گزارش تمرین
کامپیوتری دوم

ترم پائیز 1403

درس هوش مصنوعی

امیرمرتضی رضائی - 810003004

قسمت اول، الگوریتم ژنتیک:

کروموزوم‌ها به‌صورت تصاویر متشکل از ۵۰ مثلث در نظر گرفته‌شدند. جمعیت نیز متشکل از ۶۰ کروموزوم (تصاویر ساخته‌شده با مثلث‌ها) می‌باشد.
در هر بار تکرار الگوریتم، ابتدا کروموزم‌های موجود در جمعیت، به‌ترتیب fitness مرتب شده، و سپس نیمی از جمعیت که دارای fitness بهتری هستند، به جمعیت بعدی منتقل می‌شوند. باقی دیگر جمعیت جدید نیز، با انجام عملیات crossover روی کروموزوم‌های جمعیت قبلی بدست می‌آیند. عملیات crossover نیز بدین‌صورت انجام می‌پذیرد که با احتمال 0.65 برای هر یک از مثلث‌های هر دو تا از کروموزوم‌های والد، آن‌ها را با هم جابجا کرده و فرزندان حاصل را به جمعیت جدید اضافه می‌کنیم. در نهایت نیز عملیات mutation بدین صورت انجام می‌شود که در هریک از کروموزم‌ها، دو مثلث به صورت رندوم انتخاب شده که مختصات یکی و یکی از کانال های رنگ دیگری را تغییر می‌دهیم. نتایج بازسازی سه نمونه از تصاویر داده شده، به پیوست ارسال شده‌است.
واضح است که با افزایش تعداد مثلث‌ها و همچنین افزایش جمعیت، در تکرار های کمتری به فیتنس بسیار بهتری دست پیدا خواهیم کرد.

پاسخ به سوالات:

۱- در هر مثلث، برای هر راس آن به‌اندازه 100×100 مقدار ممکن وجود دارد؛ همچنین برای هر یک از پارامترهای رنگ آن نیز ۲۵۶ حالت وجود دارد. پس فضای حالت برابر است با:

$$100^2 \times 256^4$$

۲- یک روش این است که نرخ‌های انتخاب، crossover و mutation در طول روند الگوریتم ژنتیک به‌صورت پویا تغییر کنند، در ابتدای الگوریتم، برای جلوگیری از گیر افتادن در مینیمم‌های محلی، نرخ crossover و mutation می‌تواند بالاتر باشد تا تنوع بیشتری در جمعیت ایجاد شود. سپس با نزدیک شدن به بهینه‌سازی، نرخ‌ها کاهش پیدا کنند تا بهترین ویژگی‌ها از طریق عملیات‌های دقیق‌تر ترکیب شوند. این کار اجازه می‌دهد که الگوریتم در مراحل ابتدایی فضای جستجو را به طور گسترده‌ای کاوش کند و در مراحل بعدی روی بهینه‌سازی دقیق‌تر و متمرکزتر تمرکز کند.

یکی از عملیات‌های مهم در الگوریتم‌های ژنتیک mutation است که به تنوع جمعیت کمک می‌کند و از گیر افتادن در مینیمم‌های محلی جلوگیری می‌کند. اما اگر جهش به‌طور تصادفی و بدون در نظر گرفتن شرایط محیطی انجام شود، می‌تواند باعث خراب شدن راه‌حل‌ها و کند شدن همگرایی شود. استفاده هوشمندانه از mutation که به‌طور هدفمند و بر اساس تجزیه و تحلیل ویژگی‌های جمعیت انجام شود می‌تواند باعث سریعتر همگرا شدن الگوریتم گردد. برای مثال، می‌توان جهش را فقط در نواحی خاصی از جمعیت که به نظر می‌رسد بهینه نیستند، انجام داد.

۳- در الگوریتم‌های ژنتیک معمولی، بعد از هر نسل، بهترین افراد به‌طور تصادفی یا بر اساس انتخاب‌های انجام‌شده در نسل بعدی قرار می‌گیرند. اما در الگوریتم‌های مبتنی بر الیتیزم، بهترین فرد (یا افراد) در هر نسل به نسل بعدی انتقال داده می‌شوند بدون اینکه تحت تاثیر عملیات‌های انتخاب، کراس‌اوور یا جهش قرار بگیرند. در واقع حداقل یک نفر (یا چند نفر) از بهترین افراد نسل قبلی به نسل بعدی به‌طور مستقیم منتقل می‌شوند. این کار باعث می‌شود که هیچ‌گاه افراد خوب از دست نروند و همگرایی سریع‌تر صورت گیرد.

یکی از روش‌های معمول انتخاب، آن است که هر فرد در جمعیت به تناسب توانایی‌هایش (یا تابع هدف) احتمال انتخاب شدن دارد. اما در روش انتخاب گروهی، یک زیرمجموعه تصادفی از افراد جمعیت انتخاب می‌شود و از این بین بهترین فرد انتخاب می‌شود. این فرآیند باعث می‌شود که افراد با کیفیت بالاتر به احتمال بیشتری انتخاب شوند، اما همچنان تنوع جمعیت حفظ شود. کاهش احتمال گیر افتادن در مینیمم‌های محلی و تسریع همگرایی به سمت نواحی بهتر فضای جستجو از مزایای این روش هستند.

