تمرین چهارم هوش مصنوعی

امیرحسین رجبی (۹۸۱۳۰۱۳)

۱۲ اردیبهشت ۱۴۰۱

سوال اول

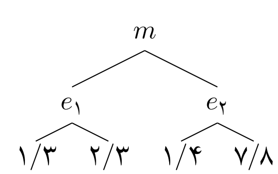
در بازی‌های مجموع غیر صفر هر بازیکن عایدی [1] جداگانه دارد و به هر نود درخت یک n-تایی نسبت داده می‌شود که هر مولفه عایدی بازیکن نظیر را نشان می‌دهد و هر بازیکن به دنبال بیشینه کردن عایدی خودش است. (به عنوان مثال عایدی یک نود می‌تواند (x, y) باشد در حالی که در بازی‌های مجموع صفر به صورت $(x, -x)$ است.) اگر عایدی بازیکنان توسط دو تابع جداگانه مشخص شود، هر بازیکن در درخت مربوط به مولفه خود (در نوبت خود) در حال انتخاب راس با عایدی بیشینه برای خودش است. ممکن است در این میان با بازیکن روبرو همکاری یا رقابت کند؛ ولی با استراتژی بیان شده این اتفاق به صورت عمدی نخواهد بود. در نتیجه اجرای آلفا-بتا روی مقادیر مولفه مربوط به یک بازیکن جواب معتبری نمی‌دهد؛ زیرا بازیکنان با هم رقابت نمی‌کنند و کاهش عایدی یک بازیکن به معنی افزایش عایدی دیگری نیست.

سوال دوم

آ) در درخت max در هر نود در حال انتخاب فرزند با عایدی بیشینه هستیم. و برگ درخت با بیشترین مقدار به عنوان عایدی بازی در نظر گرفته می‌شود. هرس نمی‌توان انجام داد زیرا نود با بیشترین مقدار در هر شاخه‌ای می‌تواند باشد و همه شاخه‌ها باید بررسی شوند. اگر زودهنگام هرس کنیم و نودی با مقدرا بزرگتر بعدا دیده شود دچار مشکل می‌شویم.

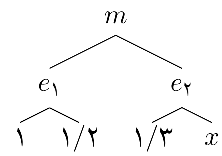
در درخت expectimax ماجرا کمی متفاوت است. نمی‌توان عملیات هرس آلفا-بتا را انجام داد چرا که عایدی راس chance از امید ریاضی فرزندان مشخص می‌شود و به دلیل کران دار نبودن عایدی، همه فرزندان باید بررسی شوند. (در صورت هرس زود هنگام و وجود راسی با عایدی بسیار بزرگ در siblingهای بعدی، ورق در امید ریاضی فرزندان می‌تواند برگردد!)

ب) برای درخت max وضعیت مانند گذشته است چرا که از کران دار نبودن مقادیر رئوس و برگ‌ها استفاده نکردیم. تنها حالت استثنا این است که مقدار یک راس ۱ باشد. در این صورت نیازی به بررسی siblingها نخواهیم داشت و می‌توان این شاخه را هرس کرد و عایدی این شاخه برابر ۱ خواهد بود. برای درخت expectimax نیز مانند مورد قبل مثال نقض وجود دارد. درخت زیر را در نظر بگیرید. احتمال انتخاب هر فرزند هر راس chance برابر $\frac{1}{2}$ در نظر می‌گیریم. شاخه e_2 به دلیل $\frac{1}{4} < \frac{1}{2} \times \frac{1}{3} + \frac{1}{2} \times \frac{2}{3} = \frac{1}{2}$ هرس می‌شود در حالی که بهین است. البته



در بعضی حالات هرس ممکن است. در درخت زیر احتمال انتخاب هر فرزند راس chance برابر $\frac{1}{2}$ است. امید فرزندان e_1 برابر $\frac{3}{4}$ است. از طرفی فرزند چپ e_2 برابر $\frac{1}{3}$ است و چون $\frac{1}{3} < \frac{3}{4}$ طبق الگوریتم شاخه e_2 هرس می‌شود و مشکلی نیز ایجاد نمی‌شود. چرا که امید فرزندان e_2 برابر است با $\frac{1}{2} \times \frac{1}{3} + \frac{1}{2} \times x = \frac{1}{6} + \frac{x}{2} < \frac{1}{6} + \frac{1}{2} = \frac{2}{3} < \frac{3}{4}$.

