Inteligência Artificial

Primeiro Trabalho de Programação

Arthur do Prado Labaki - 11821BCC017
Vinnicius Pereira da Silva - 11821BCC046

08-11, 2022

GBC063

1 Introdução

Nesse projeto buscamos obter uma solução para o problema computacional das n-rainhas. Para isso, foi utilizado os algoritmos: busca em profundidade iterativa, subida da encosta com e sem recomeço aleatório e o recristalização simulada.

1.1 Problema das N-rainhas

O problema das n-rainhas resume-se a encontrar uma solução para agrupar um número n predeterminado de rainhas de modo que nenhuma rainha consiga atacar a outra em um tabuleiro n por n.

1.2 Busca em Profundidade Iterativa

A busca em profundidade iterativa ou, mais especificamente, a pesquisa de profundidade de aprofundamento iterativo é uma estratégia de pesquisa de espaço/gráfico de estado na qual uma versão de profundidade limitada da pesquisa em primeiro lugar é executada repetidamente com limites de profundidade crescentes até que o objetivo seja encontrado.

1.3 Subida de Encosta (*Hill Climbing*)

A subida de encosta é um algoritmo iterativo que começa com uma solução arbitrária para um problema, depois tenta encontrar uma solução melhor fazendo uma mudança incremental na solução. Se a mudança produzir uma solução melhor, outra mudança incremental será feita na nova solução e assim por diante até que nenhuma outra melhoria possa ser encontrada. Ainda é possível adicionar um recomeço aleatório, caso a melhor solução encontrada não conseguiu atingir o objetivo determinado (evitar ótimo locais).

1.4 Recristalização Simulada (*Simulated Annealing)*

A recristalização simulada ou têmpera simulada é uma técnica probabilística para aproximar o ótimo global de uma determinada função. Ela busca encontrar o máximo local o mais

próximo possível do máximo global utilizando o decremento da temperatura. Além disso, sempre existe uma probabilidade atrelada a temperatura atual que permite que o algoritmo prossiga para um estado pior que o anterior, para evitar um possível ótimo local.

1.5 Criação do nosso projeto

O projeto todo foi feito na linguagem Python, que tem boa manipulação para inteligência artificial. Nossa estrutura principal tem como função exibir um menu via terminal para que o usuário possa escolher o algoritmo que deseja executar, junto com o número de rainhas que o tabuleiro terá.

No algoritmo de **busca em profundidade iterativa**, utilizamos uma recursão para buscar os nós mais profundos, porém, definimos um limite máximo para essa descida (no caso, sempre é o numero de rainhas mais um).

Já no algorítimo de **subida de encosta**, buscamos todos os vizinhos do nó atual, e atualizamos o nó atual com o vizinho de menor custo (menor número de colisões entre rainhas). Também foi criada uma versão desse algorítimo com recomeço aleatório, em que, caso não encontre vizinho de menor custo ele aleatoriza de novo o algorítimo e refaz sua busca (ele recomeça um número limitado de vezes).

Por fim, no algorítimo de **recristalização simulada** foi usada a mesma aborda da subida de encosta, porém com o uso da técnica de temperatura (controla o fluxo do código), além de uma probabilidade de aceitação, com base na temperatura, que permite atualizar o tabuleiro atual com um tabuleiro de maior custo.

Também foi criado um repositório público no Github, disponível no link abaixo, para salvar tanto os códigos quanto os testes e as imagens.
Repositório do projeto

2 Experimentos

Após o desenvolvimento dos três algoritmos, foram realizados uma bateria de testes a fim de verificar se existe problemas ou *bugs* nos códigos criados. Para isso, foi feito:

2.1 Testes

Para os testes, foi definido três números de rainha a serem testadas nos três algoritmos criados, que são com 4 rainhas, com 6 rainhas e com 10 rainha. Para melhores resultados, cada teste foi repetido três vezes. Com isso, obtemos a tabela abaixo:

Tabela de Testes (em segundos)				
	Profundidade Iterativa	**Hill Climbing**	**H.C. com Rec. Aleat.**	**Simulated Annealing**
4 Rainhas	Teste 1: 0.02792358398	Teste 1: 0.00801563262	Teste 1: 0.00398993492	Teste 1: 0.02405714988
	Teste 2: 0.11856842041	Teste 2: 0.00498628616	Teste 2: 0.00997519493	Teste 2: 0.05106353759
	Teste 3: 0.94184517860	Teste 3: 0.00299215316	Teste 3: 0.01201152801	Teste 3: 0.01097536087
	Média: 0.362779061	Média: 0.00533135732	Média: 0.00865888595	Média: 0.08609604835
6 Rainhas	Teste 1: 201.208940744	Teste 1: 0.02991914749	Teste 1: 0.188702583312	Teste 1: 0.51239156723
	Teste 2: 2.19140195846	Teste 2: 0.02792477607	Teste 2: 0.09128093719	Teste 2: 0.61734604835
	Teste 3: 28.7834041118	Teste 3: 0.03396415710	Teste 3: 0.17997407913	Teste 3: 0.65863585472
	Média: 77.3945822716	Média: 0.03060269355	Média: 0.15331919988	Média: 0.5961244901
10 Rainhas	Teste 1: Inviável	Teste 1: 0.39693689346	Teste 1: 4.28875541687	Teste 1: 2.09473156929
	Teste 2: Inviável	Teste 2: 0.43003225326	Teste 2: 0.80684781074	Teste 2: 1.85223436355
	Teste 3: Inviável	Teste 3: 0.36160159111	Teste 3: 2.12788200378	Teste 3: 1.36353802680
	Média: Indisponível	Média: 0.39619024594	Média: 2.40782841047	Média: 1.77016798655

Figura 1: Tabela de testes dos algoritmos em segundos

Vale ressaltar que os testes sublinhados na tabela são os que não alcançaram a solução para o problema descrito. Destaca-se ainda que os testes do algoritmo de profundidade iterativa com 10 rainha estão marcados como inviáveis pois ultrapassaram 10 minutos de execução, que podem ser descartados pois os outros algoritmos são muito superiores.

