دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

هوش مصنوعی

بهار ۱۴۰۴
استاد: احسان تن قطاری

مهران بختیاری،سینا محمدی،علیرضا میرشفیعیان،علیرضا ملک حسینی، عسل مسکین

تمرین دوم | جست و جوی تخاصمی و CSP | مهلت ارسال: ۷فروردین

- مهلت ارسال پاسخ تا ساعت ۲۳:۵۹ روز مشخص‌شده است.
- در طول ترم امکان ارسال با تاخیر پاسخ همه‌ی تمارین سقف ۴ روز و در مجموع ۱۰ روز، وجود دارد. پس از گذشت این مدت، پاسخ‌های ارسال‌شده پذیرفته نخواهند بود. همچنین، به ازای هر ساعت تأخیر غیر مجاز نیم درصد از نمره ی تمرین کم خواهد شد.
- هم‌کاری و هم‌فکری شما در انجام تمرین مانعی ندارد اما پاسخ ارسالی هر کس حتما باید توسط خود او نوشته شده باشد.
- در صورت هم‌فکری و یا استفاده از هر منابع خارج درسی، نام هم‌فکران و آدرس منابع مورد استفاده برای حل سوال مورد نظر را ذکر کنید.
- لطفا تصویری واضح از پاسخ سوالات نظری بارگذاری کنید. در غیر این صورت پاسخ شما تصحیح نخواهد شد.

سوالات نظری (۱۰۰ نمره)

۱. (۱۵ نمره) درستی یا نادرستی گزاره‌های زیر را با ذکر دلیل کوتاه مشخص کنید.

(آ) هر مسئله‌ی ارضای قیود، قابل تبدیل به یک مسئله‌ی ارضای قیود دودویی است. یعنی می‌توان آن را تبدیل به مسئله‌ای کرد که تمام قیود آن با حداکثر دو متغیر سر و کار دارند.

(ب) اگر یک مسئله‌ی ارضای قیود k-consistent باشد، آنگاه (k-1)-consistent است (به ازای $k > 1$).

(ج) یک مسئله‌ی ارضای قیود دودویی را می‌توان در زمان چندجمله‌ای بر حسب n (تعداد متغیرها) و d (اندازه‌ی دامنه) حل کرد.

(د) در مسائل CSP اگر Arc Consistency را با الگوریتمی مثل AC3 اعمال کنیم، دیگر نیازی به Backtracking نخواهیم داشت.

(ه) در هرس آلفا_بتا اگر گره‌ها را از چپ به راست باز کنیم، تعداد گره‌های هرس شده برابر با حالتی است که گره‌ها را از راست به چپ باز کنیم.

(و) هرس آلفا_بتا می‌تواند مقدار minimax محاسبه شده برای یک گره در درخت بازی را تغییر دهد.

(ز) در درخت Expectimax، اجرای هرس به هیچ وجه ممکن نیست.

حل.

(آ) درست. هر Non-binary CSP که دامنه‌ی متغیرهایش گسسته است، قابل تبدیل به یک Binary CSP معادل است (هرچند تعداد متغیرها زیاد خواهد شد). برای تبدیل یک CSP دلخواه به یک Binary CSP می‌توانید از روش گراف دوگان استفاده کنید. دراین روش، constraintهایی از CSP اوّلیّه که باینری نیستند را در CSP جدید تبدیل به متغیر می‌کنیم. یک مثال در این لینک موجود است.

(ب) غلط. Strong k-consistency با k-consistency فرق دارد. معنی k-consistency این است که به ازای هر k گره دلخواه در گراف قیود، اگر به k-1 گره مقدارهای سازگار نسبت داده باشیم، آنگاه برای گره kام هم حداقل یک مقدار سازگار وجود دارد. تعریف Strong k-consistency این است که هم k-consistent باشد، هم (k-1)-consistent باشد، هم ... هم 1-consistent باشد. Strong k-consistency آنقدر شرط قوی‌ای است که با داشتن آن می‌توانیم مسئله را بدون Backtracking حل کنیم (چطور؟)! شاید بپرسید چطور ممکن است که یک مسئله k-consistent باشد ولی Strongly k-consistent نباشد؟ یک مثال از این را می‌توانید در این لینک ببینید.

(ج) غلط. مسائل ارضای قیود، NP-complete هستند. پس (با فرض $P \neq NP$)، راه‌حل چندجمله‌ای ندارند. مفهوم NP-completeness در درس اتوماتا یا طراحی الگوریتم به شما تدریس خواهد شد پس اگر مفهوم آن را هنوز بلد نیستید نگران نباشید، صرفاً پاسخ این سوال را به یاد بسپارید.

(د) غلط. بعد از اعمال Arc Consistency، سه حالت ممکن است پیش بیاید:

i. اگر دامنه‌ی یکی از متغیرها خالی شد، مسئله جواب ندارد.

ii. اگر در دامنه‌ی تمام متغیرها دقیقاً یک مقدار باقی ماند، به جواب یکتای مسئله رسیدیم.

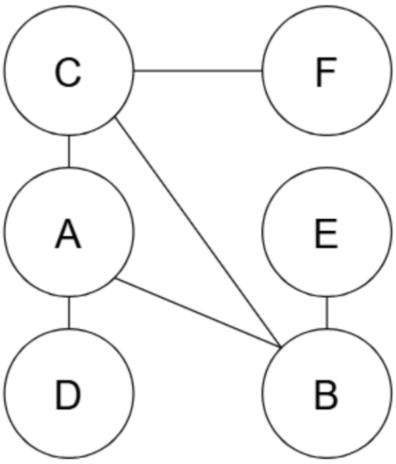
iii. در غیر از دو حالت بالا، نیاز به Backtracking داریم و مسئله ممکن است جواب داشته باشد یا نداشته باشد.

(ه) غلط.

(و) درست. دقت کنید که فقط مقدار گره ریشه است که قطعاً تغییر نمی‌کند. مقدار بقیه‌ی گره‌ها ممکن است تغییر کنند و برای همین است که جست‌وجو در یک درخت با اجرای هرس آلفا_بتا، فقط اولین حرکت بهینه‌ی ما را تعیین می‌کند و وقتی حریفان حرکت خودشان را کردند و دوباره نوبت ما شد، نمی‌توانیم از درختی که در نوبت قبل هرس کردیم استفاده کنیم و باید دوباره در درخت جست‌وجو کنیم.

(ز) غلط. اگر دامنه‌ی گره‌ها محدود باشند، می‌توانیم شبیه هرس آلفا_بتا عمل کنیم و می‌توانید مثال این را در این لینک ببینید.

۲. (۱۰ نمره) نمودار زیر گراف محدودیت یک CSP را نشان می دهد که فقط محدودیت های باینری دارد و در ابتدا هیچ متغیری مقدار دهی نشده است .



(آ) اگر متغیر A را مقداردهی کنیم، دامنه کدام متغیرها بعد از اعمال forward checking روی A تغییر خواهد کرد؟

(ب) اگر متغیر A را مقداردهی و forward checking را روی آن اجرا کنیم سپس متغیر B را مقداردهی کنیم، با اجرای forward checking روی B دامنه کدام متغیرها تغییر خواهد کرد؟

(ج) اگر متغیر A را مقداردهی کنیم، دامنه کدام متغیرها بعد از اعمال arc consistency تغییر خواهد کرد؟

(د) اگر متغیر A را مقداردهی و arc consistency را اجرا کنیم سپس متغیر B را مقداردهی کنیم، با اجرای arc consistency دامنه کدام متغیرها تغییر خواهد کرد؟

حل.

(آ) اجرای forward checking روی A، یال‌هایی را در نظر می‌گیرد که A رأس آن‌ها باشد. بنابراین دامنه رؤوس B، C و D تغییر خواهد کرد.

