Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Departamento de Computação

Inteligência Artificial - 02.270-5 Turma A
Professora Doutora Lucia Helena Machado Rino

# Trabalho 2
## Aplicação dos conceitos de busca informada.
## Comparação de heurísticas no problema do saco (knapsack).

Alexandre Yukio Harano
RA: 254274

Luiz Carlos Irber Júnior
RA: 254258

São Carlos, 29 de outubro de 2007.

## 1. Introdução.

O problema do saco é um problema de otimização combinatorial. A formulação[1] é a seguinte:

"Dado um conjunto de itens, cada um com um peso e um valor, determinar quais itens adicionar na coleção de modo que o peso total é menor que um peso máximo e que o valor seja o máximo possível".

Inicialmente, pegamos a idéia do "knapsack problem" e, ao adicionarmos os itens para a análise do programa, baseamos nos objetos da casa de um colega nosso e o que faríamos para roubá-lo. Obviamente não iremos fazer tal ato, mas essa brincadeira nos serviu de inspiração para o trabalho.

## 2. Mapeamento através da Teoria dos Grafos.

Para a resolução do problema usamos a quádrupla G = [N,A,S,GD] de modo que:

N é o conjunto de estados descritos no item 2.1;

A é o conjunto de transições descritos no item 2.2;

S é o estado inicial definido no item 2.3;

GD é o conjunto de possíveis estados finais no item 2.3.

### 2.1. Definição da estrutura de dados utilizada e do conjunto de estados.

Um item é representado através de dois predicados:

```
peso(nome_do_item, P) :- P is peso.
valor(nome_do_item, V) :- V is valor.
```

O estado, de acordo com a definição pela Teoria dos Grafos, representa o saco através de uma lista que contém os nomes dos itens atualmente no saco e as ações tomadas até atingir o determinado estado, podendo, após a tomada de uma ação, manter o mesmo estado do saco. A partir do nome do item pode-se calcular seu peso e seu valor, o que deixa a representação do estado mais simples.

### 2.2. Definição das transições possíveis.

"...determinar quais itens adicionar na coleção..."

De acordo com a definição do problema, temos um saco que comporta os itens adicionados a ele com um peso limitante superior. Temos que determinar quais serão os próximos itens a serem avaliados, de acordo com a heurística determinada, sendo, na H1, o item que possui o maior valor entre os itens restantes e, na H2, o item que possui a maior "densidade monetária", ou seja, a maior relação valor por peso. Os predicados a seguir, marcados com 'aspas simples' indicam as transições possíveis e o modo de se alcançar tais predicados está descrito.

As transições possíveis são basicamente as seguintes: primeiro, busca-se o item restante que melhor se enquadra na condição da heurística adotada sendo os predicados 'buscaItemMaisValioso' e 'buscaItemMaiorDensidade' em H1 e H2, respectivamente. Logo em seguida avalia-se o item selecionado dentro do predicado 'avaliacao' de acordo com os dados do item e a condição do saco.

Caso o peso do item em questão for maior que o peso máximo do saco, o item é descartado da análise. Se o peso somado dos itens contidos no saco somado ao peso do item for igual ou menor que o peso máximo permitido no saco, então adiciona-se o item através do predicado 'insercao'. Se a somatória citada anteriormente for maior que o peso máximo permitido no saco, então busca o item de menor valor contido no saco através do predicado 'buscaItemMenosValioso'. Se o item menos valioso do saco possuir um valor maior que do item em questão, então o item em questão é descartado. Se o item menos valioso do saco possuir um valor igual ao do item em questão, então compara-se o peso dos itens e mantém no saco o

[1] Obtida em http://www.nist.gov/dads/HTML/knapsackProblem.html

item de menor peso, sendo usado os predicados 'remocao' e 'insercao' no caso de o item em questão ser de peso menor que o item menos valioso do saco. Se o item menos valioso do saco possuir um valor menor que do item em questão, então verifica se a soma do peso do saco mais o peso do item em questão menos o peso do item menos valioso do saco é menor ou igual ao peso máximo do saco. Se for maior, então o item em questão é descartado, caso contrário é usado o predicado 'remocao' e 'insercao' para remover o item de menor valor e inserir o item em questão.

