Sveučilište u Splitu,
Prirodoslovno-matematički fakultet
Odjel za informatiku

Nikola Vidović

Alfa-Beta obrezivanje

Seminar

Split, 12.2.2025.

Sadržaj

1. Uvod

Minimax algoritam je ključni koncept u teoriji igara. Koristi se za donošenje odluka u igrama s dva igrača, poput šaha ili kružić kružića . Algoritam je posebno koristan u situacijama gdje je potrebno predvidjeti poteze protivnika i odabrati najbolji mogući potez. Međutim, s povećanjem dubine stabla odluke, računska složenost raste eksponencijalno.

Kako bi se smanjila računska složenost, koristi se tehnika alfa-beta obrezivanja, koja omogućuje eliminaciju grana koje sigurno neće utjecati na konačnu odluku. Ovaj seminar objašnjava teorijski aspekt Minimax algoritma s alfa-beta obrezivanjem te prikazuje praktičnu primjenu i usporedbu performansi s i bez alfa-beta obrezivanja.

Nadahnut projektom https://github.com/GeorgeSeif/Tic-Tac-Toe-AI postavljenim na GitHub platformu, odlučio sam napraviti svoju varijaciju koristeći Google Colab.

2. Teorijski dio

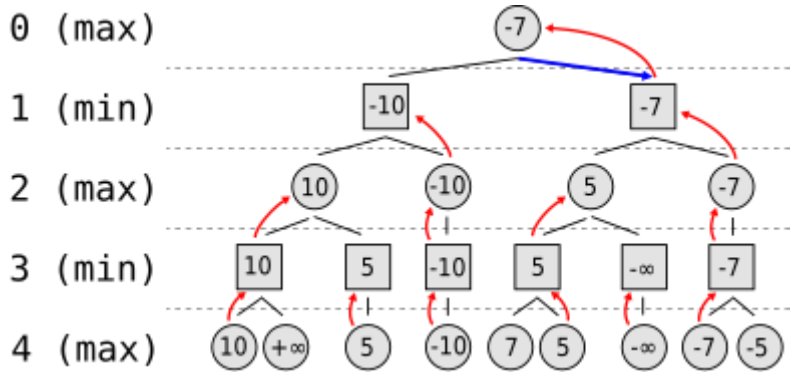
2.1. Minimax Algoritam

Minimax algoritam temelji se na pretpostavci da oba igrača igraju optimalno. Algoritam simulira sve moguće poteze i protupoteze, stvarajući stablo igre. Svaki čvor u tom stablu predstavlja neko buduće stanje igre, a grane predstavljaju moguće poteze. Algoritam dodjeljuje vrijednost svakom čvoru na temelju heurističke funkcije koja procjenjuje koliko je stanje povoljno za trenutnog igrača.

Kako bi objasnili što točno radi ovaj algoritam, prvo moramo uvesti pojmove maksimizirajućeg i minimizirajućeg igrača.

Heuristička funkcija može poprimiti vrijednosti iz skupa realnih brojeva. Svaka vrijednost iz toga skupa označava koliko je stanje igre koje evaluiramo poželjno za kojega igrača. Stoga, ako je vrijednost heurističke funkcije veća od nule, tada će maksimizirajući igrač imati prednost, u suprotnom će prednost imati minimizirajući igrač. Ako je vrijednost funkcije jednaka nuli, tada nijedan igrač nema prednost, a stanje igre se evaluira kao neriješeno.

Slika 1. Primjer Minimax algoritma.



By Nuno Nogueira (Nmnogueira) - Own work
using: http://en.wikipedia.org/wiki/Image:Minimax.svg, created in Inkscape by author, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=2276653

Algoritam rekurzivno evaluira sve moguće poteze i odabire onaj koji vodi do najboljeg mogućeg ishoda, pod pretpostavkom da protivnik igra optimalno.

