Problema das N-Rainhas - AA4

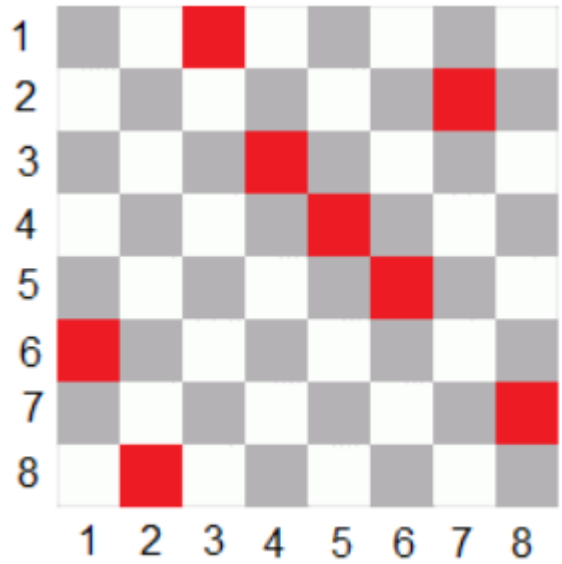
Proponha uma adaptação da metaheurística Busca Tabu para o problema das N-rainhas.Você deve projetar aadaptação seguindo todos os componentes essenciais, listados abaixo. Componentes adicionais que sejam adotados também devem ser explicados em detalhes.

- a.Defina a representação a ser utilizada.
- b.Defina uma função de avaliação adequada.
- c.Defina pelo menos uma forma de gerar uma solução inicial (heurística construtiva).
- d.Defina pelo menos uma vizinhança para o problema.0
- e.Projete uma adaptação metaheurística Busca Tabu, garantindo que todos seus compo-nentes sejamconsiderados.

Representação do Problema

O problema das n-rainhas é demonstrado em um tabuleiro de xadrez, então uma maneira simples de moldá-lo é atribuir números para as linha e coluna no tabuleiro de 1 até N.

Conjuntos com N tuplas seriam representações de N rainhas em um tabuleiro. Por exemplo: `{(1,3), (2,7), (3,4), (4,5), (5,6), (6,1), (7,8), (8,2)}` é uma representação para um conjunto com N=8.



Neste problema existe a restrição de quenenhuma rainha poderá atacar outra, de forma que torna-se 3 restrições: sem rainhas na mesma linha, coluna ou diagonal. Com a modelagem usada acima é simples identificar as duas primeiras restrições, pois, rainhas na mesma linha ou coluna terão valores iguais nas respectivas posições em suas tuplas.

Função de avaliação

Cada solução candidata do problema pode ser avaliada em função do número restrições infringidas, ou seja, rainhas na mesma linha, coluna ou diagonal.

O número máximo de colisões possíveis para um tabuleiro NxN é dada pela seguinte fórmula: $kmax = N*(N-1)$. Assim, a FO pode ser definida como uma função de maximização para `f(x) = kmax - k`. Em que

k é o número de vezes que as restrições foram infringidas.

Para identificar a quebra de uma restrição de linha ou coluna é bastante simples, deve-se verificar se a linha ou a coluna das posições de duas rainhas é igual, assim encontra-se uma colisão de linha ou coluna.

Para identificar as colisões nas diagonais é preciso assumir que existem duas diagonais, a diagonal principal e a diagonal secundária. Para encontrar a primeira basta subtrair numa coordenada a linha da coluna, assim fazendo `linha - coluna = diagonal_principal`. Já a segunda, o contrário deve ser feito, obtendo`linha + coluna = diagonal_secundaria`.

Assim, para duas rainhas nas coordenadas `{5,4} e {2,7}`, sendo diagonal principal = d_p e diagonal segundaria = d_s:

```
d_p1 = 5 - 4 = 1
d_p2 = 2 - 7 = -5

d_p1 != d_p2 # Não há colisão na d_p
```

```
d_s1 = 5 + 4 = 9
d_s2 = 2 + 7 = 9

d_s1 == d_s2 # Há colisão na d_s
```

Solução Inicial

Para necessariamente evitar problemas com colisão nas linhas, as tuplas são iniciadas com as linhas eumeradas de 1...N. E para evitar a colisão de colunas a função leva em consideração se já existe alguma rainha naquela coluna, restando apenas as colisoes nas diagonais para calcular a FO.

Pseudo-código para construção da Solução Inicial

```
verifica_coluna_tupla(lista_tuplas, tupla_candidata):
    for tupla in lista_tuplas:
      if tupla_candidata.column == tupla.column:
          return true # coluna já utilizada

    return false #tupla candidata não foi inserida


solucao_inicial(lista_tuplas, N):
  for i = i,.., N:
    tupla_candidata = {i, rand(1,..N)} #Seta Candidata
    while not verifica_coluna_tupla(lista_tuplas, tupla_candidata):
        tupla_candidata = {i, rand(1,..N)} #Seta Candidata
    lista_tuplas.append(tupla_candidata)
```

Vizinhança

Neste caso foi escolhido a utilização do melhor aprimorante, e o movimento é definido pela troca de índices de forma linear no vetor de tuplas, ou seja, dado i = 1,...,N, a troca será feita entre i e i + 1. A lista tabu irá guardar a solução completa.

```
  verifica_lista_tabu(lista_tuplas, lista_tabu):
    for sol in lista_tabu:
      if sol == lista_tuplas:
        return true
    return false


  melhor_vizinho_BT(lista_tuplas, lista_tabu):
    melhor_viz = lista_tuplas
    fo_aux = -inf

    for i = 1,...,N :
        alterna_linhas(lista_tuplas[i], lista_tuplas[i + 1])
        if verifica_lista_tabu(lista_tuplas, lista_tuplas[i]) and fo(x) > fo_aux:
          fo_aux = fo(x)
          melhor_viz = lista_tuplas
        alterna_linhas(lista_tuplas[i], lista_tuplas[i + 1])

    return melhor_viz
```

Busca Tabu

O critério de parada é um parametro maxIt, e o tamanho da lista tabu foi definido como N/2, sendo retirada a solução mais antiga quando houver a necessidade de substituição.

```
# inicializa linhas lista de tuplas
for i in range(N):
  lista_tuplas[i].line = i

busca_tabu(N, lista_tuplas, maxIt):

  sol = solucao_inicial(lista_tuplas, N)
  sol_aux  = sol
  lista_tabu.append(sol)


  while it < maxIt:
    sol = melhor_vizinho_bt(sol, lista_tabu)
    if fo(sol) > fo(sol_aux):
      sol_aux = sol
    lista_tabu.append(sol)
```

```
      if len(lista_tabu) > N/2:
        retira_elemento_mais_antigo(lista_tabu)
      it += 1
    return sol_aux
```