(ب) مشابه قسمت قبل، اجرای forward checking، یال‌هایی را در نظر می‌گیرد که B رأس آن‌ها باشد. اما از آنجایی که A قبلاً مقداردهی شده است، دامنه آن تغییری نخواهد کرد؛ بنابراین دامنه رؤوس C و E تغییر خواهد کرد.

(ج) اجرای arc consistency روی هر متغیر مقداردهی نشده که مسیری به متغیر مقداردهی شده دارد تأثیر می‌گذارد. این به این دلیل است که تغییر در دامنه یک متغیر، تمام یال‌ها را منجر به تغییر یال‌های دیگر می‌کند. بنابراین تغییرات در گراف منتشر می‌شود و همچنین دامنه تنها زمانی تغییر می‌کند که خالی شود؛ در این صورت الگوریتم نیاز به backtracking دارد. از این رو، در نظر گرفتن دامنه‌های جدید در این مورد منطقی نیست، و دامنه همه متغیرها به غیر از A تغییر خواهد کرد.

(د) پس از دادن یک مقدار به A و اعمال arc consistency، مقداردهی‌های بعدی و اجرای arc consistency منجر به تغییر در دامنه B نمی‌شود. D نیز تغییر نخواهد کرد؛ زیرا تنها یالی که ممکن است باعث تغییر شود با D سازگار است و حذف نمی‌شود. بنابراین دامنه متغیرهای E، C و F تغییر خواهد کرد.

۳. (۱۵ نمره)
تحدب مجموعه ها و یا توابع زیر را نشان دهید.

(آ)

$$A = \{(x,y) \mid ||x|| \leq y\}$$

(ب)

$$A = \{(x+y) \mid x, y \in B\}$$

با این فرض که B یک مجموعه‌ی محدب است.

(ج)

$$f(x) = x^۲ y^۲, \quad (x,y) \in \mathbb{R}^۲$$

(د)

$$f(x) = \begin{cases} x, & x > ۰ \\ a, & x = ۰ \\ \infty, & x < ۰ \end{cases}$$

با فرض اینکه a یک مقدار دلخواه نامنفی دارد.

حل.

(۱) اگر $(x,y),(a,b)\in A$ آنگاه داریم $||a||\leq b$ و $||x||\leq y$. حال طبق تعریف نرم می‌دانیم که به ازای تمام $\lambda\in[۰,۱]$ داریم:

$$||(\lambda x+(۱-\lambda)a)||\leq(\lambda||x||+(۱-\lambda)||a||)$$

حال دوباره از تعریف نرم استفاده می‌کنیم، داریم:

$$||\lambda x+(۱-\lambda)a||\leq||\lambda x||+||(۱-\lambda)a||\leq\lambda y+(۱-\lambda)b$$

و این بدان معناست که به ازای تمام $\lambda\in[۰,۱]$ داریم:

$$(\lambda x+(۱-\lambda)a,\lambda y+(۱-\lambda)b)\in A$$

که این نکته، تحدب مجموعه‌ی A را اثبات می‌کند.

(ب) حال $a,b\in A$ را در نظر می‌گیریم و طبق تعریف مجموعه‌ی A می‌دانیم که داریم: $a=a_\gamma+a_\tau$ و $b=b_\gamma+b_\tau$. پس برای هر $\lambda\in[۰,۱]$ داریم:

$$\lambda a+(۱-\lambda)b=\lambda a_\gamma+(۱-\lambda)b_\gamma+\lambda a_\tau+(۱-\lambda)b_\tau$$

حال با توجه به تحدب مجموعه‌ی B می‌دانیم که $\lambda a_\gamma+(۱-\lambda)b_\gamma\in B$ و $\lambda a_\tau+(۱-\lambda)b_\tau\in B$، پس:

$$\lambda a_\gamma+(۱-\lambda)b_\gamma+\lambda a_\tau+(۱-\lambda)b_\tau\in A$$

که این مسئله، تحدب مجموعه‌ی A را اثبات می‌کند.

(ج) تابع مورد نظر به صورت $f(x,y)=x^۲y^۲$ تعریف شده است.
برای بررسی تحدب تابع، ابتدا باید مشتقات جزئی مرتبه اول و سپس ماتریس هسین تابع را محاسبه کنیم:
مشتقات جزئی مرتبه اول:

$$\frac{\partial f}{\partial x}=۲xy^۲,\quad\frac{\partial f}{\partial y}=۲x^۲y$$

مشتقات جزئی مرتبه دوم:

$$\frac{\partial^۲ f}{\partial x^۲}=۲y^۲,\quad\frac{\partial^۲ f}{\partial y^۲}=۲x^۲,\quad\frac{\partial^۲ f}{\partial x\partial y}=\frac{\partial^۲ f}{\partial y\partial x}=۴xy$$

بنابراین، ماتریس هسین تابع به‌صورت زیر خواهد بود:

$$H=D^۲f(x,y)=\begin{pmatrix}۲y^۲ & ۴xy\\ ۴xy & ۲x^۲\end{pmatrix}$$

اکنون برای تعیین تحدب، از مینورهای اصلی (principal minors) استفاده می‌کنیم:

- مینور اول $(\Delta_۱)$: عنصر قطری اول ماتریس هسین است:

$$\Delta_۱=۲y^۲\geq ۰$$

- مینور دوم $(\Delta_۲)$: دترمینان کل ماتریس است:

$$\Delta_۲=(۲y^۲)(۲x^۲)-(۴xy)^۲=۴x^۲y^۲-۱۶x^۲y^۲=-۱۲x^۲y^۲$$

با توجه به این‌که:

$$\Delta_1 \geq 0, \quad \Delta_2 < 0$$

و طبق جدول تعیین تحدب ماتریس هسین، در این حالت ماتریس هسین نامعین (indefinite) خواهد بود.

نوع تابع	Δ_2	Δ_1
نه محدب و نه مقعر	< 0	≥ 0

در نتیجه تابع f در حالت کلی نه محدب است نه مقعر.

(د) برای این مورد دو متغیر x_1, x_2 را در نظر می‌گیریم و شرط زیر را بر روی آن‌ها چک می‌کنیم:

$$f((1-\alpha)x_1 + \alpha x_2) \leq (1-\alpha)f(x_1) + \alpha f(x_2)$$

که $\alpha \in [0, 1]$.

حال اگر هر دوی x_1, x_2 اکیداً مثبت باشند، سمت راست نابرابری بالا به تساوی تبدیل می‌شود.

اگر یکی از دو متغیر اکیداً مثبت و دیگری صفر باشد، سمت راست شرط بالا به‌صورت نابرابری برقرار می‌ماند.

اگر حداقل یکی از این دو متغیر اکیداً منفی باشد، سمت راست نابرابری بالا برابر ∞ خواهد بود و بدون توجه به حالات مختلف در مورد x_1, x_2، تساوی برقرار خواهد بود.

۴. (۲۰ نمره) درخت Minimax شکل۱ را در نظر بگیرید.

(آ) مقدار Minimax گره ریشه را تعیین کنید. آیا امکان دارد مقدار دیگری در یک وضعیت مشابه اما با ترتیب دیدن متفاوت به دست آید؟ چرا؟

(ب) هرس آلفا–بتا را در این درخت اعمال کنید، فرض کنید گره‌های فرزند از چپ به راست دیده می‌شوند. تمام گره‌هایی که به دلیل هرس دیده نمی‌شوند را مشخص کرده و دلیل هرس آنها را توضیح دهید.