Essas condicionantes parecem complicadas, mas provam que não há como manter a monotonicidade da função heurística em ambos os casos. Se preferível, o parágrafo anterior está sintetizado como pseudo código em anexo após os dumps de tela sendo o retorno o custo da função heurística dependendo da situação.

## 2.3. Estado inicial e conjunto de estados finais.

O conjunto de estado inicial é representado pela totalidade de estados que obedecem a relação

```
inicial([],[], I) :- itens(I).
```

Modificando a ordem dos itens no código em Prolog pode-se obter estados iniciais diferentes. Seu significado é:

- A primeira lista vazia significa o saco, que inicia vazio.
- A segunda lista guarda as ações realizadas até o momento, que no caso inicial é nenhuma.
- A terceira lista indica os itens a serem analisados, sendo que item é um predicado que procura o match de item que esteja no predicado valor e peso, excluindo 'invalido', sendo o último usado no caso de busca de menor valor dentro do saco.

O conjunto de estados finais inclui estados definidos de dois modos: ou a somatória do peso dos itens do saco atingiu o peso máximo do saco (previsto no predicado 'objetivo') ou a lista de itens a serem analisados se encontrar vazia (representado por um match de ItensRestantes com []).

# 3. Instruções de execução.

Fizemos uso do SWI-Prolog, pois programamos em dois sistemas operacionais (Windows e Linux). Ao iniciar a execução, necessita-se usar o predicado 'consult' e usar o predicado 'bt2' (Busca Tipo 2 - Pattern-directed search) com aridade 1, sendo o parâmetro um inteiro qualquer. O predicado usa o módulo de 2 para definir qual heurística seria utilizada, de acordo com os itens a, b e c da especificação do trabalho, ou seja, quando se entra com um parâmetro ímpar, então H1 é utilizada e quando se entra com um parâmetro par, então H2 é utilizada. Como os itens “a” e “b” da especificação podem ser verificados explicitamente no código-fonte fornecido e verificado anteriormente, então os itens c.1, c.2 e c.3 da mesma serão discutidos, respectivamente, em 3.1, 3.2 e 3.3.

## 3.1. Descrição das heurísticas.

H1 leva em consideração o valor do item, como citado anteriormente em 2.2.

H2 leva em consideração a "densidade" (relação entre valor e peso) do item, como citado anteriormente em 2.2.

## 3.2. Comparação entre esforço e trajetória obtidos.

Para a contagem do tamanho do caminho é considerada a quantidade de inserções de itens no saco somada à quantidade de avaliações de itens realizadas, pois é isso que determina o estado do saco. Uma substituição de item é coberta por esse caso, pois requer a remoção de um item e a inserção de outro, sendo, nesse caso, contado como inserção.

Para a contagem do esforço foi considerada a quantidade de ações (avaliação, inserção, remoção e busca) realizada pela heurística. Grosseiramente, representa o custo computacional do caminho.

Levando em conta isso, H1 obteve o resultado mais rapidamente, percorrendo 14 estados do problema, com um esforço computacional de 33 ações. H2 percorreu mais estados, 16, com um esforço computacional de 36 ações.

Comparando o valor obtido pelas duas heurísticas, H2 superou H1, pois obteve 8537 contra 7800 de H1. Para este caso, embora percorra mais estados, H2 é mais vantajosa, pois, por um esforço computacional um pouco maior, obteve-se um "lucro" de 737 sobre H1, fato que torna nosso ladrão uma pessoa mais contente com nosso programa.

## 3.3. Informatividade, admissibilidade e monotonicidade.

Informatividade:

- H2 é mais informada que H1, pois além de considerar as informações básicas disponíveis (peso e valor) considera uma informação secundária, derivada das básicas (densidade). H1 apenas considera o valor.

Admissibilidade:

- Nenhuma delas é admissível, pois não garantem o menor caminho, apenas que chegará a um estado objetivo.

