

INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

22/23

06. PESQUISA LOCAL – MELHORAMENTO ITERATIVO

Carlos Pereira
ISEC

Índice

2

□ Índice

- ▣ Problemas com restrições
- ▣ Técnicas de Melhoramento Iterativo
 - Trepa-Colinas
 - Recristalização Simulada
 - Pesquisa Tabu

Problemas com Restrições

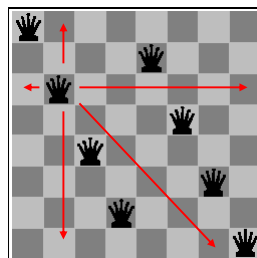
3

- Problema com Restrições (CSP-Constraint Satisfaction Problem)
 - ▣ Trata-se de um problema cuja solução só é válida se satisfazer certas condições
 - Um CSP é caracterizado por:
 - Variáveis: Os seus valores finais representam a solução
 - Domínio: Conjunto de valores que as variáveis podem assumir
 - Restrições: atuam sobre as variáveis
 - Problema: Assinalar valores às variáveis sem violar as restrições

Problemas com Restrições

4

- ...
 - ▣ Exemplo
 - Problema das 8 Rainhas



- Variáveis: Localizações das 8 rainhas no tabuleiro
- Domínio das Variáveis: As coordenadas de cada um dos 64 quadrados do tabuleiro
- Restrições: Quaisquer 2 rainhas (combinações de 8 duas a duas) não podem ficar na mesma linha, coluna ou diagonal

Problemas com Restrições

5

- ...
 - ▣ Num CSP interessa determinar apenas um “estado” final válido e não um caminho que leve a esse estado
 - O “Estado Final” é desconhecido e constitui a solução do problema:
 - ▣ No caso do exemplo das 8 rainhas
 - Dispor 8 rainhas de modo a que nenhuma seja atacada: Interessa apenas conhecer uma disposição das rainhas, um estado final, e não a ordem pela qual se devem colocar no tabuleiro (é indiferente!):
 - O estado final, ou solução, é composto por 8 rainhas dispostas de modo a não haver ataques

Problemas com Restrições

6

- ...
 - ▣ Um CSP pode ser resolvido por técnicas de pesquisa, contudo são geralmente ineficientes neste contexto, dado gerarem muitos estados desnecessariamente.
 - ▣ Existem algoritmos especialmente adaptados à resolução de CSPs:
 - Hill-Climbing
 - Simulated-Annealing
 - Pesquisa Tabu
 - ...

Problemas com Restrições

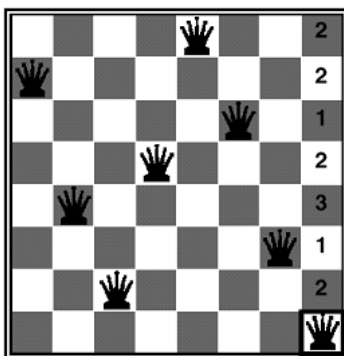
7

- Heurística dos “Conflitos Mínimos”
 - ▣ As rainhas são inicialmente dispostas 1 por coluna, e ao acaso, numa linha qualquer.
 - ▣ Os operadores de modificação movem uma rainha de cada vez.
 - ▣ O movimento faz-se colocando essa rainha na posição em que sofre menos ataques, isto é, produz menos conflitos.

Problemas com Restrições

8

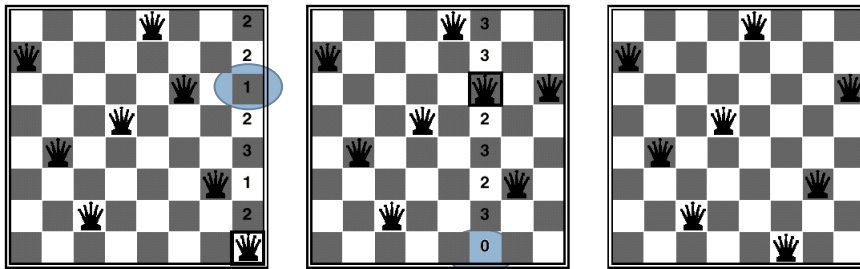
□ ...



Problemas com Restrições

9

□ ...



Melhoramento Iterativo

10

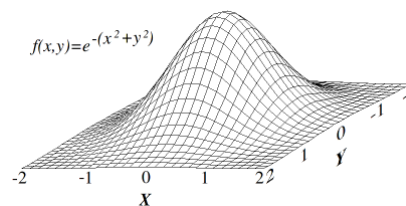
- Os algoritmos de melhoramento iterativo caracterizam-se por:
 - ▣ Não anotam os estados intermédios que conduzem a uma solução, apresentando apenas a “configuração” válida que a compõe.
 - ▣ Partem de uma configuração inicial completa (que viola as restrições), eventualmente gerada aleatoriamente, e melhoram-na sucessivamente até alcançarem uma solução.

