Agentes de Procura
Procura Heurística

Capítulo 3:
Costa, E. e Simões, A. (2015). Inteligência Artificial – Fundamentos e Aplicações, 3.ª edição, FCA.

Procura Heurística

- Nas estratégias de procura consideradas até agora admitimos a inexistência de conhecimento que nos pudesse auxiliar na travessia do espaço de procura
- A única informação utilizada foi a função g(n) que indicava o custo do caminho desde o estado inicial até ao estado n (custo uniforme)

Procura Heurística

- Nos algoritmos que se apresentam a seguir vamos admitir que possuímos informação adicional que nos permite estimar o custo do caminho do nó corrente até ao nó solução

- Essa informação é dada por uma função h(n) – função de avaliação

Procura Heurística

- No exemplo das cidades que temos vindo a considerar, esta função h(n) será a distância quilométrica em linha reta entre cada cidade e a cidade de Faro

	Faro
Aveiro	366
Braga	454
Bragança	487
Beja	99
C. Branco	280
Coimbra	319
Évora	157
Faro	0
Guarda	352
Leiria	278
Lisboa	195
Portalegre	228
Porto	418
Santarém	231
Setúbal	168
Viana	473
V. Real	429
Viseu	363

Distâncias quilométricas em linha recta

Procura Heurística

- Os próximos algoritmos distinguem-se em vários aspetos
- Por exemplo, o modo como o próximo operador é escolhido: num caso, escolhe-se o melhor à luz de uma heurística global, noutros casos a heurística é puramente local
- Também os podemos distinguir de acordo com o modo como a memória é ou não limitada
- Finalmente, podemos ainda considerar o modo como as componentes de custo, g(n) e h(n), são utilizadas.

Procura Sôfrega

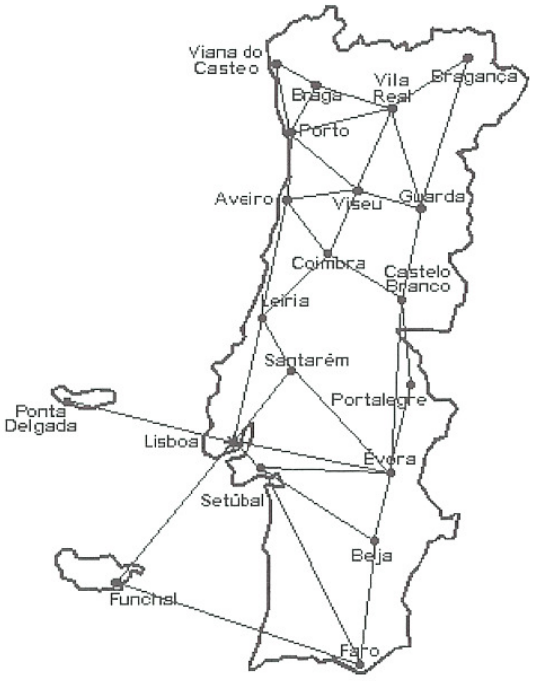
- Na procura sôfrega (*greedy search*) o princípio consiste em escolher o nó na fronteira da árvore de procura que aparenta ser o mais promissor de acordo com o valor estimado por h(n)
- Assim, o algoritmo limita-se a manter a fronteira da árvore de procura ordenada pelos valores de h(n), sendo sempre escolhido o nó de valor mais baixo, ou seja, aquele que está hipoteticamente mais próximo da solução

Procura Sôfrega

- Exemplo: Ir de Coimbra até Faro

- Partindo de Coimbra, podemos ir para as suas cidades vizinhas: Aveiro, Leiria, Viseu e Castelo Branco
- Essas possibilidades aparecem ordenadas pela distância a Faro, estimada em linha reta

