هوش مصنوعی

فصل ۵

جستجوی تخاصمی و بازی‌ها

Adversarial Search and Game

کاظم فولادی قلعه

دانشکده مهندسی، دانشکدگان فارابی

دانشگاه تهران

http://courses.fouladi.ir/ai

هوش مصنوعی

جستجوی رقابتی

۱

بازی‌ها

«رقابت» در محیط چندعاملی

هماهنگی
Coordination

همکاری
Cooperation

رقابت
Competition

اهداف عامل‌ها در تعارض با یکدیگر است.

جستجوی تخاصمی (بازی)
Adversarial Search

حریف پیش‌بینی‌ناپذیر: راه‌حل یک استراتژی است.

مشخص کردن یک حرکت برای هر پاسخ ممکن حریف

نظریه‌ی بازی

نظریه‌ی ریاضی بازی

GAME THEORY (MATHEMATICAL GAME THEORY)

نظریه‌ی بازی
Game Theory

شاخه‌ای از علم اقتصاد

محیط‌های چندعاملی به عنوان یک **بازی** دیده می‌شوند.

(با این فرض که اثر عامل‌ها بر هم چشمگیر است؛ چه همکار باشند چه رقیب)

مشخصات «بازی‌ها» در هوش مصنوعی

قطعی *Deterministic*	نوبتی *Turn-Taking*	دو نفره *Two-Player*	مجموع صفر *Zero-Sum*	با اطلاعات کامل *Perfect Information*
؟	؟	؟	؟	؟
اتفاقی *Stochastic*	پیوسته *Continuous*	چند نفره *Multi-Player*	مجموع ناصفر *Non Zero-Sum*	با اطلاعات ناکامل *Imperfect Information*

مشخصات «بازی‌ها» در هوش مصنوعی

بازی‌های ساده

محیط بازی قطعی	عامل‌ها به صورت نوبتی بازی می‌کنند	دو عامل در بازی نقش دارند	امتیاز دو عامل در پایان بازی مساوی و از نظر علامت مخالف	محیط بازی مشاهده‌پذیر کامل
قطعی *Deterministic*	**نوبتی** *Turn-Taking*	**دو نفره** *Two-Player*	**مجموع صفر** *Zero-Sum*	**با اطلاعات کامل** *Perfect Information*
؟	؟	؟	؟	؟
اتفاقی *Stochastic*	پیوسته *Continuous*	چند نفره *Multi-Player*	مجموع ناصفر *Non Zero-Sum*	با اطلاعات ناکامل *Imperfect Information*

هوش مصنوعی

جستجوی رقابتی

۲

تصمیم‌های بهینه در بازی‌ها

فرمول‌بندی مسئله‌ی «بازی»

مؤلفه‌های شش‌گانه‌ی تعریف مسئله‌ی بازی

مؤلفه‌های شش‌گانه‌ی تعریف مسئله‌ی «بازی»

حالت آغازین Initial State	حالتی که عامل از آن شروع می‌کند.
بازیکن‌ها Players	تعریف اینکه کدام عامل در یک حالت s باید حرکت کند: $\text{PLAYER}(s)$
کنش‌های ممکن Available Actions	کنش‌های ممکن عامل (حرکت‌ها: moves) در هر حالت s: $\text{ACTIONS}(s)$
مدل گذار Transition Model	نتیجه‌ی هر حرکت: در هر حالت s، هر کنش a چه می‌کند؟: $\text{RESULT}(s,a)$
آزمون پایان Terminal Test	وقتی حالت s پایان بازی باشد، **true** و گرنه **false** برمی‌گرداند: $\text{TERMINAL-TEST}(s)$ **حالت‌های پایانی** (terminal states): حالت‌هایی که در آنها بازی به پایان می‌رسد.
تابع سودمندی Utility Function	تابع ارزش‌دهی عددی نهایی برای بازی خاتمه یافته در حالت s برای بازیکن p: $\text{UTILITY}(s,p)$ مثلاً در بازی شطرنج: برد (+1)، باخت (0) و مساوی (½)
تابع هدف Objective Function	**بازی مجموع–صفر (مجموع–ثابت):** مجموع پی‌آف همه‌ی بازیکن‌ها برای همه‌ی نمونه‌های بازی **مشابه** است. مثلاً در بازی شطرنج: برد (+1−0)، باخت (−1+0) و مساوی (½+½)
تابع تسویه حساب Payoff Function	

درخت بازی

مؤلفه‌های شش‌گانه‌ی تعریف مسئله‌ی «بازی»

- **حالت آغازین** Initial State
- **بازیکن‌ها** Players
- **کنش‌های ممکن** Available Actions
- **مدل گذار** Transition Model
- **آزمون پایان** Terminal Test
- **تابع سودمندی** Utility Function

درخت بازی *Game Tree*

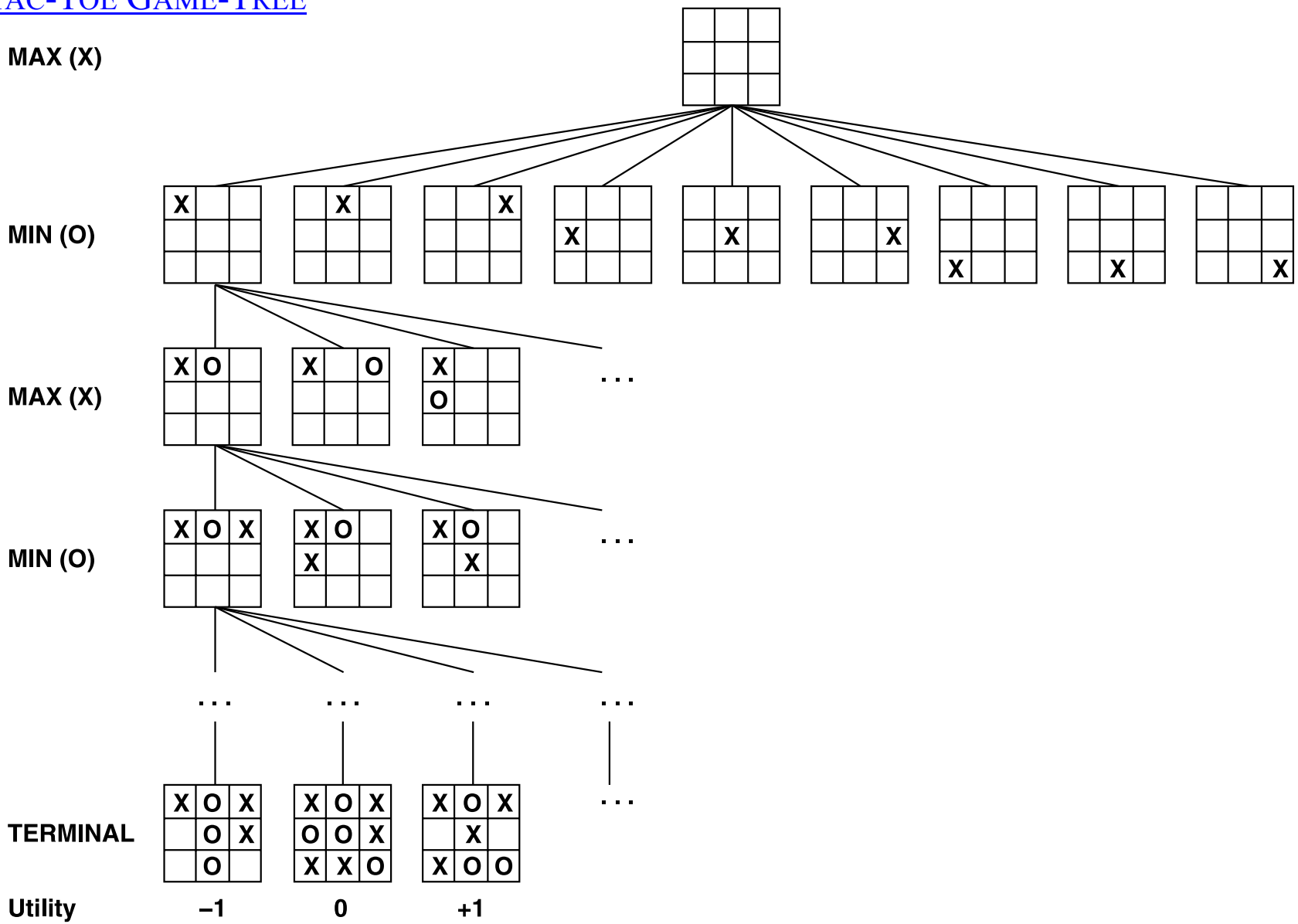
- حالتی که عامل از آن شروع می‌کند.
- کنش‌های ممکن عامل (حرکت‌ها: moves) در هر حالت s : $\text{ACTIONS}(s)$
- نتیجه‌ی هر حرکت: در هر حالت s، هر کنش a چه می‌کند؟ : $\text{RESULT}(s,a)$

درختی که گره‌های آن **حالت‌های بازی** و یال‌های آن **حرکت‌ها** است.

درخت بازی

مثال: بازی دوز

TIC-TAC-TOE GAME-TREE

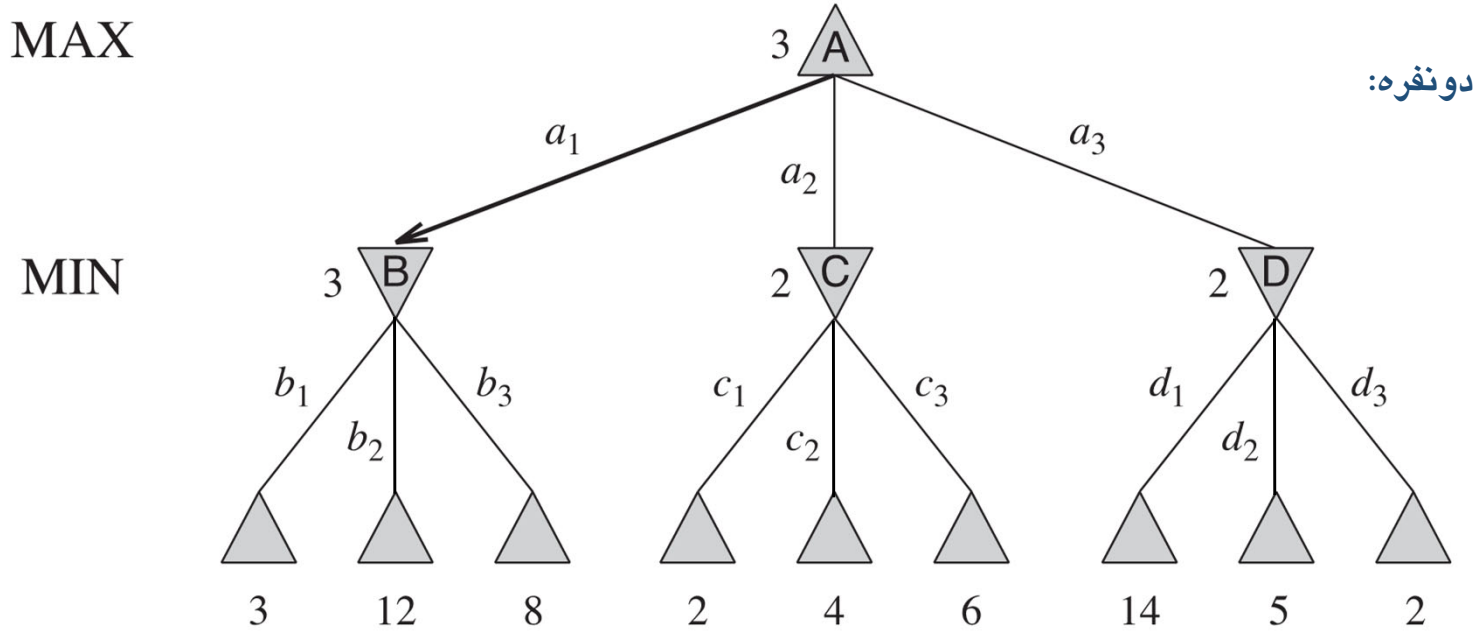


می‌نیماکس

MINIMAX

انتخاب حرکت به سمت بالاترین ارزش می‌نیماکس
= بهترین پی‌آف قابل دسترس در برابر بهترین بازیکن رقیب

مثال: **بازی دونفره:**



هر دو عامل رسیونال هستند و می‌خواهند سود خود را ماکزیمم کنند:
بازی مجموع صفر است: هر امتیاز عامل معادل با منفی امتیاز رقیب اوست.

