دانشکده مهندسی کامپیوتر

جزوه درس

هوش مصنوعی

استاد درس: سید صالح اعتمادی

نیم‌سال اول

سال تحصیلی ۰۰-۹۹

جلسه ۹

Game Trees

محمد امین قسوری - ۱۳۹۹/۷/۳۰

جزوه جلسه ۹ام مورخ ۱۳۹۹/۷/۳۰ درس هوش مصنوعی تهیه شده توسط محمد امین قسوری. در جهت مستند کردن مطالب درس هوش مصنوعی

۱.۹ مقدمه

در این قسمت می خواهیم درمورد الگوریتم هایی صحبت کنیم که در بازی هایی که Single-agent نیستند و یک رقیب در مقابل خود داریم که گاها هوشمند و در بعضی مواقع نیز ممکن است اشتباه کند.

۲.۹ تاریخچه ی بازی ها

بعضی بازی ها مانند checkers حل شده (solved) هستند وحل شده به این معنی است که اگر هر دو بازیکن به صورت بهینه بازی کنند شما می توانید یک برد یا حداقل مساوی را بصورت قطعی بدست بیاورید (نبازید). از طرفی در مورد بعضی بازی ها مانند شطرنج نمی توان گفت که حل شده هستند.

۳.۹ دسته بندی بازی ها

تا به حال بازی های زیادی انجام داده ایم. در این جا میخواهیم آنها را بر اساس ویژگی هایی دسته بندی نماییم.

۱.۳.۹ Stochastic or Deterministic

به بازی هایی که شانس در آنها معیار نباشد، deterministic می گویند مانند GO یا شطرنج و به بازی هایی که با شانس همراه باشند Stochastic گفته میشود مانند تخته نرد که دو تاس درآن داریم و حرکات و تصمیمات بعدی ما به آن وابسته است.

۲.۳.۹ تعداد بازیکن ها

تعداد بازیکن ها نیز معیاری برای دسته بندی بازی ها هستند که بعضی مانند شطرنج دونفره و بعضی دیگر میتوانند چندین نفره باشند مانند خیلی از Board Game ها.

۳.۳.۹ Zeo Sum

این ویژگی نشان میدهد که آیا بازیکن ها کاملا بر خلاف و مقابل هم بازی میکنند یا خیر. برای مثال اگر هر دو یک هدف مانند برداشتن یک الماس داشته باشند خوب قطعا بر خلاف هم هستند و الماس را یکی از آن ها بدست می آورد و دیگری از دست میدهد.

شکل ۱.۹: Zero sum

۴.۳.۹ Perfect Information

یعنی درمورد بازی تمام اطلاعات و مواردی که برای تصمیم گیری لازم داریم در اختیار داریم.

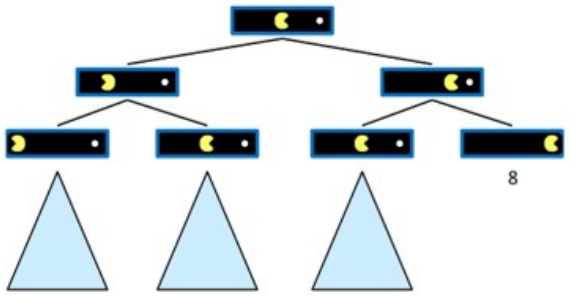
در قسمت قبل از regular planning استفاده می کردیم منتهی در این جا پارامتری داریم که بر خلاف ما عمل میکند و ما نمی توانیم تمام تصمیمات خود را برنامه ریزی کنیم پس در ادامه از تعدادی الگوریتم برای محاسبه استراتژی بازی کردن در مقابل حریف صحبت خواهیم کرد.

۴.۹ Deterministic Games

ما میتوانیم بازی های deterministic به صورت زیر نمایش بدهیم.

- States هر وضعیت بازی یک حالت است مانند حالت اولیه.
- Players همان بازیکن ها هستند.
- Actions تصمیماتی که میتواینم در یک حالت بگیریم که میتواند وابسته به حالت یا بازیکن باشد.
- Transition Function خیلی مشابه successor function در regular search است.
- Terminal Test تست میکند که آیا بازی تمام شده است یا خیر.
- Terminal Utilities همه ی نتیجه ها امتیازدهی میشوند و میتواند بجای سه مقدار بردن، باختن یا مساوی بازه ای از اعداد برای امتیازدهی داشته باشند.