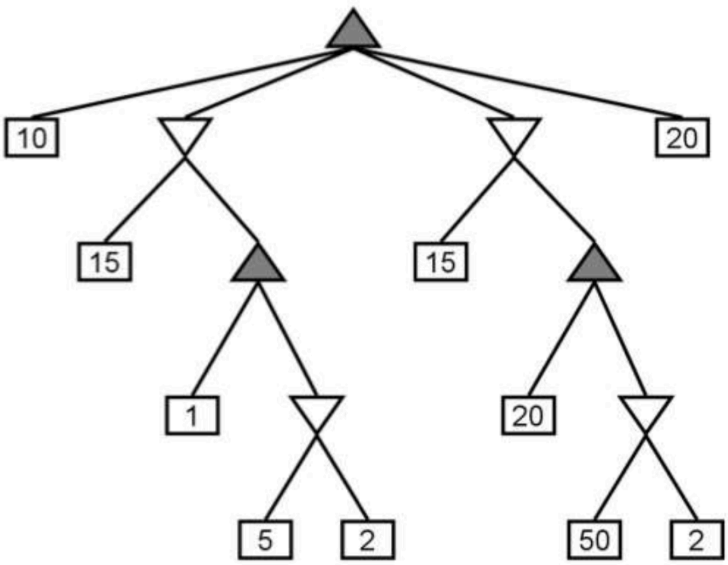
(ج) آیا ترتیب دیگری برای فرزندان گره ریشه وجود دارد که به ازای آن، هرس شدن بیشتری اتفاق بیفتد؟ اگر بله، این ترتیب را بیابید و دلیل خود را توضیح دهید.

(د) یک استراتژی کلی و عملی برای ترتیب‌دهی به فرزندان گره‌ها که منجر به حداکثر هرس شدن ممکن شود ارائه دهید. این استراتژی باید هم برای گره‌های Max و هم برای گره‌های Min به وضوح مشخص باشد.

(ه) فرض کنید که مقادیر برگ‌ها نامشخص هستند و فقط در یک محدوده داده شده‌اند (مثلاً مقادیر برگ‌ها می‌توانند در بازه $[a, b]$ باشند). در این صورت چگونه می‌توان از اطلاعات بازه‌ای برای بهینه‌سازی هرس آلفا–بتا استفاده کرد؟ آیا می‌توان بدون بررسی برخی از گره‌های داخلی، مقدار Minimax را دقیق‌تر تخمین زد؟ توضیح دهید.

(و) در نظر بگیرید که این درخت به جای گره‌های Min و Max شامل گره‌هایی است که استراتژی‌های متفاوتی را دنبال می‌کنند، مانند یک گره که بر اساس میانگین مقادیر فرزندانش مقداردهی می‌شود. چگونه می‌توان الگوریتم Minimax را برای این سناریو اصلاح کرد؟ آیا هرس آلفا–بتا هنوز قابل اعمال است؟ چرا یا چرا نه؟

حل.



شکل ۱: درخت Minimax

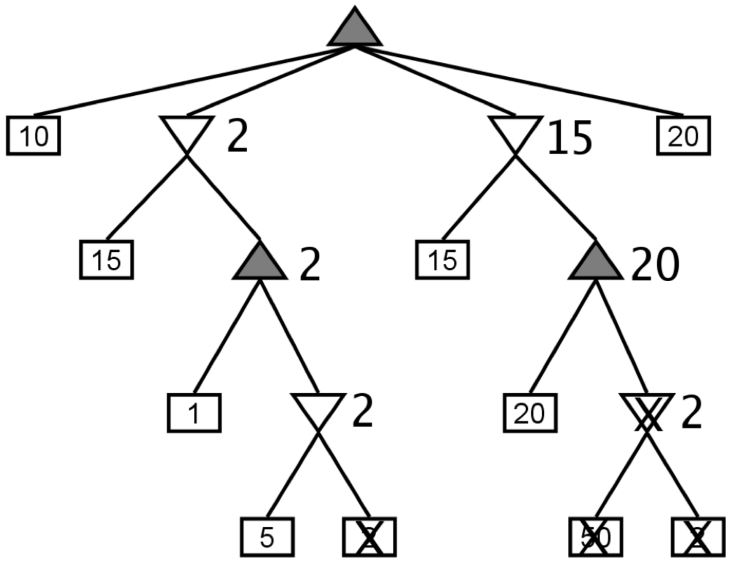
(آ) با توجه به درخت و مقادیری که در برگ‌ها قرار داده شده است (طبق شکل سؤال)، گره ریشه یک گره Max است. محاسبه دقیق نشان می‌دهد که مقدار مینی‌مکس گره ریشه برابر با ۲۰ خواهد بود.
در حالت کلی، زمانی که ساختار درخت، تعداد فرزندان و مقادیر برگ‌ها ثابت بمانند، مقدار مینی‌مکس تغییر نمی‌کند. علت آن این است که ترتیب بازدید فرزندان تأثیری بر مقدار نهایی مینی‌مکس ندارد؛ چرا که این مقدار توسط قواعد Max/Min در سطوح مختلف درخت تعیین می‌شود و مستقل از این است که ابتدا کدام فرزند بررسی شده باشد. اما تعداد گره‌هایی که جهت محاسبه بررسی می‌شوند یا نحوه‌ی هرس ممکن است تحت تأثیر ترتیب فرزندان تغییر کند، نه مقدار نهایی. بنابراین مقدار ۲۰ برای گره ریشه در هر ترتیب بازدیدی یکسان است و مقدار دیگری قابل دست‌یافتن نیست. به عبارت دیگر، مینیمکس یک الگوریتم کامل است که همیشه بهترین حرکت ممکن را با فرض بازی بهینه هر دو طرف پیدا می‌کند.

(ب) ابتدا مقدار α و β را در طول پیمایش به‌روز می‌کنیم. با فرض این‌که فرزندان گره ریشه (که یک گره Max است) از چپ به راست باشند، زمانی که به مقدارهای فرزندان می‌رسیم، شرایط زیر باعث هرس آلفا_بتا می‌شود:

- α در گره Max نمایانگر بهترین مقداری است که Max تا آن لحظه توانسته بیابد.
- β در گره Min نمایانگر بهترین مقداری است که Min تا آن لحظه توانسته بیابد.

در شکل زیر، گره‌هایی که با علامت X مشخص شده‌اند، هرس می‌شوند. به صورت خلاصه، دلیل هرس هر گره یکی از موارد زیر است:

- هرس در گره Max (هرس بتا): هنگامی که مقدار α یک گره Max (بهترین گزینه موقت برای Max در آن گره) بزرگتر یا مساوی مقدار β ارسال شده از جد Min آن شود ($\alpha \geq \beta$). در این حالت، بازیکن Min در سطح بالاتر هرگز اجازه نمی‌دهد بازی به این گره Max برسد، زیرا گزینه‌ای با ارزش حداکثر β (که کمتر از α است) برای خود دارد. بنابراین، بقیه‌ی فرزندان این گره Max هرس می‌شوند.
- هرس در گره Min (هرس آلفا): هنگامی که مقدار β یک گره Min (بهترین گزینه موقت برای Min در آن گره) کوچکتر یا مساوی مقدار α ارسال شده از جد Max آن شود ($\beta \leq \alpha$). در این حالت، بازیکن Max در سطح بالاتر هرگز این مسیر را انتخاب نمی‌کند، زیرا گزینه‌ای با ارزش حداقل α (که بیشتر از β است) برای خود دارد. بنابراین، بقیه‌ی فرزندان این گره Min هرس می‌شوند.



شکل ۲: درخت Minimax هرس شده

(ج) بله؛ اگر فرزندان ریشه به صورتی مرتب شوند که گره‌ای با مقدار بزرگ‌تر (مثلاً ۲۰) زودتر بررسی شود، آنگاه در سطوح بعدی مقدار α زودتر افزایش خواهد یافت و هرس بیشتری انجام می‌شود.
در مثال موجود، بهترین ترتیب (برای بیشینه کردن هرس) این است که فرزندان ریشه را در ترتیب

$$۲۰,\ ۱۵,\ ۱۰,\ ۲$$

بررسی کنیم. دلیل آن این است که در گره Max، پیدا شدن زودهنگام مقدارهای بزرگ باعث می‌شود α بالا رود و در نتیجه بسیاری از شاخه‌های دیگر زودتر بی‌اثر شده و هرس گردند. به عبارت دیگر، اگر ابتدا فرزندان با ارزش بالاتر بررسی شوند، مقدار آلفا سریع‌تر افزایش می‌یابد و شانس هرس در گره‌های بعدی بیشتر می‌شود. در این حالت، با بررسی ابتدا فرزند با ارزش ۲۰، آلفا به سرعت به ۲۰ افزایش می‌یابد و احتمال هرس گره‌های بعدی افزایش می‌یابد.