عامل: ماکزیمم‌کننده‌ی سود خود

رقیب: می‌نیمم‌کننده‌ی سود رقیب (= ماکزیمم‌کننده‌ی سود خود [منفی سود رقیب])

الگوریتم می‌نیماکس

تابع ریاضی

$$\textsc{Minimax}(s) = \begin{cases} \textsc{Utility}(s) & \text{if } \textsc{Terminal-Test}(s) \\ \max_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{max} \\ \min_{a \in Actions(s)} \textsc{Minimax}(\textsc{Result}(s,a)) & \text{if } \textsc{Player}(s) = \textsc{min} \end{cases}$$

الگوریتم می‌نیماکس

شبه کد

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))
```

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

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

الگوریتم جستجوی می‌نیماکس

ویژگی‌ها

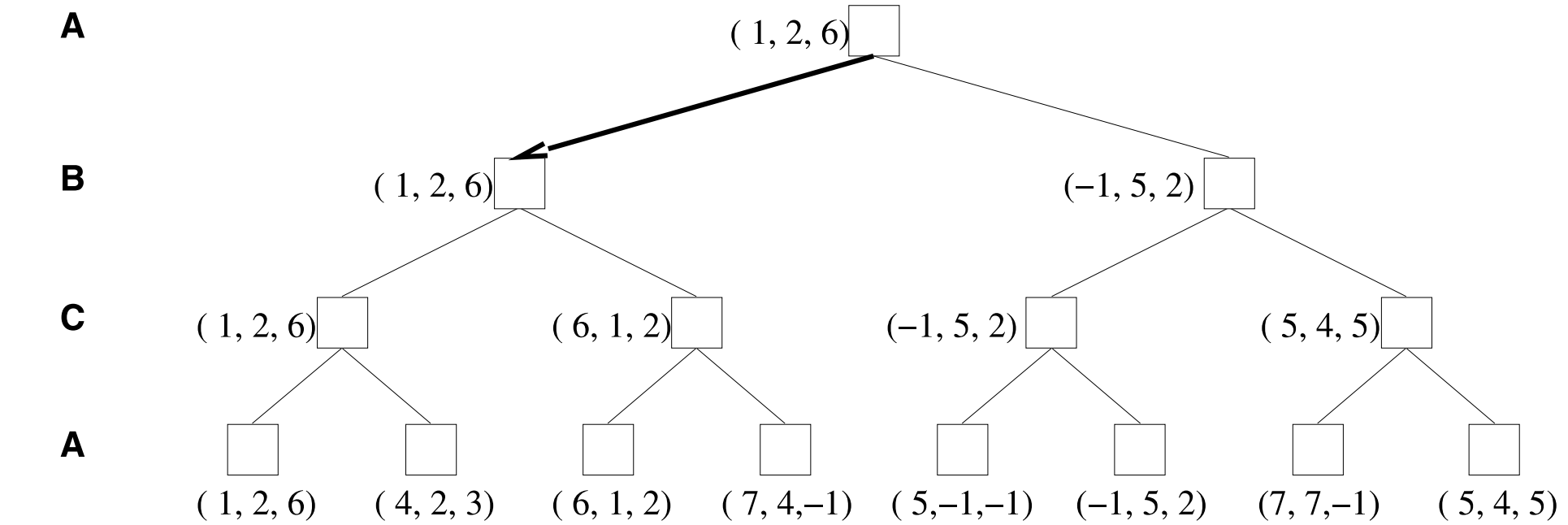
ارزیابی الگوریتم جستجوی می‌نیماکس			
۴	۳	۲	۱
پیچیدگی فضایی *Space Complexity*	**پیچیدگی زمانی** *Time Complexity*	**بهینگی** *Optimality*	**تمامیت** *Completeness*
خطی $O(bm)$ پیمایش عمق–اول	نمایی $O(b^m)$	بله (در مقابل حریف بهینه) اگر حریف بهینه بازی نکند، لزوماً بهینه نیست.	بله (در صورت متناهی بودن درخت بازی) در یک درخت نامتناهی هم ممکن است استراتژی متناهی وجود داشته باشد!

زمان جستجو بسیار بالاست (مثلاً برای شطرنج $m \approx 100$ و $b \approx 35$) ⇐ **نیاز به روش‌های برای کاهش فضای جستجو**

تصمیم‌های بهینه در بازی‌های چندنفره

به هر گره، برداری از امتیازهای هر بازیکن نسبت داده می‌شود.
هر بازیکن برای انتخاب حرکت، روی امتیاز خودش ماکزیمم می‌گیرد.

$$\langle v_A, v_B, v_C \rangle$$



در بازی‌های دونفره‌ی مجموع–صفر، بردار $\langle v_A, v_B \rangle = \langle v_A, -v_A \rangle$ را داریم که با یک عدد v_A قابل خلاصه شدن است.

هوش مصنوعی

جستجوی رقابتی

۳

هرس
آلفا ـ بتا

هرس کردن درخت بازی

مشکل بزرگ الگوریتم جستجوی می‌نیماکس:
وجود **رابطه‌ی نمایی** بین تعداد حرکت‌های بازی و تعداد حالت‌های بازی

مقابله با این مشکل

هرس کردن درخت بازی

هرس کردن *Pruning*	نادیده گرفتن بخشی از درخت جستجو که در راه‌حل تأثیر ندارد.

هرس آلفا–بتا

ALPHA-BETA PRUNING

α-β Pruning

در **هرس آلفا–بتا** همانند روش می‌نیماکس عمل می‌شود؛
