

فهرست مطالب

| | |
|---|--|
| ۱ | مدل سازی |
| ۱ | توضیحات backtrack |
| ۲ | توضیحات MRV Heuristic |
| ۲ | توضیحات LCV Heuristic |
| ۲ | توضیحات forwardcheck |
| ۲ | توضیحات ac ^۳ |
| ۳ | تاثیر افزودن ac ^۳ در حل مثال ها |
| ۳ | زمان اجرای الگوریتم با forward check |
| ۴ | زمان اجرای الگوریتم با ac ^۳ |
| ۴ | لیست کامیت های گیت هاب |

مدل سازی

من این مسئله را به این صورت پیاده کردم که هر یک خانه ی پازل به عنوان یک متغیر انتخاب شده است اما در هر بار مقدار دهی متغیر، خودش به همراه خانه ی جفتش مقدار داده میشود یعنی به نحوی هر آهنگ را یک متغیر در نظر گرفته شده است.

نکته: من در حل این مسئله از روش دوم در پیدا کردن قطب آهنگ ها استفاده کردم.

نکته: برای انجام تست میتوانید در انتهای فایل در قسمت main ابتدا مسیر فایل ورودی را در متغیر file_path ست کنید (که پیش فرض فایل input_۱_method۲ است) و همچنین میتوانید با مقدار دهی باینری به متغیر use_forwardcheck مشخص کنید آیا از forward check استفاده کند یا خیر. در صورتی که مقدار False به متغیر داده شود از ac^۳ استفاده خواهد کرد.

توضیحات backtrack

Backtrack به این صورت پیاده شده که در هر بار فراخوانی متغیری که مقدار کمتری برایش باقی مانده با mrv_heuristic انتخاب میشود و سپس دامنه ی متغیر با lcv_heuristic ترتیب دهی میشود، سپس روی دامنه ی مرتب شده for زدم و در داخل for ابتدا یک کپی از جداول table (که مقادیر هر خانه را در آن لحظه نشان میدهد) و assignment (که دامنه ی باقی مانده برای هر متغیر را نشان میدهد) گرفتم چون زمانی که یک مقدار را برای متغیر امتحان کردیم و خواستیم مقدار بعد را امتحان کنیم باید ابتدا جداول table و assignment به حالت قبل از آن مقدار دهی برگردند.

حال پس از کپی گرفتن از این دو جدول چک میشود که آیا مقدار جدید برای متغیر سازگار است یا خیر. این سازگاری به این صورت چک میشود که وقتی این مقدار را به متغیر و جفتش بدهیم همسایه هایشان قطب های ناسازگار با این مقادیر نداشته باشند.

اگر مقدار سازگار بود آن را در جدول table ست میکنیم و همچنین این مقدار را از دامنه ی assignment متغیر حذف میکنیم. پس از آن بسته به اینکه متغیر use_forwardcheck مقدار True یا False داشته باشد از forwardcheck یا ac³ استفاده میکند.

توضیحات MRV Heuristic

در mrv خانه ی مدنظر به این صورت انتخاب میشود که از بین خانه های پازل، خانه ای که کمترین مقدار در دامنه اش است و البته این متغیر هنوز مقدار داده نشده به عنوان متغیر بعدی برای مقدار دهی انتخاب میشود.

توضیحات LCV Heuristic

میدانیم که مقدار خنثی یعنی صفر هیچ محدودیتی برای متغیر های دیگر ایجاد نمیکند در نتیجه مسلماً اولین مقدار در دامنه صفر است. برای مقادیر ۱ و -۱ به این صورت عمل کردم که یک score برای مقدار ۱ و مقدار -۱ در نظر گرفتم که امتیاز این مقدار را نشان میدهد و بعد براساس این که کدام یک از این دو مقدار امتیاز بیشتری گرفته در دامنه مرتب میشوند.

اگر برای آهنبها همسایه ای باشد که در دامنه اش مثلاً مقدار ۱ دارد به این معناست که میتواند این مقدار را بگیرد و در صورتی که ما متغیر کنونی را مقدار ۱ دهیم باید مقدار ۱ را از دامنه همسایه اش حذف کنیم پس هرچه در دامنه همسایه های آهنبها مقدار ۱ کمتر باشد به این معناست که این متغیر محدودیت کمتری ایجاد میکند. برای مقدار -۱ نیز همینطور است. پس از این براساس اینکه کدام یک از مقادیر ۱ و -۱ در همسایه ها کمتر داشتند به این متغیر score بیشتری میدهیم.

مورد دیگری که چک میکنم تعداد مقادیر ۱ و -۱ در سطرو ستون آهنبهای مدنظر است. هرچه بیشتر سطر و ستون ها از مقدار مورد انتظار پر شده باشند به این معناست که با دادن این مقدار به این آهنبها محدودیت بیشتری برای خانه هایی که در سطر و ستون آن هستند ایجاد میشود. پس هرچه کمتر از مقدار مدنظر در آن سطر و ستون پر شده باشد این مقدار شانس بیشتری برای انتخاب به عنوان مقدار اول را دارد.

