Trabalho Prático 3 - Algoritmos 1

Victor Hugo Silva Moura - 2018054958

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG – Brasil

1 Introdução

O sudoku é um puzzle de lógica baseado, em sua versão tradicional, na colocação de números em uma grade nxn. Os números são colocados de forma que nenhum número seja repetido nas linhas, nas colunas e em subquadrantes de tamanho n cada.

Tradicionalmente os quadrantes do sudoku são quadrados perfeitos, ou seja, um sudoku 9x9 tem quadrantes 3x3 por exemplo, que é um quadrado perfeito. Porém existem versões que não são perfeitas, como por exemplo um sudoku 8x8 onde cada quadrante tem tamanho 2x4, o que não é um quadrado perfeito. Para este trabalho, essas versões também serão utilizadas.

O objetivo do trabalho é resolver o problema do Sudoku por meio de uma transformação possível para ele, que é o problema da Coloração de Grafos. Nesse problema, dado um grafo $S(V, A)$ deve-se encontrar o menor número de cores que podem ser utilizadas para colorir o grafo sem que haja repetições de cores em vertices adjacentes. No caso do trabalho, não se busca o menor número k de cores possíveis, uma vez que já se conhece que $\chi(S) = n$, sendo $\chi(S)$ o número de cores necessárias para colorir o grafo S.

Sendo assim, é necessário transformar o sudoku em um grafo que respeite as restrições do sudoku (vértices na mesma linha, coluna ou bloco não podem ter a mesma cor). A montagem desse grafo será discutida na seção de Implementação. Como o problema de Coloração de Grafos é NP-Completo, é necessário utilizar uma solução aproximada, visto que a solução exata exige um tempo de computação exponencial, o que a torna inviável.

2 Implementação

Para a implementação desse problema, foi utilizada uma transformação do Sudoku para um problema de Coloração de Grafos, assim como dito na introdução. Essa transformação é feita de forma a respeitar as restrições do sudoku (vértices na mesma linha, coluna ou bloco não podem ter a mesma cor).

Para montar uma instância da Coloração de Grafos utilizando o sudoku, o primeiro passo é transformar cada célula em um vértice do grafo. Sendo assim, um sudoku nxn gera um grafo de nxn vértices para a coloração. O próximo passo é fazer as conexões do grafo de modo que essas conexões permitam com que as restrições do sudoku sejam mantidas. Para cada vértice do grafo, o mesmo é conectado aos vértices que representam elementos da mesma coluna, linha ou bloco. Dessa forma, como na coloração dois vértices adjacentes não podem ter a mesma cor, estamos garantindo no sudoku que nenhuma linha, coluna ou bloco terá números repetidos. Caso tenha, quer dizer que dois vértices adjacentes foram coloridos da mesma cor, o que não é possível. Abaixo temos uma imagem do grafo fornecida no enunciado do trabalho:

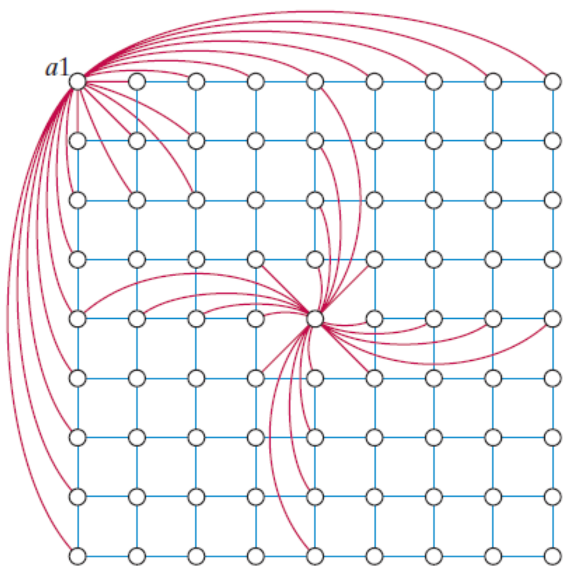

Figura 1: Exemplo de parte do grafo para um sudoku 9x9. Note que o vértice $a1$ está conectado a todos os vértices de sua linha coluna e bloco. O mesmo vale para o vértice central do grafo.