با این تفاوت که شاخه‌های غیر مؤثر در تصمیم‌گیری نهایی، دنبال نمی‌شوند.

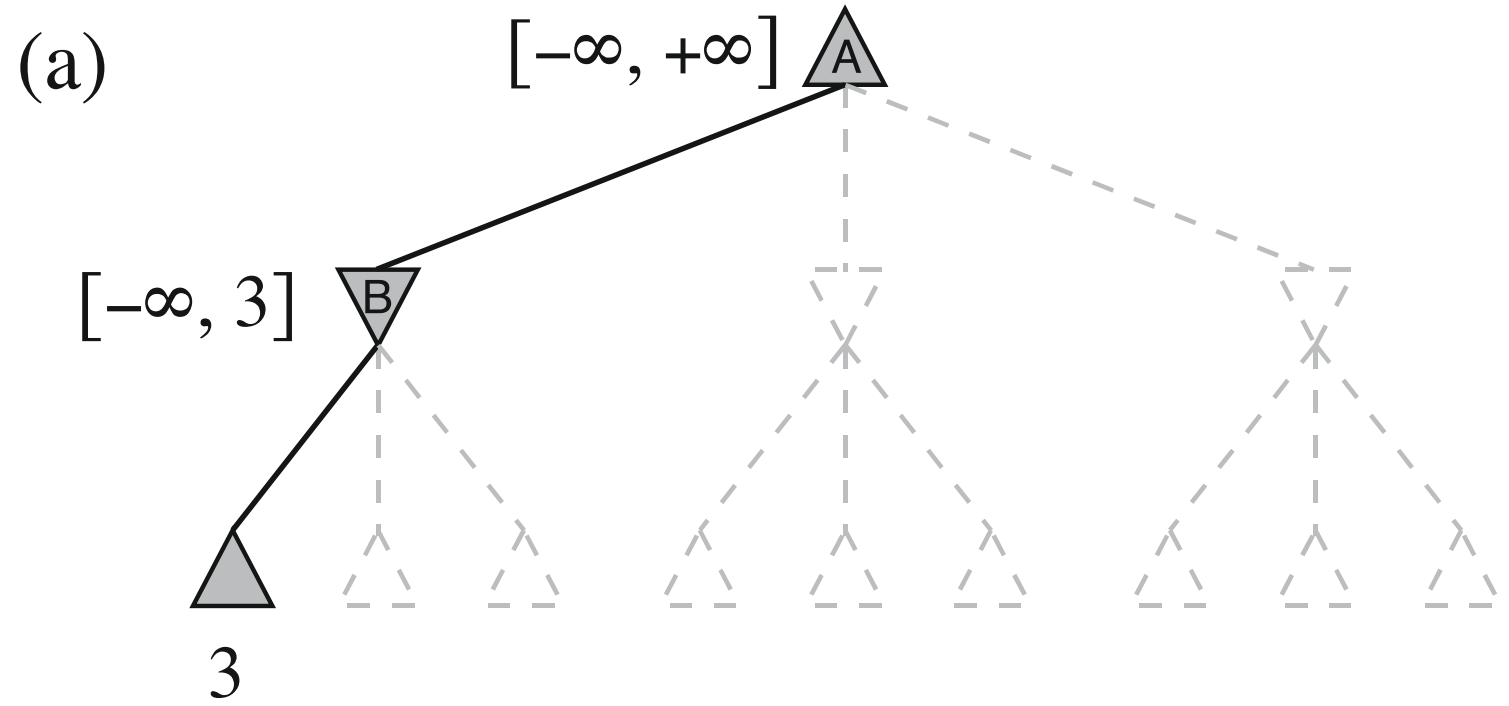
α = حد پایین‌ترین مقدار برای یک گره
β = حد بالاترین مقدار برای یک گره

هرگاه $\alpha \geq \beta$ جستجو در آن شاخه و زیرشاخه‌های آن قطع می‌شود.

هرس آلفا–بتا

مثال (۱ از ۶)

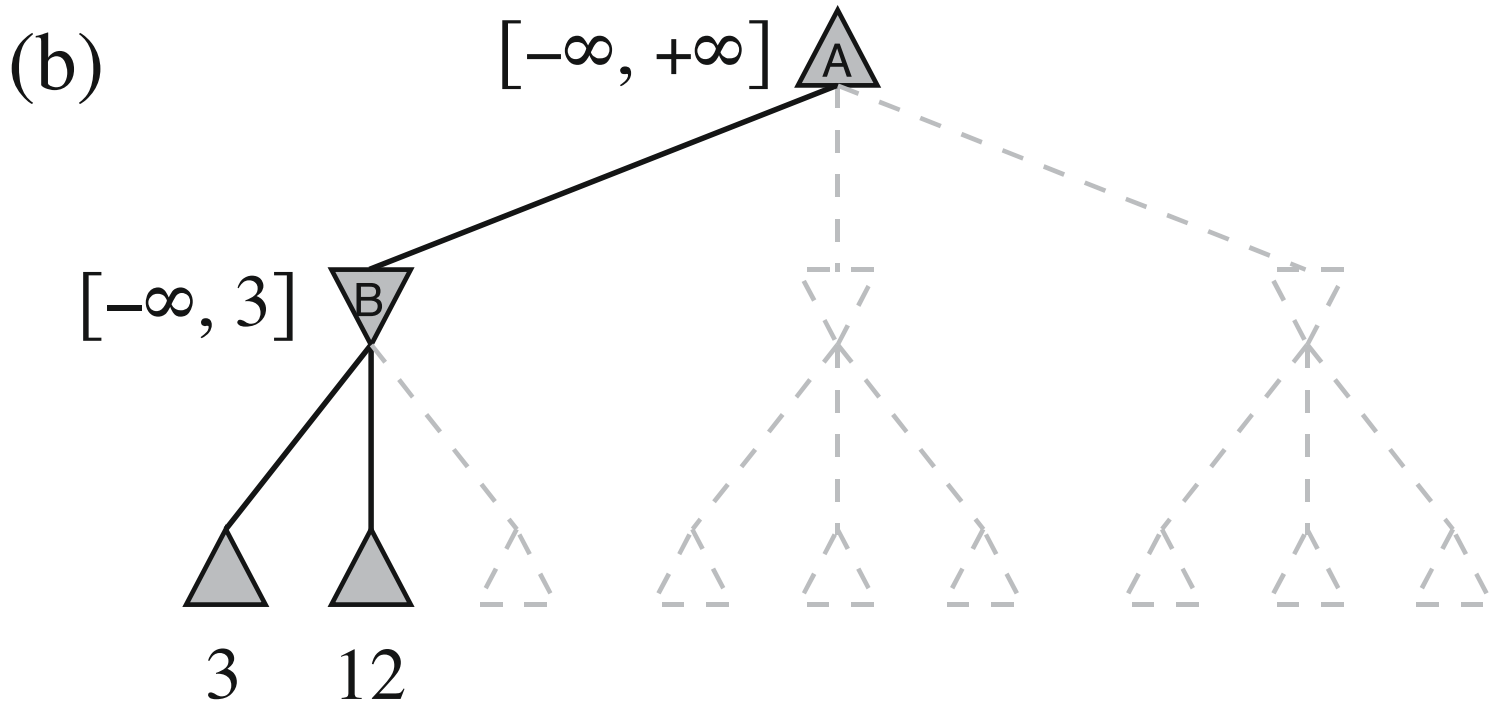
ALPHA-BETA PRUNING



هرس آلفا–بتا

مثال (۲ از ۶)

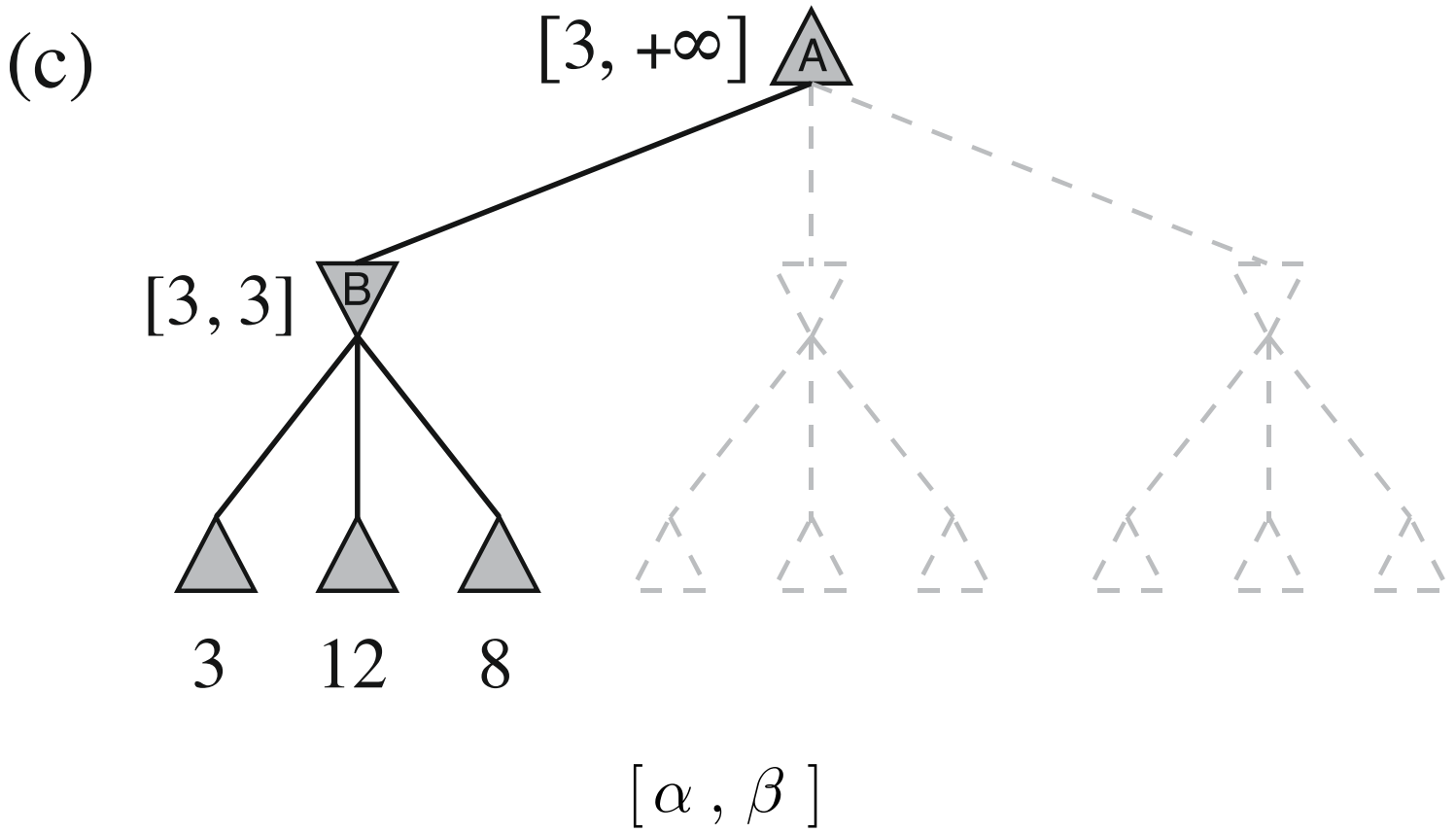
ALPHA-BETA PRUNING



هرس آلفا–بتا

مثال (۳ از ۶)

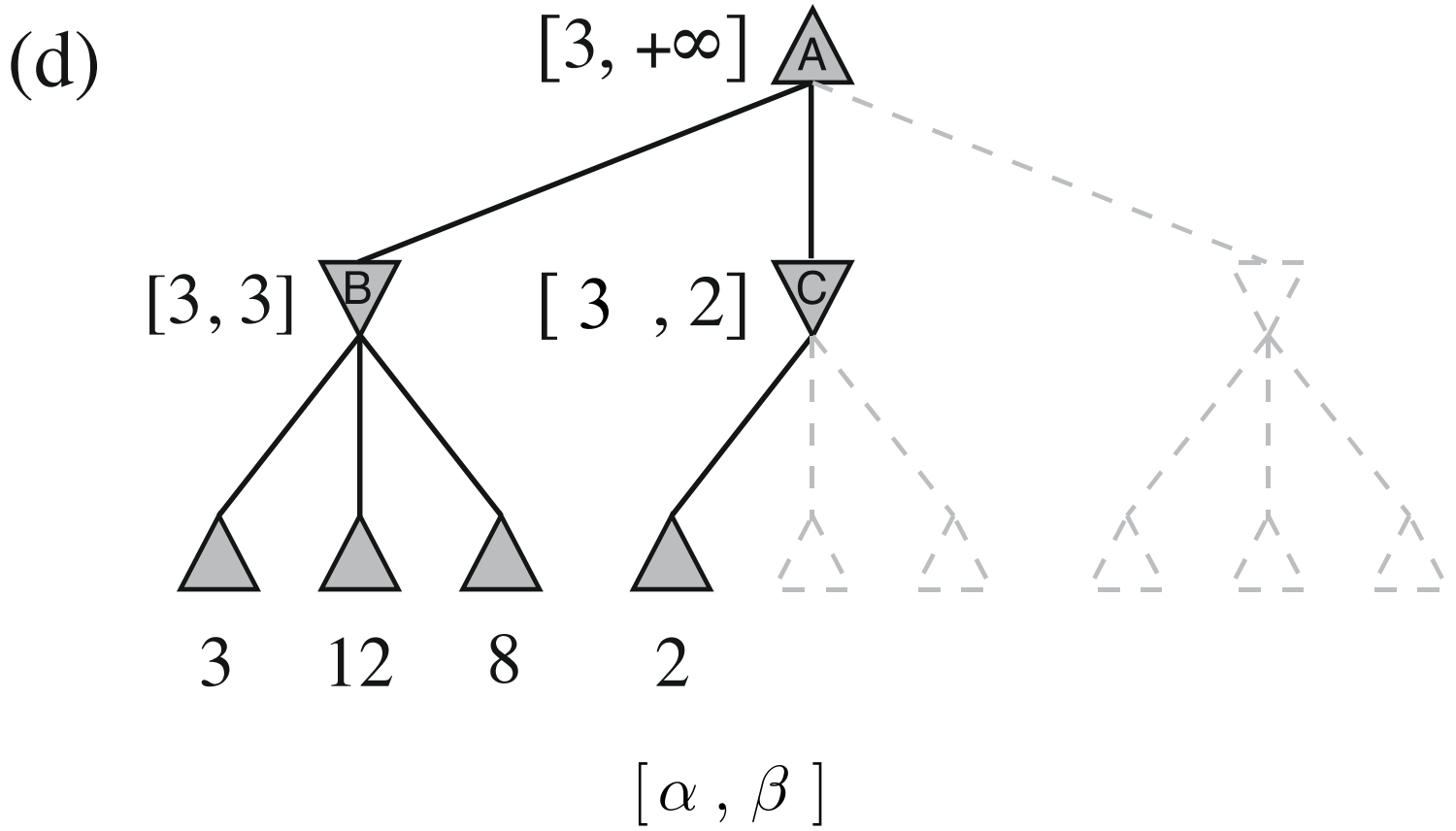
ALPHA-BETA PRUNING



هرس آلفا–بتا

مثال (۴ از ۶)

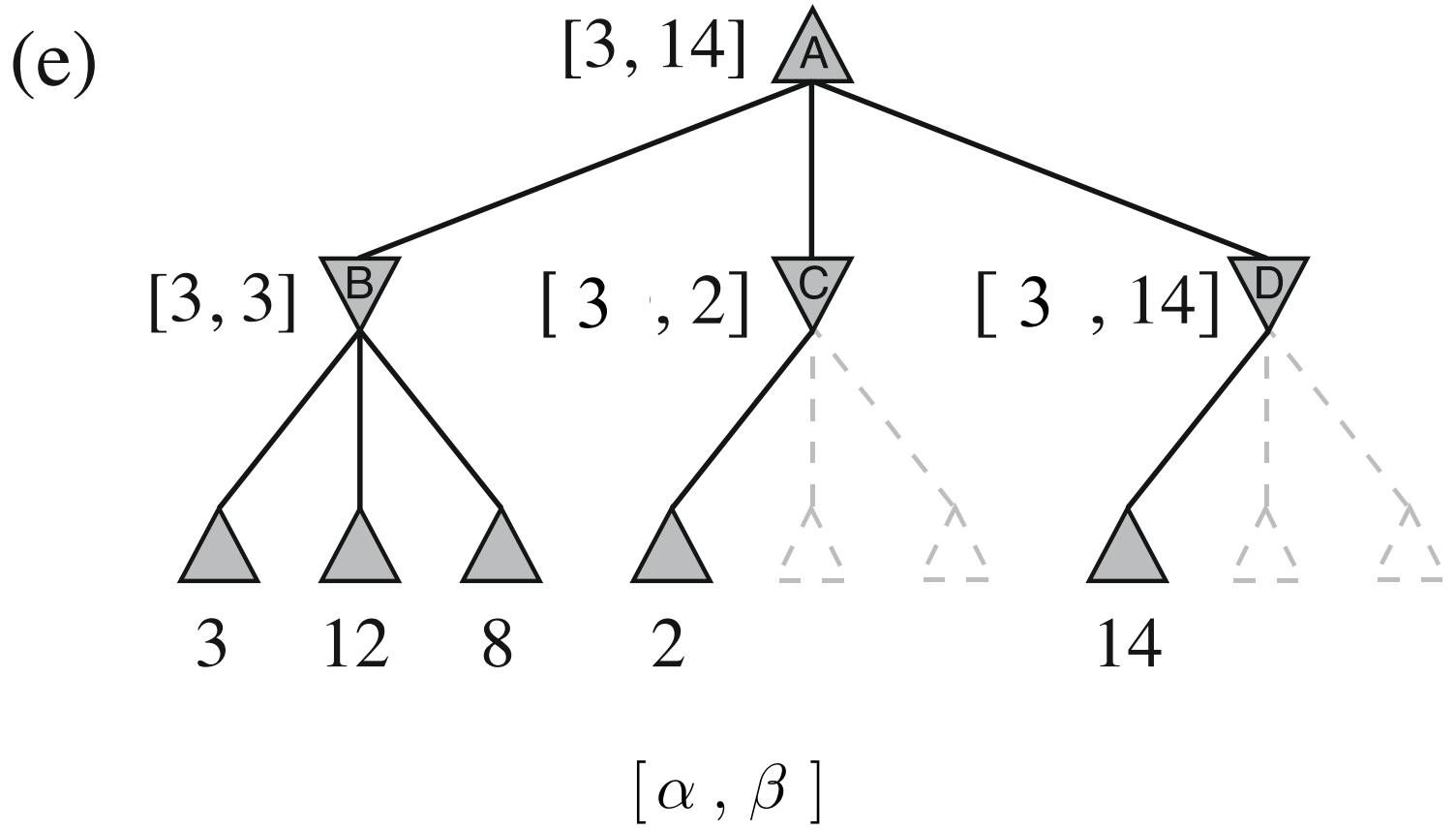
ALPHA-BETA PRUNING



هرس آلفا–بتا

مثال (۵ از ۶)

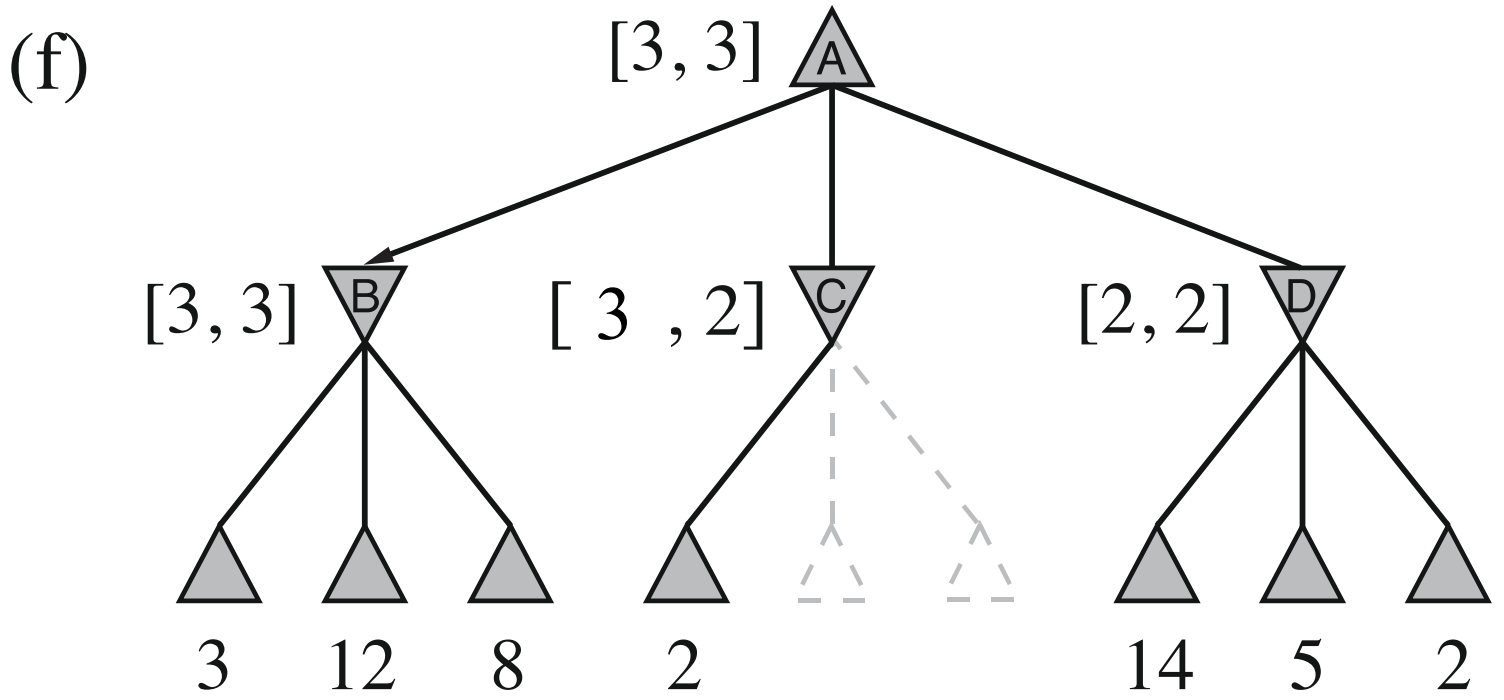
ALPHA-BETA PRUNING



هرس آلفا–بتا

مثال (۶ از ۶)

ALPHA-BETA PRUNING



$$\begin{aligned}
\text{MINIMAX}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3,z,2) \qquad \text{where } z = \min(2,x,y) \le 2 \\
&= 3.
\end{aligned}$$

هرس آلفا–بتا

شبه کد

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, −∞, +∞)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for each a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

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

هرس آلفا–بتا

الگوریتم برای اجرای بصری

۱ درخت بازی را با شروع از ریشه به صورت عمق–اول (از چپ به راست) پیمایش کنید تا به اولین برگ برسید:

برای ریشه: $\alpha \leftarrow -\infty$ $\beta \leftarrow +\infty$

در طول پیمایش:
- برای گره‌ی ▲: $\alpha \leftarrow -\infty$ $\beta \leftarrow$ مقدار پدر
- برای گره‌ی ▼: $\beta \leftarrow +\infty$ $\alpha \leftarrow$ مقدار پدر

در برگ‌ها:
- برای گره‌ی ▲: $\alpha \leftarrow U(n)$ $\beta = \alpha$
- برای گره‌ی ▼: $\beta \leftarrow U(n)$ $\alpha = \beta$

۲ بعد از پایان ارزیابی هر گره n در هر مرحله، مقدار **پدر آن** را بهنگام کنید:

- اگر پدر n، ▲ بود: $\alpha \leftarrow \max(\alpha, \beta(n))$
- اگر پدر n، ▼ بود: $\beta \leftarrow \min(\beta, \alpha(n))$

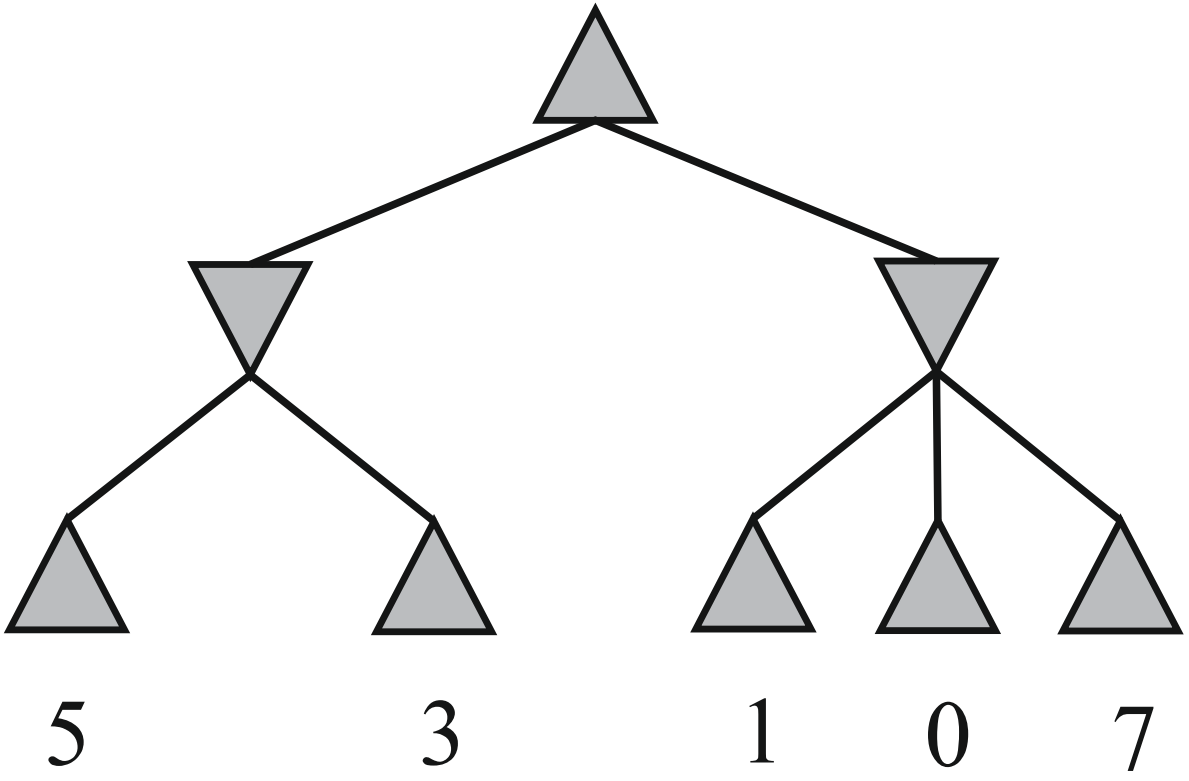
۳ هرگاه $\alpha \geq \beta$ شد، سایر فرزندان آن گره بررسی نمی‌شوند **(هرس می‌شوند)**.