[1] payoff



سوال سوم

اگر بازیکن min بهینه بازی نکند همچنان بازیکن max حداقل همان عایدی را خواهد داشت که min بهینه بازی کند ولی در این صورت ممکن است از ماجرا بهره کامل نبرد؛ یعنی بتواند بهتر از عایدی که minimax ارائه می‌دهد را نصیب خودش کند. در این حالت بهتر است از expectimax بهره برد تا به کمک امید ریاضی و احتمالی فرض کردن حریف عایدی بیشتری کسب کرد.

سوال چهارم

آ) اگر $\alpha \models (\beta \wedge \gamma)$ یعنی $M(\alpha) \subseteq M(\beta \wedge \gamma)$. اما به سادگی می‌توان دید که $M(\beta \wedge \gamma) = M(\beta) \cap M(\gamma)$. زیرا هر مدلی که هر دو جمله β و γ در آن درست باشند در اشتراک مدل هرکدام قرار دارد و برعکس. پس $M(\alpha) \subseteq M(\beta) \cap M(\gamma)$ و در نتیجه $M(\alpha) \subseteq M(\beta)$ و همچنین $M(\alpha) \subseteq M(\gamma)$ یا معادلا $\alpha \models \beta$ و $\alpha \models \gamma$.

ب) اگر $\alpha \models (\beta \vee \gamma)$ یعنی $M(\alpha) \subseteq M(\beta \vee \gamma)$. اما $M(\beta \vee \gamma) = M(\beta) \cup M(\gamma)$. چرا که هر مدلی که در آن حداقل یکی از دو جمله β و γ درست باشند، عضوی از مدل β یا γ خواهد بود و برعکس. پس $M(\alpha) \subseteq M(\beta) \cup M(\gamma)$ و در نتیجه $M(\alpha) \subseteq M(\beta)$ یا (منطقی) $M(\alpha) \subseteq M(\gamma)$. معادلا داریم: $\alpha \models \beta$ یا (منطقی) $\alpha \models \gamma$.

سوال پنجم

ابتدا همه جملات KB را به فرم نرمال عطفی [۲] تبدیل می‌کنیم. خواهیم داشت:

$$A \iff (B \vee E) \equiv (\neg A \vee B \vee E) \wedge \Big(\neg(B \vee E) \vee A\Big) \equiv (\neg A \vee B \vee E) \wedge (\neg B \vee A) \wedge (\neg E \vee A) \qquad (S_{۱})$$

$$E \implies D \equiv \neg E \vee D \qquad (S_{۲})$$

$$C \wedge F \implies \neg B \equiv \neg C \vee \neg F \vee \neg B \qquad (S_{۳})$$

$$E \implies B \equiv \neg E \vee B \qquad (S_{۴})$$

$$B \implies F \equiv \neg B \vee F \qquad (S_{۵})$$

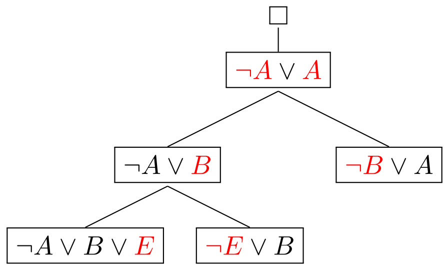
$$B \implies C \equiv \neg B \vee C \qquad (S_{۶})$$

اکنون سعی می‌کنیم به کمک قواعد resolution نشان دهیم $KB \models \neg A \wedge \neg B$ یا معادلا با فرض $KB \wedge \neg(\neg A \wedge \neg B) \equiv KB \wedge (A \vee B)$ به تناقض برسیم. پس جملات زیر را داریم:

$$\boxed{A \vee B}\ \boxed{\neg B \vee C}\ \boxed{\neg B \vee F}\ \boxed{\neg E \vee B}\ \boxed{\neg C \vee \neg F \vee \neg B}\ \boxed{\neg E \vee D}\ \boxed{\neg A \vee B \vee E}\ \boxed{\neg B \vee A}\ \boxed{\neg E \vee A}$$

درخت زیر نحوه استدلال از جملات بالا را از برگ‌ها به سمت ریشه درخت نشان می‌دهد. با توجه به درخت به جمله تهی رسیده‌ایم و از $A \vee B$ نیز استفاده نکردیم. پس KB فعلی خود دارای تناقض است و $M(KB) = \varnothing$. پس $M(KB) \subseteq M(\neg A \wedge \neg B)$ و حکم ثابت می‌شود.

