جستجوی خصمانه (بازی ها)

دانشگاه صنعتی قوچان

محمدحسین سیگاری

تعریف بازی از دید هوش مصنوعی

- بازی: یک مسئله در محیط **چند عاملی رقابتی**
  - اهداف عامل ها در تضاد هم قرار دارد
  - ممکن است محیط دارای همکاری هم باشد، اما وجود رقابت لازمه بازی است.
- نظریه بازی ها (Game Theory)، شاخه از علم اقتصاد است هر محیط چند عاملی رقابتی را به عنوان بازی مدل می کند.
- بازی ها، یکی از اولین کاربردهایی بود که در هوش مصنوعی مطرح شد. چون بازی ها به نوعی هوشمندی را بیان می کنند.

خواص بازی ها

- قابل مشاهده بودن
  - کاملا قابل مشاهده/قابل مشاهده جزیی
- ایستایی
  - ایستا/پویا
- قطعیت
  - قطعی/غیرقطعی
- گسستگی
  - گسسته/پیوسته (معمولا محیط های پیوسته مرود توجه علم تئوری بازی ها نیست)
- رویداد
  - ترتیبی/اپیزودیک (معمولا بازی ها ترتیب هستند)
- چند عاملی
  - چندعاملی رقابتی (ممکن است بین بعضی از عامل ها همکاری هم باشد)

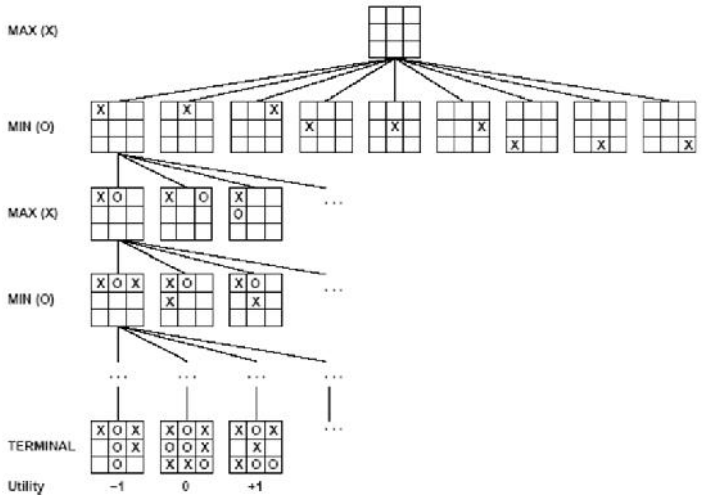
انواع بازی

- از نظر تعداد بازیکن
  - دو نفره
  - چند نفره
- از نظر امتیازدهی
  - بازی های مجموع صفر (Zero Sum)
    - بازی های مجموع ثابت
  - بازی های مجموع غیرصفر (Non-Zero Sum)

فرموله کردن بازی

- بازی دو نفره: Min و Max
  - اول Max حرکت می کند و سپس به نوبت بازی می کنند تا بازی تمام شود
  - در پایان بازی، برنده جایزه و بازنده جریمه می شود
- بازی به عنوان یک جستجو:
  - حالت اولیه (initial state): موقعیت صفحه و شناسه های قابل حرکت
  - تابع جانشین (successor): لیستی از (حالت، حرکت) که معرف یک حرکت معتبر است
  - آزمون هدف (goal): پایان بازی چه موقع است؟ حالتهای پایانه (Terminal)
  - تابع سودمندی (utility): برای هر حالت پایانه یک مقدار عددی را ارائه می کند. مثلا برنده (+۱) و بازنده (-۱)
- حالت اولیه و حرکات معتبر برای هر بازیکن، **درخت بازی** را برای آن بازی ایجاد می کند

بازی XOX (Tic-Tac-Toe)

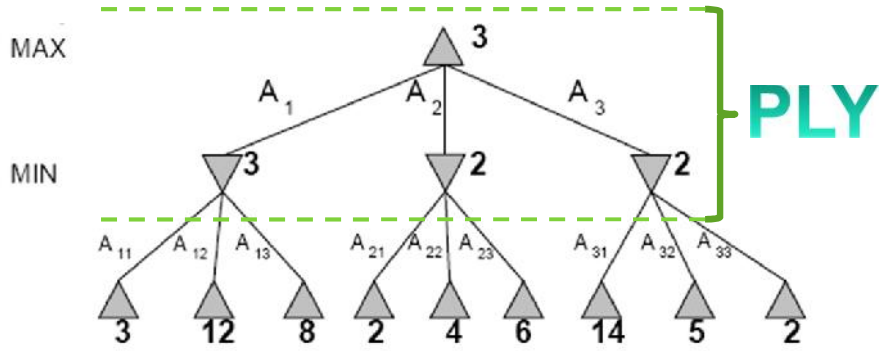


راهبرد بهینه بازی

- فرض می کنیم بازی با MAX آغاز می شود.
  - اگر هم بازی با MIN آغاز شود، بعد از انجام حرکت اول توسط MIN، بازی را شروع شده فرض می کنیم
- فرض می کنیم هر دو بازیکن بدون خطا و به صورت **کاملا بهینه** بازی می کنند. بنابراین هر بازیکن سعی دارد، امتیاز خود را بیشینه کند.
  - اگر فرض کنیم بازی مجموع صفر باشد، می توان فرض کرد MAX می خواهد امتیاز را بیشینه و MIN می خواهد امتیاز را کمینه کند.