هرس آلفا–بتا

مثال

ALPHA-BETA PRUNING

در درخت بازی زیر، اگر الگوریتم می‌نیماکس را با هرس–آلفا بتا اجرا کنیم، کدام گره‌ها بررسی نمی‌شوند؟

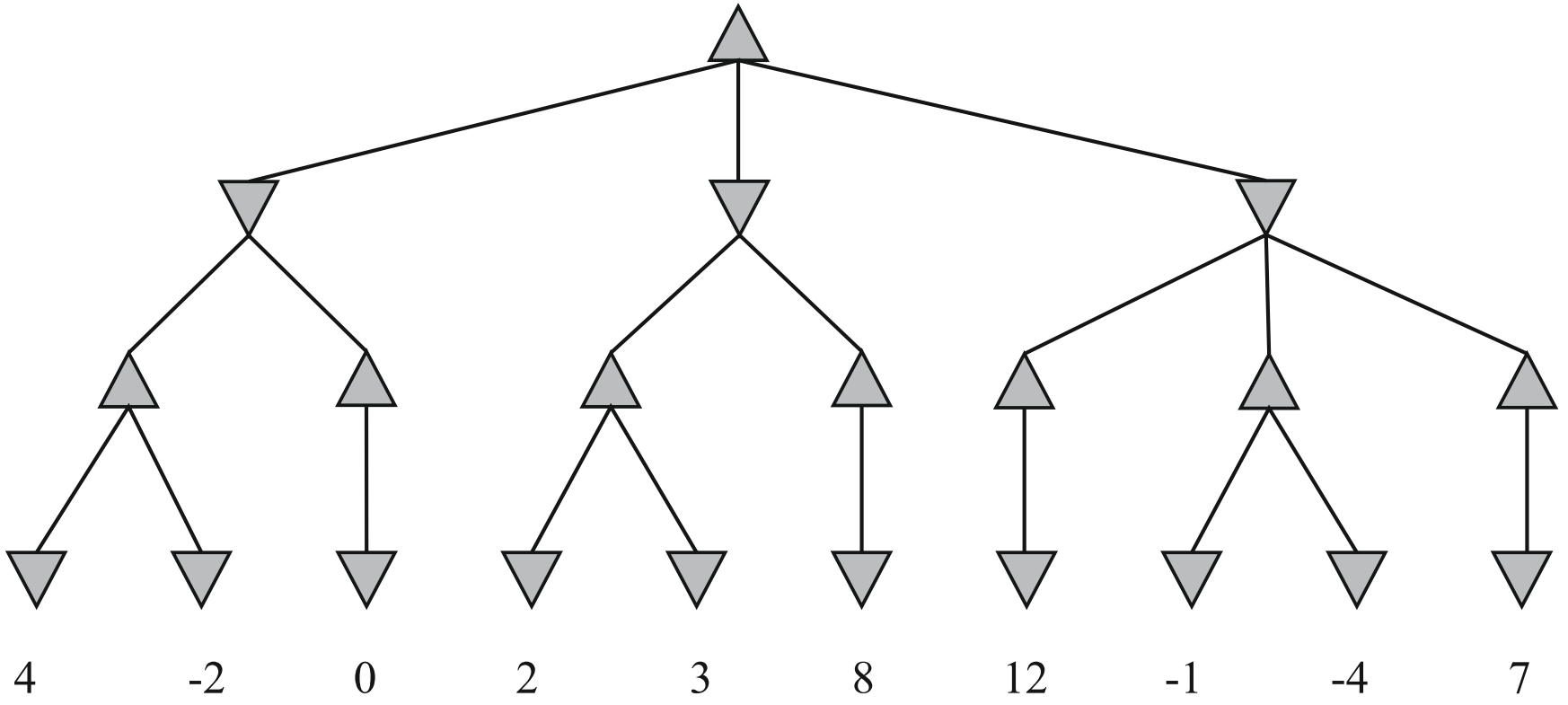


هرس آلفا–بتا

مثال

ALPHA-BETA PRUNING

در درخت بازی زیر، اگر الگوریتم می‌نیماکس را با هرس–آلفا بتا اجرا کنیم، کدام گره‌ها بررسی نمی‌شوند؟

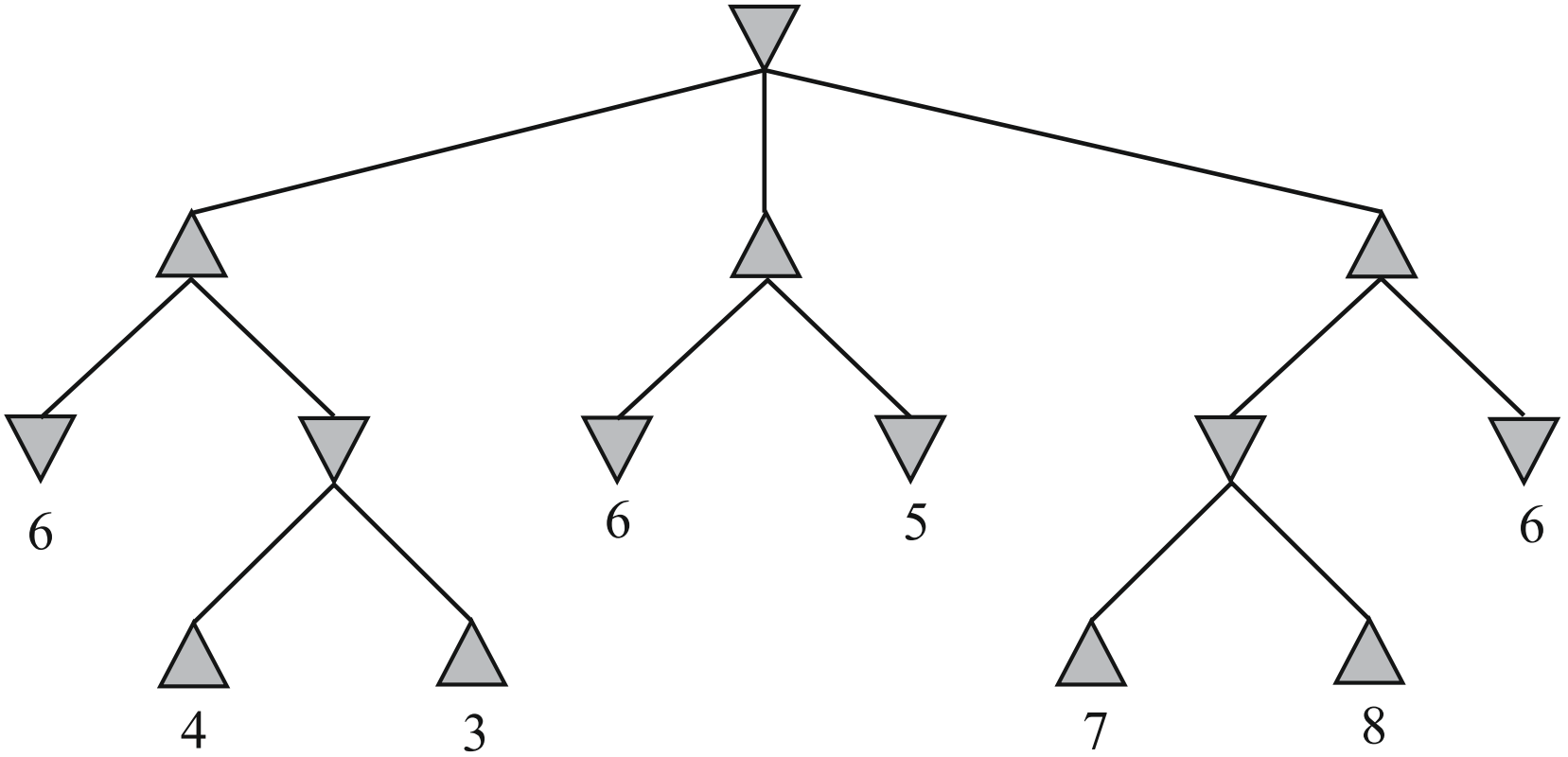


هرس آلفا–بتا

مثال

ALPHA-BETA PRUNING

در درخت بازی زیر، اگر الگوریتم می‌نیماکس را با هرس–آلفا بتا اجرا کنیم، کدام گره‌ها بررسی نمی‌شوند؟

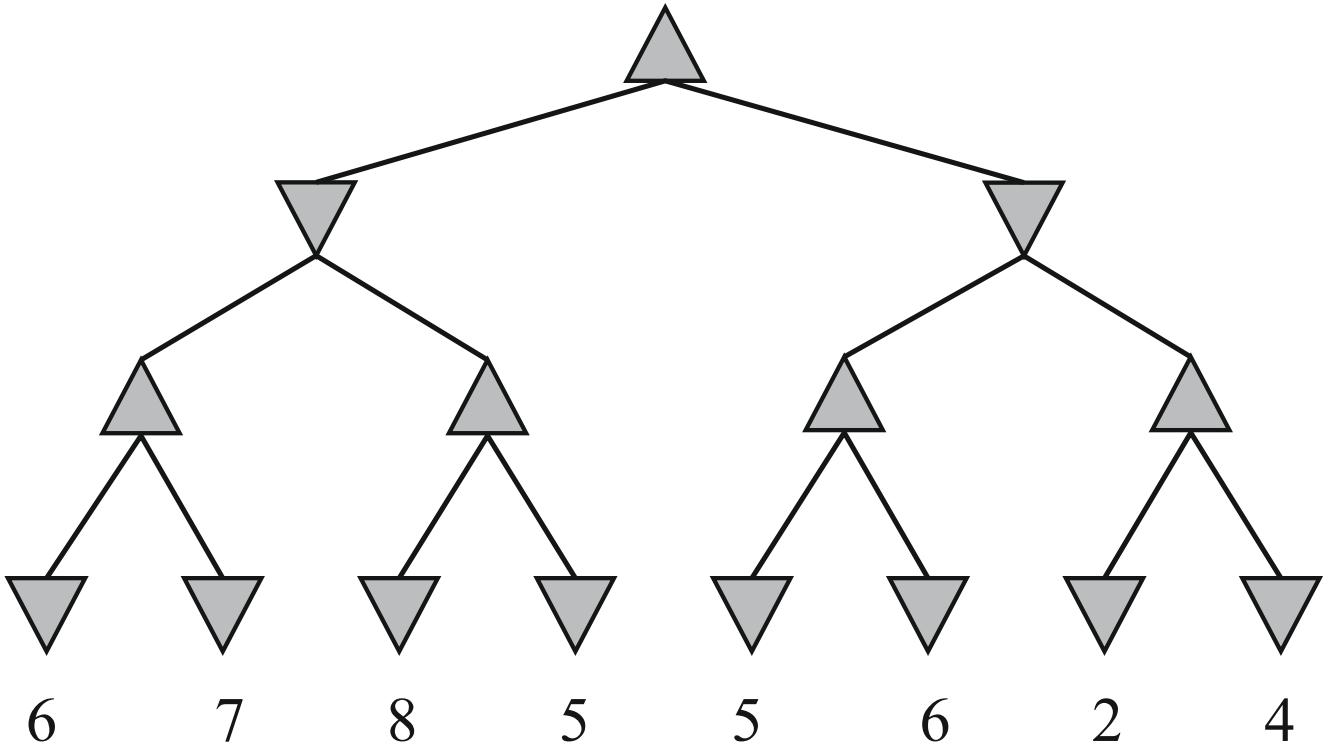


هرس آلفا–بتا

مثال

ALPHA-BETA PRUNING

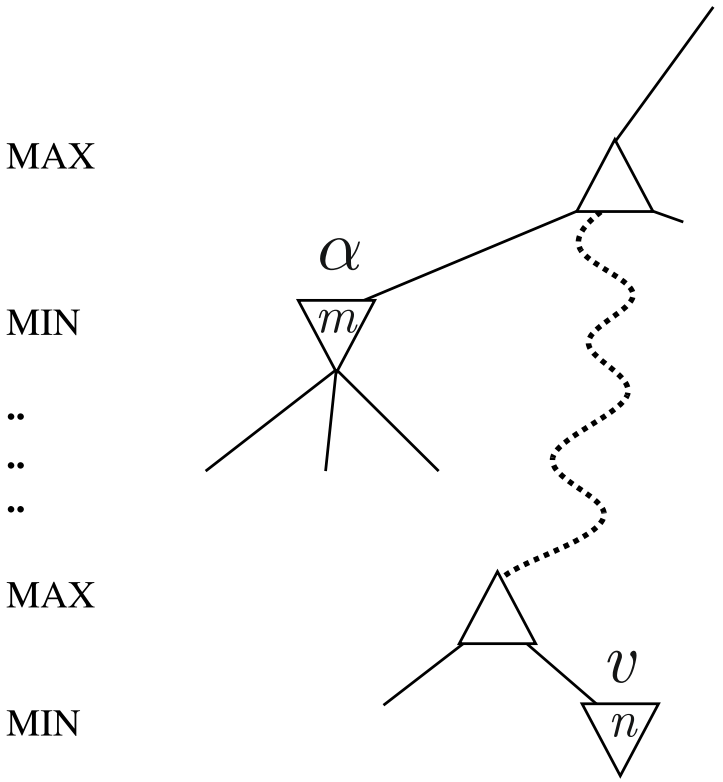
در درخت بازی زیر، اگر الگوریتم می‌نیماکس را با هرس–آلفا بتا اجرا کنیم، کدام گره‌ها بررسی نمی‌شوند؟



هرس آلفا–بتا

منطق روش

ALPHA-BETA PRUNING



α تا کنون بهترین مقدار (برای MAX) است که در طول مسیر جاری یافت شده است.
اگر v بدتر از α باشد، MAX از آن اجتناب می‌کند $\Leftarrow$ آن شاخه هرس می‌شود.
(به طور مشابه β برای MIN)

هرس آلفا–بتا

ویژگی‌ها

ALPHA-BETA PRUNING

هرس کردن بر نتیجه‌ی نهایی تأثیری ندارد.

ترتیب خوب حرکت‌ها (ترتیب گره‌های درخت جستجو)، اثربخشی هرس کردن را بهبود می‌دهد.

با ترتیب‌دهی کامل، پیچیدگی زمانی $= O(b^{m/2})$ $\Leftarrow$ دو برابر شدن عمق راه‌حل

در هر گره به طور متوسط نیمی از فرزندان بررسی نمی‌شود، پس فاکتور انشعاب نصف می‌شود.

هوش مصنوعی

جستجوی رقابتی

۴

تصمیم‌های بی‌درنگ ناکامل

جستجو در درخت بازی با وجود محدودیت منابع

۱ استفاده از تابع آزمون قطع CUTTOFF-TEST به‌جای تابع آزمون پایان TERMINAL-TEST

تابع آزمون پایان
TERMINAL-TEST

→

تابع آزمون قطع
CUTTOFF-TEST

مانند حد عمق برای جستجو
(قطع زودهنگام جستجو)

۲ استفاده از تابع ارزیابی حالت EVAL به‌جای تابع سودمندی UTILITY

تابع سودمندی
UTILITY

→

تابع ارزیابی حالت
EVAL

تخمین میزان مطلوبیت یک حالت
(تابع هیوریستیک)

$$\text{H-MINIMAX}(s,d) = \begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s,d) \\ \max_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{H-MINIMAX}(\text{RESULT}(s,a), d+1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

