مسئله کوله پشتی با Memetic Algorithm

امیرحسین انتظاری ۶۱۰۳۹۹۱۰۳

امیرعلی امینی ۶۱۰۳۹۹۱۰۲

**تعریف مسئله:**

مسئله کوله‌پشتی ۰-۱ یک مسئله بهینه‌سازی در علم کامپیوتر است که در آن باید مجموعه‌ای از اشیاء با ارزش و وزن داده شده را درون یک کوله‌پشتی با ظرفیت مشخص قرار داد. هر شیء دارای وزن و ارزشی است و باید تعیین کنیم کدام شیء‌ها را درون کوله‌پشتی قرار دهیم تا مجموعه‌ای با بیشترین ارزش را داشته باشیم، در حالی که وزن کل شیء‌ها ظرفیت کوله‌پشتی را نباید بیشتر کند.

در این مسئله، هر شیء را می‌توان به دو صورت گرفته یا نگرفته در نظر گرفت. به عبارت دیگر، مجموعه شیء‌های قابل انتخاب برای قرار دادن در کوله‌پشتی یا ۰ (شیء را نگرفتن) یا ۱ (شیء را گرفتن) است.

هدف در این مسئله انتخاب مجموعه‌ای است که مجموع ارزش شیء‌های داخل آن بیشینه شود، در حالی که وزن کلی آن مجموعه نباید بیشتر از ظرفیت کوله‌پشتی باشد. این مسئله به عنوان یک مسئله بهینه‌سازی معروف است و الگوریتم‌های مختلفی برای حل آن وجود دارد.

ما در این پروژه سعی داریم با الگوریتم ممتیک، جواب بهینه ای برای آن پیدا کنیم.

**الگوریتم ممتیک:**

االگوریتم ممتیک یا همان Memetic Algorithm را نمی‌توان یک الگوریتم ثابث فرض کرد. در واقع الگوریتم‌های ممتیک را می‌توان از دسته‌ی الگوریتم‌ها دانست که به نوعی شاخه ای از الگوریتم ژنتیک (Genetic Algorithm) هستند. در این پروژه ما به این الگوریتم‌ می پردازیم.

الگوریتم ژنتیک را به یاد بیاورید. به صورت خلاصه، این الگوریتم با تولید افراد نسل‌های مختلف و ترکیب (Crossover) و جهش (Mutation) در نسل‌ها، می‌توانست به یک حالت بهینه‌ی خوب دست پیدا کند. اما جالب این‌جاست در الگوریتم ژنتیک هر کدام از افراد (ژن‌ها) تلاشی برای بهبود خود نمی‌کنند. در واقع در الگوریتمِ ژنتیک، هر کدام از افراد (ژن‌ها) در نسل‌های مختلف فقط منتظر عملیات Crossover و Mutation هستند تا شاید بهبود حاصل شود، و نکته‌ی اصلی در الگوریتم ممتیک در این‌جا نهفته است. این الگوریتم مانند الگوریتم ژنتیک عمل می‌کند با این تفاوت که هر کدام از افراد (ژن‌ها) بعد از تولد (به وجود آمدن از نسل قبلیِ خود) به بهبود وضعیت خویش به صورت نسبی و محلی می‌پردازند. آن‌ها این کار با الگوریتم تپه‌نوردی (Hill Climbing) انجام می‌دهند.

اگر بخواهیم با مثالی شهودی الگوریتم ممتیک را متوجه شویم، به این صورت است که فرض کنید افراد مختلف (همان ژن‌ها) در جهان گسترده شده‌اند و به دنبال رشد و شکوفایی هستند. آن‌ها با عملیاتی مانند تبادل فرهنگی (Crossover در الگوریتم ژنتیک) قسمتی از فرهنگ خود را می‌دهند و قسمتی از فرهنگ دیگران را دریافت می‌کنند تا با ترکیب آن فرهنگ با فرهنگِ خود، بتوانند شاید به یک حالت بهینه‌تر و رشد و شکوفایی بیشتر دست پیدا کنند. حالا آن‌هایی که بهینه‌تر از بقیه هستند شانس بیشتری برای زنده ماندن و سپس ازدواج با بقیه‌ی بازماندگان دارند! این افرادی که زنده مانده‌اند با افراد دیگر از نقاط جغرافیاییِ دیگر ازدواج کرده و فرزندانی را به وجود می‌آورند و با این‌کار امید دارند که فرزندانشان بهتر از خودشان شوند. تا این‌جای کار الگوریتم ژنتیک بود. الگوریتم ممتیک مانند این است که فرزند متولد شده، در مکانی که قرار دارد، به دنبال بهبود و رشد و شکوفایی خویش (بدون توجه به ترکیب و جهش) باشد. مثلاً خودش به صورت حریصانه هر روز خود را بهبود دهد. سپس دوباره به چرخه‌ی ژنتیک برگردد و با عملیات Crossover و Mutation با کمک هم نسل‌های خود، فرآیند بهبود را در ادامه دهد.