Sendo assim, agora vamos à solução para o sudoku. A solução desenvolvida utiliza a seguinte heurística: "*para cada novo preenchimento de cores no grafo, escolha o vértice com menos opções disponíveis, que ainda não tenha sido preenchido, e preencha-o com a primeira cor disponível para ele*". Em outras palavras, para cada vez que uma nova cor tiver que ser preenchida no grafo, o vértice com a menor opção de cores disponível, ou seja, o vértice cujo seus vizinhos tem a maior variedade de cores (impossibilitando que essas cores sejam escolhidas para esse vértice), e que ainda esteja descolorido é escolhido para ser preenchido. A cor com a qual ele será preenchido é a primeira cor disponível para ele. Exemplo: Suponha que as cores para a coloração são {"Azul", "Verde", "Amarelo", "Vermelho"} e o vértice que será preenchido não pode ser preenchido com as cores "Azul" e "Vermelho". A primeira cor disponível, seguindo a ordem das cores do grafo é a cor "Verde" e é com essa cor que o vértice será preenchido. Tal heurística foi adotada devido à sua simplicidade de implementação e entendimento, além de possuir uma boa taxa de acertos para sudokus relativamente pequenos.

O preenchimento só termina com duas condições, que são as condições de solução do sudoku. Elas são:

- Se o sudoku estiver completamente preenchido. Nesse caso a execução termina e é retornado ao usuário que uma solução foi encontrada.

- Se ao tentar colorir algum vértice, não houverem opções de cor disponíveis para ele. Nesse caso, é retornado ao usuário que a solução não foi encontrada pela heurística.

Para o primeiro caso, se o sudoku estiver completo, não existe nenhum vértice no grafo que não foi colorido, o que indica que a coloração terminou e, por consequência, o sudoku. Além disso, para chegar nesse estado, nenhum vértice do grafo foi colorido com a mesma cor de um vértice adjacente, o que indica que no sudoku nenhuma célula foi preenchida com o mesmo número de outra célula na mesma linha, coluna ou bloco, devido à construção do grafo.

Para o segundo caso, se algum vértice não possui cor para ele, isso significa que durante o processo de coloração, todos os vértices adjacentes a ele foram coloridos de forma que todas as cores foram utilizadas. Dessa forma, o único jeito de reverter isso seria por meio de *backtracking*. Porém, a heurística proposta não utiliza essa técnica. Assim, ao chegar nesse estado, não há solução possível para a coloração, e, por consequência, para o sudoku.

3 Análise de Complexidade

Para a Análise de Complexidade dessa solução vamos avaliar individualmente a Complexidade de Tempo e a de Espaço.

3.1 Complexidade de Tempo

Assim como explicado na seção anterior, o processo de coloração deve ser feito em cada vértice do grafo até que todos estejam devidamente coloridos (ou até que não haja mais solução). O número de vértices do grafo é n^2, sendo n número de cores (ou no caso do sudoku, o tamanho de cada subquadrante). Sendo assim, poderíamos dizer que o processo de coloração é $O(n^2)$. Porém, para sabermos qual vértice será colorido e qual cor será escolhida para o vértice, temos que fazer algumas operações a mais para cada vértice.