تابع ارزیابی

شرایط لازم

Evaluation Function (Eval)

باید مقدار آن در حالت‌های پایانی با مقدار تابع سودمندی هماهنگ باشد.

باید محاسبه‌ی آن ساده و سریع باشد.

باید شانس برنده شدن را به خوبی نشان دهد.

می‌توان تابع ارزیابی را به صورت ترکیب خطی تعدادی ویژگی تعریف کرد:

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

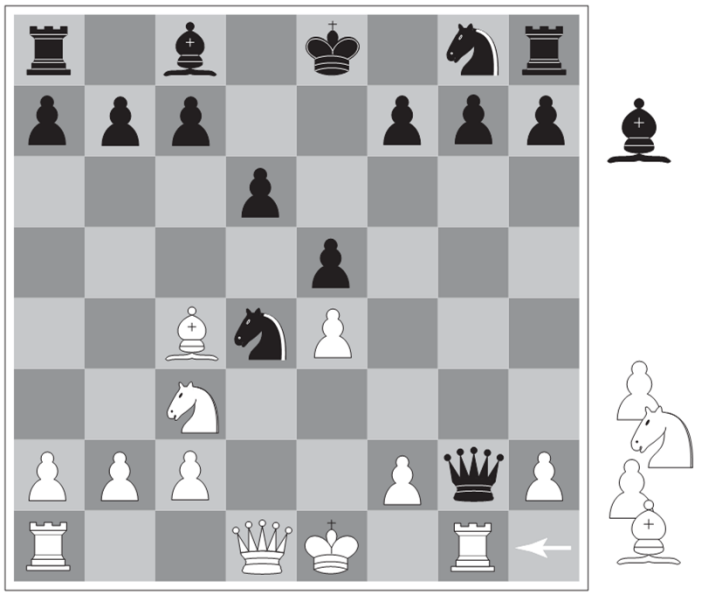
تعیین وزن‌ها را می‌توان با یادگیری ماشین انجام داد.

تابع ارزیابی

مثال: بازی شطرنج

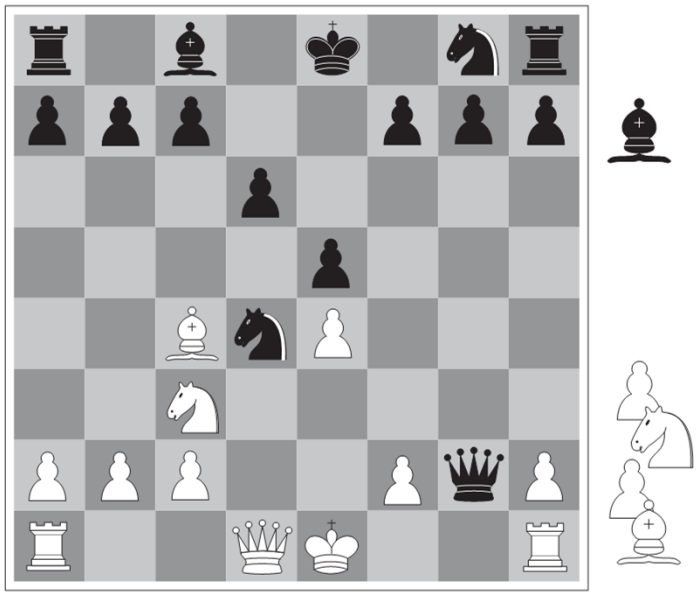
EVALUATION FUNCTION (EVAL)

سیاه با جلو بردن یک اسب و دو سرباز بازی را می‌برد.



(b) White to move

سیاه با از دست دادن وزیر بازی را می‌بازد.



(a) White to move

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s) = \sum_{i=1}^{n} w_i f_i(s)$$

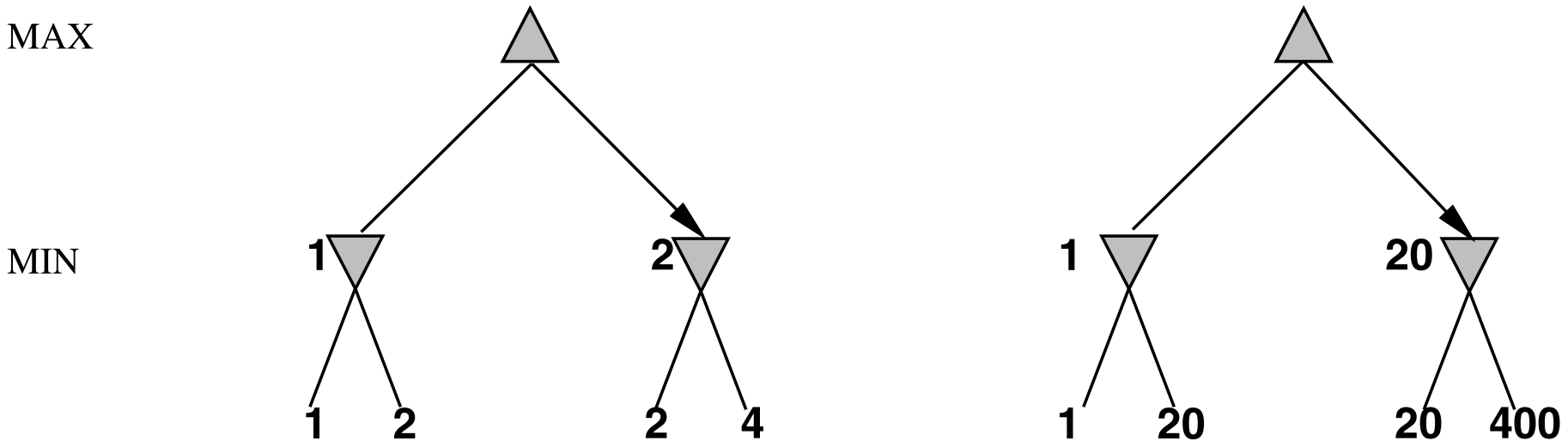
$$f_1(s) = (\text{number of white queens}) - (\text{number of black queens}) \qquad , w_1 = 9$$

وزن بر اساس ارزش هر مهره

تابع سودمندی

تابع سودمندی/ارزیابی ترتیبی

تابع سودمندی/ارزیابی در بازی‌های قطعی، به صورت یک تابع ترتیبی عمل می‌کند:
(Oridnal Utility Function)



رفتار بازی تحت هر تبدیل یکنوای صعودی تابع سودمندی/ارزیابی ثابت می‌ماند.

تابع آزمون قطع

قطع جستجو

CUT-OFF TEST FUNCTION: CUTTING-OFF SEARCH

حد عمقی: جستجو تا عمق d انجام شود.

حد زمانی: جستجو تا زمان $time\text{-}out$ انجام شود.

هوش مصنوعی

جستجوی رقابتی

۵

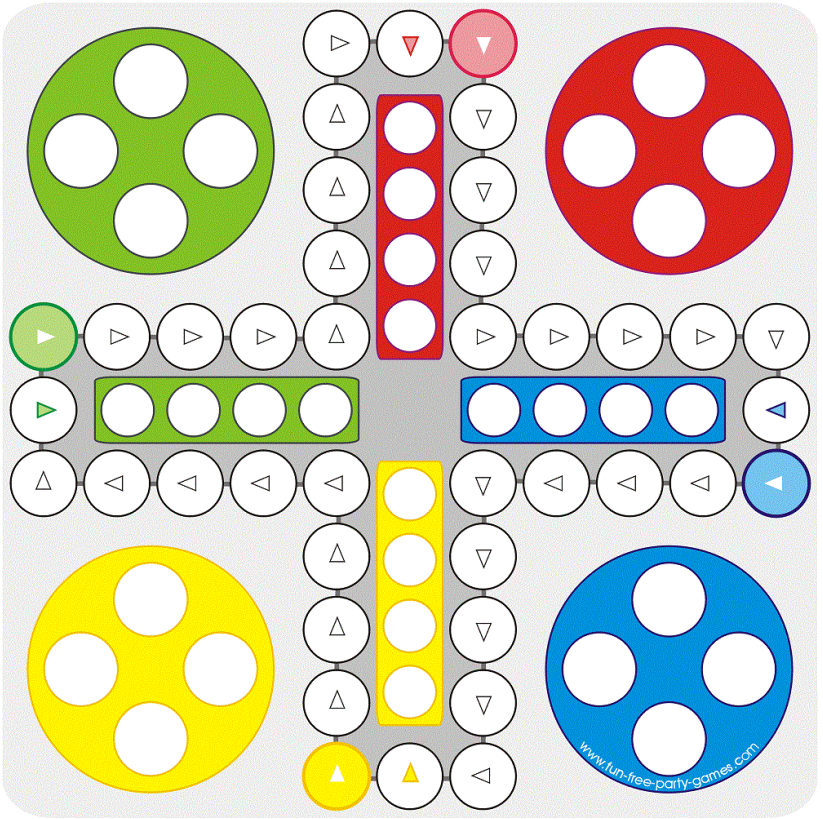
بازی‌های اتفاقی

بازی‌های اتفاقی

بازی‌هایی که در آنها عامل شانس نقش دارد

STOCHASTIC GAMES

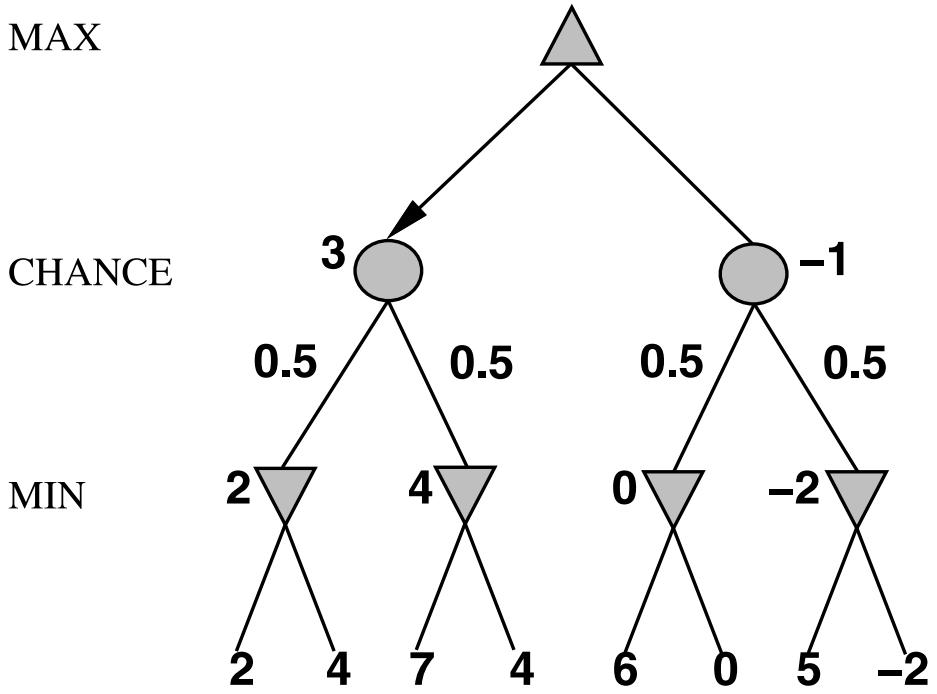
در بازی‌های اتفاقی، موردی مانند **پرتاب تاس**، حرکت مجاز را مشخص می‌کند.



بازی‌های اتفاقی

عامل شانس

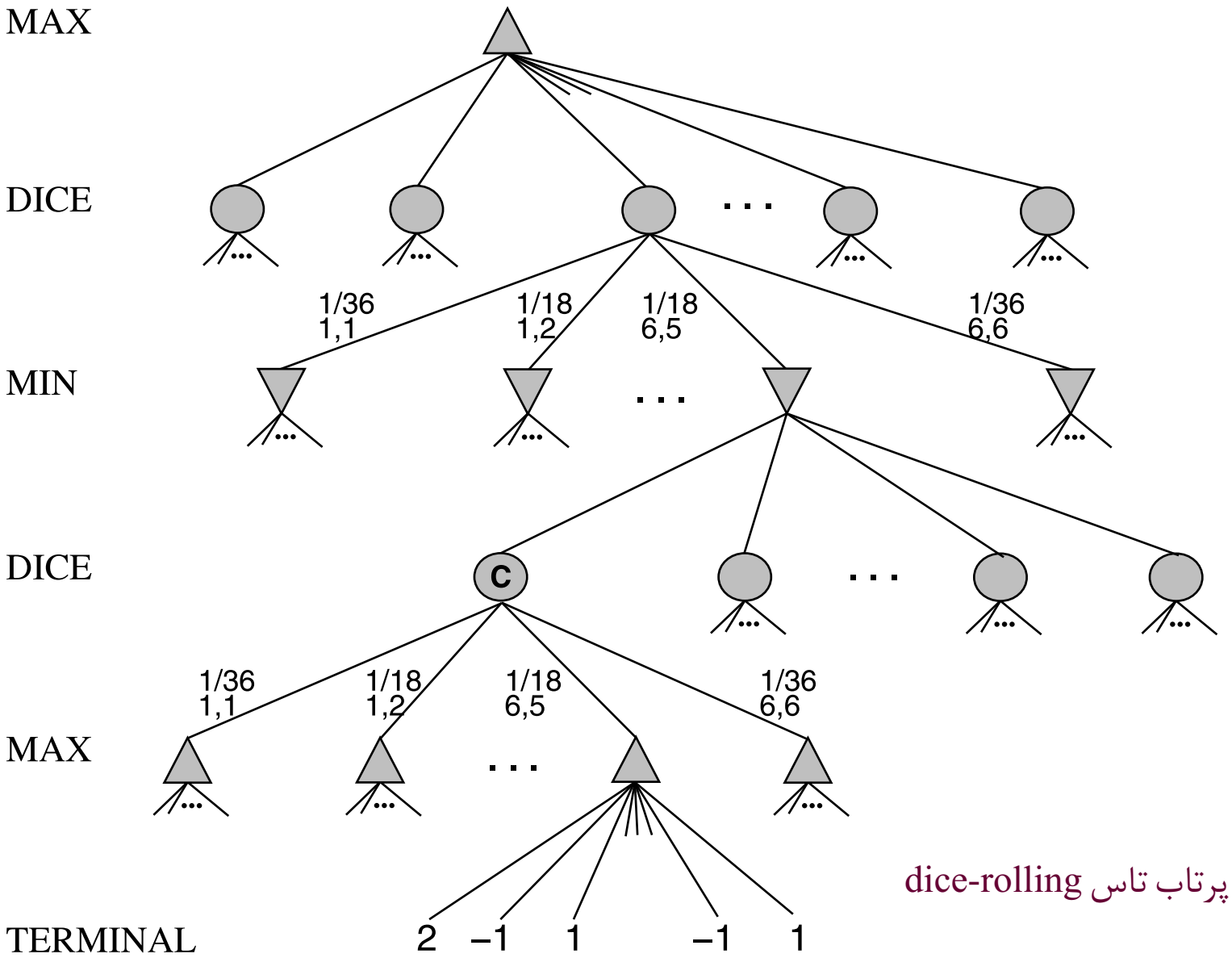
عامل شانس مانند یک بازیکن وارد می‌شود:
هر برآمد شانس، مانند یک کنش با یک احتمال مشخص است.
گره شانس در درخت بازی با دایره نشان داده می‌شود و حامل یک توزیع احتمال است.



مثال: پرتاب سکه coin-flipping

بازی‌های اتفاقی

مثال



مثال: پرتاب تاس dice-rolling