(د) به طور کلی، اگر ارزیابی درستی از ارزش گره‌ها (یا محدوده‌ی تقریبی آن‌ها) در دست داشته باشیم، می‌توانیم از آن برای ترتیب‌دهی فرزندان استفاده کنیم:

- در گره‌های Max: فرزندان را به ترتیبی بررسی می‌کنیم که پرارزش‌ترین فرزند (یا فرزندی که بیشترین تخمین را برای خروجی دارد) زودتر بررسی شود تا α سریع‌تر بالا برود و باعث هرس شدن بقیه گره‌ها شود. در واقع فرزندان را بر اساس تخمین مقدار ارزیابی آنها به ترتیب نزولی مرتب کنیم.
- در گره‌های Min: فرزندان را نیز به ترتیبی بررسی می‌کنیم که کم‌ارزش‌ترین فرزند (یا فرزندی که کمترین تخمین را دارد) زودتر بررسی شود تا β سریع‌تر پایین بیاید و باعث هرس شدن بیشتر شود. در واقع فرزندان را بر اساس تخمین مقدار ارزیابی انها به ترتیب صعودی مرتب کنیم.

البته در عمل برای بازی‌های پیچیده، همیشه نمی‌دانیم ارزش واقعی برگ‌ها چیست. بنابراین از تابع ارزیابی (evaluation function) در سطح متوسط درخت استفاده می‌کنیم تا فرزندان را به صورت تخمینی مرتب کنیم. هر چه این تخمین قوی‌تر باشد، احتمال افزایش هرس نیز بیشتر می‌شود. در واقع این استراتژی با استفاده از یک تابع ارزیابی برای تخمین مقادیر گره‌ها قبل از بررسی کامل آنها عملی می‌شود. نکته مهم این است که مرتب‌سازی بر اساس مقادیر واقعی گره‌ها عملی نیست، زیرا این مقادیر قبل از بررسی گره‌ها مشخص نیستند.

(ه) هنگامی که تنها یک بازه‌ی ممکن برای مقدار هر برگ می‌دانیم (مثلاً $[a, b]$)، می‌توان با در نظر گرفتن کمینه و بیشینه‌ی ممکن برای گره‌های Min/Max در سطوح مختلف، حد بالایی و حد پایینی برای مقدار یک گره محاسبه کرد. در واقع:

- در گره Max: اگر تا یک جایی حد پایینی ممکن از مقدارش بالاتر از β باشد، هرس می‌تواند رخ دهد.
- در گره Min: اگر تا یک جایی حد بالایی ممکن از مقدارش کمتر از α باشد، هرس می‌تواند رخ دهد.

بنابراین با استفاده از بازه‌های مقادیر هر برگ، در طول محاسبه مقدار داخلی هر گره (حتی اگر بعضی برگ‌ها یا گره‌های پایین‌تر دقیق بررسی نشده باشند) می‌توان با قطعیت گفت که مقدار حقیقی آن گره در یک بازه مشخص قرار می‌گیرد. اگر این بازه با بازه یا مقدار به‌دست‌آمده از گره‌های دیگر هم‌پوشانی نداشته باشد و معیار $\alpha \geq \beta$ (یا بالعکس در سطح Min/Max) ایجاد شود، هرس زودتر صورت می‌گیرد.

همچنین در برخی موارد ممکن است بدون محاسبه دقیق همه‌ی برگ‌ها بتوانیم یک تخمین دقیق‌تر برای مقدار مینی‌مکس گره‌های میانی ارائه دهیم. به‌طور مثال اگر بدانیم بازه‌ی ممکن یک گره Min کوچک‌تر از یک حد بالایی است که دیگر نمی‌تواند از مقدار فعلی α در سطح بالاتر عبور کند، عملاً نیازی به ادامه بررسی جزئیات آن گره Min نخواهیم داشت. هر چه بازه‌هایی که در اختیار داریم کوچک‌تر باشند (مثلاً $[۴/۹, ۵/۱]$ به جای $[۰, ۱۰]$)، امکان هرس بیشتری نیز فراهم می‌شود و حتی ممکن است گره‌های داخلی‌ای که قرار بود محاسبه شوند، اصلاً بررسی نشوند.

پس در صورتی که مقادیر برگ‌ها در یک بازه $[a, b]$ قرار داشته باشند، می‌توان از این اطلاعات برای بهبود هرس Alpha-Beta استفاده کرد:

هرس بر اساس حد بالا و پایین: اگر حد بالای مقدار یک زیردرخت (که توسط یک گره Min ریشه دارد) کوچکتر یا مساوی آلفای (Alpha) فعلی شود ($b_{\min} \leq \alpha$)، یا حد پایین مقدار یک زیردرخت (که توسط یک گره Max ریشه دارد) بزرگتر یا مساوی بتای (Beta) فعلی شود ($a_{\max} \geq \beta$)، می‌توان آن زیردرخت را بدون بررسی کامل هرس کرد.

تخمین دقیق‌تر: با استفاده از حدود بازه‌ای، می‌توان تخمین دقیق‌تری از مقدار نهایی Minimax داشت. برای مثال، اگر می‌دانیم مقادیر برگ‌ها در بازه $[۰, ۱۰۰]$ هستند و مقدار آلفای (Alpha) فعلی ۸۰ است، می‌توانیم زیردرخت‌هایی که توسط گره Min ریشه دارند و حد بالای آن‌ها کمتر یا مساوی ۸۰ است ($b_{\min} \leq ۸۰$) را هرس کنیم.

تخمین مقدار Minimax: با ترکیب حدود بالا و پایین از زیردرخت‌های مختلف، می‌توان یک بازه برای مقدار نهایی Minimax به دست آورد، حتی بدون بررسی کامل برخی گره‌ها.

این روش به هرس Alpha-Beta با اطلاعات محدوده‌ای یا Alpha-Beta with Bounds شناخته می‌شود و می‌تواند کارایی الگوریتم را به طور قابل توجهی افزایش دهد.

(و) Minimax مخصوص سناریویی طراحی شده است که دو بازیکن در تضاد (با عملگرهای Min و Max) عمل می‌کنند. اگر درخت شامل گره‌هایی باشد که روش به‌روزرسانی مقدارشان متفاوت است (مثلاً میانگین گرفتن از فرزندان)، آنگاه ساختار بازگشتی و قانون قطع هرس آلفا–بتا ($\alpha \geq \beta$) به‌صورت سنتی صحت ندارد.

به صورت دقیق‌تر: در هرس آلفا–بتا، تنها با علم به این که گره والد Max یا Min است، می‌توان تصمیم گرفت دیگر نیازی به محاسبه برخی زیرشاخه‌ها نیست. اما اگر یک گره استراتژی میانگین داشته باشد، هر فرزند روی مقدار نهایی آن گره تأثیر دارد؛ یعنی حذف (هرس) حتی یک فرزند می‌تواند میانگین را تغییر دهد. به این ترتیب، قاعده‌ی ساده‌ی $\alpha \geq \beta$ دیگر کفایت نمی‌کند و نمی‌توان بدون دیدن مقادیر بقیه فرزندان نتیجه گرفت که مقدار گره دقیقاً چه خواهد شد.