الگوریتم جستجوی Min-Max

جستجوی عمقی



تعریف Ply:

چند مرحله از بازی که در آن تمام بازیکن ها یکبار مهلت بازی کردن داشته باشند

برای بازی دو نفره، هر ply شامل دو مرحله است

الگوریتم جستجوی Min-Max

```
function MINIMAX-DECISION(state) returns an action
  inputs: state, current state in game
  v←MAX-VALUE(state)
  return the action in SUCCESSORS(state) with value v
```

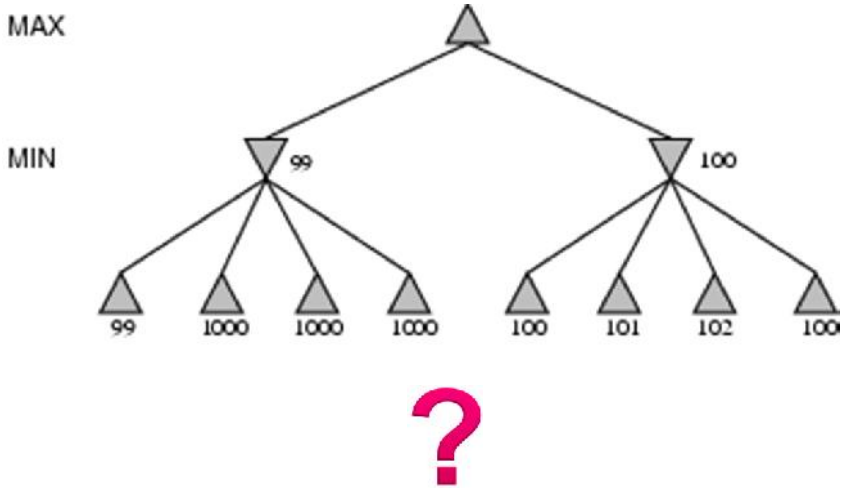
```
function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← -∞
  for a,s in SUCCESSORS(state) do
    v ← MAX(v,MIN-VALUE(s))
  return v
```

```
function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for a,s in SUCCESSORS(state) do
    v ← MIN(v,MAX-VALUE(s))
  return v
```


الگوریتم جستجوی Min-Max

- کامل بودن:
  - کامل است، اگر عمق درخت محدود باشد
- بهینه بودن:
  - بلی
- پیچیدگی زمانی:
  - $O(b^m)$
- پیچیدگی فضا:
  - $O(bm)$

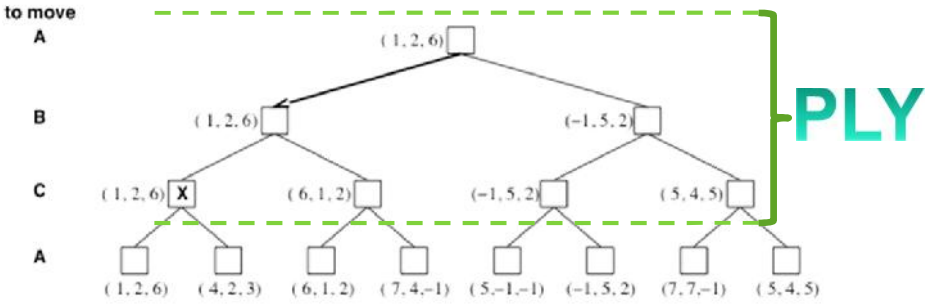
الگوریتم جستجوی Min-Max



بازی های چند نفره

- در بازی چند نفره، فارغ از بحث مجموع صفر بودن امتیازدهی، می توان از یک بردار برای نمایش میزان امتیاز بازکن ها در نظر گرفت. هر عنصر از این بردار مربوط به یک بازیکن است.
- هر مرحله از بازی، نوبت یک بازیکن است که در آن مرحله سعی می کند امتیاز خود را **بیشینه** کند.
- هر ply برابر تعداد مراحلی است که تمام بازیکن ها مهلت بازی داشته باشند. بنابراین برای یک بازی k نفره، هر ply شامل k مرحله خواهد بود.

بازی های چند نفره



بازی های چند نفره

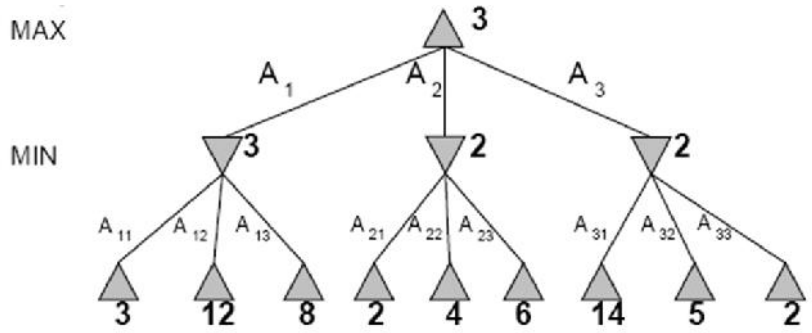
- در بازی های چند نفره ممکن است اتحاد (alliance) تشکیل شود
- انواع اتحاد:
  - اتحاد خواسته/ناخواسته: تشکیل یک اتحاد بین اعضا، ممکن است با هماهنگی قبلی یا بدون هماهنگی قبلی بین آنها باشد
  - اتحاد دائم/موقت: تشکیل یک اتحاد بین اعضا، ممکن است با دائم یا موقت باشد
- به عنوان مثال، ممکن است در یک بازی سه نفره، بین دو بازیکن ضعیف به طور موقت اتحادی شکل گیرد تا بازیکن قوی را شکست دهند، اما بعد از ضعیف کردن آن بازیکن، آن اتحاد از بین رفته و اتحاد دیگری شکل گیرد.