Monotonicidade:

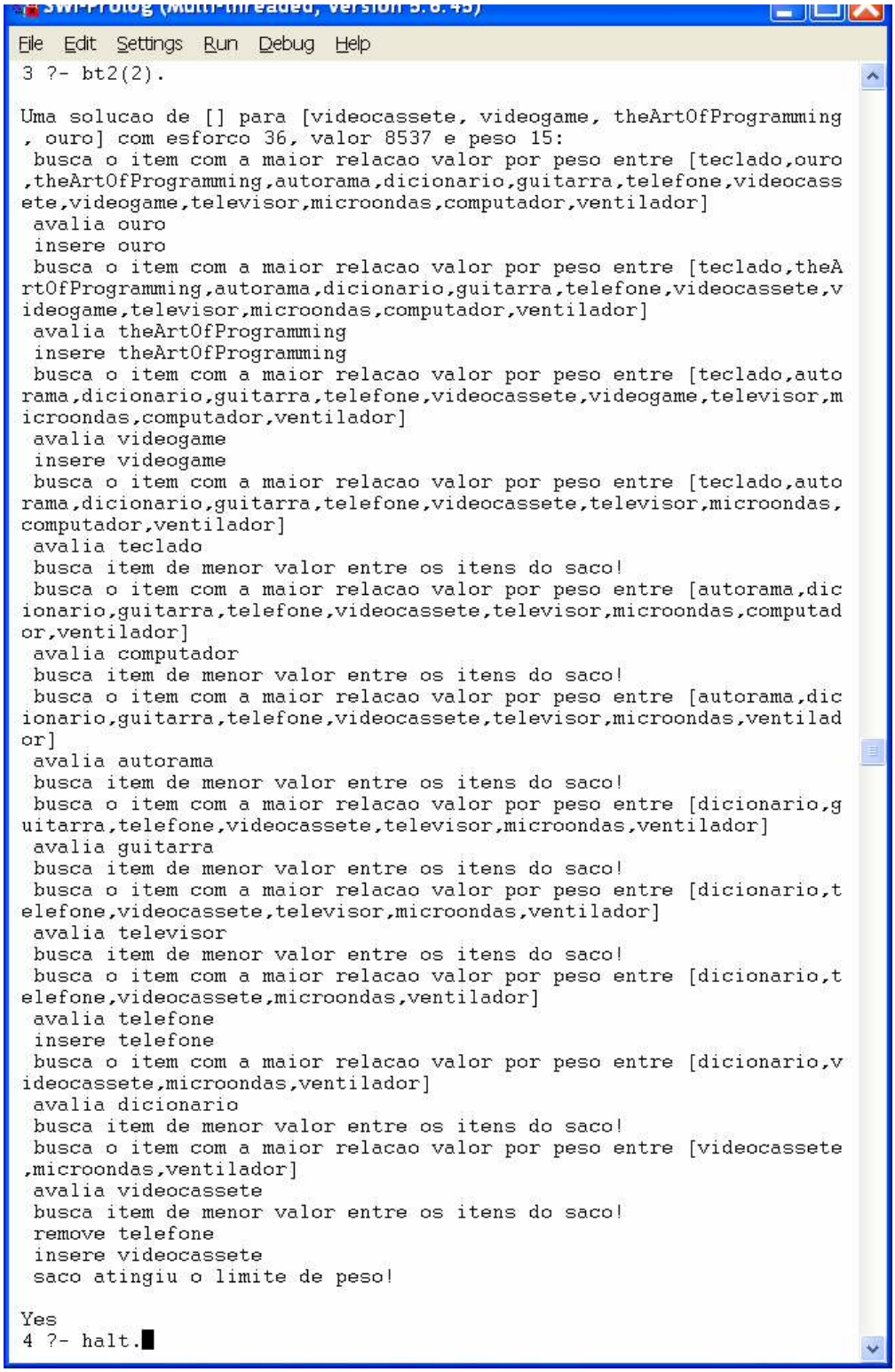
- H1 (Maior peso): não é monotônica, pois a parte H da função de avaliação varia (não cresce ou descresce, apenas).
- H2 (Maior densidade): não é monotônica, pois a parte H da função de avaliação varia (não cresce ou descresce, apenas).