اگرچه ممکن است روش‌هایی مانند برآورد بازه‌ای یا تکنیک‌های برش شاخه‌ها (branch and bound) در برخی ساختارهای خاص (مثلاً وقتی اوزان فرزندان یا بازه‌ی آن‌ها به شکل خاصی باشند) منجر به هرس شود، اما الگوریتم آلفا–بتای کلاسیک در معنای مرسوم آن قابل اعمال نیست؛ چرا که قاعده‌ی به‌روزرسانی α و β به‌صورت Max و Min دیگر معتبر نخواهد بود. در نتیجه هرس کلی و ساده مانند آلفا–بتا در چنین درختی امکان‌پذیر نیست یا به‌شدت محدود خواهد شد. با این حال، می‌توان نسخه‌های اصلاح‌شده هرس را برای برخی استراتژی‌ها طراحی کرد، اما این نیازمند تحلیل دقیق خواص ریاضی استراتژی مورد نظر است.

۵. (۲۰ نمره) گروهی از هکرها پس از نفوذ موفق به یک سیستم فوق‌امنیتی، یک الگوریتم رمزنگاری ارزشمند را به دست آورده‌اند. این گروه شامل N عضو است که بر اساس توانایی‌ها و میزان تأثیرگذاری خود، نسبت به یکدیگر قدرت و میزان نفوذ متفاوتی دارند. حال آن‌ها باید تصمیم بگیرند که چگونه منافع حاصل از فروش این الگوریتم را بین خود تقسیم کنند. هر بار یکی از اعضای گروه (به‌عنوان رئیس موقت) پیشنهاد تقسیم سود را ارائه می‌دهد. سپس اعضای باقی‌مانده در مورد این پیشنهاد رأی‌گیری می‌کنند. اگر حداقل نصف اعضای گروه (شامل خود رئیس) به پیشنهاد رأی مثبت دهند، سود طبق پیشنهاد تقسیم می‌شود. در غیر این صورت، رئیس موقت از گروه حذف شده و یکی از اعضای باقی‌مانده جای او را می‌گیرد.

شرایط و محدودیت‌ها:

- هر هکر ترجیح می‌دهد زنده بماند و پس از آن دریافت بیشترین سهم از سود را هدف اصلی خود قرار می‌دهد.
- اگر دو پیشنهاد مقدار مساوی برای یکی از اعضا داشته باشند، او به نفع پیشنهادی که رئیس قوی‌تری دارد رأی می‌دهد.
- اگر عضو X متوجه شود که رأی او به حذف رئیس فعلی منجر می‌شود و در آینده شانس بیشتری برای رئیس شدن دارد، ممکن است تصمیم بگیرد که رأی منفی بدهد.
- برخی اعضای گروه دارای اتحاد مخفی هستند که به نفع یکدیگر رأی می‌دهند، مگر اینکه منفعت شخصی‌شان به خطر بیفتد.

(آ) در حالتی که تعداد اعضای گروه $۲, ۳, ۴, ۵$ و ۶ نفر است، درخت بازی کامل را رسم کنید و تحلیل کنید که چگونه تصمیم‌گیری‌ها در هر مرحله انجام می‌شود.

(ب) برای حالت کلی، الگوریتمی ارائه دهید که به یک رئیس موقت کمک کند تا بهینه‌ترین پیشنهاد ممکن را ارائه دهد تا زنده بماند.

(ج) تحلیل کنید که آیا در این سیستم، راهکاری وجود دارد که اعضای ضعیف‌تر در برابر حذف شدن مقاومت کنند؟ اگر بله، چگونه؟

حل.

(آ) تحلیل بازی برای $N = ۲, ۳, ۴, ۵, ۶$ با استفاده از Backward Induction: از روش Backward Induction استفاده می‌کنیم؛ یعنی از پایان بازی (تعداد کمتر اعضا) شروع کرده و به سمت ابتدای بازی (تعداد اعضای بیشتر) حرکت می‌کنیم. فرض می‌کنیم ۱۰۰ واحد سود برای تقسیم وجود دارد و هکرها کاملاً منطقی عمل می‌کنند.

حالت پایه: $N = ۱$ اگر تنها یک هکر ($H_۱$) باقی بماند، او تمام ۱۰۰ واحد سود را برمی‌دارد. نیازی به پیشنهاد و رأی‌گیری نیست. نتیجه: (۱۰۰)

حالت $N = ۲$: هکرها $H_۱, H_۲$

- رئیس: $H_۱$.
- تعداد اعضای حاضر (k): ۲.
- حد نصاب رأی: $۱ = \lceil ۲/۲ \rceil$. رأی خود $H_۱$ کافی است.
- تحلیل: $H_۱$ می‌داند که رأی خودش برای تصویب پیشنهاد کافی است. بنابراین، برای حداکثر کردن سود خود (اولویت دوم)، نیازی به دادن سهمی به $H_۲$ ندارد.
- پیشنهاد بهینه $H_۱$: $(s_۱, s_۲) = (۱۰۰, ۰)$.
- رأی‌گیری:
  - $H_۱$: بله (برای حداکثر سود).

- $H_۲$: خیر (چون ۰ می‌گیرد؛ هرچند رأیش تأثیری ندارد).

- نتیجه: پیشنهاد با ۱ رأی موافق از ۲ رأی تصویب می‌شود. توزیع نهایی: $(۱۰۰, ۰)$.
- توصیف درخت بازی: گره ریشه $(H_۱, H_۲)$ با رئیس $H_۱$. شاخه پیشنهاد $(۱۰۰, ۰)$. نتیجه پذیرش. بازی تمام می‌شود.

حالت $N = ۳$: هکرها $H_۱, H_۲, H_۳$

- رئیس: $H_۱$.
- تعداد اعضای حاضر (k): ۳.
- حد نصاب رأی: $۲ = \lceil ۳/۲ \rceil$. $H_۱$ به رأی حداقل یک نفر دیگر نیاز دارد.
- تحلیل (Backward Induction):
  - اگر $H_۱$ حذف شود: بازی با $(H_۲, H_۳)$ ادامه می‌یابد و $H_۲$ رئیس می‌شود. طبق تحلیل $N = ۲$، $H_۲$ پیشنهاد $(۱۰۰, ۰)$ به $H_۳$ می‌دهد و نتیجه $(۱۰۰, ۰)$ برای $(H_۲, H_۳)$ خواهد بود؛ یعنی در این سناریو، $H_۲$ مقدار ۱۰۰ و $H_۳$ مقدار ۰ کسب می‌کند.
  - تصمیم $H_۱$: $H_۱$ برای زنده ماندن نیاز به یک رأی دیگر دارد. او باید ببیند چه کسی با کمترین هزینه حاضر به رأی دادن است.
    * $H_۲$: اگر $H_۱$ حذف شود ۱۰۰ می‌گیرد. پس برای رأی مثبت به $H_۱$ حداقل ۱۰۱ واحد می‌خواهد که غیرممکن است؛ بنابراین $H_۲$ رأی منفی خواهد داد.
    * $H_۳$: اگر $H_۱$ حذف شود ۰ می‌گیرد. $H_۱$ می‌تواند با پیشنهاد مبلغی بیش از ۰ (حداقل ۱ واحد) رأی $H_۳$ را بخرد.
  - پیشنهاد بهینه $H_۱$: $H_۱$ برای حداکثر کردن سود خود، کمترین مقدار ممکن (۱ واحد) را به $H_۳$ می‌دهد و بقیه (۹۹ واحد) را برای خود نگه می‌دارد. پیشنهاد: $(s_۱, s_۲, s_۳) = (۹۹, ۰, ۱)$.
  - رأی‌گیری:
    * $H_۱$: بله (بقا + ۹۹ سود).
    * $H_۲$: خیر (۰ در مقابل ۱۰۰ در صورت حذف $H_۱$).
    * $H_۳$: بله (۱ در مقابل ۰ در صورت حذف $H_۱$).
  - نتیجه: پیشنهاد با ۲ رأی موافق از ۳ رأی تصویب می‌شود. توزیع نهایی: $(۹۹, ۰, ۱)$.
  - توصیف درخت بازی: گره ریشه $(H_۱, H_۲, H_۳)$ با رئیس $H_۱$. شاخه پیشنهاد $(۹۹, ۰, ۱)$، نتیجه پذیرش، بازی تمام می‌شود. در صورت هر پیشنهاد دیگر، رأی کافی کسب نمی‌شود، پیشنهاد رد می‌شود، $H_۱$ حذف می‌شود، گره جدید $(H_۲, H_۳)$ با رئیس $H_۲$ شکل می‌گیرد. در این گره، پیشنهاد $(۱۰۰, ۰)$ ارائه و پذیرفته می‌شود، ولی $H_۱$ با ارائه پیشنهاد بهینه از این مسیر جلوگیری می‌کند.