۵.۹ Single Agent Tree

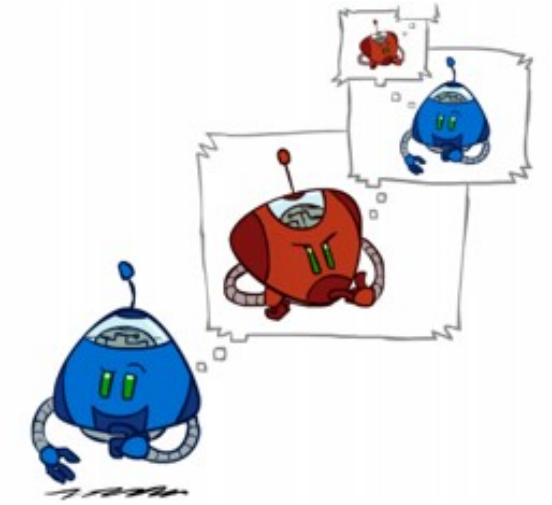


شکل ۲.۹: Single Agent Tree

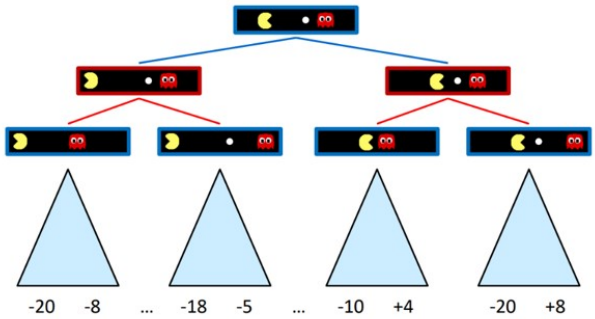
برای pacman زمانی که یک agent داشته باشیم این درخت خیلی ساده است. ما مقداری از آن درخت را میکشیم و با توجه به این که با گذشت زمان امتیاز منفی خواهیم گرفت همه ی موارد سمت چپ کمتر از ۸ خواهند شد و ما برای هر گره امتیازی در نظر میگیریم که در حقیقت ماکسیمم امتیازهای بچه هایش است.

۶.۹ Adversarial Search

در adversarial search بدلیل داشتن رقیب در بازی باید به همه ی حالات فکر کنیم و تا چندین سطح به این فکر کنید که اگر چه حرکتی را بروید در نهایت امتیاز به نفع شما خواهد بود.



شکل ۳.۹: Adversarial Search

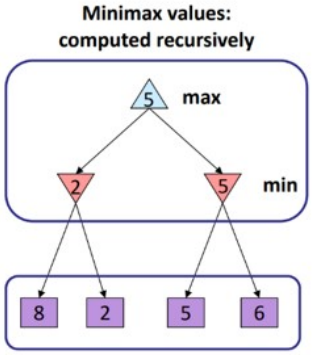


شکل ۴.۹: Adversarial Search

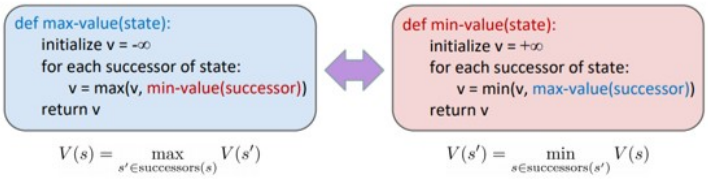
این درخت با درخت Single Agent متفاوت است و نمیتوانیم مانند قبل بیشترین امتیاز هر فرزند را به گره بالای آن بدهیم چون رقیب ما میتواند راهی را انتخاب کند که ماکسیمم نباشد بلکه مینیمم باشد و سعی کند کمترین امتیاز را به شما بدهد.

۱.۶.۹ Minimax

یکی از الگوریتم ها استفاده از minimax است که در حقیقت وقتی نوبت حریف است باید مینیمم امتیاز را برای گره قرار دهید ولی اگر نوبت شما باشد باید از فرصت استفاده کنید و بیشتری مقداری که میتوانید را برداشت کنید.