[2] Conjunctive Normal Form (CNF)



گزارش پیاده سازی

با اجرای فایل `main.py` می‌توان برنامه را اجرا کرد. برنامه از چهار کلاس `TicTacToe`، `Player`، `Agent` و `Human` تشکیل شده که دو کلاس `Agent` و `Human` از `Player` ارث برده و دو نوع پیاده سازی بازیکن هستند. کلاس `TicTacToe` به کمک تابع `run()` بازی را شبیه سازی کرده و به نوبت تابع `move(state)` هر یک از بازیکنان را با ورودی دادن وضعیت بازی به آن‌ها اجرا می‌کند. (نتیجه بازی در فیلد `winner` قرار می‌گیرد و در صورت `None` بودن به معنی تساوی است و در غیر این صورت شی متناظر بازیکن برنده برگردانده می‌شود.) کلاس `Human` این تابع رو اینگونه پیاده سازی کرده که ورودی از کاربر گرفته می‌شود و وضعیت بازی تغییر داده می‌شود. در مقابل کلاس `Agent` از الگوریتم minimax استفاده می‌کند. بنابراین می‌توان هر دو نوع حریفی را مقابل یک دیگر قرار داد و بازی کرد. (مثلا در صورت قرار دادن کامپیوتر مقابل بازی همواره به تساوی می‌کشد چرا که هر دو از استراتژی نباخت استفاده می‌کنند.) وضعیت‌ها به صورت یک آرایه به طول ۹ نگه داری می‌شوند و هر درایه یکی از سه مقدار `x`، `o` یا `-` دارد و تابع استاتیک `status(state)` از کلاس `TicTacToe` با ورودی گرفتن یک وضعیت مشخص می‌کند که آیا `x` برده است یا `o` برده است یا بازی تساوی شده است یا اصلا تمام نشده است. به کمک این تابع کلیدی، تابع `is_finished() -> bool` مشخص می‌کند بازی تمام شده است یا خیر و در صورت پایان بازی، برنده را در فیلد `winner` قرار می‌دهد. دوباره به کمک تابع `status()`، تابع `utility(state)` مقداری بین ۰، ۱ و −۱ برمی‌گرداند. (برای بهینه کردن تعداد فراخوانی‌ها در صورت عدم اتمام بازی `None` برمی‌گرداند.) در نهایت تابع `minimax(state, player_type, alpha, beta)` به کمک `utility()`، مقدار عایدی وضعیت فعلی و بهترین حرکت را برمی‌گرداند. (تعداد فراخوانی‌های این تابع در یک فیلد ذخیره شده و قبل از هر حرکت توسط `move()` صفر می‌شود و بعد از مشخص شدن بهترین حرکت به کاربر نمایش داده می‌شود.) شایان ذکر است که `minimax()` وضعیت فعلی را دست نخورده تحویل می‌دهد و در نتیجه دائما نیازی به ساخت آرایه به طول ۹ ندارد و در مصرفه حافظه صرفه جویی می‌شود. همچنین هرس به روش آلفا–بتا تنها زمانی انجام می‌شود که هنگام ساخت `Agent` ذکر شود. (به صورت پیش فرض هرش انجام می‌شود.) مقایسه تعداد رئوس بررسی شده توسط مینی ماکس عادی با مینی ماکس هرس شده هیجان انگیز است. عامل برای اولین حرکت با مینی ماکس عادی حدود ۵۴۹۹۴۶ وضعیت را بررسی می‌کند. (کافی است عامل را به صورت `agent = Agent(..., alpha_beta_pruning=False)` بسازید و هنگام ساخت بازی کامپیوتر را نفر اول قرار دهید.) اما با فعال کردن آن تنها حدود ۱۸۲۹۷ وضعیت بررسی می‌شود!