

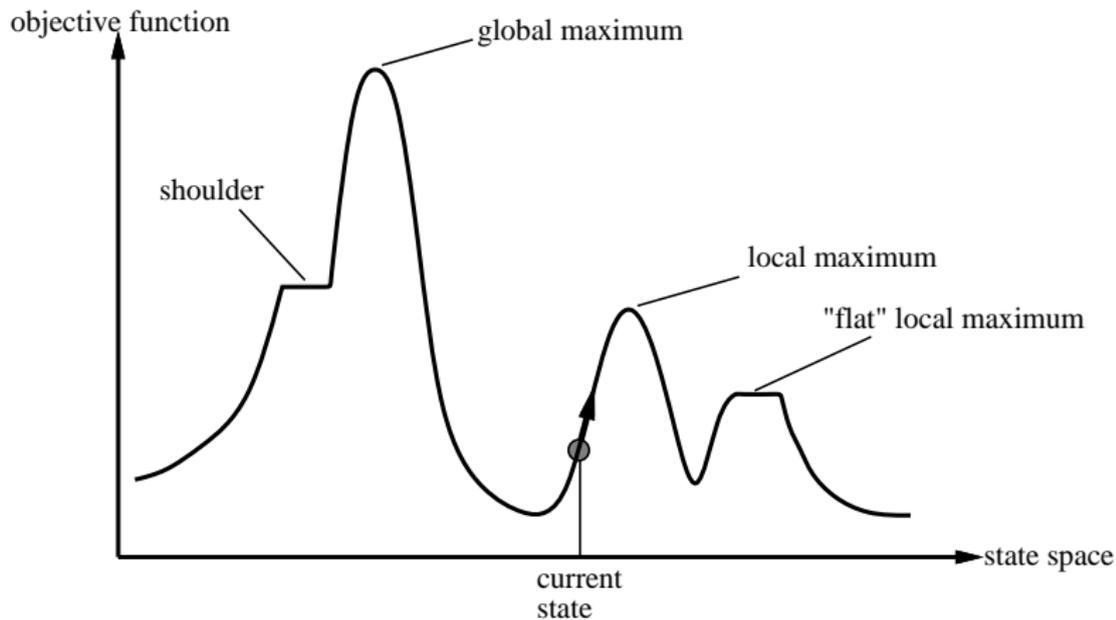
# Estratégias de controle para a procura

- irrevogáveis: nunca retornam por um caminho já explorado
- tentativa: “backtracking” (métodos não informados e informados).

# Algoritmos de Melhoria Iterativa

- Utilizados em problemas cuja descrição já contém toda a informação para encontrar a solução (ex: n-rainhas e layout de circuitos VLSI)
- parte-se de uma config inicial conhecida e tenta-se melhorar a solução.

# Algoritmos de Melhoria Iterativo



# Algoritmos de Melhoria Iterativa

- Exemplos:
  - ▶ Método Irrevogável: Hill-Climbing (busca de máximo/mínimo local)
  - ▶ Métodos de tentativa: (busca de máximo/mínimo global) podem temporariamente tornar estados piores.
    - random restart-hill-climbing
    - simulated annealing
    - busca tabu
    - busca paralela

# Hill Climbing (ou gradiente descendente)

- tenta fazer modificações que melhorem o estado corrente
- 2 desvantagens:
  - ▶ máximos locais
  - ▶ platôs: percorre estados aleatoriamente porque a função de avaliação não muda muito
- Exemplo: Jogo dos oito :-)

## Hill Climbing: exemplo

2 8 3	2 8 3	2 3	2 3	1 2 3	1 2 3
1 6 4	1 4	1 8 4	1 8 4	8 4	8 4
7 5	7 6 5	7 6 5	7 6 5	7 6 5	7 6 5
f=-4	f=-3	f=-3	f=-2	f=-1	f=0

- No caso de não se poder aplicar a regra, o processo termina e a solução é um máximo local.
- **OBS:** este exemplo é baseado no algoritmo de hill-climbing da primeira edição do livro do Russell and Norvig. O algoritmo da segunda edição foi modificado fazendo com que este exemplo termine no segundo nó, sem encontrar o máximo global (f=0).