Procura Sôfrega



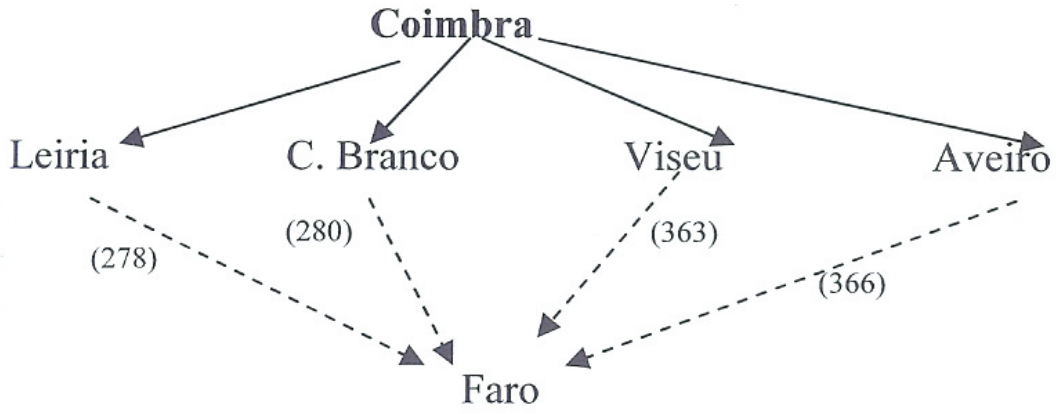
As estradas entre as capitais de distrito

	Faro
Aveiro	366
Braga	454
Bragança	487
Beja	99
C. Branco	280
Coimbra	319
Évora	157
Faro	0
Guarda	352
Leiria	278
Lisboa	195
Portalegre	228
Porto	418
Santarém	231
Setúbal	168
Viana	473
V. Real	429
Viseu	363

Distâncias quilométricas em linha recta

Procura Sôfrega

- Exemplo: Ir de Coimbra até Faro



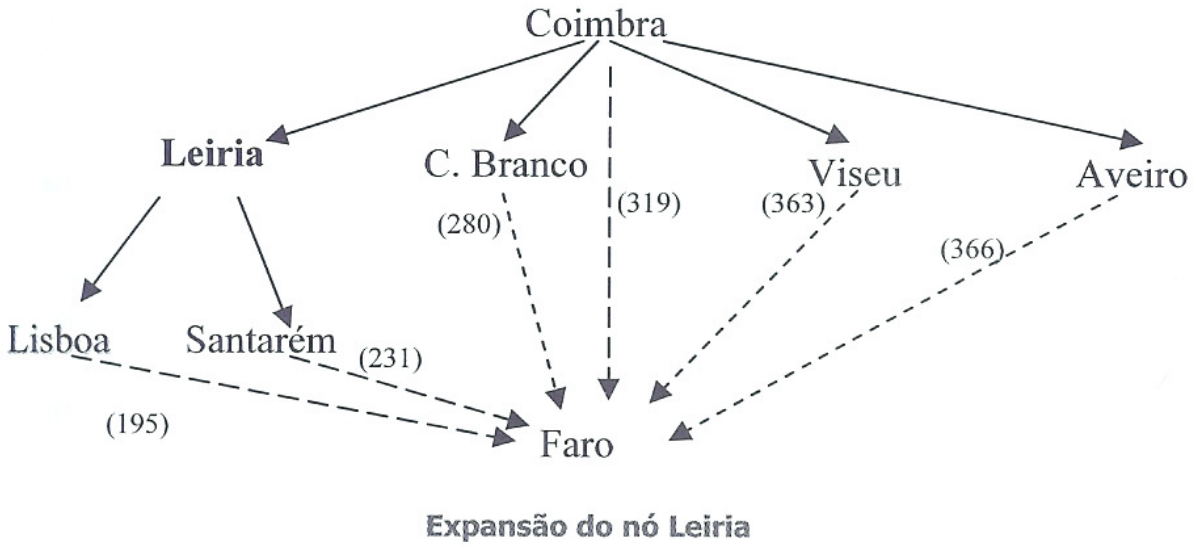
Primeira expansão da árvore de procura

Procura Sôfrega

- Exemplo: Ir de Coimbra até Faro
- Em função daqueles valores, a próxima cidade a ser visitada será Leiria
- Como Leiria não é o ponto de chegada, temos de calcular os seus descendentes e respetivas distâncias estimadas a Faro
- A nova fronteira da árvore de procura terá de ser ordenada

Procura Sôfrega

- Exemplo: Ir de Coimbra até Faro



Expansão do nó Leiria

Procura Sôfrega

- Exemplo: Ir de Coimbra até Faro

- Com estes resultados, a próxima cidade a ser escolhida seria Lisboa …

Procura Sôfrega

Algoritmo de procura sôfrega

```
Função ProcuraSôfrega(problema, InsereListaOrdenada, Heurística): solução ou falha
    1. l_nós ← FazListaOrdenada(EstadoInicial(problema))
    2. Repete
        2.1. Se VaziaListaOrdenada(l_nós) Então
            2.2.1. Devolve falha
        Fim_de_se
        2.2. nó ← RetiraListaOrdenada(l_nós)
        2.3. Se TesteObjectivo(nó) Então
            2.3.1. Devolve  nó
        Senão
            2.3.2. InsereListaOrdenada(l_nós,
                Heurística (Expansão(nó,Operadores (problema)))
        Fim_de_se
    Fim_de_Repete
Fim_de_Função
```