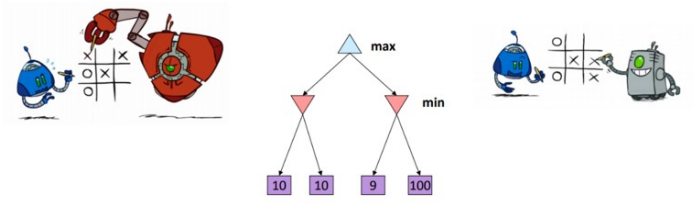


شکل ۵.۹: Minimax



شکل ۶.۹: Minimax Code

همانطوری که مشاهده میکنید ما basecase ای نداریم و برای قابل پیاده سازی شدن باید یک حالت basecase برای این الگوریتم بازگشتیمان در نظر بگیریم که اکثرا بدلیل نمایی بودن رشد درخت قادر به جستوجوی کامل آن نیستیم و این کار امکان پذیر نیست که در آنجا به عنوان مثال می گوییم که تا ۱۰۰۰۰۰۰۰ سطح پایین را بگردد و بعد دیگر ادامه ندهد.(Depth-limited)

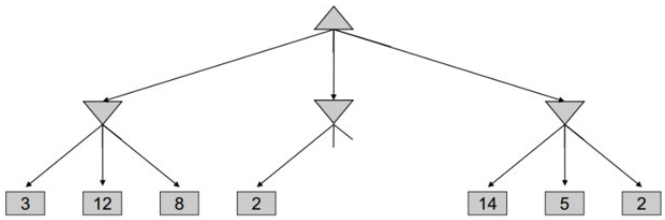


شکل ۷.۹: Minimax

اگر در مقابل یک حریف سرسخت و قوی بازی کنیم minimax نتیجه ی خوبی از بازی دریافت میکند ولی اگر رقیب ما ضعیف باشد agent به دلیل ترس از از دست دادن امتیاز ریسک نمی کند و این باعث میشود که از اشتباهات حریفش به خوبی استفاده نکند که دقیقا نقطه منفی این الگوریتم است. کارآمدی این الگوریتم

مانند dfs است. پیچیدگی زمانی این الگوریتم O(bn) است که b تعداد حالت یا branch factor است و n هم عمق ما است. در مورد حافظه وضعیت بر خلاف زمانی بهتر است و از O(bn) است. برای شطرنج factor branching ۳۵ است و باید تا ۱۰۰ سطح سرچ کنیم که امکان پذیر نیست. پس باید به دنبال راه حل دیگری باشیم.

۲.۶.۹ Minimax Pruning(Alpha-Beta)



شکل ۸.۹: Minimax Pruning

یکی از راه هایی که به ما در بهینه تر کردن الگوریتم کمک میکند pruning است به عنوان مثال درخت بالا را در نظر بگیرید در بین گرده های سمت چپ ۳ کمتر است و به عنوان best option برای گره ریشه ی ما برگردانده میشود. ولی زمانی که گره میانی را بررسی میکنیم به عدد ۲ برخورد میکنیم که از best option ما کمتر است. در حقیقت این بدین معنا است که عددی کوچکتر یا مساوی با ۲ برگردانده خواهد شد که در هر صورت در بالا انتخاب نخواهد شد چون ۳ از آن بزرگتر میباشد برای همین دیگر بقیه گره ها را بررسی نخواهیم کرد و همان ۲ را باز میگردانیم.

```
α: MAX's best option on path to root
β: MIN's best option on path to root
```

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

شکل ۹.۹: Minimax Pruning Code

همانطور که میبینید علاوه بر دادن وضعیت، آلفا و بتا نیز داده میشوند که نمادی از بیشترین و کمترین کاندیدا برای انتخاب شدن در مسیر رسیدن به root هستند. همانطوری که ملاحظه میکنید زمانی که آلفا بزرگتر

از گره ای باشد که دارد بررسی میشود با بلعکس بلافاصله برگردانده میشود تا الگوریتم ما بهینه تر باشد. نکته ای که باید به آن توجه داست این است که اگر child های یک گره به ترتیب و sort وشده باشند این بهینه بودن بیشتر خواهد شد. با این که پیچیدگی زمانی ما به $O(b(n/۲))$ کاهش پیدا میکند. در این الگوریتم و در جواب نهایی که بر روی root نوشته خواهد شد تاثیری نخواهد داشت. این در حالی است که گره های میانی ممکن است دقیق نباشند.