## 4. Conclusão.

No dump de tela, mostra-se a execução de nosso programa e a heurística H2 obteve maior valor em relação à heurística H1. Algo que ficou nítido ao adicionarmos itens e testar para cada inserção é que cada heurística se enquadra melhor de acordo com as características comuns dos itens a serem avaliados. Além das heurísticas escolhidas, existem também as heurísticas que optam por avaliar primeiramente ou os itens mais pesados ou os menos pesados. Se fizéssemos uma análise estatística, provavelmente enquadraríamos os dados de modo que poderíamos indicar qual é a melhor heurística para aqueles dados, mas, muito provavelmente, tal "seletor de heurísticas" levaria mais tempo do que qualquer uma das heurísticas citadas anteriormente. Uma variação nítida que podemos verificar para tal problema seria incluir, também, a característica de volume, pois se levarmos em conta um saco "real", esse aspecto jamais poderia ser deixado de lado!

É importante citar que utilizamos a formulação mais simples do problema, onde dado uma lista de itens únicos e seus valores e com isso tentamos encaixá-los em uma restrição simples (peso, no caso). Essa formulação é NP-Hard, e possui um método de resolução polinomial. Se tivéssemos optado por não restringir a quantidade de determinado item, permitindo repetições, e ainda assim desejássemos maximizar algo (peso ou valor) restrito a um limite, o problema passa a ser NP-Completo e até o momento não foi achada uma solução exata para o problema, apenas havendo heurísticas para a solução. Nesse sentido IA é uma opção interessantíssima para a resolução deste problema, já que muitas aplicações são baseadas nesse problema. Por exemplo, na área de telecomunicações, mais especificamente na transmissão de dados via telefonia.

## Anexo. Dumps de tela


SWI-Prolog (Multi-threaded, version 5.6.45)
File Edit Settings Run Debug Help
3 ?- bt2(2).
Uma solucao de [] para [videocassete, videogame, theArtOfProgramming, ouro] com esforco 36, valor 8537 e peso 15:
busca o item com a maior relacao valor por peso entre [teclado,ouro,theArtOfProgramming,autorama,dicionario,guitarra,telefone,videocassete,videogame,televisor,microondas,computador,ventilador]
avalia ouro
insere ouro
busca o item com a maior relacao valor por peso entre [teclado,theArtOfProgramming,autorama,dicionario,guitarra,telefone,videocassete,videogame,televisor,microondas,computador,ventilador]
avalia theArtOfProgramming
insere theArtOfProgramming
busca o item com a maior relacao valor por peso entre [teclado,autorama,dicionario,guitarra,telefone,videocassete,videogame,televisor,microondas,computador,ventilador]
avalia videogame
insere videogame
busca o item com a maior relacao valor por peso entre [teclado,autorama,dicionario,guitarra,telefone,videocassete,televisor,microondas,computador,ventilador]
avalia teclado
busca item de menor valor entre os itens do saco!
busca o item com a maior relacao valor por peso entre [autorama,dicionario,guitarra,telefone,videocassete,televisor,microondas,computador,ventilador]
avalia computador
busca item de menor valor entre os itens do saco!
busca o item com a maior relacao valor por peso entre [autorama,dicionario,guitarra,telefone,videocassete,televisor,microondas,ventilador]
avalia autorama
busca item de menor valor entre os itens do saco!
busca o item com a maior relacao valor por peso entre [dicionario,guitarra,telefone,videocassete,televisor,microondas,ventilador]
avalia guitarra
busca item de menor valor entre os itens do saco!
busca o item com a maior relacao valor por peso entre [dicionario,telefone,videocassete,televisor,microondas,ventilador]
avalia televisor
busca item de menor valor entre os itens do saco!
busca o item com a maior relacao valor por peso entre [dicionario,telefone,videocassete,microondas,ventilador]
avalia telefone
insere telefone
busca o item com a maior relacao valor por peso entre [dicionario,videocassete,microondas,ventilador]
avalia dicionario
busca item de menor valor entre os itens do saco!
busca o item com a maior relacao valor por peso entre [videocassete,microondas,ventilador]
avalia videocassete
busca item de menor valor entre os itens do saco!
remove telefone
insere videocassete
saco atingiu o limite de peso!
Yes
4 ?- halt.

## Anexo. Pseudo código para o valor heurístico do estado.

Retorno: Valor heurístico de acordo com o estado.