Melhoramento Iterativo

11

□ Trepá-Colinas

- ▣ Também conhecido por “Hill-Climbing” ou “Gradient-Descent”
- ▣ O nome e ideia base provem de uma analogia com a decisão tomada por um agente que, perdido numa encosta, pretende atingir o topo:
 - Deslocar-se-á na direção “em que o caminho sobe”.



...Trepá-Colinas

12

□ ...

```

function HILL-CLIMBING(problem) returns a solution state
  inputs: problem, a problem
  static: current, a node
           next, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    next ← a highest-valued successor of current
    if VALUE[next] < VALUE[current] then return current
    current ← next
  end

```

Trepa-Colinas

13

□ ...

□ Implementação

- Parte de um estado inicial dado ou gerado aleatoriamente
 - todas as variáveis com valores atribuídos
- Gera os estados sucessores do estado actual
- Através de uma Função de Avaliação, avalia cada estado assim gerado e escolhe o de maior valor
- Pára quando o estado seleccionado tiver um valor inferior ao escolhido na iteração anterior
 - isto significa que a solução “piorou” e que se “está a descer a colina, em vez de a subir”

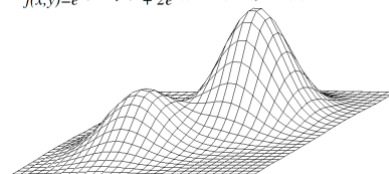
Trepa-Colinas

14

□ Problemas

- Um **máximo local** pode ser atingido sem que corresponda ao máximo absoluto (melhor solução)
- Nos “planaltos” é necessário **escolher uma direcção** aleatoriamente
- Um cume pode ter lados tão inclinados que o passo seguinte conduz ao “outro lado do cume” e não ao seu topo. Neste caso a **solução poderá “oscilar”** nunca atingido o máximo pretendido

$$f(x,y) = e^{-(x^2+y^2)} + 2e^{-((x-1.7)^2 + (y-1.7)^2)}$$



Trepa-Colinas

15

□ ...

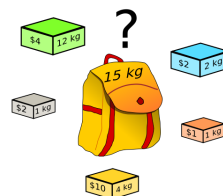
- Tentativa de resolução dos problema relativo ao atingir um ponto de não progresso:
 - Reiniciar a pesquisa partindo de um estado inicial diferente, gerado aleatoriamente (Random-Restart-Hill-Climbing)
 - Guarda o melhor resultado obtido nas pesquisas anteriores
 - até ao ponto de não-progresso
 - Pára quando atingir o número de reinícios máximo ou quando o melhor resultado guardado não for ultrapassado durante “n” iterações (valor “n” é pré-fixado)

Trepa-Colinas

16

□ Problema da Mochila

- Um conjunto de N objectos,
 - caracterizados por um peso e um lucro.
- Uma mochila com capacidade limitada.



- Pretende-se:
 - Encontrar o conjunto ideal de objectos para colocar na mochila;
 - A capacidade não pode ser excedida e o **lucro deve ser máximo**

Trepa-Colinas

17

□ ...

▣ Representação de uma solução

- Os objectos que estão na mochila, sequência binária com N posições (1 significa que o objecto está na mochila)

- Exemplo, para N=8:

■ 1 0 1 0 0 0 0 0

- Esta representação admite soluções inválidas!

Trepa-Colinas

18

□ ...

▣ Como definir a vizinhança de uma solução?

- Adicionar ou remover um objecto

■ Exemplo:

■ Solução actual : 1 0 1 0 0 0 0 0

Adicionar objecto 8

■ Solução vizinha: 1 0 1 0 0 0 0 1

Trepa-Colinas

19

□ ...

□ Como decidir se uma nova solução é relevante para a pesquisa?

■ Através da Função de avaliação

- Associa a cada solução um valor numérico (a sua qualidade)
- Permite comparar soluções alternativas

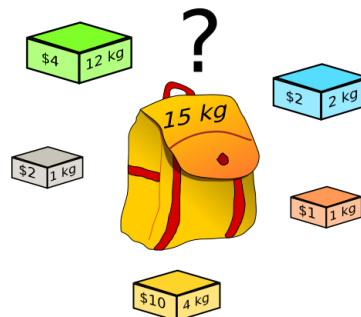
■ Neste caso, devemos considerar o lucro e garantir que a capacidade não é ultrapassada:

$$fitness(x) = \begin{cases} \sum_{i=1}^N x[i] \times L[i] & \text{se } x \text{ for uma solução legal} \\ 0 & \text{se } x \text{ for uma solução ilegal} \end{cases}$$

Trepa-Colinas

20

□ ...