Vamos chamar de v_i, o vértice escolhido para ser colorido na i-ésima iteração. Para encontrar v_i, é necessário percorrer todos os vértices do grafo procurando pelo vértice que tem a menor opções de cores disponíveis para ele. Com a ajuda de um set auxiliar que guarda quais elementos um vértice não pode ser, isso pode ser feito em $O(n^2)$, já que saber o tamanho do set de cada vértice gasta um tempo constante e isso é feito para cada vértice do grafo. Outro processo que gasta $O(n^2)$ é conferir se o sudoku já está completo, pois precisamos passar por cada vértice e verificar se ele já foi colorido ou não. Após isso, temos que ver qual cor será atribuída ao vértice. Para isso, é feita uma iteração sobre as cores e, para cada cor, é verificada se ela está presente no set de cores "proibidas" de v_i. Caso esteja, passamos para a próxima cor. Caso não, atribuimos essa cor a v_i e atualizamos os sets de cores "proibidas" dos vértices adjacentes. O número de cores é igual ao tamanho de um bloco/subquadrante, ou seja, n. Procurar por um elemento no set é logarítmico no tamanho do set e o tamanho do set é no máximo n. Assim, a operação de procurar pela cor é $O(n \log n)$. Para atualizar os sets da lista de adjacências, temos que considerar o tamanho da lista. Ela é linear no número de vértices, pois cada vértice tem $n-1$ vértices correspondentes à linha, $n-1$ vértices correspondentes à coluna e $O(n)$ vértices correspondentes ao bloco (nunca ultrapassa n pois cada bloco tem tamanho máximo n), somando no total $2(n-1)+O(n)$, que é $O(n)$. Para cada inserção no set, a complexidade é logarítmica, o que gera no final uma complexidade $O(n \log n)$ para a atualização do set dos vértices adjacentes a v_i.

Sendo assim, a complexidade para encontrar o vértice v_i, atribuir uma cor a ele e atualizar os sets da sua lista de adjacências é:

$$O(n^2) + O(n^2) + O(n \log n) + O(n \log n) = O(n^2)$$

Como isso é feito aproximadamente n^2, pois cada vértice do grafo deve ser colorido, temos que a complexidade de tempo para resolver o sudoku é:

$$O(n^2) * O(n^2) = O(n^2 n^2) = O(n^4)$$

Além disso, temos a complexidade de montar o grafo do sudoku para que seja feito o processo de coloração. Cada vértice é inicialmente conectado com todos os vértices da sua linha e da sua coluna. Como citado no paragrafo acima, temos $n-1$ vértices correspondentes à linha do vértice atual e $n-1$ vértices correspondentes à coluna, totalizando $2(n-1)$ vértices. Além disso, temos os vértices do bloco. Parte deles já foram cobertos ao conectar as linhas e parte ao conectar as colunas. Como cada bloco tem no máximo n elementos e alguns já estão cobertos, temos algo da ordem de $O(n)$ vértices faltantes. Juntando tudo isso, para cada vértice temos $O(n)$ operações, e, com n^2 vértices, a complexidade para montar o grafo é:

$$O(n^2) * O(n) = O(n^2 n) = O(n^3)$$

Juntando tudo, temos que a complexidade final de tempo é:

$$O(n^3) + O(n^4) = O(n^4)$$

3.2 Complexidade de Espaço

A complexidade de espaço do algoritmo envolve a criação de estruturas adicionais para facilitar o processo de preenchimento do sudoku. Assim como mencionado, cada vértice possui uma lista de adjacências, que indica quais vértices são adjacentes a ele, e também possui um set, que indica quais cores esse vértice não pode ter.

A lista de adjacências de cada vértice, conforme explicado na complexidade de tempo, tem tamanho $O(n)$. Além disso, temos n^2 vértices, cada um com sua própria lista. Assim, temos que as listas de adjacência ocupam um espaço total de $O(n^3)$.

O set de cada vértice guarda quais cores esse vértice não pode ter. Ao final do sudoku, onde todos os vértices tem apenas um cor possível, que é a cor que ele possui, cada set de cada vértice tem tamanho igual a $n-1$, pois temos n cores. Como cada vértice tem um set e o número de vértices é n^2, o espaço ocupado por esses sets é no máximo da ordem de $O(n^3)$.

Considerando agora o armazenamento do sudoku, temos uma matriz nxn que guarda as cores/números de cada vértice. Assim, temos que o armazenamento do sudoku é da ordem de $O(n^2)$ em espaço.

Juntando tudo temos que a complexidade de espaço final do algoritmo é:

$$O(n^3) + O(n^3) + O(n^2) = O(n^3)$$

4 Análise Experimental

Para a análise experimental, foram feitos testes para sudokus com subquadrantes de tamanho 2x2, 2x3, 2x4, 3x3. Para cada tamanho foram gerados 10 testes diferentes utilizando o website http://www.menneske.no/sudoku/eng/. Cada teste foi executado 10 vezes para garantir uma média do tempo de execução mais confiável. Sendo assim, cada tamanho de sudoku foi testado 100 vezes.