2.2 Resultados

Após analisar esses resultados, é possível construir um gráfico comparando os algoritmos com base em seus respectivos tempos de execução para determinada quantidade de rainhas. Com isso, temos:

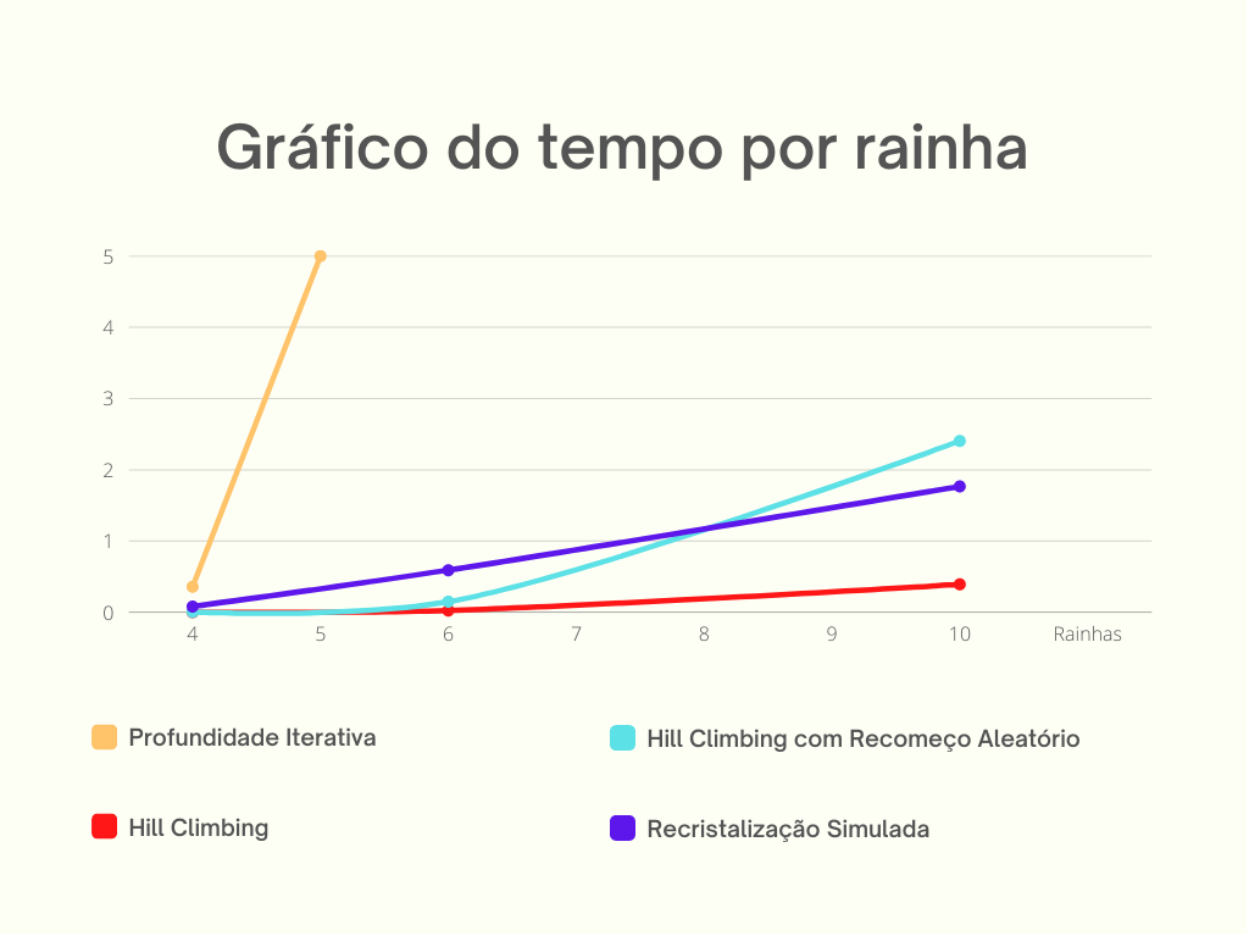

Figura 2: Gráfico de linhas comparando os algoritmos em tempo por rainhas

Analisando esse gráfico, é possível concluir que o algoritmo de Profundidade Iterativa é muito inferior aos outros, pois seu tempo de execução para 5 rainhas passa de 5 segundos, e 6 rainhas passa de 77 segundos, o que é muito distante dos outros no mesmo número de rainhas. Também é possível observar que o algoritmo *Hill Climbing* normal é superior em tempo a todos, mas ele não resolve o problema em todos os casos (para em um ótimo local).

Sobre os dois melhores, o *Hill Climbing* com recomeço aleatório, é melhor que o Recristalização Simulada quando se trata de poucas rainhas (entre 7 e 9 rainhas). Quando o número de rainha aumenta, o algoritmo de Recristalização Simulada passa a ser superior ao de subida de encosta.

3 Conclusão

Concluímos que foi possível concluir o objetivo do desse projeto utilizando os algorítimos citados. Existiram muitas dificuldades em encontrar conteúdos na internet sobre os algoritmos, principalmente o algoritmo da recristalização simulada.

Por fim, foi analisado e concluído que o algorítimo da recristalização simulada tem menor tempo de execução para encontrar a solução do problema em relação a o subida de encosta e do busca em profundidade iterativa, sendo indicado para resolver esse problema com um maior número de rainhas.