الگوریتم «امید می‌نیماکس» برای بازی‌های اتفاقی

در بازی‌های اتفاقی، هر گره به‌جای تابع ارزیابی، دارای مقدار **امید می‌نیماکس** است.

مشابه الگوریتم می‌نیماکس
فقط، در گره‌های شانس، ارزش فرزندان میانگین‌گیری می‌شود.

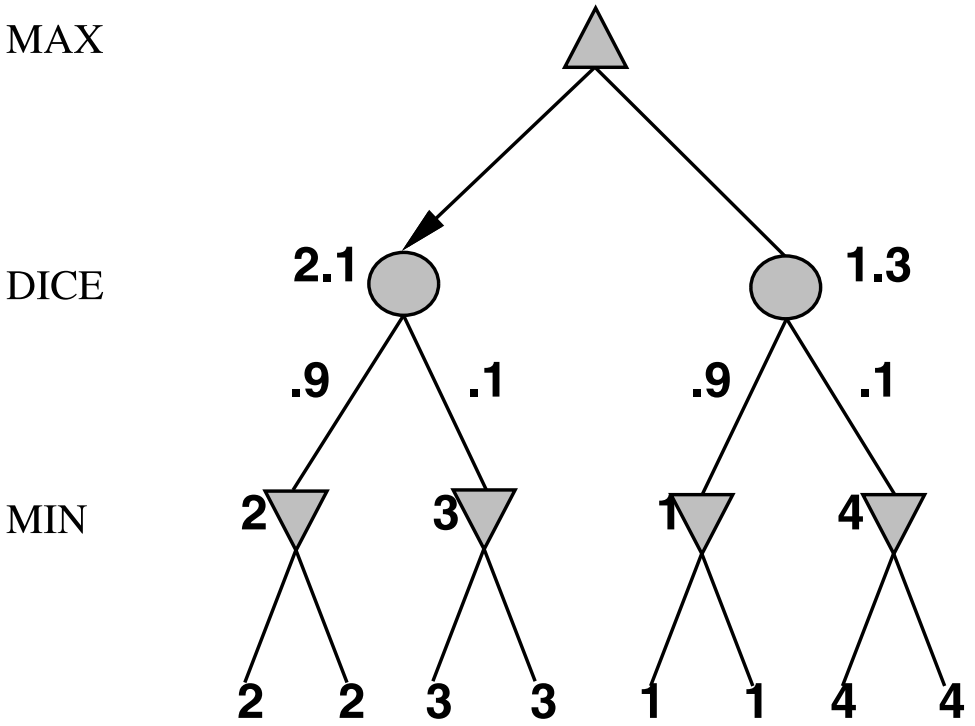
$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

پیچیدگی زمانی: $O(c^m b^m)$

c تعداد برآمدهای شانس

الگوریتم «امید می‌نیماکس» برای بازی‌های اتفاقی

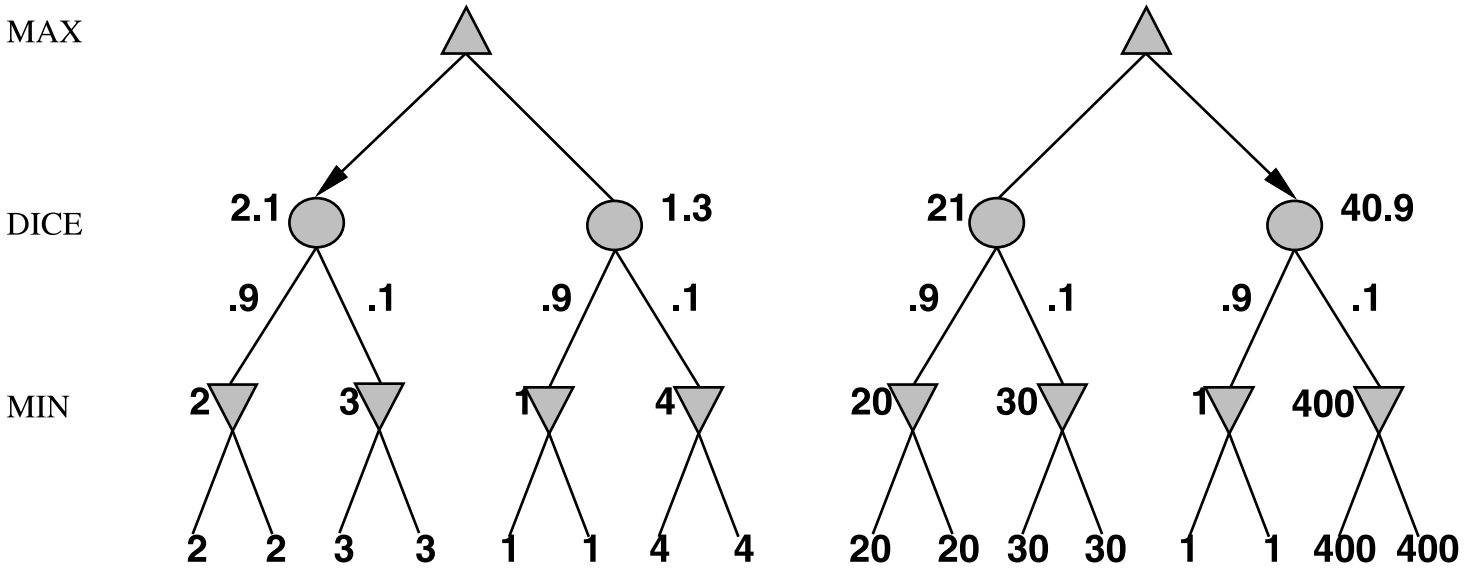
مثال



تابع سودمندی

تابع سودمندی/ارزیابی ترتیبی

مقادیر تابع سودمندی/ارزیابی در بازی‌های اتفاقی، اهمیت دارد:



رفتار بازی تنها تحت تبدیل خطی مثبت تابع سودمندی/ارزیابی ثابت می‌ماند.

هوش مصنوعی

جستجوی رقابتی

۶

بازی‌های مشاهده‌پذیر جزئی

بازی‌های مشاهده‌پذیر جزئی

بازی‌هایی با اطلاعات ناکامل

PARTIALLY OBSERVABLE GAMES

در بازی‌های مشاهده‌پذیر جزئی
وضع رقیب نامشخص است و پس از برخورد مشخص می‌شود.
⇐ لزوم گردآوری اطلاعات، جاسوسی، اختفا و بلوف برای گیج کردن رقیب

راه‌حل: مقدار می‌نیماکس هر کنش را در هر حالت محاسبه کنید،
سپس کنشی را انتخاب کنید که دارای بزرگ‌ترین مقدار متوسط بین همه‌ی حالت‌هاست:

$$\underset{a}{\operatorname{argmax}} \sum_{s} P(s)\, \textsc{Minimax}(\textsc{Result}(s,a))$$

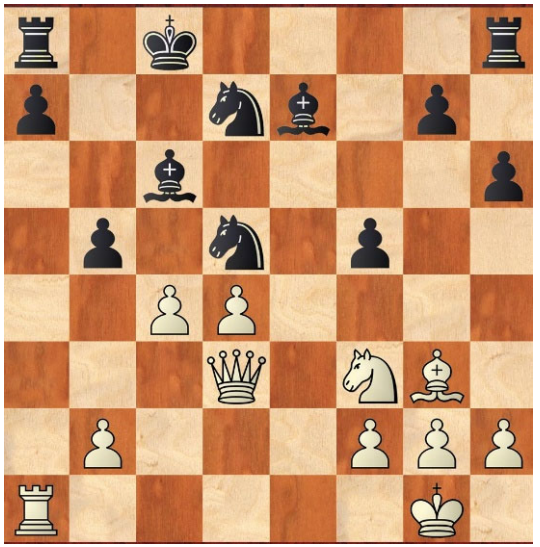
هوش مصنوعی

جستجوی رقابتی

۷

مرزهای دانش برنامه‌های بازی

بازی شطرنج



$$b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4$$

4-ply $\approx$ انسان تازه‌کار

8-ply $\approx$ کامپیوتر شخصی نوعی، انسان وارد

12-ply $\approx$ DeepBlue ،Kasparov

Kasparov, Garry: Kasparov playing against Deep Blue, 1997

Garry Kasparov playing against Deep Blue, the chess-playing computer built by IBM.

“

Rybka, winner of the 2008 and 2009 World Computer Chess Championships, is considered the strongest current computer player. It uses an off-the-shelf 8-core 3.2 GHz Intel Xeon processor, but little is known about the design of the program. Rybka's main advantage appears to be its evaluation function, which has been tuned by its main developer, International Master Vasik Rajlich, and at least three other grandmasters.