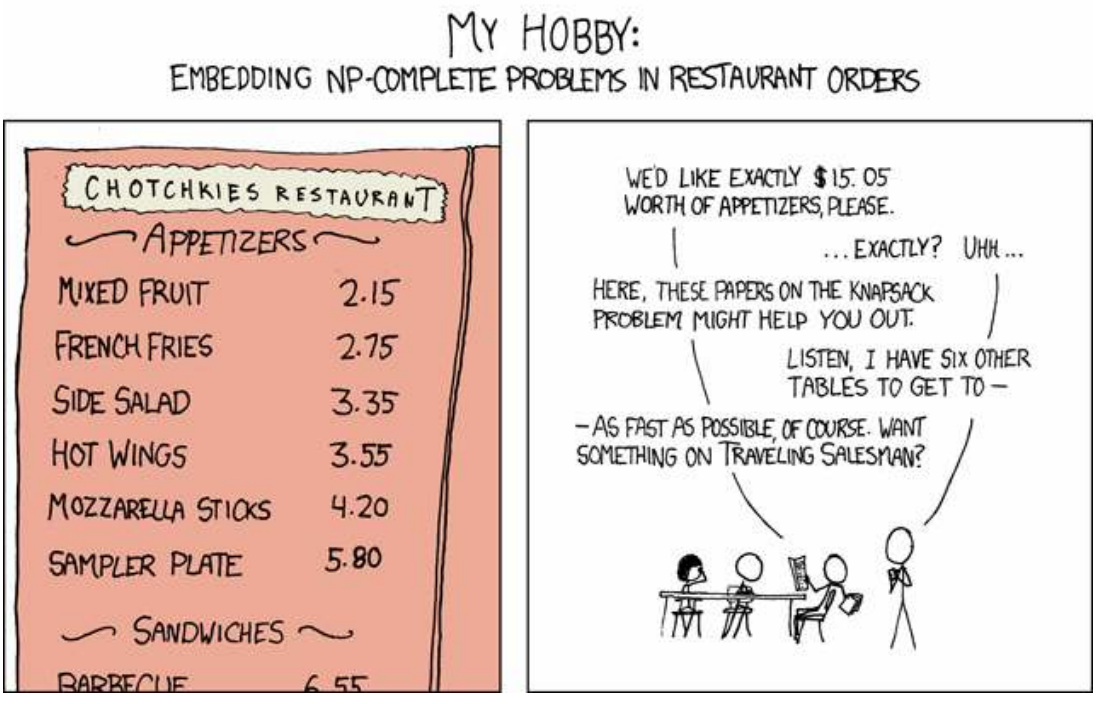
Variáveis:

`pesoMax` - peso máximo do saco
`pesoSaco` - somatória dos pesos dos itens do saco atual
`pesoItem` - peso do item em análise
`valorItem` - valor do item em análise
`pesoMenor` - peso do item de menor valor dentro do saco
`valorMenor` - valor do item de menor valor dentro do saco

Pseudo código:

```
int custo
se (pesoItem > pesoMax)
  descarta item | CUSTO_FINAL: 1
se (pesoSaco + pesoItem <= pesoMax )
  insere item | CUSTO_FINAL: 1
senão
  buscaItemMenor
  custo <- custo + 1 (1)
  se ( valorMenor > valorItem )
    descarta item | CUSTO FINAL: 1
  senão
    se ( valorMenor = valorItem )
      se ( pesoMenor <= pesoItem )
        descarta item | CUSTO FINAL: 1
      senão
        remove menor
        custo <- custo + 1 (2)
        insere item
        custo <- custo + 1 (3) | CUSTO FINAL: 3
    senão
      se ( pesoItem+pesoSaco-pesoMenor > pesoMax )
        descarta item | CUSTO FINAL: 1
      senão
        remove menor
        custo <- custo + 1 (2)
        insere item
        custo <- custo + 1 (3) | CUSTO FINAL: 3
```

Segue-se uma tirinha sobre o tema do nosso trabalho (recomenda-se ler a conclusão antes!):




Fonte: http://xkcd.com/287/ (Nosso problema é NP-hard, não NP-completo!)