توضیحات forwardcheck

برای یک متغیر ابتدا همسایه های خودش و جفتش را با متد get_neighbors میگیریم سپس در این همسایه ها چک میکنیم اگر مقدار جدیدی که برای متغیر و جفتش ست کردیم در دامنه شان بود و این همسایه ها هنوز مقدار نگرفته بودند این مقدار را از دامنه شان حذف میکنیم، توجه داریم که مقدار متغیر با همسایه های خودش و مقدار داده شده به جفتش با خانه هایی که همسایه این خانه هستند چک میشود.

توضیحات ac³

برای پیاده سازی این الگوریتم از لینک زیر کمک گرفته شد:

<http://aima.cs.berkeley.edu/python/csp.html>

در اینجا ابتدا متغیر های دارای محدودیت با متغیر کنونی و جفتش به کمک تابع csp در کلاس MagnetPuzzle دریافت میشود سپس روی لیست csp داده شده حرکت میکنیم و چک میکنیم آیا برای متغیر xi (متغیر اول) باید مقداری از دامنه اش حذف شود یا خیر. اگر مقداری از دامنه اش حذف شد چک میشود که اندازه دامنه اش اگر صفر شده باشد پس یعنی این مقداردهی جواب ندارد.

پس از این چک نیز محدودیت های xi با همسایه هایش به لیست محدودیت ها اضافه میشود.

حال مقدار به این صورت از دامنه حذف میشود که اگر برای مقدار ۱ و ۱- در دامنه ی Xi، مقدار مناسب در دامنه ی Xj نبود، این مقدار از دامنه Xi حذف میشود. مقدار صفر از دامنه Xi چون ناسازگاری ایجاد نمیکند پس چک نمیشود.

تاثیر افزودن ac^3 در حل مثال ها

این روش به جای کاهش زمان حل مسئله، مسئله را در زمان طولانی تر از زمانی که از forward check استفاده میکنیم حل میکرد.

دلیلش هم به نظرم دو مورد زیر است:

۱. تعداد زیاد ناسازگاری هایی که خیلی از آنها بی مورد است در این جا چک میشود که سربار زمانی و محاسباتی ایجاد میکند.
۲. مقدار صفری که همان مقدار خنثی هست در دامنه متغیر ها محدودیتی ایجاد نمیکند لذا اگر همسایه های یک متغیر این مقدار را در دامنه شان داشته باشند با آن متغیر ناسازگاری نخواهند داشت و در نتیجه مقدار مدنظر از دامنه آن متغیر حذف نمیشود. مثلاً فرض کنید برای متغیر Xi مقدار ۱ را میخواهیم سازگاری کمانش را چک کنیم. اگر همسایه هایش یکی از دو مقدار صفر یا ۱- را در دامنه شان داشته باشند دیگر با این مقدار ناسازگاری نخواهند داشت و در نتیجه این مقدار از دامنه Xi حذف نمیشود. این مورد دوم به نحوی به همان سربار زمانی و محاسباتی که در مورد اول گفته شد اشاره دارد چون خیلی از مقادیر بی دلیل چک میشوند.

در کل نتیجه میشود که سازگاری کمان برای این مسئله نه تنها مفید نیست بلکه باعث افزایش زمان حل مسئله میشود.

زمان اجرای الگوریتم با forward check

```

73 timer = Timer()
74 # file_path = "D:/Dars/term9/AI/project/PROJECT3/Magnet-Puzzle-Game-CSP/InputFiles/input1_method2.txt"
75 file_path = "../InputFiles/input1_method2.txt"
76 bt = Backtrack(file_path, use_forwardcheck=True)
77 ended = timer.endTime()
78 print(ended)

if __name__ == "__main__":

```

Run: Backtracking

```

+ 2 1 2 2 2 1
- 2 1 2 2 1 2
1 1 0 + - 0 0 0
2 2 0 - + 0 + -
3 1 + 0 0 + - +
1 3 - 0 + - 0 -
2 1 + 0 - + 0 0
1 2 - 0 0 - + 0
0:01:37
Process finished with exit code 0

```

زمان اجرای الگوریتم با ac^3

```
def ac3(self, assignment, row_index, col_index):
    csp_list = self.magnet_puzzle.csp(row_index, col_index)
    while csp_list:
        (xi, xj) = csp_list.pop(0)
        pair = self.magnet_puzzle.get_pair(xi[0], xi[1])
        if self.remove_inconsistent_values(assignment, xi, xj):
            if len(assignment[xi[0]][xi[1]]) == 0:
                return False
        xi_neighbors = self.magnet_puzzle.get_neighbors(xi[0], xi[1])
        xi_neighbors.remove([pair[0], pair[1]])
        for xk in xi_neighbors:
```

Run: Backtracking

```
+ 2 1 2 2 2 1
- 2 1 2 2 1 2
1 1 0 - 0 0 0
2 2 0 - + 0 -
3 1 + 0 0 - +
1 3 - 0 + 0 -
2 1 + 0 - + 0 0
1 2 - 0 0 - + 0
0:02:10
Process finished with exit code 0
```

لیست کامیت های گیت هاب

در نهایت من در طول پیاده سازی این پروژه از گیت هاب استفاده کردم که لیست کامیت های من در طول انجام پروژه به شرح زیر است:



