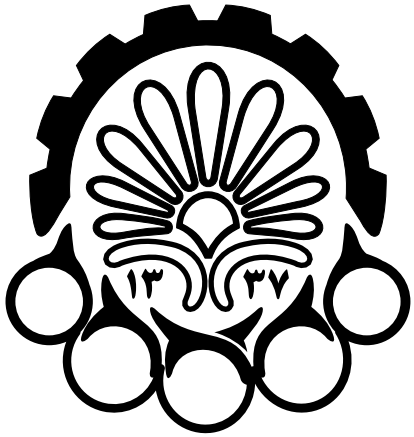

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

موضوع : پیاده سازی Sudoku با CSP

محقق : شروین غفاری

شماره دانشجویی : 9926105

استاد : یوسفی مهر

چکیده :

بازی پازلی sudoku یک بازی پر طرفدار است که روزانه توسط میلیون ها نفر در سراسر جهان انجام میشود. این بازی با توجه به سایز مسئله و نحوه مقدار دهی اولیه آن و یا تعداد خانه های تکمیل شده می تواند بسیار پیچیده شود.
استفاده از مفهوم CSP و ترکیب کردن آن با قوانین بازی میتواند مسئله را به مدلی برای پیاده سازی تبدیل کند.
CSP یا ارضای محدودیت با توجه به هر مسئله متفاوت است و شامل: domain,variable,constraints بوده و در صورتی که مسئله غیر قابل حل باشد یعنی برخی از قید ها ارضاء نشده است.

کلمات کلیدی :

- CSP
- Sudoku
- Simulated annealing
- Backtracking

مقدمه :

بازی sudoku شامل یک جدول n*n است که هر خانه آن با یکی از اعداد 1 تا n تکمیل می شود .
در خط اول ورودی برنامه مقدار n که اندازه مسئله را مشخص کرده و در خط دوم مقدار c که تعداد خانه های تکمیل شده آغازین بازی را

مشخص میکند و در خطوط بعدی به اندازه c در هر خط مقدار (i,j,value) دریافت شده که دو مقدار اول آن به ترتیب سطر و ستون و مقدار سوم عددی است که باید در آن خانه قرار بگیرد.
قوانین بازی به این صورت است که هیچ عدد تکراری در سطر یا ستون و مربع های با اندازه √n*√n وجود نداشته باشد در صورتی که هر کدام از constraint های گفته شده برآورده نشود برنامه پیغام Unsolvable CSP چاپ میکند .
domain در بازی **Sudoku** شامل بازه اعدادی است که هر variable میتواند دریافت کند و با توجه به سایز مسئله که از n است domain در بازه [1,n] قرار میگیرد.
متغیر ها یا **variables** خانه هایی **(cells)** هستند که با مقدار اولیه صفر جایگذاری شده اند و به معنای خانه خالی بوده و میتوانند ارقام گفته شده در **domain** را دریافت کنند.
Constraints در این بازه به ۳ دسته تقسیم میشود :

۱ – محدودیت اعداد غیر تکراری در هر سطر
۲ – محدودیت اعداد غیر تکراری در هر ستون
۳ – محدودیت اعداد غیر تکراری در مربع های با اندازه √n*√n

روش های گسترده ای برای حل این مسئله وجود دارد اما بیشتر آنها به دنبال حل کردن مسئله با قوانینی است که توسط انسان ها ایجاد شده و پایه علمی محکمی دارند. به طور کلی با استفاده از این قوانین است که میتوان مسئله را قابل حل کرد.

اما روش های بسیار flexible تر نیز که میتواند شمار گسترده ای از انواع مسائل بدون توجه به نوع طراحی آنها با استفاده از روش های بهینه سازی (Optimization) مانند simulated annealing حل کرد.

پیاده سازی بازی :

برنامه ابتدا با دریافت ورودی n از کاربر یک ماتریس n*n به وسیله List[List[0]] ایجاد کرده و با دریافت مقادیر اولیه صفحه بازی را ایجاد میکند. برنامه شامل یک تابع satisfy constraint است که جدول بازی به همراه سطر و ستون و مقدار عددی که قرار است در آن مکان قرار بگیرد به عنوان ورودی دریافت میشود.