2.2. Alfa-beta obrezivanje

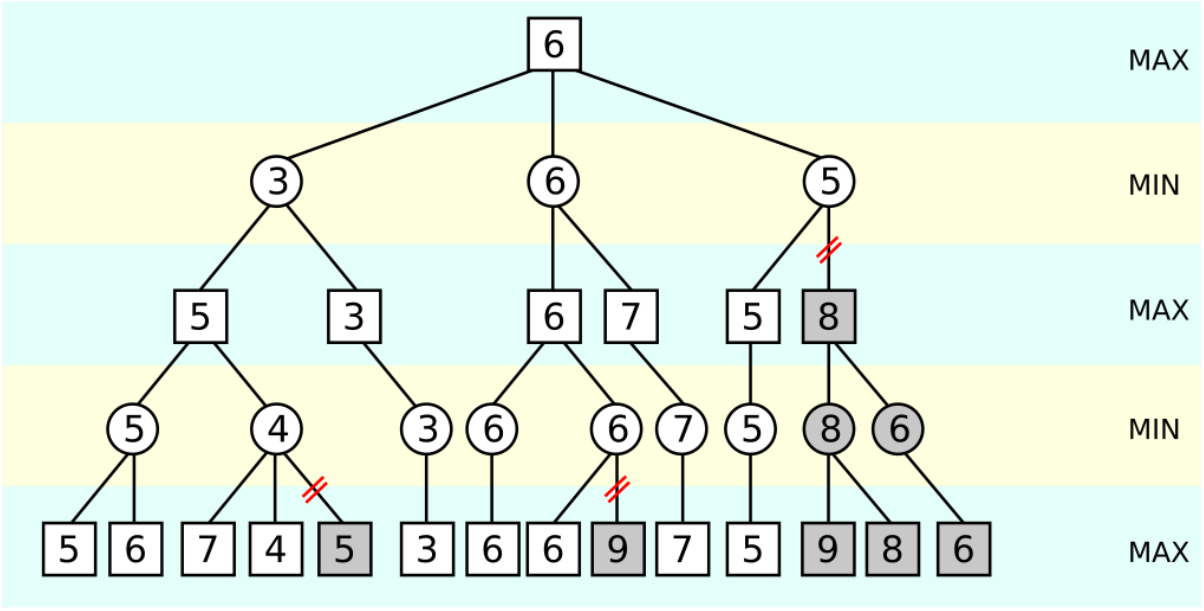
U uvodu smo se već dotakli Alfa-beta obrezivanja koji za svrhu ima reducirati računsku složenost pronalaska najboljeg poteza. Alfa-beta obrezivanje je optimizacijska tehnika Minimax algoritma koja smanjuje broj čvorova koje je potrebno evaluirati. Ova tehnika koristi dva parametra:

Alfa - Najbolja vrijednost maksimizirajućeg igrača do tog trenutka

Beta - Najbolja vrijednost minimizirajućeg igrača do tog trenutka.

Alfa-beta obrezivanje eliminira grane u stablu koje sigurno neće utjecati na konačnu odluku. Primjerice, ako je vrijednost čvora veća od beta vrijednosti, maksimizirajući igrač sigurno neće odabrati taj put, pa se daljnje evaluacije u toj grani mogu preskočiti, tj. možemo „odrezati“ granu. Slično, ako je vrijednost čvora manja od alfa vrijednosti, minimizirajući igrač sigurno neće odabrati taj put pa i u tome slučaju možemo napraviti isto.

Slika 2. Primjer Minimax algoritma sa Alfa-beta obrezivanjem.



By Jez9999, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3708424

3. Implementacija u Google Colabu

Budući da je jako kompleksno pokazati kako ovaj algoritam radi u šahu, ja sam implementirao isti u igri križić-kružić. Korištenjem Pythona u Google Colab okruženju razvio sam dva računalna igrača od kojih će jedan tražiti sljedeći potez koristeći Alfa-beta obrezivanje, dok će drugi koristiti obični Minimax algoritam bez obrezivanja.

3.1. Računalni igrač bez Alfa-beta obrezivanja

Funkcija minimax_no_pruning() vrši evaluaciju čvora. Prvo provjerava je li taj čvor list, tj. je li igra završena pa ako nije, onda će provjeriti evaluaciju svoje djece. Ako je igra završena, onda će se vratiti evaluacija popunjene ploče.

Slijedi kod ove funkcije:

```
def minimax_no_pruning(board, depth, is_maximizing):

  # Provjeri je li igra gotova.
  if check_win(board, 'O'):
    return 1  # Bot je pobijedio.
  if check_win(board, 'X'):
    return -1  # Čovjek je pobijedio.
  if check_draw(board):
    return 0  # Remi.
  # Ako nije kraj igre, nastavi s rekurzijom.
  if is_maximizing:
    best_score = -float('inf')
    for i in range(3):
      for j in range(3):
        if board[i][j] == ' ':
          board[i][j] = 'O'
          score = minimax_no_pruning(board, depth + 1, False)
          board[i][j] = ' '
          best_score = max(score, best_score)
    return best_score
  else:
    best_score = float('inf')
    for i in range(3):
      for j in range(3):
        if board[i][j] == ' ':
          board[i][j] = 'X'
          score = minimax_no_pruning(board, depth + 1, True)
          board[i][j] = ' '
          best_score = min(score, best_score)
    return best_score
```

Zatim, funkcija get_best_move() koja bira najbolji potez pozivajući unutar sebe funkciju minimax_no_pruning() kako bi saznala evaluaciju određenog poteza.

```
def get_best_move(board):

  best_score = -float('inf')
  best_move = (-1, -1)

  for i in range(3):
    for j in range(3):
      if board[i][j] == ' ':
        board[i][j] = 'O'
        score = minimax_no_pruning(board, 0, False)
        board[i][j] = ' '
        if score > best_score:
          best_score = score
          best_move = (i, j)

  return best_move
```

