby redukowania wielkości drzewa poszukiwań

- porządkowanie ruchów (by spowodować większą ilość β-cięć)
- unikajanie redundantnych obliczeń (różne sekwencje ruchów mogą prowadzić do tej samej sytuacji, co nie musi oznaczać konieczności wielokrotnego jej sprawdzania)
- manipulowanie oknem (α, β)
- głębsze przeszukiwanie „ciekawszych” gałęzi (szczędzenie czasu na mniej obiecujących)
- wykonywanie ruchów z bazy

Przykładowe metody

tablica transpozycji

Sposoby redukowania wielkości drzewa poszukiwań

- porządkowanie ruchów (by spowodować większą ilość β-cięć)
- unikajanie redundantnych obliczeń (różne sekwencje ruchów mogą prowadzić do tej samej sytuacji, co nie musi oznaczać konieczności wielokrotnego jej sprawdzania)
- manipulowanie oknem (α, β)
- głębsze przeszukiwanie „ciekawszych” gałęzi (szczędzenie czasu na mniej obiecujących)
- wykonywanie ruchów z bazy

Przykładowe metody

algorytm aspirującego okna, Principal Variation Search, rodzina algorytmów MTD (ang. Memory-enhanced Test Driver)

Sposoby redukowania wielkości drzewa poszukiwań

- porządkowanie ruchów (by spowodować większą ilość β-cięć)
- unikajanie redundantnych obliczeń (różne sekwencje ruchów mogą prowadzić do tej samej sytuacji, co nie musi oznaczać konieczności wielokrotnego jej sprawdzania)
- manipulowanie oknem (α, β)
- głębsze przeszukiwanie „ciekawszych" gałęzi (szczędzenie czasu na mniej obiecujących)
- wykonywanie ruchów z bazy

Przykładowe metody

Quiescence Search, ProbCut, Multi-ProbCut

Sposoby redukowania wielkości drzewa poszukiwań

- porządkowanie ruchów (by spowodować większą ilość β-cięć)
- unikajanie redundantnych obliczeń (różne sekwencje ruchów mogą prowadzić do tej samej sytuacji, co nie musi oznaczać konieczności wielokrotnego jej sprawdzania)
- manipulowanie oknem (α, β)
- głębsze przeszukiwanie „ciekawszych" gałęzi (szczędzenie czasu na mniej obiecujących)
- wykonywanie ruchów z bazy

Przykładowe metody

baza debiutów, baza końcówek

Iteracyjne pogłębianie

- polega na wielokrotnym, coraz głębszym przeszukiwaniu grafu gry (wywoływaniu α-β), dopóty, dopóki dostatecznie dużo czasu pozostało na wykonanie ruchu
- pozwala dostosować głębokość poszukiwań do czasu jakim dysponujemy na wykonanie ruchu
- uporządkowanie następników badanego stanu według ostatnio uzyskanych ocen, tak, by najbardziej obiecujące ruchy były sprawdzane na początku powoduje zwiększenie ilości β-cięć w kolejnych wywołaniach α-β (iteracjach)
- (algorytm aspirującego okna) okno (α, β) można ograniczyć do pewnego otoczenia wartości znalezionej w poprzedniej iteracji (jeśli odpowiedź nie zmieści się w tym otoczeniu to trzeba powtórzyć obliczenia dla danej głębokości w większym oknie)

Iteracyjne pogłębianie

- polega na wielokrotnym, coraz głębszym przeszukiwaniu grafu gry (wywoływaniu α-β), dopóty, dopóki dostatecznie dużo czasu pozostało na wykonanie ruchu
- pozwala dostosować głębokość poszukiwań do czasu jakim dysponujemy na wykonanie ruchu
- uporządkowanie następników badanego stanu według ostatnio uzyskanych ocen, tak, by najbardziej obiecujące ruchy były sprawdzane na początku powoduje zwiększenie ilości β-cięć w kolejnych wywołaniach α-β (iteracjach)
- (algorytm aspirującego okna) okno (α, β) można ograniczyć do pewnego otoczenia wartości znalezionej w poprzedniej iteracji (jeśli odpowiedź nie zmieści się w tym otoczeniu to trzeba powtórzyć obliczenia dla danej głębokości w większym oknie)