برای جلوگیری از جایگذاری عدد تکراری ابتدا مقدار row را ثابت گرفته و iteration بر روی ستون ها انجام می شود اگر عددی برابر با عدد دریافت شده یافت شد یعنی عدد مورد نظر نمی تواند در این سطر قرار بگیرد به همین ترتیب برای سطر ها نیز بررسی میکنیم و اگر عددی برابر پیدا شد مقدار False return میشود در غیر اینصورت constraint آخر که مربوط به عدد تکراری در مربع های با سایز √n*√n است بررسی میشود به طوری که با فرمول:

Row-=Row%Sqrt(n)

Col-=Col%Sqrt(n)

مقدار سطر و ستون را به اولین cell مربوط به این مربع تغییر میدهیم . درنهایت اگر تمام ۳ قید برقرار بود مقدار True برگردانده میشود که به این معنی است که عدد مورد نظر میتواند در cell فعلی قرار بگیرد. اما قرار گیری

عدد به معنای جایگاه نهایی آن نخواهد بود چون ممکن است اعداد دیگری نیز با قرارگیری در این cell تمام قید ها را satisfy کنند و در صورتی که با عدد فعلی مسئله قابل حل نشود با روش Backtracking عدد دیگری را انتخاب میکنیم.

در تابع Sudoku جدول بازی و سطر و ستون آغازین به عنوان ورودی دریافت شده و چون Sudoku از روش Backtracking استفاده میکند به صورت بازگشتی پیاده سازی شده است به طوری که base case وضعیتی است که به آخرین سطر و ستون بررسی شده باشد در این صورت مقدار True برگردانده میشود.

در صورتی که تمامی خانه ها مقداری غیر از صفر نداشته باشند در سطر فعلی بررسی میکنیم که ستون فعلی آخرین ستون است یا خیر در صورتی که آخرین ستون باشد مقدار col را reset و یک واحد مقدار row را افزایش میدهیم و در هر مرحله بررسی میکنیم که مقدار value خانه فعلی صفر است یا خیر در صورتی که بزرگتر از صفر باشد یعنی در پیمایش های قبلی مقدار گرفته و خالی نیست در اینصورت تابع sudoku را برای cell بعدی فراخوانی میکنیم و درصورتی که مقدار value صفر شود تمام انتخاب های موجود در domain را بررسی میکنیم و درصورتی که تابع satisfy constraint مقدار True را بازگرداند به صورت موقت عدد را در cell جایگذاری میکنیم و تابع Sudoku را برای cell بعدی فراخوانی میکنیم.

اگر مقدار بازگشتی sudoku درست بود یعنی عدد موقت قرار گرفته میتواند به صـورت دائمی قرار گیرد و مقدار true بازگردانده شـده در غیر اینصـورت مقدار آن به صفر reset میشود و عدد بعدی انتخاب میشود.

در صـــورتی که عددی برای cell فعلی به طوری constraint را تضـــمین کند یافت نشد مقدار False بازگردانده میشود.

نمونه عملکرد :

عملکرد ۱ :

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

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

عملکرد ۲ :

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

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

نتیجه گیری :

در این پیاده سازی توانستیم با کمک CSP و روش Backtracking مسئله sudoku را برای اندازه های مختلفی پیاده سازی کنیم اما این روش به دلیل پیچیدگی زمانی نمایی آن برای مقادیر بسیار بزرگ به شکل بهینه عمل نمیکند و ممکن پاسخی را پیدا نکند.
اما استفاده از توابع heuristic میتواند علاوه بر افزایش سرعت رسیدن به پاسخ مسئله را برای مقادیر بزرگتر حل کند.

منابع :

- **Combining Metaheuristics and CSP Algorithms to solve Sudoku**

- **https://www.geeksforgeeks.org/sudoku-backtracking-7**