3.2. Računalni igrač sa Alfa-beta obrezivanjem

Funkcija minimax_alpha_beta() radi na istom principu kao i minimax_no_pruning(), samo što je još dodano obrezivanje.

```
def minimax_alpha_beta(board, depth=0, alpha=-float('inf'),
beta=float('inf'), is_maximizing=True):

  # Provjeri je li igra gotova.
  if check_win(board, 'O'):
    return 1  # Bot je pobijedio.
  if check_win(board, 'X'):
    return -1  # Čovjek je pobijedio.
  if check_draw(board):
    return 0  # Remi.

  # Ako nije kraj igre, nastavi s rekurzijom.
  if is_maximizing:
    best_score = -float('inf')
    for i in range(3):
      for j in range(3):
        if board[i][j] == ' ':
          board[i][j] = 'O'
          score = minimax_alpha_beta(board, depth + 1, alpha, beta,
False)
```

```
          board[i][j] = ' '
          best_score = max(score, best_score)
          alpha = max(alpha, best_score)
          if beta <= alpha:
            break  # Beta obrezivanje
    return best_score
  else:
    best_score = float('inf')
    for i in range(3):
      for j in range(3):
        if board[i][j] == ' ':
          board[i][j] = 'X'
          score = minimax_alpha_beta(board, depth + 1, alpha, beta,
True)
          board[i][j] = ' '
          best_score = min(score, best_score)
          beta = min(beta, best_score)
          if beta <= alpha:
            break  # Alfa obrezivanje
    return best_score
```

Funkcija za dobivanje najboljeg poteza nije se mnogo mijenjala. Samo se u pozivu funkcije Alfa-beta obrezivanja proslijede i početni parametri alfa i beta.

```
def get_best_move_ABP(board):

  best_score = -float('inf')
  best_move = (-1, -1)

  for i in range(3):
    for j in range(3):
      if board[i][j] == ' ':
        board[i][j] = 'O'
        score = minimax_alpha_beta(board, 0, -float('inf'),
float('inf'), False)
        board[i][j] = ' '
        if score > best_score:
          best_score = score
          best_move = (i, j)

  return best_move
```

4. Performanse algoritama

Prvi potez u našoj igri igra čovjek, a sljedeći računalo i tako se izmjenjuju. U sljedećoj tablici su navedeni podaci izmjereni tokom igranja protiv oba računalna igrača u slučaju kada čovjek započinje igru potezom 1 (X u gornjem lijevom kutu).

Tablica 1. Usporedba izmjerenih podataka oba oblika Minimax algoritma.

Broj mogućnosti	Sa obrezivanjem		Bez obrezivanja	
	Vrijeme	Čvorovi	Vrijeme	Čvorovi
8	0.049271583557128906 s	26725	0.16448616981506348 s	120371
6	0.0014920234680175781 s	527	0.002568483352661133 s	934
4	0.00016760826110839844 s	43	0.00013256072998046875 s	46
2	3.695487976074219e-05 s	4	1.9073486328125e-05 s	4

4.1. Analiza rezultata

Iz tablice je vidljivo da Alfa-beta obrezivanje značajno smanjuje vrijeme izvođenja i broj evaluiranih čvorova kada imamo puno stanja igre koja moramo istražiti. No ukoliko imamo manje stanja za istražiti, vidimo da nema nekog znatnog poboljšanja. Ovo potvrđuje da Alfa-beta obrezivanje može značajno poboljšati performanse Minimax algoritma ukoliko imamo mnogo čvorova koje moramo istražiti.

5. Zaključak

Minimax algoritam je moćan alat za donošenje odluka u igrama s dva igrača, ali njegova eksponencijalna složenost ograničava njegovu primjenu na duboka stabla odlučivanja. Alfa-beta obrezivanje predstavlja efikasnu optimizaciju koja smanjuje broj čvorova koje treba evaluirati, čime se ubrzava vrijeme izvođenja.

Možda nam se ovo poboljšanje u križić-kružiću ne čini impresivnim jer u tome slučaju imamo ograničen broj poteza, ali kada se ovaj algoritam iskoristi u igrama poput šaha, gdje se već na trećem potezu možete naći u jednom od preko 9 milijuna stanja igre, ovo poboljšanje će sigurno znatno ubrzati izračun sljedećeg poteza.

Analizirajući podatke, zaključujemo da je Alfa-beta obrezivanje neizostavan dio Minimax algoritma kada radimo evaluacije nad velikim skupom mogućih stanja u igrama za dva igrača.

6. Literatura

http://shelf2.library.cmu.edu/Tech/17700646.pdf

https://www.whitman.edu/documents/Academics/Mathematics/2019/Felstiner-Guichard.pdf