The most recent matches suggest that the top computer chess programs have pulled ahead of all human contenders.

”

AIMA3e, pg. 186

هوش مصنوعی

جستجوی رقابتی

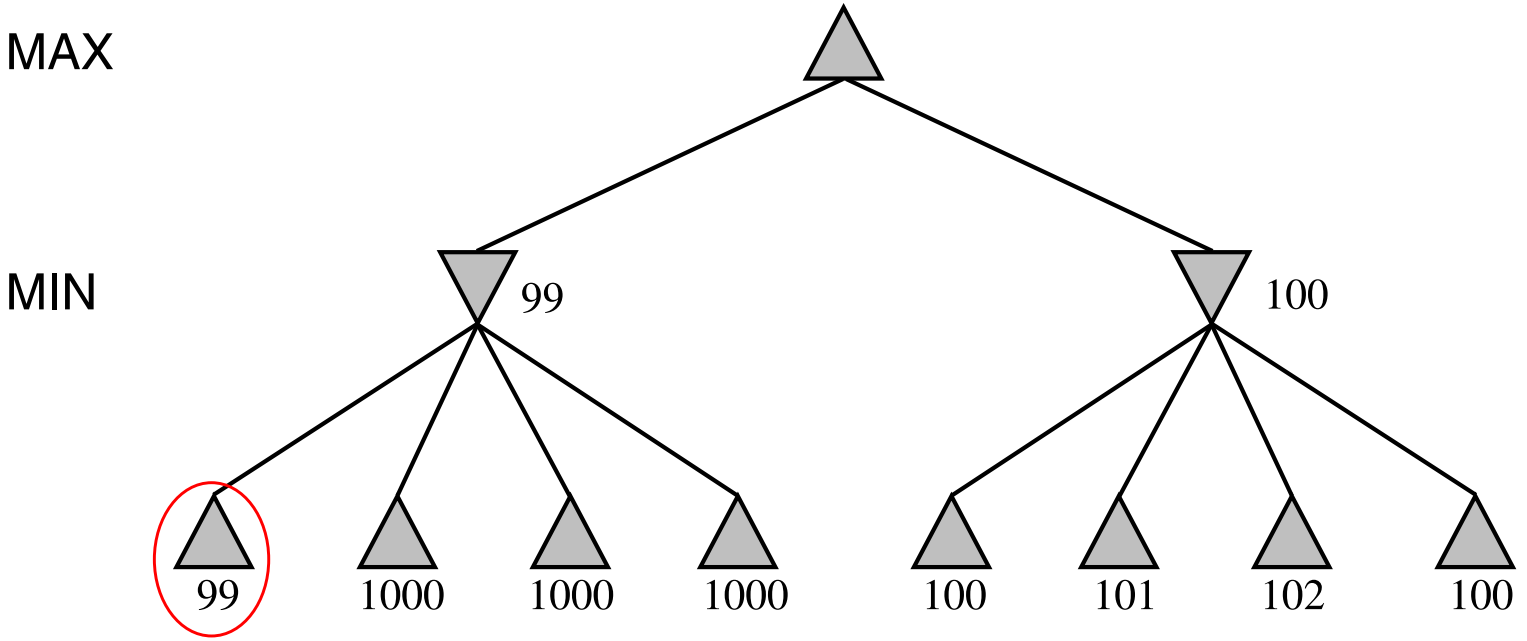
۸

روی‌کردهای جایگزین

رویکردهای جایگزین

مواردی که الگوریتم می‌نیماکس مناسب نیست

ALTERNATIVE APPROACHES



می‌نیماکس: انتخاب حرکت بهینه به شرط درست بودن ارزیابی‌ها در گره‌ی برگ
(در عمل ارزیابی‌ها تخمین خام از ارزش وضعیت‌ها و دارای خطای زیاد هستند)

روی‌کردهای جایگزین

ALTERNATIVE APPROACHES

فرا استدلال *Meta-reasoning*	استدلالی برای انتخاب مناسب‌ترین نوع محاسبات (استدلال در مورد استدلال) مانند: جستجوی آلفا–بتا
استدلال هدایت‌شده با هدف *goal-directed reasoning*	در نظر گرفتن یک هدف ویژه و تولید و انتخاب طرح‌های ممکن به کمک این هدف

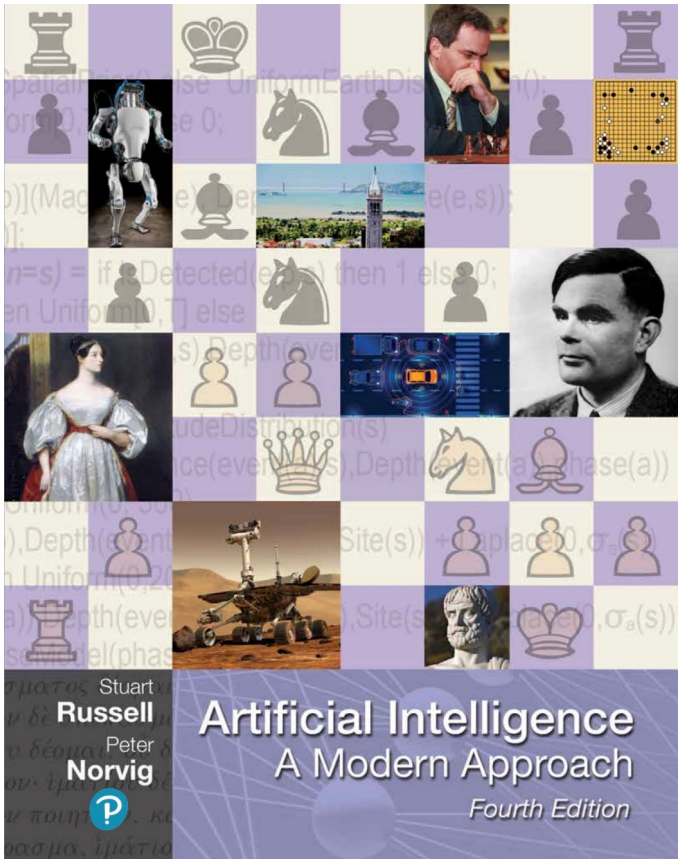
هوش مصنوعی

جستجوی رقابتی

۹

منابع

منبع اصلی



Stuart Russell and Peter Norvig,
Artificial Intelligence: A Modern Approach,
4th Edition, Prentice Hall, 2020.

Chapter 5

CHAPTER 5

ADVERSARIAL SEARCH AND GAMES

In which we explore environments where other agents are plotting against us.

Adversarial search

In this chapter we cover **competitive environments**, in which two or more agents have conflicting goals, giving rise to **adversarial search** problems. Rather than deal with the chaos of real-world skirmishes, we will concentrate on games, such as chess, Go, and poker. For AI researchers, the simplified nature of these games is a plus: the state of a game is easy to represent, and agents are usually restricted to a small number of actions whose effects are defined by precise rules. Physical games, such as croquet and ice hockey, have more complicated descriptions, a larger range of possible actions, and rather imprecise rules defining the legality of actions. With the exception of robot soccer, these physical games have not attracted much interest in the AI community.

5.1 Game Theory

Economy