۳.۶.۹ Depth limited search

با وجود تلاشی که برای بهینه کردن minimax با عملیات pruning داشتیم هنوز امکان سرچ کامل را برای شطرنج نخواهیم داشت و مجبوریم از Depth limited search استفاده کنیم که همانطور که قبلا هم گفته شد بررسی و سرچ کردن گره ها تا سطح خاصی ادامه پیدا کند. منتهی مشکل این الگوریتم این است که دیگر بهینه بودن جواب را تضمین نخواهد کرد.

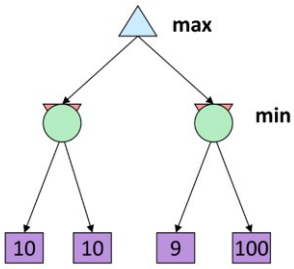
۴.۶.۹ Evaluatoin function

Evaluatoin function چیزی شبیه به heuristic در *a است که مشخص میکند تا چه زمانی این سرچ ادامه پیدا کند. بعنوان مثال Evaluatoin function برای بازی شطرنج می تواند تعداد وزیر باشد. چون معمولا زمانی که وزیر بیرون خواهد رفت بازی برای حریف سخت تر خواهد شد. البته مشکل Evaluatoin function ها دقیقا همین است که همیشه عالی و کامل نیستند و شاید به درستی نتوانند موقعیت مناسب را پیدا کنند. در این جا هم یک Trade-off داریم که هر چقدر برای Evaluatoin function کمتر وقت بگذاریم عمیق تر میتوانیم برویم و هر چقدر بیشتر برای آن وقت بگذاریم Evaluatoin function ما دقیق تر خواهد شد. دقیقا مانند اتفاقی که در *A و برای Heuristic داشتیم.

۷.۹ Expectimax Search

همانطور که گفتیم اگر از الگوریتم minimax استفاده کنیم به دلیل آن که minimax خیلی محتاطانه رفتار میکند نمی تواند از اشتباهات رقیب به عنوان یک فرصت برای امتیاز بیشتر استفاده کند و آن را انتخاب نمیکند. در روش جستجوی expectimax ما درخت جدیدی را در نظر میگیریم که به جای مینیمم بچه های آن گره میانگین را در آن قرار میدهد که به شکل زیر است.

به عنوان مثال در شکلی که ملاحظه می کنید گره سمت چپ برابر ۱۰ و گره سمت راست برابر میانگین ۹ و ۱۰۰ خواهد شد که ۵.۵۴ است. و بنابراین گره max ما مسیر سمت راست را انتخاب خواهد کرد.



شکل ۱۰.۹: Expectimax

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is EXP: return exp-value(state)

def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v

def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

شکل ۱۱.۹: Expectimax Code

سودوکد الگوریتم نیز خیلی شبیه به minimax است فقط با این تفاوت که بجای min گرفتن average گرفتن داریم (محاسبه احتمال). نکته ای که وجود دارد ما در این جا عملیات pruning نداریم و برای محاسبه میانگین لازم است که تمام بچه های یک گره دیده شوند و محاسبه را برای انها انجام دهیم پس روش ما Depth-Limited است یعنی به علت محدودیت زمانی تا عمق خاصی این عملیات را می توانیم که انجام بدهیم و بعد از رسیدن به عمقی خاص باید عملیات جستوجو را متوقف نماییم. دلیلی که باعث میشود ما از این روش استفاده کنیم مشخص نبودن نتیجه در صورت انتخاب یک action هست که میتواند دلایل خاصی داشته باشد.

- طراحی بازی به گونه ای است که deterministic نباشد و متغییری مانند تاس داشته باشیم.
- حرکت طرف مقابل قابل پیشبینی نباشد و بخواهیم جوری بازی را مدل کنیم که از اشتباهات رقیبمان نیز استفاده کنیم. مانند ghost ها در بازی .pacman
- به دلیل fail شدن action ای که انتخاب کردیم. مثلا یک روبات در حرکت کردن به سمت شمال ممکن است موفق نباشد و باید احتمالی را در نظر بگیریم. (برای مواقعی که امکان درست اجرا نشدن action

ها وجود دارد.)