قسمت دوم، GAME :

بخش‌های مربوط به پیاده سازی الگوریتم minimax ساده و همراه با alpha-beta pruning طبق سودوکدهای بیان شده در کلاس پیاده‌سازی شد.
نتایج بدست‌آمده پس از 100 بار اجرای برنامه با هر یک از عمق‌های 1 تا 4 در دوحالت عادی و با هرس کردن، به شرح زیر می‌باشد:

```
Depth: 1 | Pruning: Enabled -> User Wins:  68, CPU Wins: 31, Ties:  1, Time: 2.80s
Depth: 2 | Pruning: Enabled -> User Wins:  96, CPU Wins:  4, Ties:  0, Time: 9.38s
Depth: 3 | Pruning: Enabled -> User Wins:  98, CPU Wins:  2, Ties:  0, Time: 48.68s
Depth: 1 | Pruning: Disabled -> User Wins:  67, CPU Wins: 33, Ties:  0, Time: 11.36s
Depth: 2 | Pruning: Disabled -> User Wins:  95, CPU Wins:  5, Ties:  0, Time: 49.89s
Depth: 3 | Pruning: Disabled -> User Wins:  98, CPU Wins:  2, Ties:  0, Time: 191.58s
```

پاسخ به سوالات:

۱- همان‌طور که انتظار داشتیم، با افزایش عمق جستجو در هر یک از روش‌ها، درصد برد بازیکن افزایش پیدا می‌کند؛ هر چند این کار باعث افزایش مدت جستجو نیز می‌شود. همچنین در حالت pruning، مدت زمان جستجو به طور قابل توجه نسبت به حالت عادی کاهش یافته است.

۲- در این صورت باید به این گونه عمل کرد که در گره‌های min و max، فرزندان هر گره را به ترتیب به‌صورت کوچکتر به بزرگترر و بزرگتر به کوچکتر قرار داد. اگر بتوان همچین کاری کرد، تعداد هرس‌ها بسبار افزایش یافته و مدت زمان جستجو بسیار کاهش پیدا می‌کند. اما این کار مستلزم آن است که از قبل تمام فرزندان هر گره را به صورت کامل بررسی کرده باشیم! که گویی یکبار روی تمام گره های مسئله جستجو را انجام داده‌ایم. پس عملا این کار غیرممکن است.

۳- در ابتدا که بازی تازه شروع شده و تمام ستون‌ها آزاد هستند، branching factor بالاست، زیرا بازیکن می‌تواند هر کدام از ستون‌ها را انتخاب کند.در وسط بازی، تعداد ستون‌های آزاد کاهش می‌یابد، چون مهره‌ها در حال پر شدن هستند. در پایان بازی، زمانی که اکثر یا تمام ستون‌ها پر شده‌اند، branching factor به حداقل می‌رسد، زیرا بازیکن تنها می‌تواند در ستون‌های خاصی حرکت کند. با پیشرفت بازی و پر شدن بیشتر ستون‌ها، تعداد مکان‌های ممکن برای هر حرکت کاهش می‌یابد و در نتیجه **branching factor** کاهش پیدا می‌کند. این به این معنی است که الگوریتم در مراحل ابتدایی بازی با تعداد زیادی وضعیت ممکن روبه‌رو است که باید بررسی کند، ولی در مراحل بعدی، این تعداد وضعیت‌ها کاهش می‌یابد و الگوریتم می‌تواند به سرعت بیشتر و بهینه‌تر عمل کند.

۴- زمانی که الگوریتم متوجه می‌شود که بررسی یک گره دیگر مفید نیست (چون هیچ تأثیری بر نتیجه نهایی نخواهد داشت)، دیگر آن گره‌ها را بررسی نمی‌کند. این عمل کاهش فضای جستجو و در نتیجه کاهش زمان محاسباتی را به همراه دارد. در واقع، هرس کردن باعث می‌شود که عمق جستجو کوتاه‌تر شود و بخش‌هایی از درخت که نتیجه مشابهی خواهند داشت، نادیده گرفته شوند. این امر به الگوریتم اجازه می‌دهد که از وضعیت‌های غیرضروری صرف‌نظر کند و زمان اجرای آن را به کاهش دهد.

۵- در minimax، ما هیچ ریسکی را انجام نمی‌دهیم و در هر مرحله گره‌هایی انتخاب می‌شوند که مطمئن باشییم به نفعمان است.(کاملا adversarial). اما در حالتی که رقیب به صورت رندوم عمل می‌کند (یعنی احتمال اشتباه دارد)، بهتر است که گاهی اوقات برای کسب امتیازهای بالاتر، کمی ریسک انجام داده و کاری را انجام دهیم که شاید در نگاه اول به سودمان نباشد، اما در آینده با یک احتمالی ممکن است به سودمان باشد! در این حالت می‌توانیم از الگوریتم expectimax استفاده کنیم.