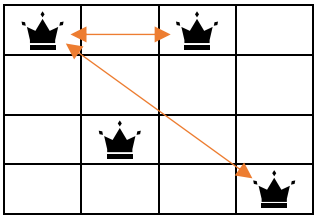
Tarefa AG: Problema das N Rainhas

Objetivos de Aprendizagem

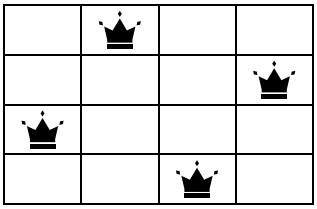
- Compreender algoritmos genéticos na sua forma canônica
- Compreender a importância dos parâmetros de configuração em AGs
- Compreender como realizar avaliação de AGs

Enunciado

Resolva o problema das N rainhas para N=10. O objetivo é encontrar uma configuração do tabuleiro onde não há ataques entre rainhas. Rainhas se atacam se estiverem na mesma linha, coluna ou diagonal. Veja o exemplo reduzido para N=4.



Estado com dois ataques entre rainhas



Estado objetivo: zero ataque entre rainhas

Objetivo da tarefa

Analisar e comparar o comportamento de duas implementações de um AG canônico para o problema em questão:

1. implementação com uma função de **reparação** de cromossomos infactíveis e
2. implementação com uma função de **penalização** de cromossomos infactíveis.

Cromossomos infactíveis são aqueles que representam tabuleiros onde as rainhas.

Método

Faça os passos abaixo para a implementação com reparação de indivíduos infactíveis e, depois, para a implementação com penalização. Elas serão comparadas.

a) Implemente os passos básicos do AG canônico conforme visto em sala (as etapas de reparação/penalização ocorrem após a geração dos filhos e antes da seleção dos sobreviventes para a próxima geração). *Você pode utilizar o esqueleto de implementação fornecido pelo professor ou fazer um a partir do zero.*

b) Escolha uma configuração para o AG constituída de:
   - máximo execuções (MAX_EXECUCOES)
   - tamanho da população (TAM_POP)
   - máximo de gerações por execução (MAX_GERACOES)
   - probabilidade de crossover (PROB_CROSS)
   - probabilidade de mutação (PROB_MUT)

Esta escolha de configuração deve ser feita após alguns testes preliminares (neste processo avalie a influência dos parâmetros nos resultados). Condição de parada definir como sendo o máximo de gerações e/ou fitness = <valor>.

c) Para cada configuração, execute **MUITAS** vezes (MAX_EXECUCOES) o algoritmo com diferentes sementes do gerador aleatório

d) Para cada execução, guarde *<ex_i, ct_call_fitness, fitness_best_ind_ex_i, best_ind_ex_i> tal que:*
   - *ex_i: número sequencial da execução*
   - *ct_inv_fitness: número de vezes que o método de fitness foi invocado*
   - *fit_melhor_crom: fitness do melhor cromossomo da execução i*
   - *melhor_crom: melhor cromossomo da execução i*

e) Ao final da última execução, calcule a média dos *fit_melhor_crom* sendo esta a métrica de qualidade do algoritmo para a configuração escolhida.

f) Ao final da última execução, dê o número de soluções encontradas dentre as MAX_EXECUCOES (neste problema, o fitness máximo é conhecido).

Para quem for usar o código fornecido pelo professor

Baixe os códigos fonte em Java para a tarefa em questão. Há dois pacotes:

- **ag**: *contém os códigos do algoritmo genético*
  - **AG.java**: executa um AG até que a condição de parada seja atingida
  - **Cromossomo.java**: contém a codificação de um cromossomo, métodos de cálculo de fitness, reparação e penalização
  - **Operadores.java**: contém os operadores de crossover e mutação
  - **ConfigAG.java**: classe de interface que contém os parâmetros de configuração do AG e de codificação do cromossomo
- **agNRainhas**: *contém a classe principal*

a) o número sequencial da geração (id) e o fitness do melhor <u>indivíduo por geração</u>. Por exemplo, abaixo, na primeira iteração o método de cálculo de fitness foi invocado 17.280 vezes, o valor de fitness do MELHOR CROMOSSOMO foi de 43 e as posições subsequentes representam a posição das rainhas em cada uma das colunas.

```
run:
1,17280,43.0,0,9,6,3,5,7,1,4,6,8
2,17280,44.0,4,7,4,2,9,6,1,3,8,0
3,17280,44.0,2,2,5,7,9,3,6,4,1,8
4,17280,44.0,3,8,0,4,1,7,2,6,2,9
5,17280,44.0,7,2,4,6,9,0,3,1,8,5
...
MAX_EXECUCOES
```

PARA IMPLEMENTAR:

1. Na classe **ConfigAG.java** defina a configuração para o AG
2. Na classe **Cromossomo.java** implemente os seguintes métodos:
   a. private void reparar()
   b. private int infactível()
   c. protected void calcularFitness()
   d. public void inicializarCromossomo()
3. Na classe **AGNRainhas.java**, defina o parâmetro MAX_FIT
4. Na classe AG.java, para coletar dados de uma execução tirar os comentários da linha abaixo a esta:
   // Para coletar dados geracao a geracao de uma execucao especifica descomentar...

PARA ENTREGAR:

1. O que é um estado neste problema?
2. Qual o tamanho do espaço de estados?
3. Explique o método de cálculo de fitness utilizado.
4. Como você codificou um cromossomo (quantos genes, quantos locus por gene e quais os valores possíveis para os alelos)?
5. Na codificação que você adotou é possível que o AG produza cromossomos infactíveis (que não representam um estado do problema)? Explique. Como fica o cálculo de fitness para cromossomos infactíveis?
6. Caso tenha respondido sim na pergunta anterior, como você lidou com esta situação?
7. Escolha três configurações diferentes para o AG e execute **5.000** vezes para cada uma delas. Calcule a métrica de qualidade do algoritmo para cada configuração. Faça isto para cada uma das implementações (penalização e reparação) utilizando as mesmas configurações. **Compare os valores da métrica de qualidade obtidos** na forma de uma tabela e, também, analise com base nas configurações escolhidas:
    a. qual parâmetro do AG influenciou mais na qualidade das soluções obtidas (observar o número de soluções encontradas e a média de fitness das execuções). Há uma tendência de penalização ou reparação produzirem melhores resultados?
    b. qual configuração produziu melhor custo computacional-qualidade da solução. Há uma tendência de penalização ou reparação produzirem melhores resultados?

	PENALIZAÇÃO			**REPARAÇÃO**		
CONFIGURAÇÕES	**1**	**2**	**3**	**1**	**2**	**3**
TAM_POP						
MAX_GERACOES						
PROB_CROSS						
PROB_MUT						
Média fitness das execuções						
Soluções (inclusive repetidas)						
Desempenho (soluções/MAX_EXECUCOES)						
Número chamadas fitness						
Soluções/1.000 chamadas fit						
Tempo (s)						

8. Liste três soluções encontradas para o problema das 10 rainhas.