گزارش ۵ درس هوش مصنوعی

پیاده سازی حل یک مسئله تصمیم‌گیری با استفاده از ارضای محدودیت

به قلم
امیر بابامحمودی

استاد
دکتر مهدی قطعی

فروردین ۱۴۰۰

مقدمه:

دسته ای از مسائل وجود دارند که در آن ها شاید جایگشت انجام دادن حالت های محتلف تفاوتی نداشته باشد و بتوان به گونه ای در هر مرحله یکی از متغیر های مسئله را با توجه به دامنه ی آن مقدار دهی کرد و صرفا نیاز به توجه به یک سری محدودیت ها باشد که یک متغیر ممکن است نسبت به متغیر های دیگر مسئله داشته باشد. در اینگونه از مسائل از الگوریتم های CSP استفاده میکنیم. برتری ای که استفاده از این الگوریتم نسبت به سرچ کردن کامل فضای استیت های یک مسئله دارد این میباشد که درختی بسیار کوچ تر خواهد داشت. برای مثال اگر n متغیر داشته باشیم و هر کدام دامنه ای به طول d داشته باشند(برای فهم راحت تر میتوان گسسته در نظر گرفت) در این حالت n^d اندازه کل درخت خواهد بود.در این گزارش به پیاده سازی دو تا از الگوریتم های CSP روی مسئله ی n-queen و مقایسه آن دو میپردازیم.

تعریف مسئله:

در این مسئله یک صفحه ی n*n شطرنج داشته و قصد داریم که n وزیر رو به گونه ای که در این صفحه بچینیم که هیچ دوتایی از آن ها با یکدیگر هیچ برخورید نداشته باشند.

BackTrack:

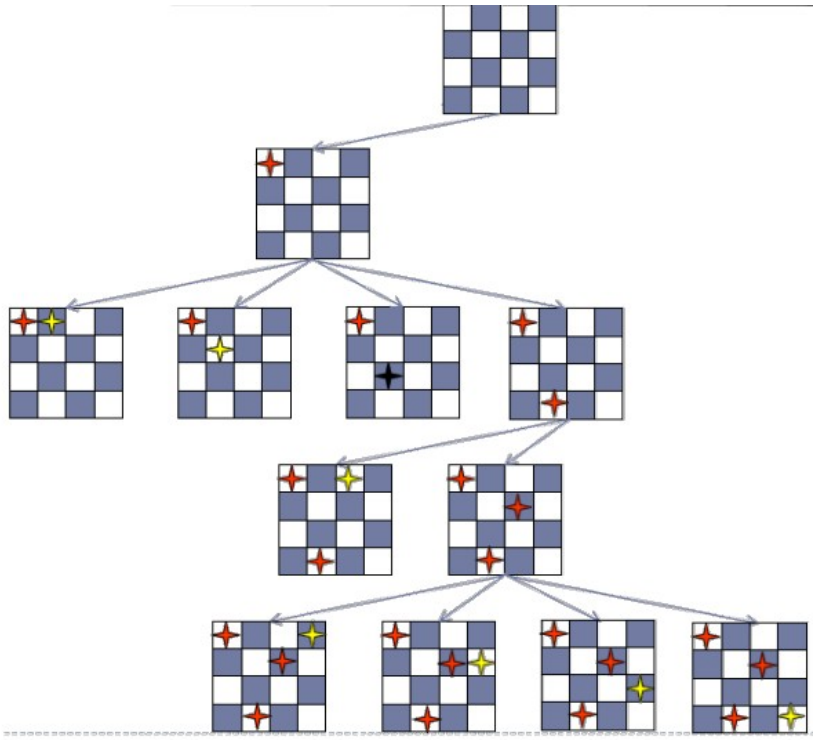
در این الگوریتم سعی در ساختن راه حل مسئله به صورت تدریجی و طی عملیاتی بازگشتی داریم. در هر مرحله راه حلی که نتواند محدودیت را ارضا کند را کنار گذاشته و هرگاه به همچین جوابی برسیم به یک مرحله قبل بازگشته تا راهی دیگر را امتحان کنیم.

برای مثال جدول سودوکو شکل مقابل را در نظر بگیرید. در هر مرحله یک عدد را قرار داده اگر با محدودیت های ما در تضاد باشد به مرحله ی قبل رفته و عدد بعدی را انتخاب میکنیم تا جایی که تمامی محدودیت ها را ارضا بکند.