مثالی برای نشان دادن پیچیدگی بازی ها

- در بازی شطرنج
  - متوسط فاکتور انشعاب برابر ۳۵ است
  - هر بازیکن حدودا ۵۰ حرکت انجام می دهد
- تعداد برگ های درخت بازی: $۳۵^{۱۰۰}$ معادل با $۱۰^{۱۵۴}$
- مدت زمان جستجوی کل درخت با استفاده از کامپیوتری که در هر ثانیه ۱ میلیارد گره را بررسی کند: بیش از ۳×۱۰^{۱۳۷} سال

- تعداد برگ های درخت بازی، میزان پیچیدگی زمانی الگوریتم Min–Max را نشان می دهد

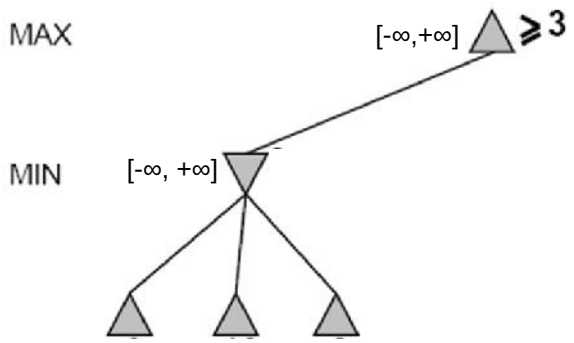
هرس آلفا–بتا (α–β Pruning)



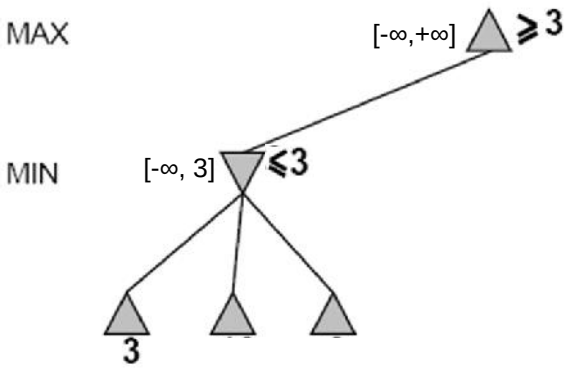
هرس آلفا–بتا (α–β Pruning)

- بسیاری از شاخه های درخت بازی در جستجوی Min–Max را می توان حذف نمود، بدون آنکه بهینه بودن جستجو نقض شود
- آلفا α: مقدار بهترین انتخابی است که تاکنون در هر نقطه انتخاب در مسیر مربوط به Max پیدا شده
- بتا β: مقدار بهترین انتخابی است که تاکنون در هر نقطه انتخاب در مسیر مربوط به Min پیدا شده

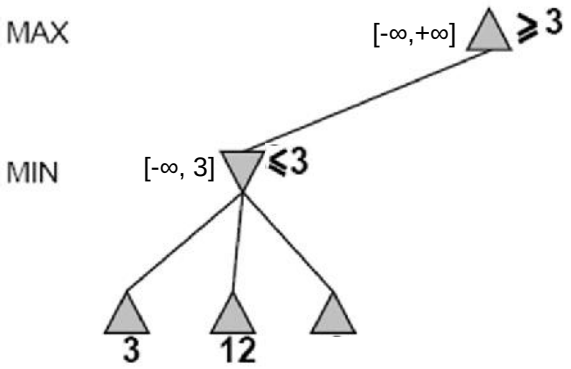
هرس آلفا–بتا (α–β Pruning)



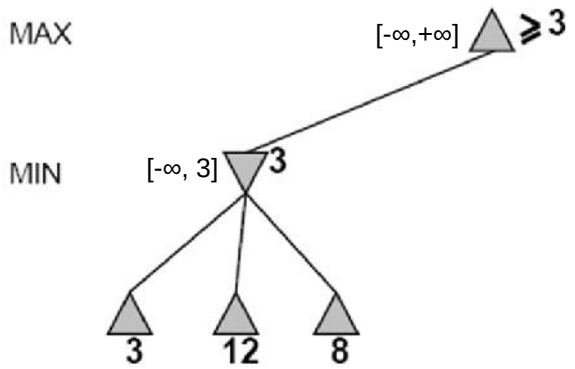
هرس آلفا–بتا (α–β Pruning)



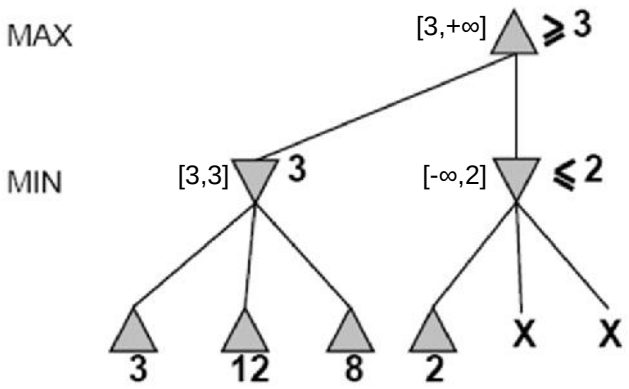
هرس آلفا–بتا (α–β Pruning)



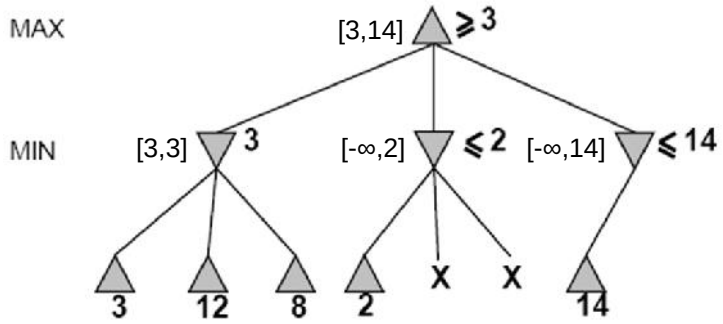
هرس آلفا–بتا (α–β Pruning)