در این الگوریتم باید به نکته ای توجه کنیم و آن نتیجه ی این بازی است. نتیجه بازی هیچ وقت برابر ۵۴ و نیم نمی شود نتیجه همیشه یا ۹ است یا ۱۰۰ ولی این عدد بیانگر این موضوع است که اگر به دفعات این بازی را انجام دهیم امتیازی که از آن می گیریم برابر ۵۴ و نیم خواهد بود. در ضمن در MDP بیشتر در مورد اینگونه مواردی که نتیجه مشخص نیست صحبت خواهد شد.

اگر شرایطی داشته باشیم که رقیب ما در ۸۰ درصد مواقع یک minimax به عمق ۲ را اجرا نماید و در غیر اینصورت بطور رندوم حرکت کند بهترین کار این است که ما نیز در ۲۰ درصد مواقع expectimax را اجرا نماییم تا از فرصت رندوم بودن و غیر هوشمند بودن تصمیمات رقیبمان استفاده کنیم. همانطور که میبینید می توانیم یکسری inform-probability نیز داشته باشیم و ساختار درخت ما بهگونه ای mix شده و ترکیبی باشد و هدف از این مثال نیز همین بود.

۸.۹ Pessimism And Optimism

در اینجا می خواهیم در مورد خوشبین بودن یا محتاط بودن بیش از اندازه صحبت کنیم. اگر بسیار خوش بین باشید در بعضی مواقع به مواردی برخواهید خورد که به دلیل خوش بین بودن شما به مسئله، امتیازی را از دست خواهید داد. برعکس آن نیز درست است اگر خیلی محتاط باشید ممکن است از خیلی از فرصت هایی که به سود شما است استفاده نکنید.

	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

شکل ۱۲.۹: Pessimism And Optimism in Pacman

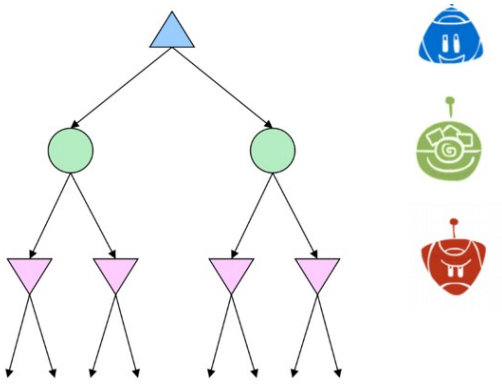
در اینجا ما حالت های مختلف بازی pacman را در نظر گرفته ایم. همانطور که می بینید وقتی بصورت minimax بازی کنیم چه در مقابل یک ghost رندوم و چه در مقابل یک ghost باهوش باشیم بازی را میبریم. اما این نکته حائز اهمیت است که اگر زمانی که ghost رندوم است از الگوریتم expectimax استفاده میکردیم امتیاز بیشتری را میتوانستیم کسب کنیم. اما expectimax نیز در مقابل یک ghost هوشمند خیلی خوش بین خواهد بود و امتیاز از دست میدهد.

۹.۹ Other Game Types

۱.۹.۹ Mix Layer types

شکل ۱۳.۹: Backgammon

ما بازی هایی را با درخت minimax و expectimax بیان کردیم و بازی های دیگری نیز وجود دارند که درخت های آن ها به شکل های دیگری هستند. در حقیقت همه ی آنها همین گره های min ، max یا average را دارند ولی بصورت ترکیبی هستند. که به آن ها Mix Layer types می گویند. به عنوان مثال در بازی تخته نرد یک متغییر به نام تاس داریم که باعث میشود شکل درخت ما بصورت زیر بشود.