حالت $N = ۴$: هکرها $H_۱, H_۲, H_۳, H_۴$

- رئیس: $H_۱$.
- تعداد اعضای حاضر (k): ۴.
- حد نصاب رأی: $۲ = \lceil ۴/۲ \rceil$. $H_۱$ به رأی حداقل یک نفر دیگر نیاز دارد.
- تحلیل (Backward Induction):
  - اگر $H_۱$ حذف شود: بازی با $(H_۲, H_۳, H_۴)$ ادامه می‌یابد و $H_۲$ رئیس می‌شود. طبق تحلیل $N = ۳$، $H_۲$ پیشنهاد $(۹۹, ۰, ۱)$ به $(H_۳, H_۴)$ می‌دهد و نتیجه $(۹۹, ۰, ۱)$ برای $(H_۲, H_۳, H_۴)$ خواهد بود؛ یعنی $H_۲ = ۹۹, H_۳ = ۰, H_۴ = ۱$.
  - تصمیم $H_۱$: $H_۱$ نیاز به یک رأی دیگر دارد.
    * $H_۲$: در صورت حذف $H_۱$ مقدار ۹۹ می‌گیرد؛ رأی منفی می‌دهد.
    * $H_۳$: در صورت حذف $H_۱$ مقدار ۰ می‌گیرد؛ $H_۱$ می‌تواند با پیشنهاد ۱ واحد رأی او را بخرد.

* H_4: در صورت حذف H_1 مقدار ۱ می‌گیرد؛ برای جلب رأی او، H_1 باید حداقل ۲ واحد پیشنهاد دهد.
- پیشنهاد بهینه H_1: H_1 ارزان‌ترین رأی را می‌خرد که رأی H_3 با ۱ واحد است. پیشنهاد: $(s_1, s_2, s_3, s_4) = (۹۹, ۰, ۱, ۰)$.
- رأی‌گیری:
  * H_1: بله (بقا + ۹۹ سود).
  * H_2: خیر (۰ در مقابل ۹۹).
  * H_3: بله (۱ در مقابل ۰).
  * H_4: خیر (۰ در مقابل ۱).
- نتیجه: پیشنهاد با ۲ رأی موافق از ۴ رأی تصویب می‌شود. توزیع نهایی: $(۹۹, ۰, ۱, ۰)$.

حالت $N = ۵$: هکرها H_1, H_2, H_3, H_4, H_5

- رئیس: H_1.
- تعداد اعضای حاضر (k): ۵.
- حد نصاب رأی: $\lceil ۵/۲ \rceil = ۳$. H_1 به رأی حداقل دو نفر دیگر نیاز دارد.
- تحلیل (Backward Induction):
- اگر H_1 حذف شود: بازی با (H_2, H_3, H_4, H_5) و رئیس H_2 ادامه می‌یابد. طبق تحلیل $N = ۴$، H_2 پیشنهاد $(۹۹, ۰, ۱, ۰)$ به (H_3, H_4, H_5) می‌دهد. نتیجه: $H_2 = ۹۹, H_3 = ۰, H_4 = ۱, H_5 = ۰$.
- تصمیم H_1: H_1 نیاز به دو رأی دیگر دارد. باید دو نفر ارزان‌تر را پیدا کند.
  * H_2: در صورت حذف H_1 مقدار ۹۹ می‌گیرد؛ رأی منفی می‌دهد.
  * H_3: در صورت حذف H_1 مقدار ۰ می‌گیرد؛ هزینه خرید رأی: ۱ واحد.
  * H_4: در صورت حذف H_1 مقدار ۱ می‌گیرد؛ هزینه خرید رأی: ۲ واحد.
  * H_5: در صورت حذف H_1 مقدار ۰ می‌گیرد؛ هزینه خرید رأی: ۱ واحد.
- پیشنهاد بهینه H_1: H_1 دو رأی ارزان‌تر را می‌خرد: H_3 و H_5 هر کدام با ۱ واحد. پیشنهاد: $(s_1, s_2, s_3, s_4, s_5) = (۹۸, ۰, ۱, ۰, ۱)$.
- رأی‌گیری:
  * H_1: بله (بقا + ۹۸ سود).
  * H_2: خیر (۰ در مقابل ۹۹).
  * H_3: بله (۱ در مقابل ۰).
  * H_4: خیر (۰ در مقابل ۱).
  * H_5: بله (۱ در مقابل ۰).
- نتیجه: پیشنهاد با ۳ رأی موافق از ۵ رأی تصویب می‌شود. توزیع نهایی: $(۹۸, ۰, ۱, ۰, ۱)$.

حالت $N = ۶$: هکرها $H_1, H_2, H_3, H_4, H_5, H_6$

- رئیس: H_1.
- تعداد اعضای حاضر (k): ۶.
- حد نصاب رأی: $\lceil ۶/۲ \rceil = ۳$. H_1 به رأی حداقل دو نفر دیگر نیاز دارد.
- تحلیل (Backward Induction):
- اگر H_1 حذف شود: بازی با $(H_2, H_3, H_4, H_5, H_6)$ و رئیس H_2 ادامه می‌یابد. طبق تحلیل $N = ۵$، H_2 پیشنهاد $(۹۸, ۰, ۱, ۰, ۱)$ به (H_3, H_4, H_5, H_6) می‌دهد. نتیجه: $H_2 = ۹۸, H_3 = ۰, H_4 = ۱, H_5 = ۰, H_6 = ۱$.
- تصمیم H_1: H_1 نیاز به دو رأی دیگر دارد.
  * H_2: در صورت حذف H_1 مقدار ۹۸ می‌گیرد؛ رأی منفی می‌دهد.

* H_3: در صورت حذف H_1 مقدار ۰ می‌گیرد؛ هزینه خرید رأی: ۱ واحد.
* H_4: در صورت حذف H_1 مقدار ۱ می‌گیرد؛ هزینه خرید رأی: ۲ واحد.
* H_5: در صورت حذف H_1 مقدار ۰ می‌گیرد؛ هزینه خرید رأی: ۱ واحد.
* H_6: در صورت حذف H_1 مقدار ۱ می‌گیرد؛ هزینه خرید رأی: ۲ واحد.

– پیشنهاد بهینه H_1: H_1 دو رأی ارزان‌تر را می‌خرد: H_3 و H_5 هر کدام با ۱ واحد. پیشنهاد: $(s_1, s_2, s_3, s_4, s_5, s_6) = (98, 0, 1, 0, 1, 0)$.

– رأی‌گیری:
* H_1: بله (بقا + ۹۸ سود).
* H_2: خیر (۰ در مقابل ۹۸).
* H_3: بله (۱ در مقابل ۰).
* H_4: خیر (۰ در مقابل ۱).
* H_5: بله (۱ در مقابل ۰).
* H_6: خیر (۰ در مقابل ۱).

– نتیجه: پیشنهاد با ۳ رأی موافق از ۶ رأی تصویب می‌شود. توزیع نهایی: $(98, 0, 1, 0, 1, 0)$.