Após a realização dos testes, foi calculado a média do tempo de execução e o desvio padrão do mesmo, e uma tabela foi gerada. A tabela de média do tempo de execução e desvio padrão está abaixo:

Tamanho	**Média de tempo (μs)**	**Desvio padrão**
2x2	153.36	63.99
2x3	203.57	63.15
2x4	327.29	111.37
3x3	402.39	144.79

Por meio dessa tabela, um gráfico de tempo x tamanho foi gerado. O gráfico citado se encontra abaixo:

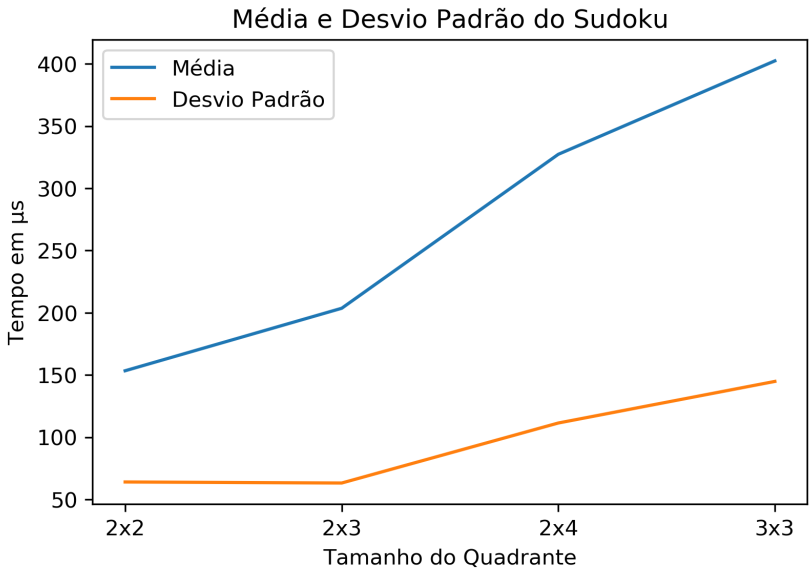

Figura 2: Gráfico de tempo x tamanho para os testes do sudoku

Ao observar o gráfico, pode-se perceber que o tempo cresce à medida que o tamanho do sudoku também cresce. Os valores de n (tamanho do quadrante) ainda são bem pequenos para que seja possível observar claramente a complexidade $O(n^4)$, mas já é possível notar uma leve curva de crescimento para esses tamanhos.

Agora, observando a acurária para cada tamanho, temos o seguinte:

Tamanho	2x2	2x3	2x4	3x3
Acurácia	100%	100%	40%	80%

Com isso, podemos concluir que a medida que o sudoku aumenta, a precisão diminui. Além disso, quanto mais complexo o preenchimento do sudoku, como o sudoku 2x4 por exemplo, mais difícil é de se acertar. Sendo assim, a heurística é esperada de ter um maior número de acertos em instâncias de tamanho menor e mais regulares, ou seja, com quadrantes perfeitamente quadrados ou próximos a isso.

5 Conclusão

Por meio da realização deste trabalho, foi possível concluir que existem heurísticas muito boas e simples para a resolução de problemas difíceis (NP-Completo, como é o caso deste trabalho), porém nenhuma delas consegue ser perfeita para resolver todas as possíveis instâncias desses problemas.

Sendo assim, essa relação é um tipo de *tradeoff* entre a velocidade de resposta e precisão da mesma. Heurísticas mais simples tendem a dar respostas mais rápidas, porém mais imprecisas, enquanto que heurísticas mais complexas tendem a ser mais lentas e mais precisas. Cabe ao usuário, neste caso, decidir o que é mais importante para ele.

Referências

[1] B. Kleinberg and É. Thardos. *Algorithm Design.* Pearson, 2005.

[2] N. Ziviani. *Projeto de Algoritmos com Implementações em Pascal e C.* Cengage, 2011.