شکل ۱۴.۹: Backgammon Tree

اگر بخواهیم محاسباتی برای درخت تخته نرد تا سطح دوم داشته باشیم:

در هر حرکت ما به طور میانگین ۲۰ حرکت میتوانیم انجام دهیم و همچنین ۲۱ امکان برای تاس وجود دارد چون دو تاس داریم و تعداد حالات غیر تکراری آن برابر ۲۱ خواهد شد. حال با توجه به این که در ابتدا

احتمالی برای تاس نداریم (چون تاس ها ریخته شده اند و مقادیر آنها مشخص شده است) محاسبه سطح دوم درخت به شکل زیر است:

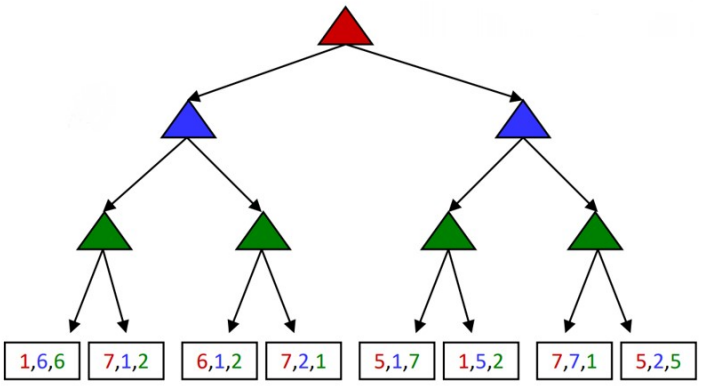
$$20 \times (21 \times 20)^3 = 1.2 \times 10^9 \quad (۱.۹)$$

که در حقیقت عبارت زیر بوده است ولی چون تاس ها در لحظه تصمیم گیری ریخته شده اند یک تقسیم بر ۲۱ دارد.

$$(21 \times 20)^4 \quad (۲.۹)$$

۲.۹.۹ Multi-agent types

در بعضی از بازی ها ما چندین بازیکن داریم و پیشبینی را باید به تعداد بازیکن ها و با توجه به امتیازهایی که میگیرند انجام دهیم.



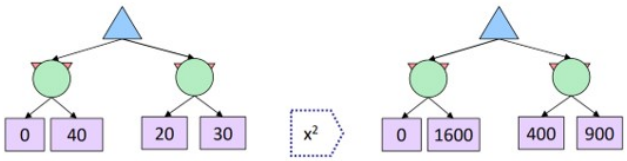
شکل ۱۵.۹: Multi-agent Tree

به عنوان مثال ما در اینجا ۳ بازیکن داریم و باید از پایین شروع کنیم و درگره های سبز رنگ مواردی را بر می داریم که برای سبز بیشترین سودمندی (utility) را داشته باشد. در ادامه همین روند را پیش می گیریم و از بین آنهایی که انتخاب شدند آنهایی که برای آبی بیشترین سودمندی را داشته باشند درون گره مقدار دهی می شوند در نتیجه گره قرمز رنگ از بین (۱،۶،۶) و (۵،۲،۵) آن را که سودمندی بیشتری دارد انتخاب میکند.

(۵،۲،۵)

۱۰.۹ Utilities

در این جا کمی میخواهیم در مورد خود سودمندی ها صحبت کنیم.

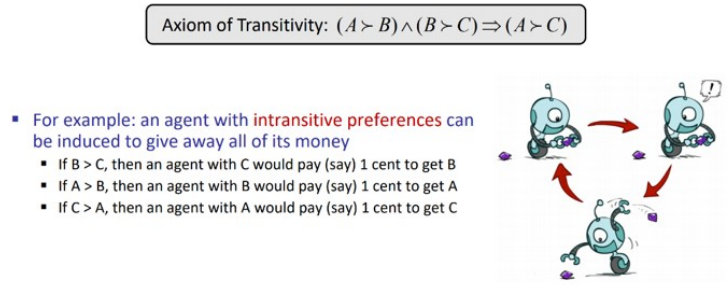


شکل ۱۶.۹: Utilities Difference

نکت ای که وجود دارد خود امتیازها و سودمندی ها حتی میتواند روی الگوریتم و شیوه تصمیم گیری ما تاثیر بگذارند. به عنوان مثال درخت رو به رو را در نظر بگیرید که در آن ما ارزش هر گره را مجذوز کرده ایم. برای minimax ما دقیقا همان نتیجه قبلی را داریم ولی برای expectimax نتیجهی متفاوتی خواهیم داشت که این نتیجه به ما نشان میدهد حتی مقادیر سودمندی میتواند تصمیمات ما را تغییر دهد.