در تحلیل تأثیر شرایط اضافی:

- ترجیح رئیس قوی‌تر: در محاسبات بالا، پیشنهادها به گونه‌ای بودند که برای رأی‌دهندگان کلیدی، سهم پیشنهادی دقیقاً برابر یا بیشتر از سهم مورد انتظارشان در صورت حذف رئیس بود. شرط ترجیح رئیس قوی‌تر تنها زمانی مؤثر است که یک عضو بین دو پیشنهاد با سهم یکسان مردد باشد؛ در استراتژی بهینه محاسبه شده، چنین وضعیتی پیش نیامد.
- رأی‌دهی استراتژیک: تحلیل Backward Induction به طور خودکار این را در نظر می‌گیرد. هر عضو با مقایسه سهم فعلی با سهم محتمل در آینده (اگر رئیس فعلی حذف شود) تصمیم می‌گیرد.
- اتحادها: اتحادها می‌توانند محاسبات را تغییر دهند. مثلاً در $N = 5$، اگر H_3 و H_4 متحد باشند و H_1 پیشنهاد $(98, 0, 1, 0, 1)$ بدهد. H_3 (با دریافت ۱) باید به نفع H_4 (که ۰ می‌گیرد ولی در صورت حذف H_1 مقدار ۱ می‌گیرد) رأی دهد؟ طبق شرط «مگر اینکه منفعت شخصی به طور قابل توجهی به خطر بیفتد»، H_3 احتمالاً رأی مثبت می‌دهد ($1 > 0$). اما اگر اتحاد بسیار قوی باشد یا تعریف «قابل توجه» متفاوت باشد، H_3 ممکن است رأی منفی دهد تا به H_4 کمک کند؛ در این صورت H_1 برای جلب رأی اتحاد H_3, H_4 ممکن است مجبور شود به هر دو سهم بدهد (مثلاً به H_3 یک واحد و به H_4 دو واحد) و یک رأی دیگر هم بخرد، که سهم خودش را کاهش می‌دهد. تحلیل دقیق نیازمند تعریف دقیق قدرت اتحادها است.

(ب) الگوریتم کلی برای پیشنهاد بهینه رئیس موقت: هدف رئیس موقت (H_L) در گروهی k نفره، ارائه پیشنهادی است که با کسب حداقل $\lceil k/2 \rceil$ رأی، بقای خود را تضمین کرده و سهم خود (s_L) را حداکثر کند. الگوریتم مبتنی بر Backward Induction به شرح زیر است:

i. شناسایی وضعیت: تعیین اعضای حاضر (k نفر)، رئیس فعلی (H_L) و حد نصاب رأی $V = \lceil k/2 \rceil$.

ii. تحلیل سناریوی شکست: محاسبه پیامد حذف H_L. این یعنی حل مسئله برای $k - 1$ عضو باقی‌مانده با رئیس جدید (نفر با بالاترین رتبه بعدی). نتیجه این تحلیل مشخص می‌کند که هر عضو H_i (به جز H_L) در صورت حذف H_L چه سهمی E_i (Expected payoff) کسب خواهد کرد. این محاسبه نیازمند اجرای بازگشتی همین الگوریتم برای $k - 1$ است (پایه بازگشت $k = 1$ است).

iii. تعیین هزینه خرید رأی: برای هر عضو H_i ($i \neq L$):
- اگر $E_i = 0$ باشد، حداقل سهم لازم برای جلب رأی او $s_i^* = 1$ است.
- اگر $E_i > 0$ باشد، حداقل سهم لازم برای جلب رأی او $s_i^* = E_i + 1$ است (یا اگر سهم‌ها پیوسته باشند، $E_i + \epsilon$). در عمل با واحدهای گسسته، $E_i + 1$ در نظر گرفته می‌شود.

(توجه: این هزینه ممکن است تحت تأثیر اتحادها تغییر کند).

iv. انتخاب رأی‌دهندگان: H_L به $V - ۱$ رأی دیگر نیاز دارد (چون رأی خودش را دارد). او باید $V - ۱$ نفر از اعضای دیگر را با کمترین هزینه کل انتخاب کند؛ یعنی اعضایی را انتخاب کند که کمترین مقدار s_i^* را دارند.

v. فرموله کردن پیشنهاد: H_L به $V - ۱$ عضو منتخب، سهم s_i^* مربوط به آن‌ها را پیشنهاد می‌دهد. به بقیه اعضای غیر از خود، سهم ۰ پیشنهاد می‌دهد. سهم خودش برابر خواهد بود با $s_L = ۱۰۰ - \sum_{i \in \text{Selected}} s_i^*$.

vi. بررسی اتحادها (اختیاری ولی مهم): اگر اعضای منتخب شامل بخشی از یک اتحاد باشند، آیا سایر اعضای اتحاد که سهمی نمی‌گیرند (یا کمتر از انتظارشان می‌گیرند) رأی منفی می‌دهند و باعث شکست اتحاد می‌شوند؟ اگر بله، H_L ممکن است مجبور شود ترکیب دیگری از رأی‌دهندگان را انتخاب کند یا به تمام اعضای کلیدی یک اتحاد سهم بدهد که ممکن است سهم خودش را کاهش دهد. انتخاب بهینه باید این پویایی‌ها را در نظر بگیرد.

vii. ارائه پیشنهاد: H_L پیشنهاد نهایی محاسبه شده را ارائه می‌دهد.

این الگوریتم، با فرض عقلانیت کامل و اطلاعات کامل، بهترین پیشنهاد ممکن را برای بقا و حداکثرسازی سود رئیس فعلی ارائه می‌دهد.

(ج) راهکارهای مقاومت اعضای ضعیف‌تر در برابر حذف شدن: اعضای ضعیف‌تر (آن‌هایی که رتبه پایین‌تری دارند، مانند $\ldots, H_{N-۱}, H_N$) در این بازی در معرض خطر حذف یا گرفتن سهم صفر هستند، به خصوص اگر تعداد اعضا زیاد باشد و رأی آن‌ها برای تشکیل اکثریت لازم نباشد. با این حال، راهکارهایی برای مقاومت وجود دارد:

i. تشکیل اتحاد (ائتلاف): چند عضو ضعیف می‌توانند با هم متحد شوند و به صورت یک بلوک رأی دهند. اگر تعداد آن‌ها به اندازه‌ای باشد که بتوانند در رأی‌گیری تأثیرگذار باشند (مثلاً رأی آن‌ها برای رسیدن رئیس به حد نصاب لازم باشد، یا بتوانند با رأی منفی هماهنگ، رئیس را حذف کنند)، قدرت چانه‌زنی پیدا می‌کنند؛ رئیس ممکن است مجبور شود برای جلب رأی بلوک، به همه یا نمایندگان آن‌ها سهم بدهد. همچنین یک عضو ضعیف می‌تواند با یک یا چند عضو قوی‌تر (که رقیب رئیس فعلی هستند) متحد شود. این اتحاد می‌تواند با هدف حذف رئیس فعلی و به قدرت رساندن عضو قوی‌ترِ متحد شکل بگیرد، به این امید که عضو ضعیف در توزیع سود بعدی سهمی کسب کند.

ii. استفاده از نقش رأی کلیدی (Pivotal Vote): حتی یک عضو ضعیف به تنهایی، اگر رأیش برای رسیدن رئیس به حد نصاب $V = \lceil k/۲ \rceil$ حیاتی باشد (یعنی بدون رأی او، تعداد آرا $V - ۱$ شود)، می‌تواند قدرت چانه‌زنی قابل توجهی داشته باشد. او می‌تواند در ازای رأی مثبت، سهم بیشتری طلب کند تا جایی که برای رئیس همچنان دادن آن سهم بهتر از حذف شدن باشد.