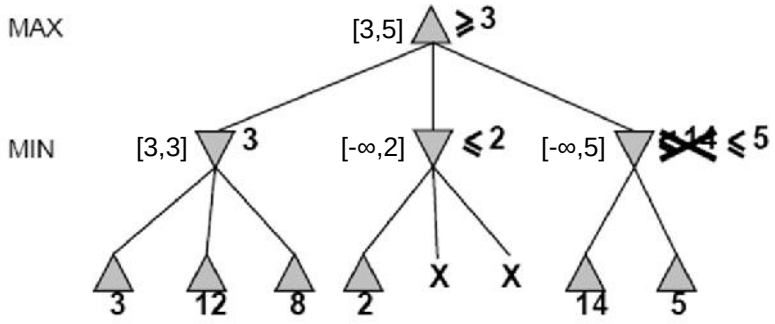
هرس آلفا–بتا (α–β Pruning)



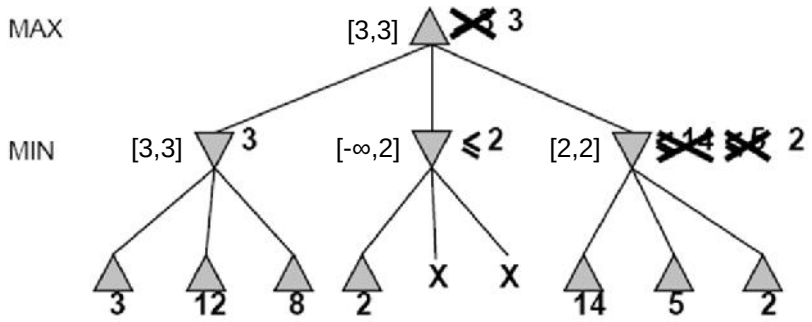
هرس آلفا–بتا (α–β Pruning)



هرس آلفا–بتا (α–β Pruning)



هرس آلفا–بتا (α–β Pruning)



الگوریتم هرس آلفا–بتا (α–β Pruning)

function ALPHA-BETA-SEARCH(*state*) returns *an action*
  inputs: *state*, current state in game
  v←MAX-VALUE(*state*, -∞ , +∞)
  return the *action* in SUCCESSORS(*state*) with value *v*

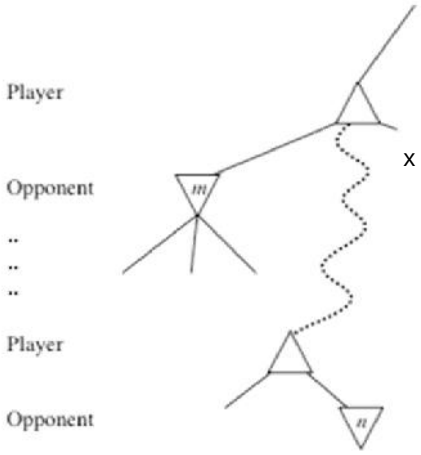
الگوریتم هرس آلفا–بتا (α–β Pruning)

function MAX-VALUE(*state*,~ ,) returns *a utility value*
  if TERMINAL-TEST(*state*) then return UTILITY(*state*)
  v ← - ∞
  for *a,s* in SUCCESSORS(*state*) do
    v ← MAX(*v,*MIN-VALUE(*s*, ~ ,))
    if *v* ≥ then return *v*
    ~ ← MAX(~ ,*v*)
  return *v*

الگوریتم هرس آلفا–بتا (α–β Pruning)

function MIN-VALUE(*state*, ~ ,) returns *a utility value*
  if TERMINAL-TEST(*state*) then return UTILITY(*state*)
  v ← + ∞
  for *a,s* in SUCCESSORS(*state*) do
    v ← MIN(*v,*MAX-VALUE(*s*, ~ ,))
    if *v* ≤ ~ then return *v*
      ← MIN(,*v*)
  return *v*

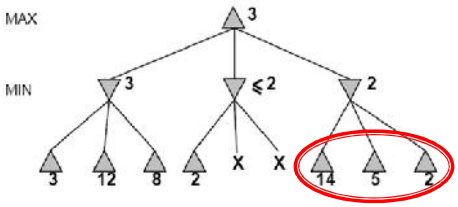
هرس آلفا–بتا (α–β Pruning)

