ESCOLA POLITÉCNICA

RODRIGO MASARU OHASHI

EXERCÍCIO PRÁTICO - SUDOKU

São Paulo
2019

ESCOLA POLITÉCNICA

RODRIGO MASARU OHASHI

EXERCÍCIO PRÁTICO - SUDOKU

Primeiro exercício prático desenvolvido na disciplina PCS3838 - Inteligência Artificial

Prof. Dr. Jaime Sichman

São Paulo
2019

Resumo

O trabalho em questão mostra como um Sudoku pode ser implementado utilizando uma técnica de satisfação de restrições, através da linguagem Prolog, utilizando predicados embutidos no SWI-Prolog. Outras formas de solução como a utilização da biblioteca de restrições para conjuntos finitos também são abordadas, para fins de comparação.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

O Sudoku, ou Su Doku, é um jogo onde é disponibilizado uma matriz 9x9, dividida em 9 blocos 3x3 [1]. Em algumas células números de 1 a 9 devem estar preenchidos previamente, sendo que o objetivo do jogo é preencher todos os espaços vazios, de modo que em cada linha, coluna e bloco 3x3 devam existir os números de 1 a 9, sem repetições.

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

Figura 1: Exemplo de um jogo de Sudoku. Fonte: Web Sudoku

Atualmente existem diversos portais com inúmeras combinações de jogos de Sudoku, desde revistas de passatempo até sites como o Web Sudoku[1].

O problema proposto pelo Sudoku pode ser modelado com através da satisfação de restrições [2], pela combinação da verificação de números diferentes nas linhas, colunas e blocos.

Para o exercício em questão, foi proposto o desenvolvimento de um programa na linguagem **Prolog** para o problema do Sudoku.

[1] https://www.websudoku.com/

2 Abordagem Proposta

Para resolver o problema proposto, primeiramente foi revisada a sua definição. Dado isto, é posto que dentro de uma matriz 9x9, todas as linhas, colunas e blocos 3x3 devem ter valores de 1 até 9, sendo que estes devem ser diferentes entre si.

A primeira restrição notada é em relação a estrutura do campo do jogo: linhas e colunas de tamanho 9. Dessa forma, pode ser imposto esse comprimento para tal.

Dado o campo com suas devidas dimensões, é necessário ter todos os valores diferentes entre si para cada linha, coluna e bloco. Para isso, podem ser criadas restrições onde deve ser imposto a diferença de todos os valores para essas estruturas.

O último passo é a restrição sobre os valores possíveis para cada uma das células do tabuleiro. Para tal, pode-se simplesmente dizer que estes estão entre um valor mínimo (1) e um máximo (9).

3 Estrutura do Software

Para o desenvolvimento do software, foi utilizado o ambiente fornecido pelo SWI-Prolog[2]. Também foi utilizada a biblioteca *clpfd.pl*, a qual providencia restrições para domínios finitos.

3.1 Estrutura base

```
1   sudoku(Rows) :-
2       length(Rows, 9),
3       maplist(same_length(Rows), Rows),
4       maplist(all_diff, Rows),
5       transpose(Rows, Columns),
6       maplist(all_diff, Columns),
7       Rows = [R0, R1, R2, R3, R4, R5, R6, R7, R8],
8       subgrid(R0, R1, R2),
9       subgrid(R3, R4, R5),
10      subgrid(R6, R7, R8),
11      append(Rows, Values),
12      sudoku_numbers(Values).
```

Figura 2: Código principal do algoritmo.

Inicialmente foi definido um predicado *Sudoku/1*, o qual deveria receber uma matriz (lista de listas) 9x9, Representando as linhas do tabuleiro.

Para restringir o tamanho das linhas e colunas, foram utilizados os predicados *length/2* e *same_length/2* já definidos pelo ambiente.

Um predicado *all_diff/1* foi criado para restringir os elementos passados, de forma a estes serem todos diferentes entre si.

Para realizar o processo descrito acima para as colunas, foi criado um predicado *transpose/2*, o qual forma a matriz transposta do primeiro elemento no segundo. Dessa forma, as colunas passam a ser tratadas como listas e podem ser restringidas da mesma maneira.

[2]https://www.swi-prolog.org/

Como os blocos são formados por diferentes elementos da matriz, foi criado um predicado *subgrid/3*, o qual recebe 3 listas (representando 3 linhas do tabuleiro) e utiliza *all_diff/1* para os elementos que formam um bloco.

A última restrição criada e imposta em *sudoku/1* foi a *sudoku_numbers/1*, a qual recebe uma lista de elementos, onde estes devem estar contidos no intervalo de 1 até 9. Para tal, foi utilizado *append/1* de forma a concatenar as listas da matriz e obter uma lista de todos os elementos, a qual foi passada para a restrição em questão.

3.2 Modificações Realizadas

```
1   :- use_module(library(clpfd)).
2
3   sudoku(Rows) :-
4     length(Rows, 9),
5     maplist(same_length(Rows), Rows),
6     append(Rows, Values),
7     Values ins 1..9,
8     maplist(all_distinct, Rows),
9     transpose(Rows, Columns),
10    maplist(all_distinct, Columns),
11    Rows = [R0, R1, R2, R3, R4, R5, R6, R7, R8],
12    subgrid(R0, R1, R2),
13    subgrid(R3, R4, R5),
14    subgrid(R6, R7, R8).
```

Figura 3: Código modificado do algoritmo.

