Inteligência Artificial

Aula 05- Solução de Problemas (Busca com Informação - Algoritmos por Refinamentos Sucessivos)[1]

Sílvia M.W. Moraes

Faculdade de Informática - PUCRS

March 26, 2018

Sinopse

- Nesta aula, apresentamos uma **introdução a solução de problemas por algoritmos de busca local por refinamentos sucessivos**.
- Este material foi construído com base nos capítulos:
  - 4 do livro Artificial Intelligence – a Modern Approach de Russel & Norvig
  - 4 do livro Inteligência Artificial de Luger.

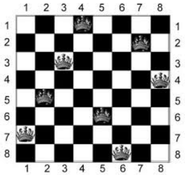
Sumário

Aulas anteriores

- Agente
    - Reativos
    - Cognitivos
- Solução de Problemas
    - Representação, Espaço de Estados
    - Busca Cega
        - Plano: sequência de ações
    - A*

Algoritmos de Busca Local por Refinamentos Sucessivos

- Os algoritmos vistos até agora exploram sistematicamente espaços de busca mantendo na memória um ou mais caminhos que levam do estado inicial do problema até o estado objetivo.
- **Para muitos problemas**, **o caminho até a solução é irrelevante**.
    - **Exemplos:** Problema das 8 rainhas, projeto de circuitos integrados, layout de instalações industriais, escalonamento de jornadas de trabalho, roteamento de veiculos, alocação de recursos, etc.



- Para esses problemas, usamos **algoritmos de busca local por refinamentos sucessivos**.

Características

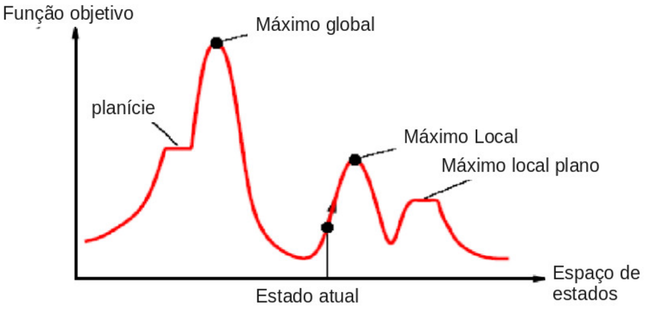
- **Algoritmos de busca local por refinamentos sucessivos**, em geral:
  - **partem de soluções propostas** e tentam melhorá-las.
  - **operam sobre um único estado corrente**, ao invés de vários caminhos.
  - **se movem** apenas para os **vizinhos** desse estado.
  - **não mantêm a árvore de pesquisa** (não consideram o caminho para solução), guardam apenas os estados e suas avaliações.
  - são úteis para resolver **problemas de otimização**.

Características

- Algoritmos de busca local por refinamentos sucessivos
  - **Vantagens:**
    - ocupam **pouca memória**
    - podem **encontrar soluções razoáveis em grandes ou infinitos espaços de estados**, para os quais os algoritmos sistemáticos são inadequados.

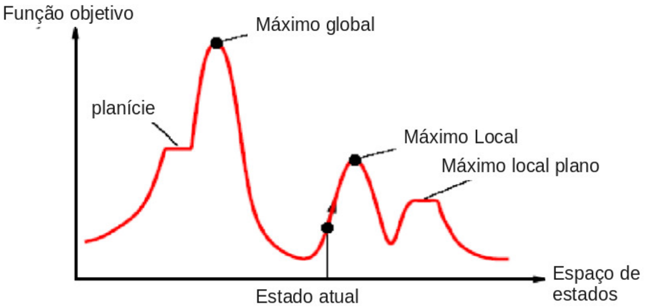
Espaço de Busca

- **Topologia** consiste em:
  - **estado** (posição)
  - elevação definida pelo **valor da função** heurística (mínimo global) ou função objetivo (máximo global).



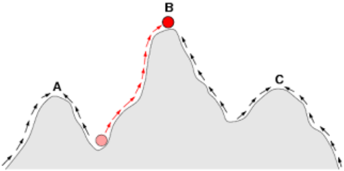
Espaço de Busca

- **Máximo global**: pico mais alto
- **Máximo local:** picos mais altos que os vizinhos, mas menor que o global.
- **Platôs** - planíces e máximos locais planos:
  - planície: plano em que é possível progredir.
  - máximo local plano: plano em que não há saída.



Alguns Algoritmos de Busca por Refinamentos Sucessivos

- **Hill-climbing** (Subida da encosta): o algoritmo tenta realizar mudanças que podem melhorar o estado corrente.
- **Simulated annealing** (Têmpera simulada): podem realizar mudanças que podem piorar o estado corrente, temporariamente.



