پروژه ارضای محدودیت

سید رضا فیروزی – سید علی فرخ

۱- مدل سازی بازی

برای مدل سازی بازی و تبدیل آن به یک مسئله ارضای محدودیت، هر کدام از قطب های آهن ربا (خانه های جدول) را به صورت یک متغییر در نظر گرفته ایم. مقادیری که این متغییر ها میتوانند اتخاذ کنند (دامنه متغییر ها) عبارتست از کاراکتر های + و − و × که به ترتیب نشانه قطب مثبت آهن ربا، قطب منفی آهن ربا و خالی بودن خانه مذکور است. نکته مهم در زمان مقدار دهی به هر کدام از این متغییر ها این است که با مقدار گرفتن یک متغییر، در صورتی که مقدار آن مثبت و یا منفی باشد، قطب دیگر آهن ربا در جدول بازی نیز مقدار خواهد گرفت و مقدار آن عکس مقدار متغییر فعلی خواهد بود. به این ترتیب میتوانیم مسئله دو قطبی بودن آهن ربا ها را پیاده سازی کنیم.

در این حالت بدیهی است که محدودیت های مسئله عبارتند از این اینکه اولا هیچ دو متغییر مجاوری مقدار یکسان نداشته باشند (بجز مقدار × که نشانه خالی بودن آن خانه است) و همچنین مجموع قطب های مثبت و منفی هر سطر/ستون برابر خواسته های بازی باشد.

لازم به ذکر است که در این پروژه از نوع دوم تست کیس ها استفاده میکنیم.

۲- فایل های پروژه

فایل puzzle.py شامل کلاس Puzzle است که وظیفه مدل سازی بازی را بر عهده دارد. فایل main.py نقطه شروع برنامه است و شامل کد پیاده سازی الگوریتم Backtracking نیز میباشد. فایل AC3.py شامل توابع مربوط به پیاده سازی الگوریتم Arc Consistency است و فایل FC.py مربوط به اجرای برنامه با در نظر گرفتن Forward Checking است.

لازم به ذکر است که الگوریتم AC3 و هیروئستیک MRV قابل استفاده در هر دو فایل main.py و FC.py میباشند.

۳- الگوریتم های پیاده سازی شده

- **الگوریتم Backtracking:** حالت ساده و پایه ای حل مسائل ارضای محدودیت است و در هر فراخوانی بازگشتی یک مقدار را به یک متغییر که هنوز مقداری ندارد نسبت میدهد. روش انتخاب متغییر و مقدار آن میتواند در کارایی الگوریتم برای حل مسئله تاثیر گذار باشد، به همین دلیل از هیروئیستیک های MRV و LCV برای انجام این کار استفاده میکنیم تا کارایی الگوریتم را افزایش دهند.
- **الگوریتم MRV:** یک هیروئیستیک برای انتخاب متغییر است و متغییری را که دارای بیشترین میزان محدودیت است اول انتخاب میکند، زیرا با این کار در صورت امکان رسیدن به بن بست در حل مسئله،

در کمترین زمان ممکن به این بن بست میرسیم و نیازی نیست پیمایش طولانی در درخت حالت های مسئله داشته باشیم تا به این بن بست برخورد کنیم. بنابر این مسیر های نامطلوب سریعتر از بین میروند و وقت کمتری گرفته میشود.

- **الگوریتم LCV:** یک هیروئستیک برای انتخاب مقدار هر متغییر است. این هیروئستیک مقداری را برای هر متغییر انتخاب میکند که کمترین میزان محدودیت برای سایر متغییر ها را بوجود می آورد. بدین ترتیب شانس موفقیت در ادامه مسیر حل مسئله افزایش پیدا میکند و به طور کلی احتمالا سریعتر به جواب میرسیم.
- **الگوریتم Forward Checking:** یکی از روش های ساده انتشار محدودیت در مسئله است و به این صورت عمل میکند که پس از انتساب یک مقدار به یک متغییر، همسایه های آن را نیز بررسی میکند و در صورت خالی شدن domain همسایه ها از ادامه مسیر فعلی جلوگیری میکند. استفاده از این تکنینک تا حدودی کارایی حل مسئله را افزایش میدهد.
- **الگوریتم AC3:** یک الگوریتم برای شناسایی زودهنگام تناقضات احتمالی میان متغییر هاست و backtrack کردن از مسیر فعلی حل مسئله در صورت نیاز است. این الگوریتم قادر به تشخیص برخی از تناقضات میان جفت متغییر ها است به نوعی که الگوریتم Forward Checking قادر به تشخیص آن ها نمیباشد. با توجه به این که این الگویتم در هر دور فراخوانی بازگشتی الگوریتم Backtracking اجرا میشود، کارایی و زمان اجرای آن حائز اهمیت است. علاوه بر این باید در نظر داشت که کارایی این الگوریتم به نوع مسئله و همچنین ورودی های آن مرتبط است و لزوما همیشه باعث افزایش کارایی کلی در حل مسئله نمیشود. به عنوان مثال استفاده از این الگوریتم برای حل تست کیس اول پروژه تغییر چندانی در زمان حل مسئله ایجاد نمیکند، اما استفاده از آن در تست کیس دوم باعث افزایش سرعت میشود.

لینک پروژه در گیتهاب:

https://github.com/RezaFirouzii/magnet-puzzle-csp-project