Procura Heurística e Estocástica:

1. Procura Heurística:

↳ Usa informação / conhecimento sobre o problema para fazer uma procura mais eficiente e eficaz. Este conhecimento caracteriza-se por:

- **Custo:** (1) $g(k)$: custo desde o início até ao nó k.
  (2) $h(k)$: custo estimado desde o nó atual até ao objetivo
  (3) $f(k) = g(k) + h(k)$: custo total estimado.
- **Visão:** (1) Local: Foco apenas nos vizinhos do nó atual.
  (2) Global: Considera o estado geral do espaço de procura
- **Memória:** Quantidade de info que o algoritmo, armazena, muitas vezes limitada.

1.1. Trepa-Colinas:

- **Visão:** Local.
- **Memória:** Limitada.
- **Custo:** Usa $h(k)$.

Pega no nó com menos custo h derivado da expansão do atual.

1. nó_corrente (pega no nó inicial)
2. Loop
   2.1. nó_seguinte = Best(h, expand(nó_corrente)).
   2.2. Se h(nó_seguinte) > h(nó_corrente), devolve nó_corrente
   2.3. nó_corrente = nó-seguinte
3. Fim da função.

- **completo:** Não (pode ficar preso em máximos locais).
- **Discriminador:** Não
- **Espaço mínimo:** O(1)
- **Tempo mínimo:** $O(r \times n)$, $r \rightarrow$ número de estados sucessores possíveis.
  $n \rightarrow$ é a profundidade.

1.2. Pesquisa sôfrega (Greedy Search):

- **Visão:** Global
- **Memória:** Não limitada
- **Custo:** usa $h(k)$

1. nós (cria uma lista ordenada apenas c/ nó inicial).
2. Loop
   2.1 Se nós está vazio, devolve falha.
   2.2. nó = nós [0]
   2.3. Se Teste(nó) devolve nó
   2.4. nós.append(nó.expand) → Dá appena mas já com a ordem heurística (menor→maior)
3. Fim Função.

- completo : Não.
- Discriminador : Não.
- Espaço mínimo: O (r^n), r → número de estados sucessores possíveis.
- Tempo mínimo: O(rm) , n → é a profundidade.

1.3. Pesquisa A* (A-Star):

- Visão : Global.
- Memória : Não limitada.
- CUSTO : Usa f(k).

1. nós (cria lista c/ nó inicial sempre ordenado por f(k)) ← Crescente
2. Loop
   - 2.1 Se nós está vazio devolve falha.
   - 2.2. nó = nós [0].
   - 2.3. Se Teste (nó) devolve nó.
   - 2.4. nós. append (nó. expand) → coloca já ordenado.
3. Fim da Função

- completo : Sim, se $f(x) \leq g(x,y) + h(y)$, h(k) não sobrestima o custo real até ao objetivo, r é finito e o custo de transição c(n,m) é positivo.
- Discriminador : Sim.
- Espaço mínimo: Exponencial.
- Tempo mínimo: Exponencial.

2. Procura Estocástica

↳ Questiona se faz sentido explorar a aleatoriedade, quando métodos deterministicos (cegos e heuristicos) podem falhar.

2.1. Procura Aleatória (Random Search):

- Visão : Global.
- Memória : Não limitada.

↳ Dá jeito para espaços de procura gigantes c/ poucas soluções (rainhas)

1. Tree (inicia árvore apenas com o nó inicial).
2. Loop
   - 2.1 Se Não dá para expandir mais nós, devolve falha
   - 2.2. nó = random_nó (Tree)
   - 2.3. Se nó for objetivo, devolve solução correspondente.
   - 2.4. Tree. append (nó. expand).
3. Fim da Função

- completo : Não
- Discriminador : Não.
- Espaço mínimo: Exponêncial.
- Tempo mínimo: Exponêncial.

2.2. Recristalização Simulada (simulated Annealing):

- Visão: Local
- Memória: Não limitada
- Custo: usa h(k)

1. nó_corrente ← estado inicial
2. From t=1 to ∞:
   - 2.1. T = pol_Escalona(t)
   - 2.2. se: T = ∅, devolve nó_corrente.
   - 2.3. nó_seguinte = Random(nó_corrente.expand) — escolhe 1 sucessor aleatório.
   - 2.4. Se: h(nó_seguinte) > h(nó_corrente), nó_corrente = nó_seguinte
     senão: nó_corrente = Prob(nó_seguinte, $e^{\Delta e/T}$).
3. Fim da função

- Completo: Não.
- Discriminador: Não.
- Espaço Mínimo: O(1)
- Tempo Mínimo: O(kxn)

Como escolher o melhor Algoritmo?

(1) Usar o algoritmo mais simples!
(2) Posso usar info do problema? Heuristica Vs Cega
(3) Posso definir heurística fixe? A*
(4) Espaço de procura grande?