## Hill Climbing: exemplo

- não funciona para:

1	2	5		1	2	3
	7	4	=>		7	4
8	6	3		8	6	5

f=-2

- Qq movimento diminui o valor da função de avaliação.

# Random Restart Hill Climbing

- executa uma série de buscas hill-climbing a partir de estados iniciais aleatórios
- cada um roda até terminar ou até não ter nenhum progresso
- salva o melhor resultado obtido até então
- pode ter no. finito de iterações ou continuar até não conseguir melhorar o melhor valor encontrado
- se superfície de busca contém muitos máximos locais, busca exponencial
- geralmente, solução boa pode ser encontrada em um número pequeno de iterações.

# Simulated Annealing

- No lugar de começar novamente aleatoriamente quando passa num máximo local, permite que a busca escape do máximo local “descendo a montanha”.

# Simulated Annealing

```
function SA(problem,schedule) return a solution state
  current <- MAKE_NODE(INITIAL_STATE[problem])
  for t <- 1 to infinito do
    T <- schedule(t)
    if T = 0 then return current
    next <- sucessor de current selecionado
      aleatoriamente
    deltaE <- value[next] - value[current]
    if deltaE > 0 then current <- next
    else current <- next with prob  $e^{-(\text{deltaE}/T)}$ 
    endif
  endfor
```

# Simulated Annealing

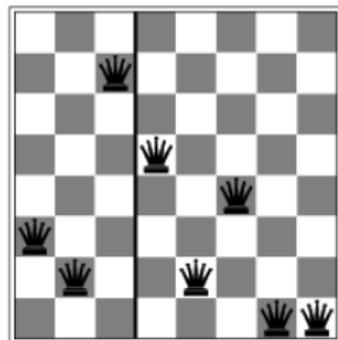
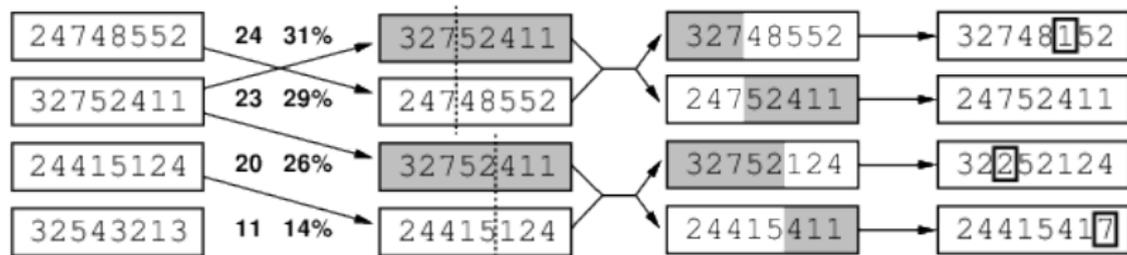
```
P = e^(deltaE/T)
n = sorteio de um no. de 0 a 1
if n < P then current <- next
```

- ou seja: quanto maior a probab mais chance de aceitar mover para passos de custo pior.

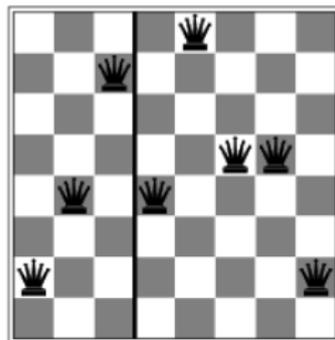
# Outros Algoritmos

- Algoritmos Genéticos
  - ▶ Operações: “crossover”, mutação e reprodução
  - ▶ começa de uma população inicial
  - ▶ aplica as operações
  - ▶ calcula uma função de “fitness” para cada indivíduo da população
  - ▶ pode eliminar indivíduos menos “fit”

# Algoritmos Genéticos



+



=