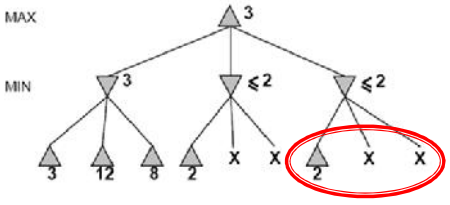


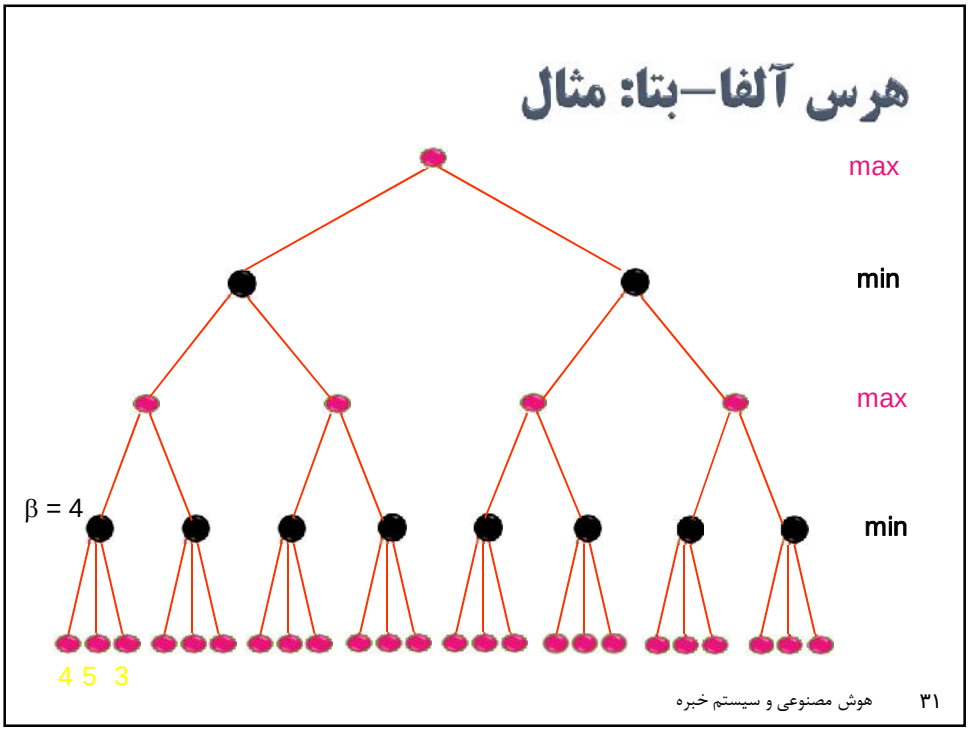
- حالت کلی هرس آلفا-بتا
  - اگر برای گره x فرزندی مانند m و نواده ای مانند n وجود داشته باشد که مقدار سودمندی m بیشتر از سودمندی n باشد و m قبل از n مشاهده شده باشد، گره n مشاهده نشده و هرس خواهد شد.

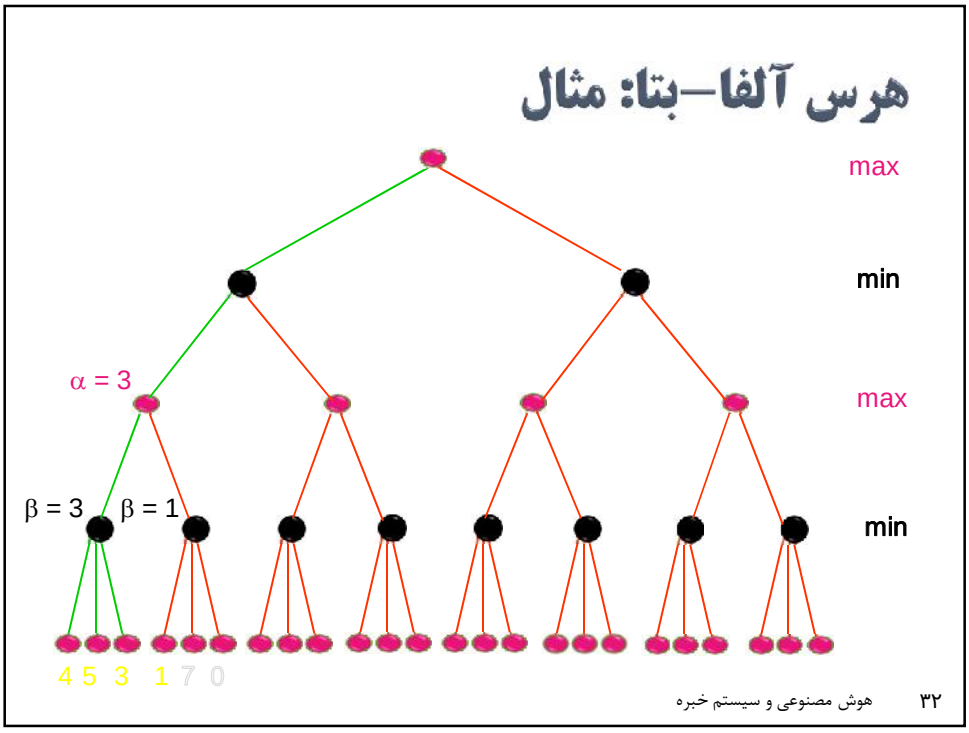
هرس آلفا–بتا (α–β Pruning)

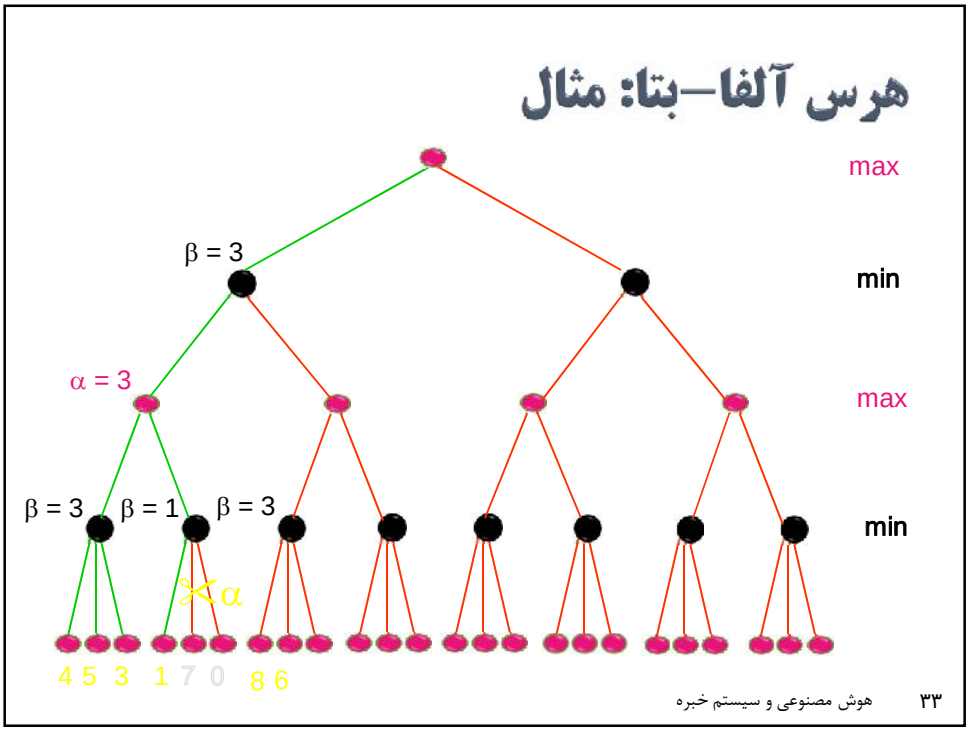
- در هرس آلفا-بتا، ترتیب قرار گرفتن فرزندان نسبت به گره والد در تعداد شاخه های هرس شده بسیار موثر است