■ Qual a solução ótima?

Trepa-Colinas

21

□ ...

▣ Considere-se outro problema:

■ 8 objectos

■ Capacidade da mochila: 35

■ Propriedades dos objectos

| Objectos | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|----------|----|----|----|----|----|----|----|----|
| Peso | 11 | 18 | 12 | 14 | 13 | 11 | 10 | 16 |
| Lucro | 5 | 8 | 7 | 6 | 9 | 6 | 5 | 3 |

■ Solução inicial: 1 0 1 0 0 0 0 0

■ $Q=12$

Trepa-Colinas

22

□ ...

▣ Iteração 1

■ Vizinhos (considere-se a distância de hamming de 1):

| Vizinhos | Qualidade |
|----------|-----------|
| 00100000 | 7 |
| 11100000 | 0 |
| 10000000 | 5 |
| 10110000 | 0 |
| 10101000 | 0 |
| 10100100 | 18 |
| 10100010 | 17 |
| 10100001 | 0 |

■ Nova solução: 1 0 1 0 0 1 0 0; $Q=18$

Trepa-Colinas

23

□ ...

□ Iteração 2

| Vizinhos | Qualidade |
|-----------------|-----------|
| 0 0 1 0 0 1 0 0 | 13 |
| 1 0 1 0 0 1 0 0 | 0 |
| 1 0 1 0 0 1 0 0 | 11 |
| 1 0 1 1 0 1 0 0 | 0 |
| 1 0 1 0 1 1 0 0 | 0 |
| 1 0 1 0 0 0 0 0 | 12 |
| 1 0 1 0 0 1 1 0 | 0 |
| 1 0 1 0 0 1 0 1 | 0 |

- Fim da Optimização!
- Solução: 1 0 1 0 0 1 0 0 **Q=18**

Será a solução óptima?

Variantes do Trepa-Colinas

24

□ Variantes

- Permitir o deslocamento ao longo de um planalto
 - Deverá ser sempre autorizado este tipo de deslocamento?
- Trepa Colinas “First-Choice”
 - Visita os vizinhos de forma aleatória
 - Aceita um vizinho de melhor qualidade e termina iteração
 - Útil quando a vizinhança é grande
 - Algoritmo não determinista
- Random Restart
 - Aplicar o algoritmo diversas vezes com diferentes pontos de partida

Recristalização Simulada

25

- Método de Recristalização simulada (*simulated annealing*)
 - ▣ Usa a seguinte estratégia para ultrapassar máximos locais:
 - Quando encontra um máximo (pode ser apenas um máximo local!), o algoritmo prossegue “durante algum tempo” a pesquisa no sentido descendente.

Recristalização Simulada

26

- ...
 - ▣ Em vez de se escolher sempre o estado seguinte de maior valor, escolhe-se um, aleatoriamente
 - ▣ Se a sua avaliação
 - for superior à do estado anterior, é sempre escolhido
 - for inferior, é escolhido mas apenas com uma certa probabilidade (<1) que baixa à medida que um parâmetro ‘T’ tende para zero ao longo das sucessivas iterações.

Recristalização Simulada

27

□ ...

- Quando T for muito pequeno, a escolha de estados de pior avaliação quase nunca ocorre, e o “Simulated Annealing” comporta-se (quase) como o “Hill-Climbing”.
- Com a continuação, as descidas vão sendo cada vez menos permitidas, de modo que no final apenas a subida para um máximo (que parte já de um valor elevado) é permitida: Com maior probabilidade, este poderá ser o máximo absoluto

Recristalização Simulada

28

□ ...

- O nome do algoritmo (Têmpera Simulada) provém de realizar uma analogia com o processo de têmpera de certos metais ou arrefecimento de um líquido até que congele.
- O parâmetro T simula a temperatura, que baixa com o tempo

Recristalização Simulada

29

□ ...

```

function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to “temperature”
  static: current, a node
           next, a node
           T, a “temperature” controlling the probability of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T=0 then return current
    next ← a randomly selected successor of current
     $\Delta E \leftarrow \text{VALUE}[\textit{next}] - \text{VALUE}[\textit{current}]$ 
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 

```

- $e^{\Delta E/T} = 1 / e^{|\Delta E/T|}$ porque só ocorre com $\Delta E < 0$:
- A probabilidade de um “movimento para um estado pior” decresce com o tempo

Recristalização Simulada

30

□ ...

□ Algoritmo **probabilístico**

- O resultado é não determinista,
- Deve-se executar o algoritmo mais do que uma vez.