Iteracyjne pogłębianie

- polega na wielokrotnym, coraz głębszym przeszukiwaniu grafu gry (wywoływaniu α-β), dopóty, dopóki dostatecznie dużo czasu pozostało na wykonanie ruchu
- pozwala dostosować głębokość poszukiwań do czasu jakim dysponujemy na wykonanie ruchu
- uporządkowanie następników badanego stanu według ostatnio uzyskanych ocen, tak, by najbardziej obiecujące ruchy były sprawdzane na początku powoduje zwiększenie ilości β-cięć w kolejnych wywołaniach α-β (iteracjach)
- (algorytm aspirującego okna) okno (α, β) można ograniczyć do pewnego otoczenia wartości znalezionej w poprzedniej iteracji (jeśli odpowiedź nie zmieści się w tym otoczeniu to trzeba powtórzyć obliczenia dla danej głębokości w większym oknie)

Iteracyjne pogłębianie

- polega na wielokrotnym, coraz głębszym przeszukiwaniu grafu gry (wywoływaniu α-β), dopóty, dopóki dostatecznie dużo czasu pozostało na wykonanie ruchu
- pozwala dostosować głębokość poszukiwań do czasu jakim dysponujemy na wykonanie ruchu
- uporządkowanie następników badanego stanu według ostatnio uzyskanych ocen, tak, by najbardziej obiecujące ruchy były sprawdzane na początku powoduje zwiększenie ilości β-cięć w kolejnych wywołaniach α-β (iteracjach)
- (algorytm aspirującego okna) okno (α, β) można ograniczyć do pewnego otoczenia wartości znalezionej w poprzedniej iteracji (jeśli odpowiedź nie zmieści się w tym otoczeniu to trzeba powtórzyć obliczenia dla danej głębokości w większym oknie)

Tablica transpozycji

Struktura w której, dla każdego odwiedzonego węzła (stanu gry S), zapisujemy:

- wartość (ocenę) stanu gry S
- głębokość na jaką badany był stan S
- informację, czy zapisana wartość jest dokładna, czy też jest górną lub dolną granicą (wynikłą z β-odcięć) tej dokładnej
- opcjonalnie: wskazanie najlepszego następnika S
- opcjonalnie: informacja czy dany węzeł drzewa poszukiwań leży na ścieżce od korzenia do węzła aktualnie sprawdzanego

Uwagi

tablica transpozycji najczęściej realizowana jest za pomocą tablicy haszującej z adresowaniem otwartym

Tablica transpozycji

Struktura w której, dla każdego odwiedzonego węzła (stanu gry S), zapisujemy:

- wartość (ocenę) stanu gry S
- głębokość na jaką badany był stan S
- informację, czy zapisana wartość jest dokładna, czy też jest górną lub dolną granicą (wynikłą z β-odcięć) tej dokładnej
- opcjonalnie: wskazanie najlepszego następnika S
- opcjonalnie: informacja czy dany węzeł drzewa poszukiwań leży na ścieżce od korzenia do węzła aktualnie sprawdzanego

Uwagi

pozwoli to na odczytanie wartości S z tablicy zamiast ponownego jego przeszukiwania...

Tablica transpozycji

Struktura w której, dla każdego odwiedzonego węzła (stanu gry S), zapisujemy:

- wartość (ocenę) stanu gry S
- głębokość na jaką badany był stan S
- informację, czy zapisana wartość jest dokładna, czy też jest górną lub dolną granicą (wynikłą z β-odcięć) tej dokładnej
- opcjonalnie: wskazanie najlepszego następnika S
- opcjonalnie: informacja czy dany węzeł drzewa poszukiwań leży na ścieżce od korzenia do węzła aktualnie sprawdzanego

Uwagi

...jeśli tylko został przeszukany odpowiednio głęboko...

Tablica transpozycji

Struktura w której, dla każdego odwiedzonego węzła (stanu gry S), zapisujemy:

- wartość (ocenę) stanu gry S
- głębokość na jaką badany był stan S
- informację, czy zapisana wartość jest dokładna, czy też jest górną lub dolną granicą (wynikłą z β-odcięć) tej dokładnej
- opcjonalnie: wskazanie najlepszego następnika S
- opcjonalnie: informacja czy dany węzeł drzewa poszukiwań leży na ścieżce od korzenia do węzła aktualnie sprawdzanego

Uwagi

...i w odpowiednim oknie

Tablica transpozycji

Struktura w której, dla każdego odwiedzonego węzła (stanu gry S), zapisujemy:

- wartość (ocenę) stanu gry S
- głębokość na jaką badany był stan S
- informację, czy zapisana wartość jest dokładna, czy też jest górną lub dolną granicą (wynikłą z β-odcięć) tej dokładnej
- opcjonalnie: wskazanie najlepszego następnika S
- opcjonalnie: informacja czy dany węzeł drzewa poszukiwań leży na ścieżce od korzenia do węzła aktualnie sprawdzanego

Uwagi

jeśli nawet S nie został sprawdzony dostatecznie głęboko poprzednim razem, to możemy zacząć przeszukiwanie od potencjalnie najlepszego jego następnika (i liczyć na dużą ilość β-cięć)

Tablica transpozycji

Struktura w której, dla każdego odwiedzonego węzła (stanu gry S), zapisujemy:

- wartość (ocenę) stanu gry S
- głębokość na jaką badany był stan S
- informację, czy zapisana wartość jest dokładna, czy też jest górną lub dolną granicą (wynikłą z β-odcięć) tej dokładnej
- opcjonalnie: wskazanie najlepszego następnika S
- opcjonalnie: informacja czy dany węzeł drzewa poszukiwań leży na ścieżce od korzenia do węzła aktualnie sprawdzanego

Uwagi

umożliwia to wykrywanie powtórek sytuacji (pętli w grafie gry)

Quiescence Search

- polega na unikaniu statycznej oceny stanów niecichych (niespokojnych)
- stan niecichych to taki którego statyczna ocena może znacznie różnić się od dynamicznej, np. w szachach lub warcabach może być to sytuacja w której strona idąca może wykonać bicie (zmieniając ważną cechę stanu jaką jest przewaga materialna)
- statyczna ocena zostaje zastąpiona oceną dynamiczną, tj. dalszym pogłębieniem danej gałęzi drzewa poszukiwań

Quiescence Search

- polega na unikaniu statycznej oceny stanów niecichych (niespokojnych)
- stan niecichych to taki którego statyczna ocena może znacznie różnić się od dynamicznej, np. w szachach lub warcabach może być to sytuacja w której strona idąca może wykonać bicie (zmieniając ważną cechę stanu jaką jest przewaga materialna)
- statyczna ocena zostaje zastąpiona oceną dynamiczną, tj. dalszym pogłębieniem danej gałęzi drzewa poszukiwań