مراحل اصلی الگوریتم ممتیک عبارت‌اند از:

1. ایجاد جمعیت اولیه تصادفی از افراد.

2. ارزیابی برازش کورومزوم افراد بر اساس پارامتر های تعریف شده.

3. تولید نسل بعدی با استفاده از عملیات شامل ترکیب ژنتیکی (تهیه‌ی نسل جدید از ترکیب دو یا چند جهش) و ایجاد نسل جدید.

4. تولید تعدادی از همسایه های یک جواب و پیدا کردن بهترین جواب در کل همسایگی.(local search)

5. انتخاب جهش‌هایی برای تولید نسل بعدی.

6. تکرار مراحل 2 تا 5 تا رسیدن به شرایط خاتمه مسئله یا دستیابی به جواب بهینه.

با تکرار مراحل فوق، الگوریتم ممتیک به تدریج به راه‌حل‌های بهینه نزدیک‌تر می‌شود و می‌تواند در مسائل بهینه‌سازی پیچیده و فضاهای جستجوی بزرگ عملکرد خوبی داشته باشد. الگوریتم ممتیک در مسائل مختلفی مانند یافتن کوتاه ترین مسیر، برنامه‌ریزی زمان،بهینه‌سازی تولید و … مورد استفاده قرار می‌گیرد.

**روش حل مسئله:**

برای حل مسئله کوله پشتی ۰-۱ با الگوریتم ممتیک، ابتدا باید تعیین کنیم که جواب های مسئله ما چگونه باید در قالب کورموزوم پیاده سازی شوند. یک راه پیشنهادی این است که بیایم یک جواب مسئله که شامل تعدادی شی هست را مستقیما به عنوان کورموزوم انتخاب کنیم. کوروموزوم گفته شده به صورت زیر خواهد بود:

Chromosome = [80, 82, 85, 70, 72, 70, 66, 50, 55, 25, 50, 55, 40]

که اعداد درون آرایه، وزن اشیا میباشند. روش دیگر برای نمایش راه حل در کورموزوم، استفاده از نمایش باینری است، که به نظر من روش بهتری است چرا که اگر چند شی با وزن تکراری داشته باشیم، پیاده سازی چک کردن موجود بودن آن در کوله پشتی راحت تر است. کوروموزوم گفته شده به صورت زیر خواهد بود:

Chromosome = [0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,]

هر چند بدلیل محاسبات بیشتر در این الگوریتم، بهتر است یک کلاس شی جدا برای افراد تعیین کرد، اما اینکار تاثیری روی الگوریتم مسئله ندارد.

پس از انتخاب نحوه نمایش کوروموزم، ابتدا یک جمعیت اولیه از کورموزوم ها ایجاد میکنیم. من اینکار را با کوروموزم اولیه با مقادیر ۰ انجام دادم. تعداد جمعیت اولیه نیز بسته به بزرگی ورودی مسئله تغییر میکند. سپس، مقدار فیتنس هر کورموزوم را در یک آرایه ذخیره میکنیم که خانه i ام آرایه مربوط به کورموزوم i ام از جمعیت اولیه است.

حال باید یک تابع بازترکیب(crossover) برای ترکیب دو کورموزوم با یکدیگر ارائه دهیم. من از روش بازترکیب دو نقطه ای برای تابع بازترکیب استفاده کردم. در این روش ابتدا بچه اول برابر با والد اول و بچه دوم برابر با بچه دوم میشود، سپس دو اندیس رندوم انتخاب میشود و از اندیس اول تااندیس دوم کوروزموم والد دوم به بچه اول و از اندیس اول تا اندیس دوم والد اول به بچه دوم ارث می رسد. در نهایت بچه با برازش بیشتر انتخاب می شود.

پس از بازترکیب و تولید یک بچه جدید، باید پوینت اصلی الگوریتم ممتیک، یعنی جستجوی محلی را نیز انجام دهیم. برای اینکار، ابتدا یک تابع همسایه تعریف میکنیم که یک کورموزوم را گرفته و همسایه ای رندوم از آن را تولید میکند. قبل از تعریف خود تابع، باید تعریف کنیم که همسایگی در مسئله با چگونه تعریف می شود؟ یک همسایگی از جواب موجود را میتوان اینگونه در نظر گرفت که یک شی را که درون کوله پشتی قرار داده ایم، خارج کنیم، یا شی ای که آن را درون کوله پشتی قرار نداده ایم، درون کوله پشتی بگذاریم. بدین ترتیب ما دو دسته همسایگی داریم. حال به تعریف تابع همسایه می پردازیم. برای تولید یک همسایه از جواب، میتوانیم ابتدا یک اندیس رندوم از اشیائی که برداشته نشده اند و یک اندیس رندوم از اشیائی که برداشته شده اند تولید کنیم. سپس هرکدام از این اشیا انتخاب شده را، معکوس میکنیم(اگر داخل کوله پشتی است، خارج میکنیم، اگر خارج کوله پشتی است، داخل کوله پشتی میگذاریم). حال دو همسایه از جواب را تولید کرده ایم. کدام را انتخاب کنیم؟ من برای این انتخاب بدین گونه عمل کردم که ابتدا میبینم که اگر هر دو همسایه، هردو شرط قابل قبول بودن(feasible) را برآورده یا هر دو شرط قابل قبول بودن را رد میکنند، همسایه با برازش بهتر را خروجی میدهیم. اگر هم یکی از آن ها شرط قابل قبول بودن را برآورده میکرد، فقط همان را خروجی میدهیم.

پس از تعریف تابع همسایه، باید تابع جستجوی محلی را تعریف کنیم. این تابع را اینگونه تعریف میکنیم که یک پارامتر n میگیرد و به تعداد n تا، همسایه رندوم تولید میکند که شامل خود کورمورزوم هم می شود. سپس از بین این همسایه ها، همسایه ای با بهترین برازش را خروجی می دهد. هر چند میتوانیم برای جلوگیری از پویش بیش از حد، با احتمال یک پارامتر(مثلا ۰.۲) به جای بهترین همسایه، یک همسایه رندوم خروجی دهیم.

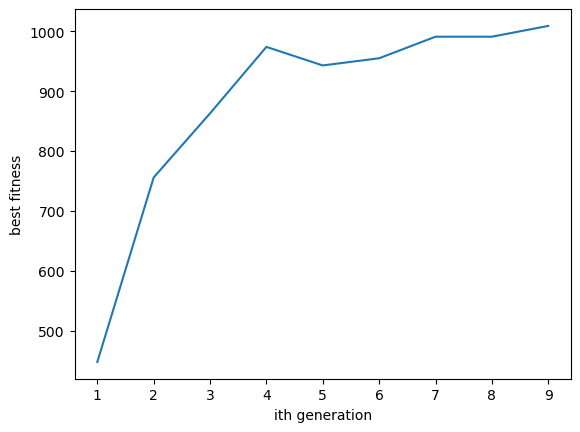
در کنار تابع بازترکیب باید تابع جهش را نیز تعریف کنیم، تا بتوانیم به الگوریتممان پویش بدهیم. از بین روش های جهش موجود، من دو روش جا به جایی و معکوس سازی بیت را انتخاب کرده ام. هر چند که روش جا به جایی برای مسئله با کورموزوم دودویی خیلی مناسب نیست و در الگوریتم نیز استفاده نشده است. در روش معکوس سازی بیت، روی کوروموزوم حرکت کرده، و با احتمال یک پارامتر تصادفی، بیتی که روی آن هستیم معکوس می شود. من با پارامتر ۰.۲ جواب های خوبی گرفتم.

حال که تابع بازترکیب و جهش را تعریف کردیم، میتوانیم به تعریف تابع انتخاب بپردازیم. از بین روش های موجود برای تابع انتخاب، من روش تورنومنت و رنک را پیاده سازی کرده ام. هر چند که روش رنک بسیار انتفاع را زیاد میکند و سریع در نقطه بهینه محلی گیر میوفتیم و در نتیجه استفاده ای از آن نکردم. در روش تورنومنت، ابتدا n عضو برای والد اول، سپس m عضو برای والد دوم انتخاب کردم، و بهترین والد ها را از هر دوی این دسته برای بازترکیب انتخاب میکنم. پس از تولید هر بچه، کوروموزوم را به تابع جهش داده و با احتمال ۰.۲ جهش میدهیم.

تفاوت اصلی این تابع انتخاب با تابع انتخاب در الگوریتم ژنتیک، این است که باید در این تابع، پس از تولید هر فرزند، نقطه بهینه محلی فرزند را نیز پیدا کنیم. همانند الگوریتم های دیگر برای این مسئله، همواره به توجه قابل قبول بودن(feasible) جواب ها میکنیم. به طوری که هر سری، چک میکنیم که آیا بچه تولید شده قابل قبول است یا خیر.

در نهایت عملیات انتخاب را تا زمان برقراری شرط مورد نظر که من برای این الگوریتم، رسیدن به جواب پیشنهادی و محدودیت نسل را انتخاب کرده ام، ادامه میدهیم.

تحلیل نمودار:

تست کیس اول:

ملاحظه میکنیم که الگوریتم برای تست کیس اول به شدت خوب عمل کرده، و در مقایسه با الگوریتم ژنتیک، خیلی سریع تر به جواب میرسد. که این موضوع احتمالا بدلیل جستجوی بهینه محلی است.

تست کیس دوم:

در تست کیس دوم نیز عملکرد به شدت بهتر از الگوریتم ژنتیک است، و در نسل های زودتری به جواب میرسیم. اما همانند ژنتیک، از فیتنس ۱۶۰۰، در مدت زمان کمی به فیتنس ۳۰۰۰ نزدیک میشویم.