- Se o arrefecimento for “suficientemente” lento é sempre atingido o **ótimo global!**

Pesquisa Tabu

31

- Princípio de funcionamento
 - ▣ Durante a pesquisa, **forçar a exploração de novas zonas do espaço de procura.**
 - Pode assim evitar-se entrar em ciclos
- ▣ Implementação:
 - Recurso a uma memória de curta-duração
 - Indica quais os **movimentos proibidos** (movimentos TABU)

Pesquisa Tabu

32

- ...
 - função Tabu (problema, vizinhança, Memória): solução**
 - $It \leftarrow 0$
 - Solução \leftarrow Gera_solução_aleatoriamente(Espaço de procura)
 - Repete enquanto ($it < Max_it$)**
 - Lista \leftarrow Obtém_vizinhos (Solução, vizinhança)
 - Lista \leftarrow Retira_Tabus (Lista, Memória)
 - Solução \leftarrow Escolhe_melhor(Lista)
 - Actualiza_memória
 - $it \leftarrow it + 1$
 - fim_Repete**
 - Devolve Solução
 - fim_de_função**

Pesquisa Tabu

33

□ ...

■ Aplicação ao Problema da mochila

- Solução inicial: 10100000
- Vizinhaça: soluções a uma distância de Hamming = 1
- Memória:
 - Guarda os movimentos recentes (Tabu)
 - Janela temporal escolhida neste caso: **2** movimentos
- Solução inicial: 1 0 1 0 0 0 0 0
 - **Q=12**

Pesquisa Tabu

34

□ ...

■ Iteração 1

■ Vizinhos

| Vizinhos | Qualidade |
|----------|-----------|
| 00100000 | 7 |
| 11100000 | 0 |
| 10000000 | 13 |
| 10110000 | 0 |
| 10101000 | 0 |
| 10100100 | 18 |
| 10100010 | 17 |
| 10100001 | 0 |

- Nova solução: 1 0 1 0 0 1 0 0; **Q=18**
- Memória
 - Posição 6 – n. de iterações Tabu=2

Pesquisa Tabu

35

...

Iteração 2

Vizinhos

| Vizinhos | Qualidade |
|------------|-----------|
| 00100100 | 13 |
| 11100100 | 0 |
| 10000100 | 11 |
| 10110100 | 0 |
| 10101100 | 0 |
| ✕ 10100000 | 12 |
| 10100110 | 0 |
| 10100101 | 0 |

Nova solução: 00100100; Q=13

Memória

- Posição 1 – n. de iterações Tabu=2
- Posição 6 – n. de iterações Tabu=1

Pesquisa Tabu

36

...

Iteração 3

Vizinhos

| Vizinhos | Qualidade |
|------------|-----------|
| ✕ 10100100 | 18 |
| 01100100 | 0 |
| 00000100 | 6 |
| 00110100 | 0 |
| 00101100 | 0 |
| ✕ 00100000 | 7 |
| 00100110 | 18 |
| 00100101 | 0 |

Nova solução: 00100110; Q=18

Memória

- Posição 1 – n. de iterações Tabu=1
- Posição 7 – n. de iterações Tabu=2

Pesquisa Tabu



37

...

Iteração 4

Vizinhos

| Vizinhos | Qualidade |
|--|-----------|
|  10100110 | 0 |
| 01100110 | 0 |
| 00000110 | 11 |
| 00110110 | 0 |
| 00101110 | 0 |
| 00100010 | 12 |
|  00100100 | 13 |
| 00100111 | 0 |

Nova solução: 0 0 1 0 0 0 1 0; Q=12

Memória

- Posição 6 – n. de iterações Tabu=2
- Posição 7 – n. de iterações Tabu=1

Pesquisa Tabu





38

...

Iteração 5

Vizinhos

| Vizinhos | Qualidade |
|--|-----------|
| 10100010 | 17 |
| 01100010 | 0 |
| 00000010 | 5 |
| 00110010 | 0 |
| 00101010 | 21 |
|  00100110 | 18 |
|  00100000 | 7 |
| 00100011 | 0 |

Nova solução: 0 0 1 0 1 0 1 0; Q=21

Memória

- Posição 5 – n. de iterações Tabu=2
- Posição 6 – n. de iterações Tabu=1

Pesquisa Tabu

39

□ Características

▣ Vantagens

- Escolhe sempre o melhor vizinho, desde que seja válido, exibindo assim um comportamento determinista
- Ao aceitar soluções de pior qualidade, pode evitar ótimos locais

▣ Desvantagens

- Nem sempre é fácil ajustar o limite de memória e número máximo de iterações