دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

دانشکده ریاضی و علوم کامپیوتر

گزارش ۵ درس هوش مصنوعی

پیاده سازی حل یک مسئله تصمیم‌گیری با استفاده از ارضای محدودیت

نگارش
مهسا دیباجی
۹۶۱۳۰۱۰

استاد درس
دکتر قطعی

۳۱ فروردین ۱۴۰۰

مقدمه

سودوکو استاندارد شامل ۸۱ خانه در یک شبکه ۹ × ۹ و دارای ۹ جعبه است که هر جعبه محل تلاقی ۳ ردیف اول ، میانی یا آخر و ۳ ستون اول ، میانی یا آخر است. هر سلول ممکن است حاوی یک عدد از یک تا نه باشد و هر شماره فقط یک بار در هر سطر ، ستون و جعبه می تواند وجود داشته باشد. سودوکو با تعدادی سلول حاوی اعداد (سرنخ) شروع می شود و هدف حل سلولهای باقیمانده است. سودوکوس مناسب یک راه حل دارد. بازیکنان و محققان از طیف گسترده ای از الگوریتم های رایانه برای حل سودوکوس ، مطالعه خواص آنها و ساخت معما های جدید از جمله سودوکوس با تقارن های جالب و سایر خصوصیات استفاده می کنند. برای حل سودوکو تولید همه شبکه‌ها و انتخاب راه حل کارآمد نیست. برای به حداقل رساندن این کار ، می توانیم از روش Backtracking استفاده کنیم.

الگوریتم Backtracking

این یک روش الگوریتمی برای یافتن همه راه حل های یک مسئله محاسباتی است، به ویژه مسائل ارضای محدودیت، که به تدریج کاندیداهایی را برای راه حل‌ها ایجاد می‌کند و به محض اینکه تشخیص می‌دهد کاندیدایی نمی تواند برای یک راه حل معتبر تکمیل شود، آن را رها می کند.
در این مسئله، باید هر بلوک از سودوکو را طی کنیم و بررسی کنیم کدام رقم را می توان در این بلوک قرار داد. این عمل را تکرار کنیم تا زمانی که راه حلی پیدا کنیم. اگر نمی‌شد رقمی را در بلوک فعلی قرار داد، به بلوک قبلی می رویم و سعی می کنیم رقم دیگری را در آن بلوک قرار دهیم و راه حل بعدی را تغییر دهیم.
این اساساً بدان معناست که هر زمان به بن بست می رسیم، به انتخاب قبلی که انجام داده‌ایم برمی‌گردیم و انتخاب خود را تغییر میدهیم، به گونه‌ای که راه حل متفاوتی خواهیم داشت.
همچنین در این گزارش برای انتخاب بلوک بعدی برای مقدار دهی از دو هیوریستیک minimum-remaining-values و degree نیز استفاده شده‌است تا مدت زمان حل مسئله و حافظه بهینه‌تر شود.

پیاده سازی

حال به توضیح توابع پیاده‌سازی شده برای این الگوریتم می‌پردازیم.

- **تابع is_complete(grid)**

ورودی این تابع جدول سودوکو است و تعیین می‌کند که آیا مسئله حل شده(تمام خانه‌ها مقدار گرفته‌اند) یا خیر.

- **تابع is_consistent(grid,row,col,num)**

ورودی این تابع جدول سودوکو، مختصات یک خانه و عدد num برای قرار دادن در آن خانه است. این تابع بررسی می‌کند که عدد num در سطر، ستون و مربع ۳*۳ متناظر با این خانه وجود دارد یا خیر. اگر وجود داشت نتیجه می‌شود که عدد num را نمی‌توان در این خانه قرار داد.

- **تابع get_domain(grid,row,col)**

ورودی این تابع جدول سودوکو و مختصات خانه مورد نظر است. خروجی یک لیست از اعدادی است که می‌توان در آن خانه قرار داد. به این منظور اعداد ۱ تا ۹ را در نظر گرفته و هرکدام که در سطر، ستون

و یا مربع ۳*۳ نظیر آن خانه تکرار شده بود را حذف می‌کنیم. اعداد باقی مانده دامنه متناظر با این خانه هستند.

- **تابع mrv_domains(grid)**

در این تابع ابتدا دامنه تمام خانه‌های جدول را توسط تابع get_domain محاسبه می‌کنیم. سپس خانه یا خانه‌هایی را که دارای کمترین سایز دامنه هستند به همراه مختصات آن‌ها خروجی می‌دهیم.

- **تابع get_degree(grid,row,col)**

ورودی این تابع، جدول سودوکو و مختصات یک خانه است. در این تابع تعداد خانه‌های مقدار داده نشده در سطر و ستون و مربع ۳*۳ نظیر این خانه را محاسبه کرده و خروجی می‌دهیم.