Algoritmo de procura sôfrega

Procura Sôfrega

Algoritmo de procura sôfrega

- A estrutura de dados usada neste caso é uma lista ordenada
- A variável l_nós irá manter a lista ordenada pela heurística dos nós ainda não visitados
- A heurística aparece aqui passada como argumento da função, o que torna o algoritmo mais genérico

Procura Sôfrega

Algoritmo de procura sôfrega

- Vejamos o conteúdo de l_nós para algumas iterações:

ITERAÇÃO	L_NÓS
0	[(Coimbra, 319)]
1	[(Leiria, 278), (C. Branco, 280), (Viseu, 363), (Aveiro, 366)]
2	[(Lisboa, 195), (Santarém, 231), (C. Branco, 280), (Coimbra, 319), (Viseu, 363), (Aveiro, 366)]

Conteúdo de l_nós

- Verifica-se o reaparecimento de nós já visitados, como é o caso de Coimbra

Procura Sôfrega

Caracterização desta estratégia

- Não é, em geral, completa

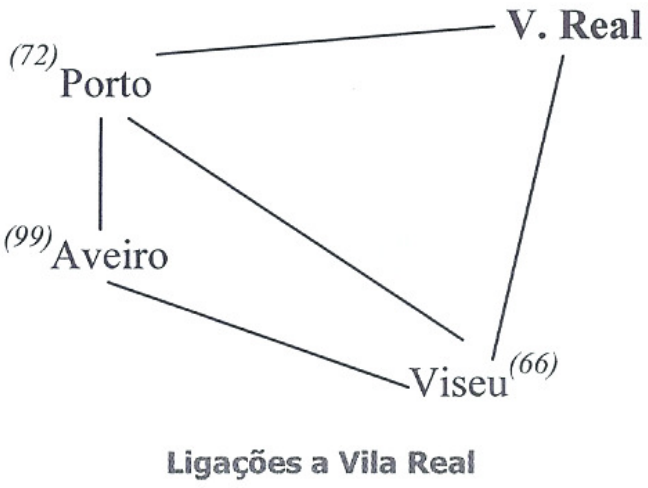
  Basta notar que podem aparecer, como no exemplo anterior, nós repetidos que podem originar caminhos infinitos

Procura Sôfrega

Caracterização desta estratégia

- Não é discriminadora

  Consideremos o seguinte exemplo



Ligações a Vila Real

Procura Sôfrega

- Admitamos que queremos ir de Aveiro para Vila Real e que usamos como heurística as distâncias em linha reta

- O algoritmo começa por expandir Aveiro em Viseu e Porto, optando por Viseu. Ao expandir Viseu encontraremos a solução

Procura Sôfrega

- Mas, na realidade, o caminho

  Aveiro – Viseu – Vila Real

  significa percorrer 205 Km, enquanto que o caminho não escolhido

  Aveiro – Porto – Vila Real

  obrigaria a percorrer apenas 184 Km

Procura Sôfrega

Caracterização desta estratégia

- Quanto à complexidade
- Nesta estratégia os nós vão sendo expandidos num misto entre uma procura em profundidade e uma procura em largura

  Procura em profundidade primeiro, uma vez que a procura prefere seguir um único caminho até à solução, mas terá que regressar a um nível anterior quando encontra um caminho sem saída (procura em largura primeiro)

Procura Sôfrega

Caracterização desta estratégia

- Assim, pode acontecer que, no pior caso, todos os nós tenham de ser expandidos e visitados, o que significa uma complexidade temporal da ordem de $O(r^n)$

  r – fator de ramificação

  n – nível da solução

Procura Sôfrega

Caracterização desta estratégia

- Como a fronteira terá de ser mantida toda em memória, e atendendo ao modo como ela evolui, teremos também no limite uma complexidade espacial da ordem de $O(r^n)$

Procura A*

- O algoritmo de procura sôfrega foi a primeira tentativa de usar informação heurística para dominar a complexidade

