Inteligência Artificial - 2021/1
Hill Climbing, Simulated Annealing e Algoritmo Genético

Nesta tarefa você vai implementar os algoritmos Hill Climbing, Simulated Annealing e Algoritmo Genético e usá-los para resolver o problema das N-Rainhas. Os programas devem ser feitos preferencialmente na linguagem Python. Recomenda-se a utilização do *Colab Notebooks*.

1. **Modelagem**
   (a) Descreva como um tabuleiro $N \times N$ com N rainhas é representado no seu programa. Justifique a escolha desta representação.

2. **Implementação Base:** As seguintes funções devem ser implementadas de maneira que possam ser testadas individualmente.
   (a) Defina a função *tabuleiro* que gere aleatoriamente um ou mais tabuleiro $N \times N$ com N rainhas. A entrada desta função deve ser o número de rainhas N e a quantidade de tabuleiros Q que devem ser gerados. Exemplo: $tabuleiro(8, 10)$ deve gerar aleatoriamente 10 tabuleiros de tamanho 8×8. Os tabuleiros gerados devem estar de acordo com a modelagem definida no item anterior.

   (b) Defina a função $todosVizinhos$ que dado um tabuleiro qualquer T, retorna todos os tabuleiros vizinhos a T.

   (c) Defina a função $umVizinho$ que dado um tabulleiro qualquer T, retorna um dos vizinhos de T. A escolha do vizinho a ser retornado pela função deve ser aleatória.

   (d) Defina a função $numeroAtaques$ que dado um tabuleiro qualquer T, retorna o número de ataques entre as rainhas de T. A função deve contar apenas uma vez a quantidade de ataques entre as rainhas. Exemplo: se a rainha A ataca a rainha B, então B ataca A. Conte isso como 1 ataque.

3. **Hill Climbing** Os grupos que foram indicados para fazer o Hill Climbing, devem implementar as seguintes versões:

   (a) **Versão Primeira Escolha**: o tabuleiro sucessor T_s do tabuleiro corrente T_c será o primeiro vizinho de T_c que tem uma avaliação melhor. Assim, se a avaliação do tabuleiro corrente T_c for igual a k, o primeiro tabuleiro vizinho de T_c encontrado com avaliação menor que k deve passar a ser o novo tabuleiro corrente. Você deve usar a função $umVizinho$ para gerar um único vizinho do nó T_c. O programa deve parar quando não houver nenhum vizinho que melhore a avaliação do nó corrente T_c.

   (b) **Versão Melhor Escolha**: todos os tabuleiros vizinhos do tabuleiro corrente T_c são gerados e avaliados. O novo tabuleiro corrente deve ser aquele que mais melhora a avaliação do tabuleiro corrente T_c. No caso de haver mais de um tabuleiro, a escolha deve ser feita de forma aleatória. O programa deve parar quando não houver nenhum vizinho que melhore a avaliação do nó corrente T_c.

   (c) **Análise**: Produza um relatório com o desempenho de cada umas das implementações. Considere:

- O tabuleiro inicial deve ser gerado usando a função *tabuleiro*.
- A avaliação dos tabuleiros deve ser feita considerando a função *numeroAtaques*.
- A análise deve ser feita para tabuleiros de tamanho 4, 8, 16 e 32.
- Para cada tamanho de tabuleiro:
  - Execute o programa uma única vez e gere um gráfico com os valores da função de avaliação obtidos nesta execução. O que você pode concluir?
  - Caso você não tenha encontrado a solução do problema na primeira execução, execute novamente o programa até que uma solução seja encontrada. Gere o gráfico da solução encontrada com os valores da função de avaliação obtidos nesta execução. Quantas vezes você precisou executar o programa até que o resultado fosse encontrado?

(d) Quais conclusões você consegue tirar destes experimentos? Quais propostas você faria para tentar melhorar o resultado usando o Hill Climbing?

4. **Simulated Annealing**

(a) Implemente o algoritmo Simulated Annealing apresentado no slide da aula correspondente (página 5).

ATENÇÃO: a linha

"if $e^{-\Delta/t} >$ random(0,1) then ..."

é na verdade:

"if $e^{-\Delta/Temp} >$ random(0,1) then ..."