Algoritmo Hill-Climbing

- **Hill-climbing** também chamado de subida da encosta, busca gulosa local ou ainda gradiente descente (função heurística). O algoritmo
  - consiste em um laço que continuamente move-se na direção de um valor melhor (encosta acima).
  - termina quando encontra um pico em que nenhum vizinho tem um valor mais alto.
  - não precisa manter a árvore de pesquisa, guarda apenas o estado atual e o valor da sua função objetivo (ou heurística).
  - examina apenas os estados vizinhos imediatos ao estado atual.

Algoritmo Hill-Climbing

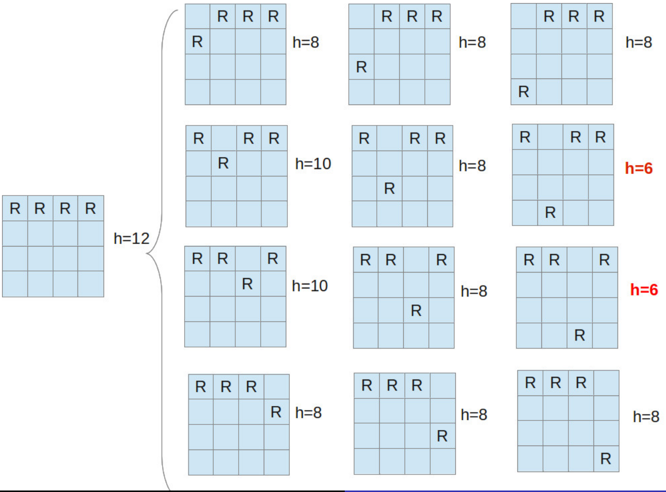
```
função SUBIDA-DE-ENCOSTA() retorna um estado-solução{
    entrada: umProblema
    variáveis locais: nóAtual, nóVizinho

    nóAtual = CRIAR-NÓ(ESTADO-INICIAL(umProblema))
    repetir{
        nóVizinho = um sucessor do nóAtual com melhor avaliação
        se valor(nóVizinho) < valor(nóAtual)
        então nóAtual = nóVizinho
        senão retornar ESTADO(nóAtual)
    }
}
```

Algoritmo Hill-Climbing

- Precisa de uma **função heurística para avaliar as soluções**.
- O nó criado inicialmente corresponde à uma **solução gerada aleatoriamente**.
- Precisa de uma função **sucessor** que gera uma solução próxima à atual: introduz uma "perturbação" (modificação) na solução atual, gerando outra solução.
- O algoritmo executa enquanto houver uma mudança significativa.

Algoritmo Hill-Climbing

- **Exemplo** - Problema das 4 rainhas
  - Vizinhos ao estado atual



Algoritmo Hill-Climbing

- **Exemplo** - Problema das 8 rainhas
  - Versão 1 : implementação tradicional do algoritmo (sucessor melhor em amplitude, variando a linha de uma rainha)

```
Hill-Climbing Simples
Dimensão: 8
Ciclo: 1 - h=17     Ciclo: 2 - h=12     Ciclo: 3 - h=7
- - - - - - - -     - - - - - - - -     - * - - - - - -
- - - - - - - -     - - - - * - - -     - - - - * - - -
- - - - - - - -     - - - - - - - -     - - - - - - - -
- - - * - - - -     - - - * - - - -     - - - * - - - -
* - - - * - - -     * - - - - - - -     * - - - - - - -
- * - - - * - *     - * - - - * - *     - - - - - * - *
- - * - - - * -     - - * - - - * -     - - * - - - * -
- - - - - - - -     - - - - - - - -     - - - - - - - -

Ciclo: 4 - h=3      Ciclo: 5 - h=2      Ciclo: 6 - h=1
- * - - - - - -     - * - - - - - -     - * - - - - - -
- - - - * - - -     - - - - * - - -     - - - - * - - -
- - - - - - - -     * - - - - - - -     * - - - - - - -
- - - * - - - -     - - - * - - - -     - - - * - - - -
* - - - - - - -     - - - - - - - -     - - - - - * - -
- - - - - * - *     - - - - - * - *     - - - - - - - *
- - * - - - - -     - - * - - - - -     - - * - - - - -
- - - - - - * -     - - - - - - * -     - - - - - - * -
```

Algoritmo Hill-Climbing

- **Exemplo** - Problema das 8 rainhas
  - Versão 2: Implementação do algoritmo com uma pequena variação na função sucessor (considera melhor em profundidade):
    - dada a configuração inicial, o movimento da próxima rainha será a partir do melhor estado da rainha anterior.

```
Hill-Climbing Simples
Dimensão: 8
Ciclo: 1 - h=17      Ciclo: 2 - h=4       Ciclo: 3 - h=1
- - - - - - - -      - - - * * - - -      - - - - * - - -
- - - - - - - -      * - - - - - * -      - - - - - - * -
- - - - - - - -      - * - - - - - -      * * - - - - - -
- - - * - - - -      - - - - - - - -      - - - * - - - -
* - - - * - - -      - - - - - * - -      - - - - - * - -
- * - - - * - *      - - - - - - - *      - - - - - - - *
- - * - - - * -      - - * - - - - -      - - * - - - - -
- - - - - - - -      - - - - - - - -      - - - - - - - -
```