There are at least three stances we can take towards multi-agent environments. The first stance, appropriate when there are a very large number of agents, is to consider them in the aggregate as an **economy**, allowing us to do things like predict that increasing demand will cause prices to rise, without having to predict the action of any individual agent.

Second, we could consider adversarial agents as just a part of the environment—a part that makes the environment nondeterministic. But if we model the adversaries in the same way that, say, rain sometimes falls and sometimes doesn't, we miss the idea that our adversaries are actively trying to defeat us, whereas the rain supposedly has no such intention.

Pruning

The third stance is to explicitly model the adversarial agents with the techniques of adversarial game-tree search. That is what this chapter covers. We begin with a restricted class of games and define the optimal move and an algorithm for finding it: minimax search, a generalization of AND–OR search (from Figure 4.11). We show that **pruning** makes the search more efficient by ignoring portions of the search tree that make no difference to the optimal move. For nontrivial games, we will usually not have enough time to be sure of finding the optimal move (even with pruning); we will have to cut off the search at some point.

For each state where we choose to stop searching, we ask who is winning. To answer this question we have a choice: we can apply a heuristic **evaluation function** to estimate who is winning based on features of the state (Section 5.3), or we can average the outcomes of many fast simulations of the game from that state all the way to the end (Section 5.4).

Imperfect information

Section 5.5 discusses games that include an element of chance (through rolling dice or shuffling cards) and Section 5.6 covers games of **imperfect information** (such as poker and bridge, where not all cards are visible to all players).