Independência no Tabuleiro de Xadrez com Cavalos - Documentação

Matheus Ribeiro Alencar - 494711

1 Introdução

Nesse problema, trabalhamos com um tabuleiro n x n, com $n \in \mathbb{N}$, o problema consiste em inserir a maior quantidade de cavalos de modo que nenhum cavalo consiga atacar outro cavalo.

A ideia central da heurística é aplicar cavalos no tabuleiro de xadrez de forma que eles restrinjam o menor número de casas possíveis. Essa foi a heurística utilizada para avaliar o quão bom determinada casa é para posicionarmos um cavalo nela. Posteriormente, vemos a melhor solução e a comparamos com a heurística.

1.1 Link para o programa

Link para o collab com o programa: `https://colab.research.google.com/drive/1ydR2qHZi3Zcku05g1HZCbIRr9mgzQlCQ?usp=sharing`

1.2 Heurística

Utilizamos uma matriz para representar nosso tabuleiro de xadrez, cada número $i \in \{1, 2, 3, ..., n^2\}$ da matriz indica o i-ésimo cavalo posto no tabuleiro. Note que n^2 é o número de casas, configurando nosso limite superior, porém raramente vamos chegar a pôr n^2 cavalos. Os caracteres 'X' na matriz representam posições onde existe pelo menos um cavalo em que a ataca.

Além da configuração do tabuleiro retornamos também o número de cavalos postos, também dado pelo maior número pertencente a matriz.

Para a formulação da heurística, criamos uma matriz cópia, onde para cada posição disponível vemos caso alocarmos um cavalo naquela posição, quantas casas restringiria. Para não conflitarmos com os cavalos já postos, decrementamos das posições com 0 $\{-8, -7, ..., -1, 0\}$ ou seja, ao final checaremos na matriz o melhor lugar para colocarmos um cavalo (o que possua valor v tal que $v \leq 0$ e v seja maior possível).

1.3 Para o programa principal, árvore binária e todas as possíveis configurações

Adicionamos toda possível solução do tabuleiro em uma fila, de modo que o primeiro elemento a entrar e sair da fila, será um tabuleiro com nenhuma casa sendo ocupada. Quando o tabuleiro vazio deixa a fila, cria dois filhos, o primeiro, sendo o tabuleiro com um 'X' marcado na primeira posição em que encontramos um espaço disponível, e o outro com um 'C'.

A marcação 'X' representa a não existência de cavalo nessa casa, e decidimos não pôr nada nessa casa, ao longo a execução a mantemos. Já a marcação 'C' representa uma casa ocupada por um cavalo, essa marcação irá restringir algumas casas baseado nos movimentos que o cavalo pode fazer.

1.3.1 Poda

Com o intuito de otimizar o programa e reduzir o escopo de instâncias de tabuleiros no decorrer do programa, implementei uma poda (poda por não otimalidade) com base na solução da heurística. Ela está implementada no início da função *branch* e funciona da seguinte forma:

1. Percorremos nosso tabuleiro P;
2. Para cada número de elementos 0 e 'C' incrementamos 1 em um contador;
3. Ao final, o contador terá o maior número possível de cavalos colocados naquela configuração (ignorando a restrição do movimento dos cavalos);
4. Comparamos valor do contador com o da boa solução encontrada com a heurística;
5. Caso o valor do contador seja igual ou melhor que o da heurística, continuamos seguindo naquele ramo;
6. Caso contrário podamos aquele ramo.

2 Funções

2.1 initialize_chessboard(dim)

Função para inicializarmos um tabuleiro de dimensões *dim x dim* com 0s. Tabuleiro é representado por uma matriz inicialmente com todos os elementos zerados.

2.2 place_attacks(pos_x,pos_y,chessboard)

Funçao para restringirmos as casas do tabuleiro (*chessboard*) baseado na posição em que colocaremos o cavalo (pos_x, pos_y).

2.3 check_attacks(chessboard)

Função principal da heurística, nela que criamos a matriz auxiliar e checaremos quantas posições cada casa, se ocupada por um cavalo, tomaria. Retorna as posições da melhor casa possível (casa que restringe menos casas no tabuleiro)

2.4 place_knights(chessboard)

Função que executa nossa heurística, a partir dela chamaremos funções secundárias como *check_attacks* e *place_attacks* ao final dela chamaremos *reset_chessboard* que adicionará a última configuração validada pela heurística e seu número de cavalos.

2.5 reset_chessboard()

Função que irá finalizar nossa heurística. Nela que alocamos a configuração do tabuleiro em que ,pela heurística, conseguimos o maior número de cavalos. Também guardamos esse número para comparar posteriormente com soluções que encontrarmos na nossa fila e árvore binária.

2.6 branch(P)

Função que ramificará P em P' e $P1$ onde P' é o tabuleiro P só que ao invés da próxima posição 0, teremos um $'X'$. De forma análoga para $P1$ só que marcaremos com um $'C'$. C representando uma casa ocupada por um cavalo e $'X'$ uma casa onde não poderemos por um cavalo.

Na função $branch(P)$ está implementada a estratégia de poda por não-otimalidade, comparando sempre com o resultado vindo da heurística.

3 Testes

As soluções por meio da heurística no meu **programa** estão representadas,na matriz, com números inteiros e caracteres'X'. Números inteiros representam os cavalos, expicado na seção **1.2**. Para facilitar a visualização, nos testes a seguir, apenas rotularemos como 'C'.

3.1 Tabuleiro 3x3

Para a $dimensao\_do\_tabuleiro = 3$ levamos menos de 1 segundo para calcular um total de 16 soluções possíveis, encontrando um valor igual ao valor da heurística para o número máximo de cavalos.

- Valor_Heurística = 5
- Valor_Máximo = 5

$$Config\_Heuristica = \begin{bmatrix} X & C & X \\ C & C & C \\ X & C & X \end{bmatrix}$$

$$Config\_Maxima = \begin{bmatrix} C & X & C \\ X & C & X \\ C & X & C \end{bmatrix}$$

3.2 Tabuleiro 4x4

Para a $dimensao\_do\_tabuleiro = 4$ levamos cerca de 2.12 segundos para calcular um total de 62 soluções possíveis, encontrando um valor igual ao valor da heurística para o número máximo de cavalos.

- Valor_Heurística = 8
- Valor_Máximo = 8

$$Config\_Heuristica = \begin{bmatrix} C & C & C & C \\ X & X & X & X \\ X & X & X & X \\ C & C & C & C \end{bmatrix}$$

$$Config\_Maxima = \begin{bmatrix} C & X & C & X \\ X & C & X & C \\ C & X & C & X \\ X & C & X & C \end{bmatrix}$$

3.3 Tabuleiro 6x6

Para a $dimensao\_do\_tabuleiro = 6$ levamos cerca de 220.76 segundos para calcular um total de 25106 soluções possíveis, encontrando um valor melhor que o valor da heurística para o número máximo de cavalos.

- Valor_Heurística = 15
- Valor_Máximo = 18

$$Config\_Heuristica = \begin{bmatrix} C & X & C & X & C & X \\ X & C & X & C & X & C \\ C & X & C & X & C & X \\ X & X & X & X & X & X \\ X & X & X & X & X & X \\ C & C & C & C & C & C \end{bmatrix}$$

$$Config\_Maxima = \begin{bmatrix} C & X & C & X & C & X \\ X & C & X & C & X & C \\ C & X & C & X & C & X \\ X & C & X & C & X & C \\ C & X & C & X & C & X \\ X & C & X & C & X & C \end{bmatrix}$$