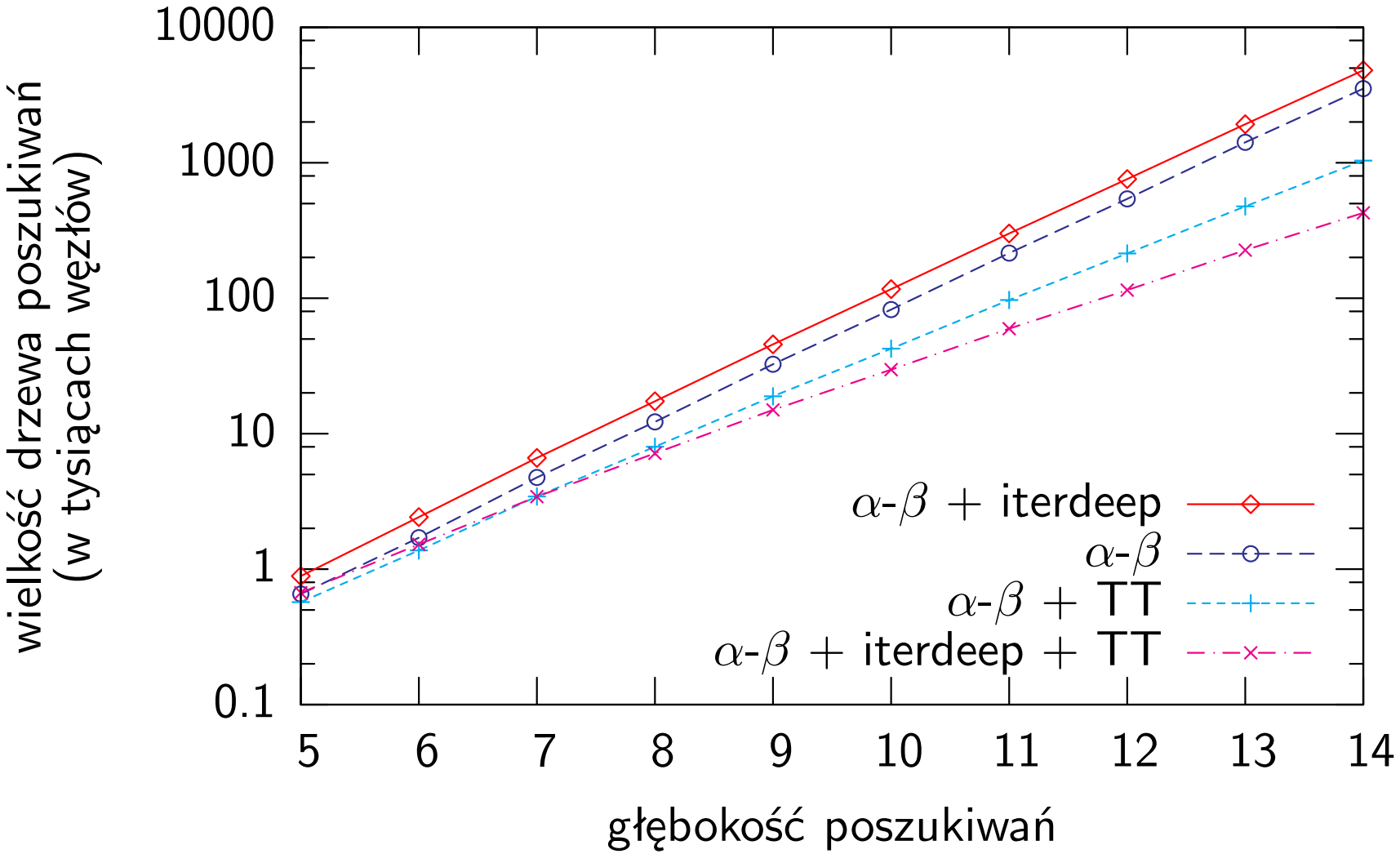
Quiescence Search

- polega na unikaniu statycznej oceny stanów niecichych (niespokojnych)
- stan niecichych to taki którego statyczna ocena może znacznie różnić się od dynamicznej, np. w szachach lub warcabach może być to sytuacja w której strona idąca może wykonać bicie (zmieniając ważną cechę stanu jaką jest przewaga materialna)
- statyczna ocena zostaje zastąpiona oceną dynamiczną, tj. dalszym pogłębieniem danej gałęzi drzewa poszukiwań

Przykładowe porównanie efektywności algorytmów przeszukiwania (dla programu Little Polish grającego w warcaby klasyczne)

zastosowane algorytmy	średnia ilość węzłów odwiedzonych przy poszukiwaniu na głębokość		
	6	10	14
α-β + iterdeep	2 423	116 790	4 818 576
PVS	1 946	87 873	3 537 031
α-β	1 706	82 428	3 522 847
α-β + TT	1 374	42 448	1 037 559
α-β + iterdeep + TT	1 524	29 708	427 175
PVS + iterdeep + TT	1 512	29 082	417 349
PVS + AspWin + TT	1 510	29 043	416 945
PVS + AspWin + TT + ED5	1 501	28 683	412 156

Zależność wielkości drzewa poszukiwań od jego wysokości



Literatura

- **Bruce Moreland** *http://www.seanet.com/~brucemo/topics/topics.htm* (WWW)
- **T.A. Marsland** *Game Tree Searching and Prunning*
- **M. N. J. van Kervinck** *The Design and Implementation of the Rookie 2.0 Chess Playing Program* (2002, mgr)
- **Aske Plaat** *Research, Re: Search & RE-SEARCH* (1996, dr)
- **Michael Buro** *From Simple Features to Sophisticated Evaluation Functions* (1999)
- **Gerald Tesauro** *Temporal Difference Learning and TD-Gammon*

Literatura

- **Halina Kwaśnicka, Artur Spirydowicz** *Uczący się komputer. Programowanie gier logicznych.* (2004, Wrocław)
- **Artur Michalski** *Współczesne techniki przeszukiwania grafów gier dwuosobowych* (slajdy)
- **Adam Kujawski** *Programowanie gry w szachy* (1994, mgr)
- **Arkadiusz Paterek** *Modelowanie funkcji oceniającej w szachach* (2004, mgr)
- **Piotr Beling** *Praktyczne aspekty programowania gier logicznych* (2006, mgr)