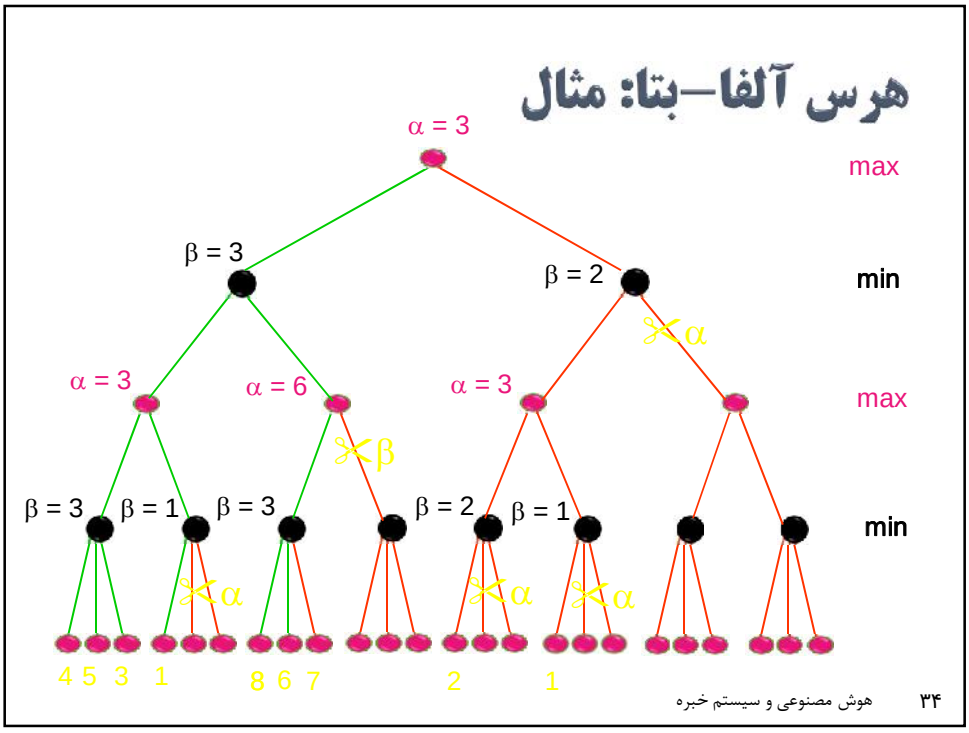




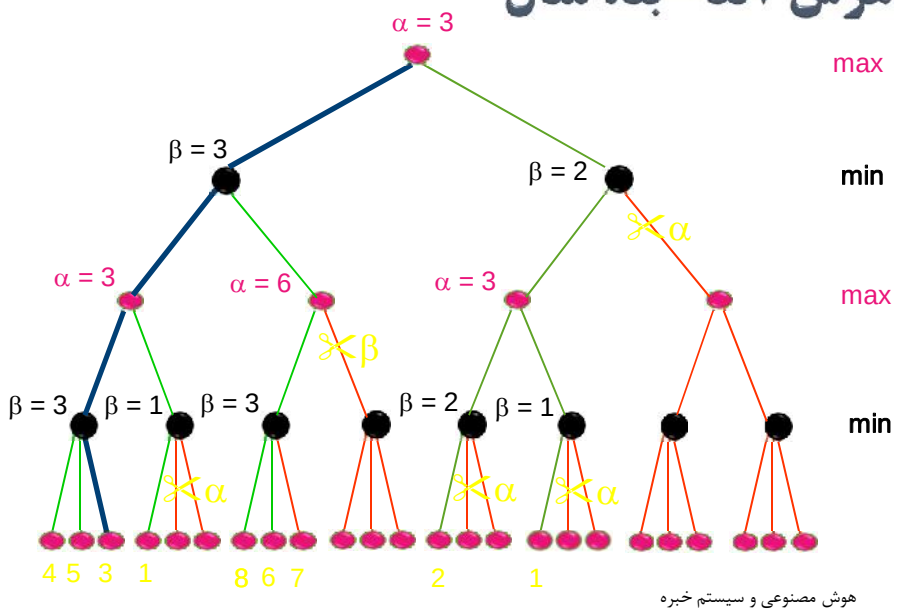

هرس آلفا–بتا: مثال
max
min
max
min
β = 4
4 5 3
هوش مصنوعی و سیستم خبره
۳۱


هرس آلفا–بتا: مثال
max
min
max
min
α = 3
β = 3
β = 1
4 5 3 1 7 0
هوش مصنوعی و سیستم خبره
۳۲