De forma a tentar melhorar a performance do algoritmo descrito anteriormente, foi feita uma implementação com a biblioteca *clpfd.pl*. Esta traz novos predicados úteis para o problema em questão como:

- *ins/2*: Elementos do primeiro argumento estão dentro do segundo argumento;
- *all_distinct/1*: Verdadeiro se todos os elementos são diferentes;

- *transpose/2*: Transpõe os elementos de uma lista de listas.

Dessa forma foi possível resolver o problema em menos linhas, além de melhorar a performance, devido à logica envolvida nesses predicados.

Além dessa modificação, foi acrescentado um predicado *print_sudoku/1*, de modo a imprimir na tela o resultado do algoritmo, caso o problema seja resolvível.

4 Descrição dos Experimentos

Para os experimentos realizados, de forma a avaliar o algoritmo desenvolvido, foram utilizados problemas do site Web Sudoku. Este disponibiliza diversos desafios, os quais são nivelados em 4 categorias: Easy, Medium, Hard e Evil.

Foram utilizados 4 problemas, sendo cada um de nível diferente, segundo a classificação do site. A figura 4 mostra a disposição inicial de cada um deles.

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

(a) Problema Easy.

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

(b) Problema Medium.

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

(c) Problema Hard.

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

(d) Problema Evil.

Figura 4: Problemas utilizados para testar o algoritmo. Fonte: Web Sudoku

De modo a verificar o desempenho em cada um dos problemas para as

implementações realizadas, foi utilizado o predicado *time/1*, o qual imprime o tempo de execução, o número de inferências lógicas e o valor médio do número de inferências lógicas por segundo.

Na obtenção dos valores analisados, foram feitas 5 execuções para cada problema e utilizadas as médias dos resultados destes.

5 Análise dos Resultados

Como dito na seção anterior, foi utilizado o predicado *time/1* para conseguir definir as métricas para as comparações entre os algoritmos criados. A tabela 1 mostra o tempo de execução até a solução dos experimentos da figura 4.

	Base	**Modificado**
Easy	0,0296 s	0,0266 s
Medium	0,0294 s	0,0516 s
Hard	0,1472 s	0,0586 s
Evil	1,2988 s	0,0612 s

Tabela 1: Tempo de execução para os experimentos.

É notável que o aumento do nível de dificuldade faz com que o algoritmo base tome mais tempo até chegar numa solução. Quanto ao algoritmo utilizando a biblioteca *clpfd.pl*, esse tempo se manteve mais constante durante todos os testes.

De modo a entender o comportamento observado na tabela 1, foram também analisados os dados à respeito do número de inferências lógicas realizadas pelo algoritmo até a solução.

	Base	**Modificado**
Easy	326771	219494
Medium	448178	349533
Hard	4532294	383018
Evil	43096530	440724

Tabela 2: Número de inferências lógicas para os experimentos.

Conforme mostrado na tabela 2, o número de inferências lógicas aumenta de forma semelhante ao tempo de execução, quanto ao algoritmo base. Além disso, o número para o algoritmo modificado também não altera tanto, assim como nos resultados anteriores.

Analisando a forma como foi implementada a versão base, o aumento visto nos resultados anteriores se deve ter sido por conta do predicado *between/1*,

já que o mesmo deixa as variáveis atreladas a todos os inteiros do intervalo passado. Assim, é feita a verificação de diferença até mesmo para os valores que a célula não poderia assumir, devido aos valores pré-estabelecidos da linha/coluna/bloco que esta pertence.

Na implementação do *clpfd.pl*, o algoritmo deve ter a capacidade de saber quais valores não podem ser assumidos antes mesmo de atrelar um valor. Assim, o número de inferências lógicas deve depender apenas da posição e quantidade dos elementos pré-estabelecidos.

6 Conclusão

Foi possível realizar a implementação do algoritmo para a resolução de problemas de Sudoku em Prolog com sucesso, mesmo sem a utilização de bibliotecas extras, como o *clpfd*.

Quanto ao algoritmo base, pode-se dizer que o mesmo é razoável, pois consegue resolver mesmo os exercícios considerados como muito difíceis em pouco mais de 1 segundo. Esse tempo de resolução é inimaginável para um ser humano conseguir alcançar.

Pode-se afirmar também que a biblioteca *clpfd* tem um bom desempenho para tarefas cujo o problema envolve conjuntos finitos. Assim como a solução descrita aqui, seus predicados escalam bem mesmo para exercícios mais complexos, já que o tempo de execução tende a se manter semelhante em qualquer caso.

Em trabalhos futuros, pode ser trabalhado melhor a forma como foram colocados os predicados e como estes funcionam no algoritmo base. Apesar de já resolver o problema em questão, alguma otimização poderia ser feita, de forma a diminuir o número de inferências lógicas, e consequentemente, o tempo de execução.

Referências

[1] FELGENHAUER, B.; JARVIS, F. Mathematics of sudoku i. *Mathematical Spectrum*, Citeseer, v. 39, n. 1, p. 15–22, 2006.

[2] SIMONIS, H. Sudoku as a constraint problem. In: CITESEER. *CP Workshop on modeling and reformulating Constraint Satisfaction Problems*. [S.l.], 2005. v. 12, p. 13–27.