3		6	5		8	4		
5	2							
	8	7					3	1
		3		1			8	
9			8	6	3			5
	5			9		6		
1	3					2	5	
							7	4
		5	2		6	3		

شکل۱)نمونه ای از یک بازی سودوکو

Back Track در n-queen :



شکل۲)Back Track در n-queen

اگر بخواهیم مسئله ی n-queen را مدل سازی کنیم میتوان گفت که هر یک از ستون های ما یکی از متغیر های ما بوده که دامنه ی هر یک از این متغیر ها هم در اصل یکی از ردیف های ۱ تا n به حساب میاید. در ابتدا ستون ۱ را انتخاب کرده و در اولین عضو دامنه ی آن یعنی ردیف ۱ وزیری را میگذاریم حال نوبت به ستون دوم میرسد . در ردیف اول آن وقتی مهره را بگذاریم به تضاد میرسیم پس در ردیف دو میگذاریم . حال این روند را ادامه داده تا به جواب نهایی برسیم.

کد پیاده سازی شده:

در این تابع بررسی میکنیم که اگر در خانه ای از یک ستون وزیری قرار بدهیم آیا این وزیر با وزیر هایی که در ستون های قبلی گذاشته ایم مداخله داشته, تهدید میشود یا خیر. به این صورت که بررسی میکنیم با فرض اینکه صفحه ی بازی یک ماتریس n*n باشد آیا در همان ردیف و ستون های قبلی مقداری برابر با ۱ یعنی وجود وزیر هست یا خیر. سپس در دو حلقه ی دیگر قطر ها با شیب منفی و مثبت را بررسی میکنیم.

```python
def is_consistent(board, row, col):

    for j in range(col):
        if board[row][j] == 1:
            return False

    a, b = row, col
    while a >= 0 and b >= 0:
        if board[a][b] == 1:
            return False
        a = a - 1
        b = b - 1

    c, d = row, col
    while c < len(board) and d >= 0:
        if board[c][d] == 1:
            return False
        c = c + 1
        d = d - 1
```

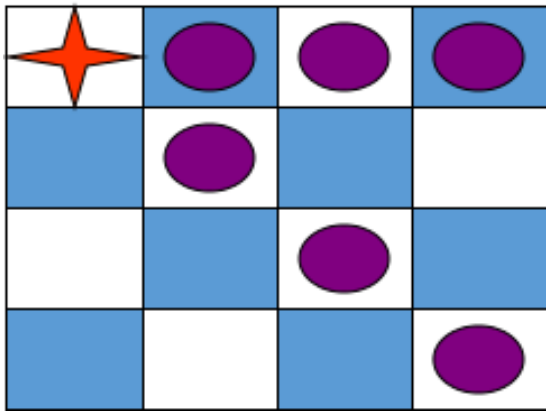
شکل۳)تابع is_consistent

```python
def back_track(board, col):
    if col >= len(board):
        print(board)
        return True

    for i in range(len(board)):
        if is_consistent(board, i, col):
            board[i][col] = 1
            if back_track(board, col + 1):
                return True
            board[i][col] = 0
    return False
```

شکل۴)تابع back_track

تابع اصلی حل مسئله یک تابع بازگشتی بوده که به عنوان آرگومان ورودی صفحه شطرنج را در حال فعلیبه همراه ستون(متغیر مسئله) که بخواهد مقدار دهی بشود گرفته . ابتدا بررسی میکند که در ستون آخر نباشیم . زیرا این بدان معناست که در n-1 ستون قبلی هیچ برخورد و تضادی نبوده و کار تمام میشود.سپس در یک حلقه از ردیف ۱ تا n را دونه دونه بررسی میکند که اگر مهری ای در آن قرار گیرد برخوردی پیش میاید یا نه. اگر نه مهره را در آن قرار داده و به صورت بازگشتی تابع را صدا تا پیش برود. حال هرجا قرار دادن مهره ای باعث ایجاد برخوردی میشد صفر را قرار میدهیم.

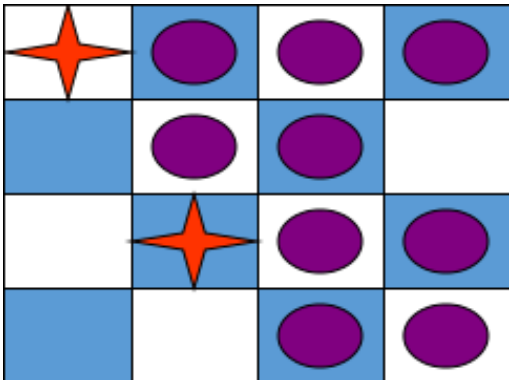
الگوریتم ForwardCheck در n-queen:



شکل۵)خانه های تهدید شده توسط ۱ وزیر

در این الگوریتم هر گاه در خانه ای وزیری بخواهد قرار بگیرد بررسی میکنیم که این وزیر چه خانه هایی ستون های بعدی رو اشغال کرده با این کار میتوان درخت حالت را از حالت back track کوچک تر کرد زیرا دیگر نیاز نیست همه ی خانه ها بررسی شوند و دیگر خانه های تهدید شده چک نمیشوند.