Os parâmetros de entrada do programa devem ser *temperatura inicial* ($TempInicial$), *o número máximo de iterações* ($MaxIt$) e *o fator de decaimento* (α). Você deve usar a modelagem do tabuleiro e as funções *tabuleiro*, *todosVizinhos* e *numeroAtaques* definidas anteriormente.

Inclua as seguintes alterações na sua implementação:

- **Número de melhoras**: Conte quantas vezes há troca do nó corrente por um vizinho que seja melhor que ele (quando $\Delta \leq 0$). Esta informação deve ser apresentada no final da execução.
- **Número de trocas aleatórias**: Conte quantas vezes há troca do nó corrente por um vizinho que seja pior que ele (quando $\Delta > 0$). Esta informação deve ser apresentada no final da execução.

  Sempre que houver uma troca aleatória, imprima (durante a execução) os valores da função de avaliação do nó corrente atual e do próximo nó corrente e de $e^{-\Delta/Temp}$.

(b) A análise deve ser feita para tabuleiros de tamanho 4, 8, 16 e 32.

(c) Considere o caso das 4 rainhas.

- Execute o programa 10 vezes com os seguintes valores $MaxIt = 50$, $TempInicial = 100$ e $\alpha = 0.9$. Em quantas execuções você obteve uma resposta? Em cada execução que encontrou uma solução, quantas iterações foram necessárias até que a solução fosse encontrada?
- Aumente o número de iterações para 100 ($MaxIt = 100$) e faça outras 10 execuções. Em quantas execuções você obteve uma resposta? Em cada execução que encontrou uma solução, quantas iterações foram necessárias até que a solução fosse encontrada?
- Fixando o número de iterações em 50 e 100 ($MaxIt = 50$ e $MaxIt = 100$), faça 10 execuções do programa, variando os parâmetros $TempInicial$ e α. Altere apenas um parâmetro por vez. Comente os resultados obtidos.

(d) Repita as análises para os casos de 8, 16 e 32 rainhas. Use a experiência do item anterior para definir o valor dos parâmetros do programa.

5. **Algoritmo Genético**

(a) Você deve implementar o algoritmo genético básico apresentado em aula. Você deve usar a modelagem do tabuleiro e as funções *tabuleiro* e *numeroAtaques* definidas anteriormente.

(b) Defina uma função que dado o tamanho n de uma população, gera aleatoriamente um conjunto de n indivíduos.

(c) **Operadores**: Defina as seguintes funções:

- função de adaptação usada para avaliar um tabuleiro T.
- função que dada uma população P, constrói a roleta viciada correspondente a P.
- função que dada uma população P contrói uma população intermediária correspondente a P. (**seleção**)
- função que faz o crossover entre dois indivíduos.(**crossover**)
- função que faz a mutação em um dado indivíduo. (**mutação**)

(d) **Algoritmo Genético Básico**

- Sua implementação deve ter como parâmetros de entrada:
  - Tamanho da população
  - Número de gerações
  - Probabilidade de Crossover
  - Probabilidade de Mutação
  - Utilização de Elitismo: no elitismo, uma cópia do melhor indivíduo da geração P_{i-1} é passada para a geração P_i, sem passar pelos operadores.

  A saída de uma execução do programa deve ser composta por 2 gráficos:
  - geração × função de adaptação do melhor indivíduo da geração
  - geração × média da função de adaptação dos indivíduos da geração

  e o melhor indivíduo da última geração e o valor da sua função de adaptação.

(e) Considere o problema das 4 rainhas.

- Defina um conjunto de valores para os parâmetros tamanho da população, número de gerações, probabilidade de crossover e probabilidade de mutação, e execute o programa 10 vezes sem elitismo e 10 vezes com elitismo. Alguma solução foi encontrada? Comente os resultados obtidos.
- Repita o caso anterior, alterando os valores dos parâmetros de entrada (um de cada vez) e verifique o que ocorre. Execute o programa 10 vezes para o caso com elitismo. Justifique os valores escolhidos e comente os resultados obtidos.

(f) Faça uma análise para os tabuleiro com 8, 16 e 32 rainhas.