iii. تهدید معتبر به رأی منفی استراتژیک: اگر یک عضو ضعیف بتواند به طور معتبر نشان دهد که حذف رئیس فعلی، موقعیت او را در آینده بهبود می‌بخشد (مثلاً شانس رئیس شدن خودش را افزایش می‌دهد یا باعث می‌شود رئیس بعدی برای جلب رأی او مجبور به دادن سهم بیشتری شود)، ممکن است بتواند رئیس فعلی را متقاعد کند که برای جلوگیری از این سناریو، سهم بیشتری به او بدهد. این تهدید زمانی معتبرتر است که تحلیل Backward Induction واقعاً نشان دهد که حذف رئیس فعلی به نفع عضو ضعیف است.

iv. ایجاد عدم قطعیت یا استفاده از شهرت: در دنیای واقعی (خارج از مدل ساده)، یک عضو ضعیف ممکن است با ایجاد شهرت غیرقابل پیش‌بینی بودن یا تمایل به انتقام (حتی اگر غیرعقلانی به نظر برسد)، دیگران را مجبور به دادن سهم به خود کند تا از رفتارهای مخرب او جلوگیری شود. البته این خارج از مفروضات استاندارد عقلانیت کامل است.

نتیجه‌گیری برای بخش مقاومت: اعضای ضعیف‌تر راهکارهایی برای مقاومت دارند، اما موفقیت آن‌ها بستگی به تعداد اعضا، قوانین دقیق رأی‌گیری، توانایی تشکیل اتحادهای پایدار و موقعیت استراتژیک

آن‌ها در هر مرحله از بازی دارد. به‌هرحال، راهکارهای اصلی آن‌ها از توانایی تشکیل ائتلاف‌های رأی‌دهی و داشتن رأی کلیدی ناشی می‌شود.

۶. (۲۰ نمره)

مسئله‌ی N Queens را در نظر بگیرید. در این مسئله باید N وزیر را در یک صفحه‌ی N x N شطرنج قرار دهید طوری که هیچ یک از وزیرها یکدیگر را تهدید نکنند.

(آ) یک مسئله‌ی CSP با سه چیز تعریف می‌شود. آنها را نام ببرید. سپس مسئله‌ی N Queens را در قالب یک مسئله‌ی Binary CSP مدل کنید که در آن اندازه‌ی دامنه‌ی هر متغیر n است.

(ب) مسئله‌ی ۴ وزیر را در نظر بگیرید. آقای ع.م.۱ می‌گوید در مدل‌سازی او، تعداد Complete Assignmentها ۱۶۴ است. امّا آقای ع.م.۲ می‌گوید این عدد در مدل‌سازی او ۲۵۶ است. حدس می‌زنید هر کدام از آنها مسئله‌ی ۴ وزیر را به چه مسئله‌ی CSP ای تبدیل کرده باشد؟

(ج) مسئله‌ی ۶ وزیر را با مدل‌سازی‌ای که کردید با استفاده از Backtracking و Forward Checking و MRV Heuristic حل کنید. نیازی به توضیح نیست، صرفا یک جا که از Forward Checking استفاده کردید را مشخص کنید (برای MRV هم همچنین). می‌توانید پاسخ خود را با اسکرین‌شات از این سایت بنویسید.

حل.

(آ) یک مسئله‌ی CSP با متغیرهایش، دامنه‌های متغیرهایش، و قیدهایی که باید روی متغیرهایش اعمال شوند تعریف می‌شود. یک مدل‌سازی خوب مسئله‌ی N Queens را می‌نویسیم:

Variables: $R_1, R_2, \ldots, R_n$

Variable R_i would denote the row of the queen that is in the i-th column.

Domain (for every R_i): $\{1, 2, \ldots, n\}$

Constraints:

$\forall ۱ \leq i < j \leq n:$

$$R_i \neq R_j \text{ (Row constraints)}$$

$$j - i \neq |R_j - R_i| \text{ (Diagonal constraints)}$$

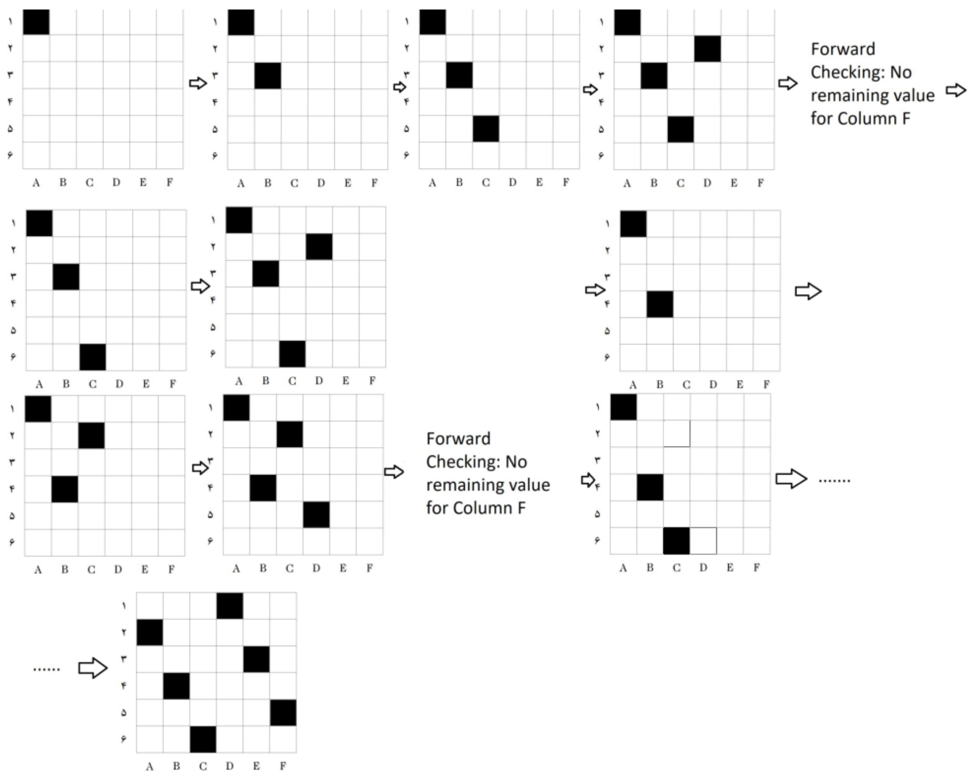
(ب) این بخش از مسئله دچار تغییر کوچکی شده و برای همین به پاسخ‌های منطقی‌ای که با پاسخنامه فرق کنند نیز نمره تعلّق می‌گیرد.

در مدل‌سازی‌ای که ما در بخش الف انجام دادیم، گفتیم چون در جواب مسئله در هر ستون دقیقاً یک وزیر خواهد بود، پس به ازای هر ستون یک متغیر تعریف می‌کنیم که شماره سطر وزیر در آن ستون را تعیین کند و سپس شروط سطرها و قطرها را در قیود مسئله قرار می‌دهیم. با اینکار دیگر لازم نبود شروطی

مربوط به ستون‌ها در قیود مسئله ذکر کنیم. دیدید که n متغیر هر یک با اندازه‌ی دامنه‌ی n داشتیم، پس به n^n طریق مختلف می‌توانیم به متغیرها مقدار بدهیم.

در مقابل، اگر هر متغیر را مکان وزیر در یکی از $n^۲$ خانه‌ی جدول (به جای یکی از n خانه‌ی یکی از ستون‌ها) تعریف می‌کردیم، آنگاه باید شرط مربوط به ستون‌ها را نیز به قیود مسئله اضافه می‌کردیم و همچنین متغیرها را می‌توانستیم به $(n^۲)^n$ حالت مقداردهی کنیم.

(ج) شکل زیر را ببینید. قابل ذکر است MRV هم در تمام مراحل رعایت شده است.



شکل ۳: حل با Backtracking