شکل های روبرو خانه های تهدید شده توسط وزیر هارا نشان میدهند.



شکل۶)خانه های تهدید شده توسط دو وزیر

کد پیاده سازی شده:

به وسیله ی این تابع خانه هایی که توسط خانه با مختصات ورودی تابع تهدید میشوند را برابر با مقدار n میکند. این خانه ها شامل خانه های هم ردیف مختصات مذکور در ستون های جلوتر و همچنین قطر ها با شیب منفی و مثبت میباشند.

```
def put_threat(board , row , col , n):
    for c in range(col + 1 , len(board)):
        board[row][c] = n

    a , b = row - 1 , col + 1
    while a >= 0 and b < len(board):
        board[a ][b ] = n
        a = a-1
        b = b+1

    c , d = row + 1 , col + 1
    while c < len(board) and d < len(board):
        board[c ][d ] = n
        c = c+1
        d = d+1
```

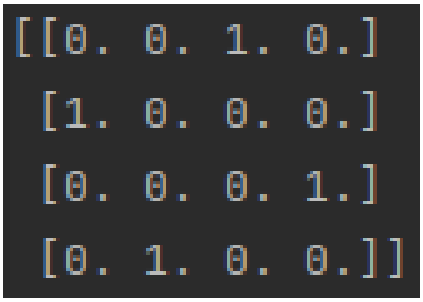
شکل۶)تابع put_threat

این تابع ساز و کاری شبیه تابع back track داشته با این تفاوت که هر بار بخواهد در خانه ای مهره ای قرار بدهد بررسی میکند که خانه مورد نظر مقدار ۲ نداشته باشد(یعنی مورد تهدید از خانه های قبلی نباشد) اگر نبود مانند back track مقدار ۱ را در آن قرار داده سپس تابع put_threat را صدا کرده تا خانه هایی که این خانه تهدید میکند را ۲ کند سپس اگر این خانه در راه حل جا نگرفت خانه را صفر کرده و دوباره تابع put_threat را صدا کرده تا خانه هایی که ۲ کرده است را صفر کند.

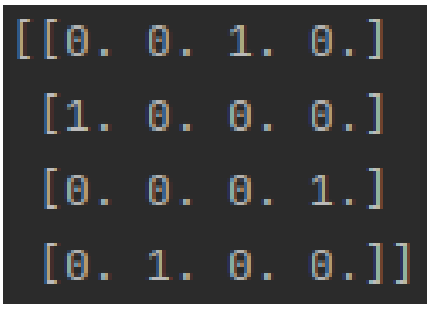
```
def forward_check(board, col):
    if col >= len(board):
        #print(board)
        return True

    for i in range(len(board)):

        if is_consistent(board, i, col) and board[i][col] != 2:
            put_threat(board , i , col , 2)
            board[i][col] = 1
            if forward_check(board, col + 1):
                return True
            board[i][col] = 0
            put_threat(board, i, col, 0)

    return False
```

شکل۷)تابع forward_check

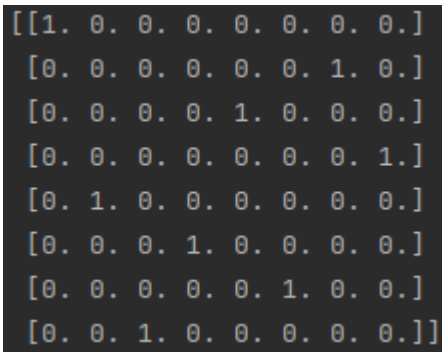
خروجی:



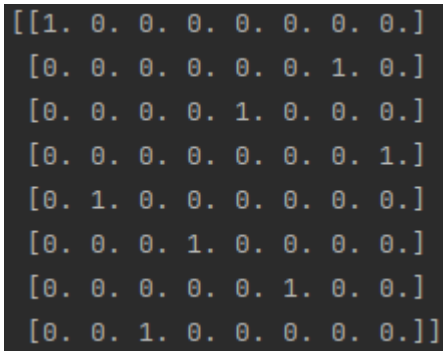
شکل۸)خروجی به ازای n=4
برای الگوریتم back track