- **تابع get_max_degree(blocks)**

ورودی این تابع لیستی از مختصات خانه‌هاست. با استفاده از تابع get_degree درجه هر خانه را محاسبه شده و خروجی تابع مختصات خانه‌ایست که بیشترین درجه را دارد.

- **تابع select_unassigned_var(grid)**

هدف این تابع انتخاب یک متغیر مناسب برای مقدار دهی است. ابتدا توسط تابع mrv_domains مختصات و دامنه خانه‌هایی که کمترین سایز دامنه را داشتند، گرفته می‌شود. اگر تنها یک خانه با این ویژگی وجود داشت مختصات و دامنه همان خانه در خروجی تابع داده می‌شود. در غیر این صورت توسط تابع get_max_degree از بین این خانه‌ها، خانه‌ای که بیشترین درجه را دارد را تعیین شده و خروجی تابع مختصات و دامنه‌ی آن خانه می‌شود.

- **تابع backtracking_search(grid)**

در نهایت با استفاده از این تابع سودوکوی ورودی حل شده و پاسخ آن به عنوان خروجی داده می‌شود.
ابتدا توسط تابع select_unassigned_var یک خانه برای مقداردهی انتخاب می‌شود. به ازای هر مقدار در دامنه‌ی این خانه، توسط تابع is_consistent بررسی می‌شود که می‌توان آن مقدار را در خانه مذکور قرار داد یا خیر. اگر این امکان وجود داشت، خانه را با این عدد مقدار دهی کرده و دوباره تابع backtracking_search فراخوانی می‌شود. در غیر این صورت، مقدار بعدی در دامنه بررسی می‌شود. این تابع به صورت بازگشتی فراخوانی می‌شود تا جایی که یک خانه را نتواند مقدار دهی کند. در این صورت به خانه قبلی که مقداردهی شده بود برگشته و مقدار دیگری را برای آن امتحان می‌کند. و یا اینکه جدول کامل حل شود(توسط تابع is_complete بررسی می‌شود) که در این صورت پاسخ را خروجی می‌دهد.

لینک colab کد پیاده سازی شده:

https://colab.research.google.com/drive/1cPZB1StRndi3nlY_UNHulXO2HNhXtn3e?usp=sharing

اجرای الگوریتم

برای اجرای الگوریتم از یک دیتاست ککل[۱] که شامل یک میلیون سودوکو و پاسخ متناظر آن‌ها بود استفاده شده‌است. الگوریتم backtracking با استفاده از دو هیوریستیک ذکر شده روی ۵۰۰ سودوکو اجرا شد. زمان کلی برای حل این تعداد سودوکو حدود ۵۰ ثانیه و زمان حل هر سودوکو به طور متوسط ۰/۱ ثانیه بود.

```
[[0, 0, 4, 0, 8, 3, 0, 0, 2],
 [0, 5, 1, 0, 0, 4, 3, 0, 0],
 [0, 0, 0, 0, 9, 6, 7, 1, 0],
 [1, 2, 0, 8, 0, 0, 0, 0, 6],
 [0, 4, 0, 0, 0, 0, 5, 0, 0],
 [8, 3, 0, 6, 0, 7, 9, 0, 0],
 [0, 6, 0, 3, 0, 9, 0, 4, 0],
 [0, 0, 7, 0, 0, 0, 2, 0, 5],
 [0, 9, 0, 0, 5, 0, 8, 0, 3]]
```

شکل ۱: یک نمونه سودوکو

```
[[9, 7, 4, 1, 8, 3, 6, 5, 2],
 [6, 5, 1, 2, 7, 4, 3, 8, 9],
 [2, 8, 3, 5, 9, 6, 7, 1, 4],
 [1, 2, 9, 8, 3, 5, 4, 7, 6],
 [7, 4, 6, 9, 1, 2, 5, 3, 8],
 [8, 3, 5, 6, 4, 7, 9, 2, 1],
 [5, 6, 8, 3, 2, 9, 1, 4, 7],
 [3, 1, 7, 4, 6, 8, 2, 9, 5],
 [4, 9, 2, 7, 5, 1, 8, 6, 3]]
```

شکل ۲: سودوکوی حل شده

مراجع

[1] https://www.kaggle.com/bryanpark/sudoku/data#

[2] https://en.wikipedia.org/wiki/Sudoku_solving_algorithms

[3] https://github.com/mahdavipanah/SudokuPyCSF

[4] https://medium.com/daily-python/solving-sudoku-puzzle-using-backtracking-in-python-daily-python-29-99a825042e