هرس آلفا–بتا: مثال
max
min
max
min
β = 3
α = 3
β = 3
β = 1
β = 3
α
4 5 3 1 7 0 8 6
هوش مصنوعی و سیستم خبره
۳۳


هرس آلفا–بتا: مثال
α = 3
max
β = 3
β = 2
min
α
α = 3
α = 6
α = 3
max
β
β = 3
β = 1
β = 3
β = 2
β = 1
min
α
α
α
4 5 3 1
8 6 7
2
1
هوش مصنوعی و سیستم خبره
۳۴

هرس آلفا–بتا: مثال



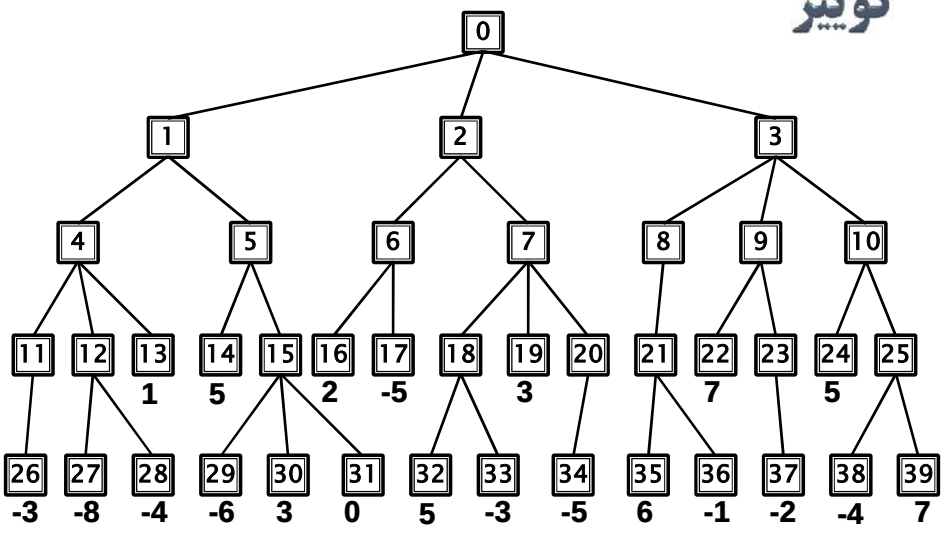
هرس آلفا–بتا (α–β Pruning)

- اگر ترتیب قرار گرفتن گره ها به گونه ای باشد که ابتدا گره های بهتر مورد بررسی قرار گیرند، هرس آلفا-بتا تعداد شاخه های بیشتری از درخت بازی را هرس خواهد کرد.
- در این صورت مرتبه زمانی الگوریتم Min–Max با هرس آلفا-بتا از مرتبه $O(b^{m/2})$ خواهد شد. به عبارت دیگر فاکتور انشعاب موثر از b به $\sqrt{b}$ تغییر می یابد.
- مثلا در بازی شطرنج که فاکتور انشعاب حدود ۳۵ است، به ۶ تقلیل می یابد.

هرس آلفا–بتا (α–β Pruning)

- اگر ترتیب قرار گرفتن گره ها به صورت تصادفی باشد، هرس آلفا-بتا باز هم بعضی شاخه های درخت بازی را هرس خواهد کرد، اما این تعداد شاخه ها چندان زیاد نیست.
- در این صورت مرتبه زمانی الگوریتم Min–Max با هرس آلفا-بتا از مرتبه $O(b^{3m/4})$ خواهد شد. در این حالت نیز فاکتور انشعاب موثر کاسته شده است.

کوییز

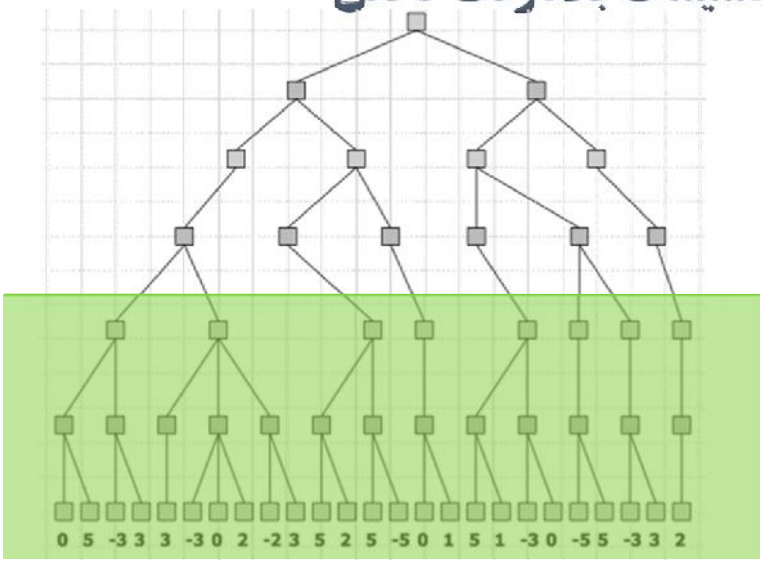


اعمال هرس آلفا-بتا بر روی درخت

تصمیمات بلادرنگ ناقص

- هرچند هرس آلفا-بتا باعث کاهش فاکتور موثر انشعاب و در نتیجه کاهش حجم محاسبات می شود، اما برای بسیاری از بازی ها، باز هم درخت جستجو بسیار بزرگ است و امکان جستجوی درخت تا رسیدن به گره های پایانه (terminal node) میسر نیست.
- برای این که بتوان در مدت زمان معینی (حدود چند دقیقه)، تصمیم مناسب را اتخاذ کرد، باید فقط تا حد مشخصی از عمق درخت بازی جستجو شود.
  - اگر نتوانیم تا گره برگ ادامه دهیم، میزان ارزش هر گره غیربرگ چگونه تعیین شود؟

تصمیمات بلادرنگ ناقص



تصمیمات بلادرنگ ناقص

- راه حل:
  - جایگزینی تابع ارزیابی (evaluation) به جای تابع سودمندی (utility)
  - جایگزینی تست گره پایانه (terminal test) با گره توقف (cut-off test)

- تابع ارزیابی یک تابع اکتشافی است که تخمینی از تابع سودمندی می باشد
- خواص تابع ارزیابی (evaluation):
  - محاسبه آن پیچیده نباشد
  - گره ها را طوری ارزیابی کند که شانس برنده شدن زیاد باشد (تخمین خوبی از تابع سودمندی باشد)

- معمولا گره های توقف، گره های عمق معینی از درخت بازی هستند.

مثال: بازی شطرنج

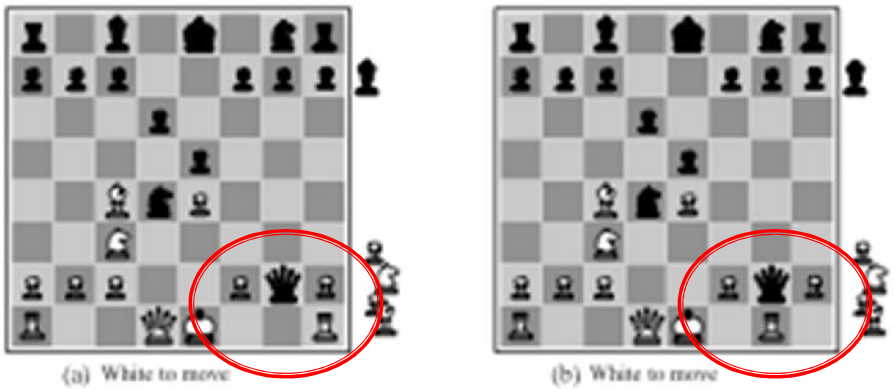
- تابع ارزیابی:
  - یک تابع خطی وزن دار

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

  - i: نوع خاصی از مهره، مثل سرباز، رخ و ...
  - f_i: تعداد مهره موجود از آن نوع
  - w_i: وزن آن مهره (میزان اهمیت مهره)

- معایب:
  - مکان قرارگیری مهره ها در صفحه شطرنج بی اهمیت است
  - تغییر وزن (میزان اهمیت) مهره ها، باعث تغییر تابع ارزیابی و تغییر در روند بازی می شود
  - این روش، روش خوبی برای بیان میزان سودمندی واقعی هر حالت نیست

مثال: بازی شطرنج



(a) White to move (b) White to move

مثال: بازی شطرنج

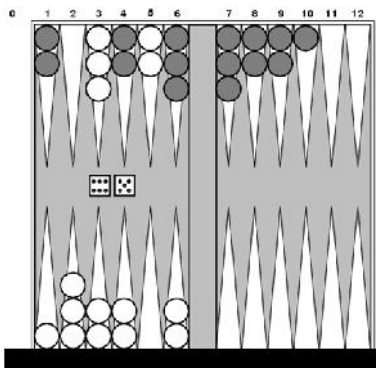
اثر افق

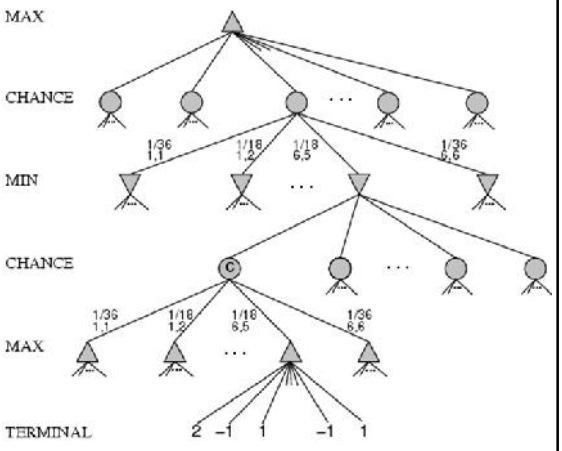
Horizon Effect

اثر افق (Horizon Effect)

- علل اصلی وجود اثر افق:
  - محدودیت عمق جستجو
  - تخمین ناصحیح تابع ارزیابی برای تخمین مقدار سودمندی گره ها

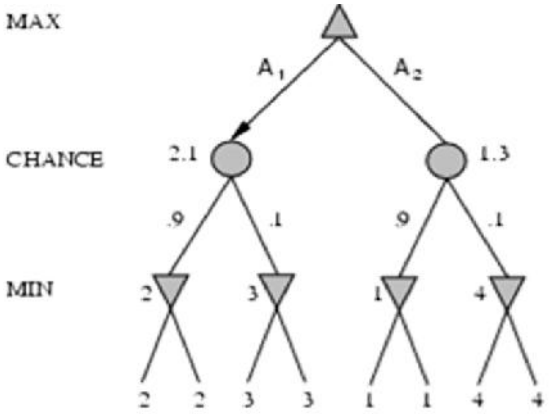
بازی های شامل عنصر شانس





بازی های شامل عنصر شانس

- در بازی های شامل عنصر شانس، باید از امید ریاضی (Expected Value) میزان سودمندی هر گره استفاده کرد.



بازی های شامل عنصر شانس

- تاثیرات منفی عنصر شانس و مقدار سودمندی گره ها در تصمیم گیری!!!