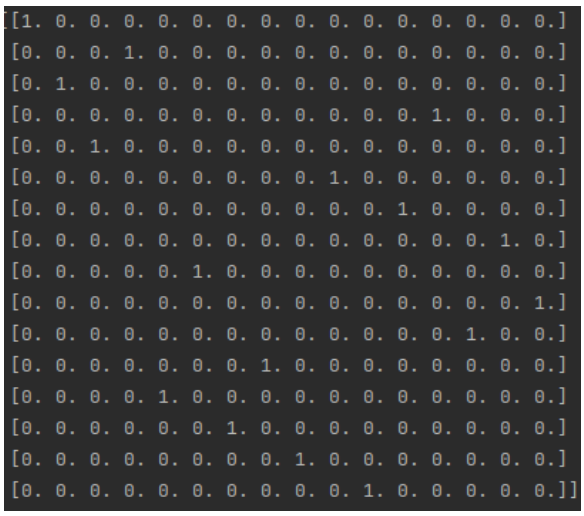
شکل۹)خروجی به ازای n=4
برای الگوریتم forwardcheck



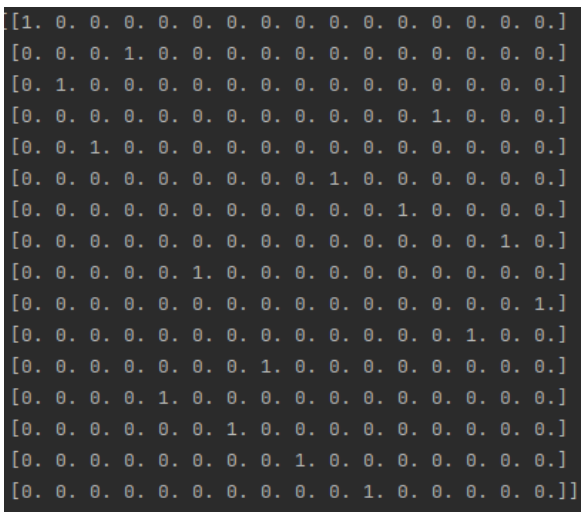
شکل۱۰)خروجی به ازای n=8
برای الگوریتم back track



شکل۱۱)خروجی به ازای n=8
برای الگوریتم forward check



شکل۱۲)خروجی به ازای n=16 برایback track



شکل۱۳)خروجی به ازای n=16 برای forward check

تحلیل زمانی:

n/الگوریتم	back track	forward check
4	0.00086	0.00021
8	0.0028	0.0031
16	0.732	0.829

تحلیل:

همانگونه که انتظار میرفت خروجی کد در برای هر دو الگوریتم یکسان شد. این امر به این دلیل میباشد که روش در این الگوریتم این میباشد که اولین راه حلی که بتواند محدودیت های مسئله را ارضا کند الگوریتم متوقف شده و خروجی ثبت میگردد.

اتفاقی که در این مسئله ی به خصوص افتاد این است که ما انتظار داشتیم زمان رسیدن به خروجی در الگوریتم forward check بهتر باشد زیرا درخت حالتی که تشکیل میدهد به دلیل شرط اضافه ای که در اول اجرای الگوریتم میباشد کوچک تر شده و از نظر زمانی و حافظه ای باید مطلوب تر باشد اما میبینیم که به ازای n= 8 و n= 16 این اتفاق نیوفتاد. تفسیری که میتوان کرد این میباشد که هزینه محاسباتی ای که تابع put_threat که در الگوریتم forward check فراخوانی شد به وجود آورد بر این کاهش اندازه ی درخت در هنگام سرچ غلبه میکند. شاید دلیل این امر به طور کلی کوچک بودن درخت حالت این مسئله ی بخصوص بوده و اگر مسئله ای دیگر مانند سودوکو بود اینگونه نمیشد. البته همینجا هم میبینیم که در حالتی که n = 4 بود چون هزینه ی محاسباتی put_threat بسیار ناچیز است تایم بهتری در نهایت برای الگوریتم forward check رقم خورد.

دریافت کامل کد:

https://github.com/amirbabamahmoudi/AI-projects/tree/main/n_queen_CSP

منابع:

۱ـ Artificial Intelligence A Modern Approach , Stuart J.Russell and Peter Norvig

۲ـ /https://www.geeksforgeeks.org/backtracking-algorithms

۳ـ /https://www.geeksforgeeks.org/n-queen-problem-backtracking-3