Algoritmo Hill-Climbing

- **Exemplo** - Problema das *n* rainhas
  - Teste do algoritmo Versão 2 com quantidades diferentes de rainhas

#rainhas	hInicial	hFinal	#ciclos
20	22	2	3
50	64	2	4
100	116	2	4
200	215	2	4

Algoritmo Hill-Climbing

- **Variantes do algoritmo**:
  - **Subida da Encosta Estocástica**: escolhe ao acaso entre os movimentos possíveis de encosta acima.
    - A probabilidade de seleção pode variar com a declividade do movimento.
    - Em geral converge mais lentamente que a subida mais íngreme (algoritmo usual).
  - **Subida da Encosta pela primeira escolha**: gera sucessores ao acaso até ser gerado um sucessor melhor que o estado atual.
    - Boa estratégia, quando há muitos sucessores (ex: milhares).

Algoritmo Hill-Climbing

- **Variantes do algoritmo (continuação)**:
  - **Subida da Encosta com reinício aleatório:** Consiste em várias buscas (executa várias vezes o algoritmo) a partir de estados iniciais gerados ao acaso, parando quando encontrar o objetivo.
    - Diferente das versões anteriores encontra o objetivo (as anteriores podem ficar presas em máximos locais).
- De uma maneira geral, **o sucesso do algoritmo depende da topologia do espaço de estados.**

Algoritmo Hill-Climbing

- **Exemplo** - Problema das *n* rainhas
  - Teste do algoritmo com reinicio aleatório com quantidades diferentes de rainhas

#rainhas	hFinal	#execuções
8	0	7
20	0	20
50	0	28

Algoritmo **Simulated Annealing**

- Este algoritmo de busca local por refinamentos sucessivos também é conhecido como **Têmpera Simulada**.
  - **Inspiração:**
    - Têmpera: processo usado para temperar ou endurecer metais e vidros, aquecendo-os a altas temperaturas e depois resfriando-os gradualmente.
  - **A ideia é combinar o algoritmo subida da encosta com um percurso aleatório que temporariamente pode piorar a solução**, mas que, ao longo da execução, pode encontrar um máximo global.
    - **Tenta resolver o problema do Algoritmo Hill Climbing: máximos locais**.

Algoritmo **Simulated Annealing**

- Implementa um laço de repetição semelhante ao do algoritmo Hill Climbing.
- Diferente do algoritmo Hill Climbing, **escolhe movimento (estado vizinho) sucessor de forma aleatória.**
  - Se o estado vizinho for melhor, será aceito
  - Se o estado vizinho não for melhor, será aceito em menos de 1% dos casos
    - Essa probabilidade diminuirá exponencialmente à medida que os estados piorarem.
    - A probabilidade também diminui à medida que a "temperatura" se reduz (movimentos ruins têm mais chances de serem aceitos no inicio da execução).

Algoritmo **Simulated Annealing**

```
função TEMPERA-SIMULADA() retorna um estado-solução{
    entrada: umProblema
    escalonamento: mapeamento de tempo para "temperatura"
    variáveis locais: nóAtual, nóProximo, T (temperatura atual)

    nóAtual = CRIAR-NÓ(ESTADO-INICIAL(umProblema))
    para t = 1 até ∞{
        T = escalonamento(t)
        se T=0 então retornar ESTADO(nóAtual)
       nóProximo = um sucessor do nóAtual (estado atual) gerado ao acaso
        △E = valor(nóPróximo) − valor(nóAtual)
        se △E > 0 então nóAtual = nóPróximo
        senão nóAtual = nóPróximo somente com a probabilidade e^(−△E/T)
    }
}
```

Algoritmo **Simulated Annealing**

- $\triangle E$: é chamado de variação de energia
- Para calcular a probabilidade pode-se a distribuição de Boltzmann-Gibbs, ou seja, $P(\Delta E) = exp(-\Delta E/T)$.
- O método termina quando
  - nenhuma melhora significativa é alcançada,
  - um número fixo de iterações foi efetuado, ou
  - a temperatura T atingiu seu valor mínimo.
- O parâmetro T é iniciado com um valor elevado e é lentamente reduzido durante o processo de busca.
  - Geralmente, implementa-se um decrescimento geométrico para T, tal como $T = k \times T$.

Algoritmo **Simulated Annealing**

- **Exemplo** - Retomando o problema das *n* rainhas
  - $T_0 = 100$
  - $T = k \times T$, onde $k = 0.5$
  - Critérios de parada:
    - 50.000 iterações
    - ausência de melhora

#rainhas	hInicial	hFinal	#ciclos
8	9	0	1.946
20	23	0	3.938
50	56	0	8.485