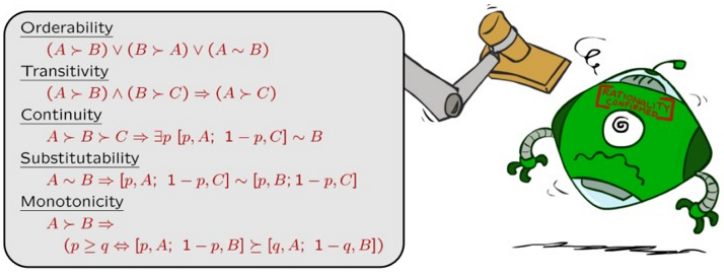
۱.۱۰.۹ Rational Preferences

در اینجا ما روابطی داریم که باید برای سناریو و شروط ما (Constraints) برقرار باشند. به عنوان مثال اگر A سودمند تر از B باشید و B هم سودمند تر از C باشد حتما باید A از C سودمند تر باشد. که به شکل زیر نمایش میدهیم. حال اگر این رابطه برقرار نباشد چه اتفاقی میافتد.



شکل ۱۷.۹: Transitivity

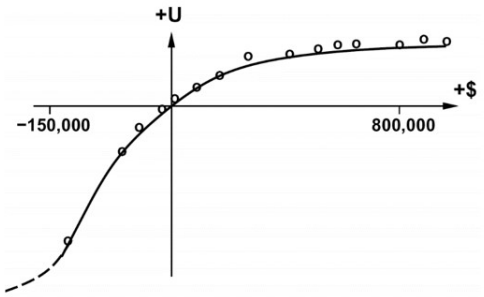
اتفاقی که می افتد این است که Agent ما قادر به تصمیم گیری نخواهد بود و پی در پی تصمیم خود را به دلیل این که دیگری تصمیم بهتری است عوض می کند. مانند نمونه بالا روابط و شروطی وجود دارند که باید برقرار باشند و بعد از داشتن این موارد سناریوی ما اصطلاحا rationality confirmed میشود.



شکل ۱۸.۹: Rational Constraints

۲.۱۰.۹ Human Utilities

در زندگی روزمره، انسان ها تصمیماتی میگیرند که در آنها احتمالات و سودمندی را سنجش مینمایند ولی در مواردی مانند لاتری و درمورد پول کمی این موضوع متفاوت است. ارزش پول نسبی است برای کسی که پولی ندارد یک میلیون دلار خیلی زیاد است ولی برای کسی که میلیاردر است شاید این مقدار سودمندی ای نداشته باشد و همینطور برعکس کسی که مقدار زیادی بدهکار باشد آمدن مقداری بدهی بروی آن برایش کم اهمیت تر است. با وجود این صحبت ها به همچین نموداری خواهیم رسید.



شکل ۱۹.۹: Money Utility Chart

حال می خواهیم آزمایشی انجام دهیم کدام یک از لاتریهای زیر را انتخاب میکنید.

اکثر آدم ها لاتری A و C را انتخاب میکنند و این نشان دهنده این است که همیشه آدم ها منطقی و ریاضیاتی فکر نمیکند. برای مورد C این تصمیم منطقی است ولی برای مورد اول نه:

- A: [0.8, $4k; 0.2, $0]
- B: [1.0, $3k; 0.0, $0]
- C: [0.2, $4k; 0.8, $0]
- D: [0.25, $3k; 0.75, $0]

شکل ۲۰.۹: lotteries

$$B > A \Rightarrow U(\$3k) > 0.8\ U(\$4k)$$

شکل ۲۱.۹: Calculate Utility

همانطوری که احساس می کنید با این که سودمندی و یا utility بیشتری دارد اما بیشتر آدم ها آن را انتخاب می کنند که یکی از دلایل آن می تواند این باشد که اعصاب خوردی آن ۲۰ درصد خیلی تاثیر منفی روی انسان ها داشته باشد. البته این که در یک لاتری داریم شرکت میکنیم با دنباله ای از لاتری ها باشد ممکن است در تصمیم گیری بعضی افراد تاثیر گذار باشد. چون اگر ۱۰۰ لاتری بدین شکل داشته باشیم بعضی از انسان ها میانگین سودهایشان از لاتری را محاسبه میکنند و ممکن است مورد B را انتخاب کنند.

Bibliography