- Usámos o custo estimado do nó corrente n ao nó solução, dado por uma função de avaliação h(n), para guiar a nossa escolha

Procura A*

- O algoritmo de procura de custo uniforme utilizava o custo para chegar do nó inicial ao nó corrente, n, dado por uma função g(n)

Procura A*

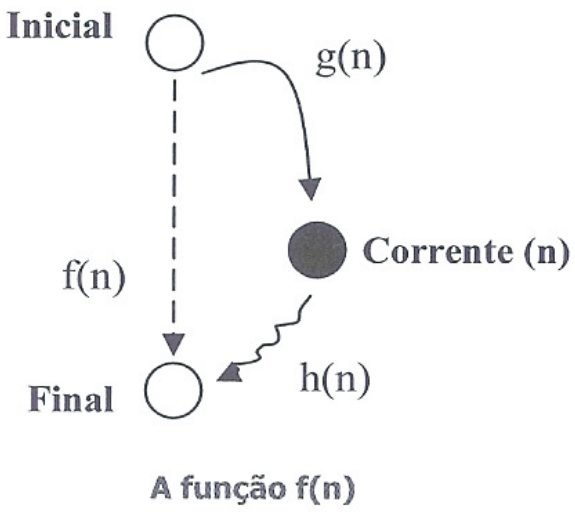
- O algoritmo A* junta as duas ideias

- Tenta escolher a cada instante o melhor caminho passando pelo nó n, utilizando para tal a função

$$f(n) = g(n) + h(n)$$

Procura A*

- Dado que h(n) nos dá um valor estimado, então f(n) também quantifica o valor estimado de um caminho passando por n



A função f(n)

Procura A*

- O aspeto interessante deste algoritmo é o de garantir, no caso de a função h(n) ter boas propriedades, não apenas que é completo, mas que encontra a melhor solução

Procura A*

- Qual é então a propriedade que h(n) deve respeitar?

  h(n) deve ser admissível

- ou seja, nunca sobrestima o custo real do caminho que passa por n:

  $h(n) \leq h_{real}(n)$

Procura A*

- A heurística utilizada no exemplo das estradas é admissível: nunca a distância em linha reta pode ser estritamente superior à distância real

Procura A*

- A descrição do algoritmo A* é a seguinte:

```
Função A*(problema, InsereListaOrdenada, g+h): solução ou falha
    1. l_nós ← FazListaOrdenada(EstadoInicial(problema))
    2. Repete
            2.1. Se VaziaListaOrdenada(l_nós) Então
                    2.2.1. Devolve falha
                Fim_de_Se
            2.2. nó ← RetiraListaOrdenada(l_nós)
            2.3. Se TesteObjectivo(nó) Então
                    2.3.1. Devolve  nó
                Senão
                    2.3.2. InsereListaOrdenada(l_nós,
                                    g+h(Expansão(nó,Operadores(problema))))
                Fim_de_Se
        Fim_de_Repete
Fim_de_Função
```

Algoritmo da procura A*

Procura A*

- A única diferença relativamente ao algoritmo de procura sôfrega está no uso combinado das funções g(n) e h(n)
- A estrutura de dados utilizada também é uma lista ordenada

Procura A*

- Exemplo: Ir de Coimbra a Faro
- Conteúdo da lista ordenada com a fronteira da árvore de procura (variável l_nós) para as primeiras iterações:

ITERAÇÃO	L_NÓS
0	[(**Coimbra, 0+319=319**)]
1	[(**Leiria, 67+278=345**), (Aveiro, 68+366=434), (C. Branco, 159+280=439), (Viseu, 96+363=459)]
2	[**(Santarém, 137 + 231 = 368)**, (Lisboa, 196+195=391), (Aveiro, 68+366=434), (C. Branco, 159+280=439), (Coimbra, 134+319=453), (Viseu, 96+363=459), (Aveiro, 182+366=548)]
3	[**(Lisboa, 196+195=391)**, (Lisboa, 215+195=410), (Évora, 254+157=411), (Aveiro, 68+366=434), (C. Branco, 159+280=439), (Coimbra, 134+319=453), (Viseu, 96+363=459), (Leiria, 207+278=485), (Aveiro, 182+366=548)]

Conteúdo de I_nós

Procura A*

- Nesta tabela a primeira componente representa o valor de g(n) e a segunda o valor de h(n)
- Verifica-se que é possível ir até à mesma cidade por caminhos diferentes

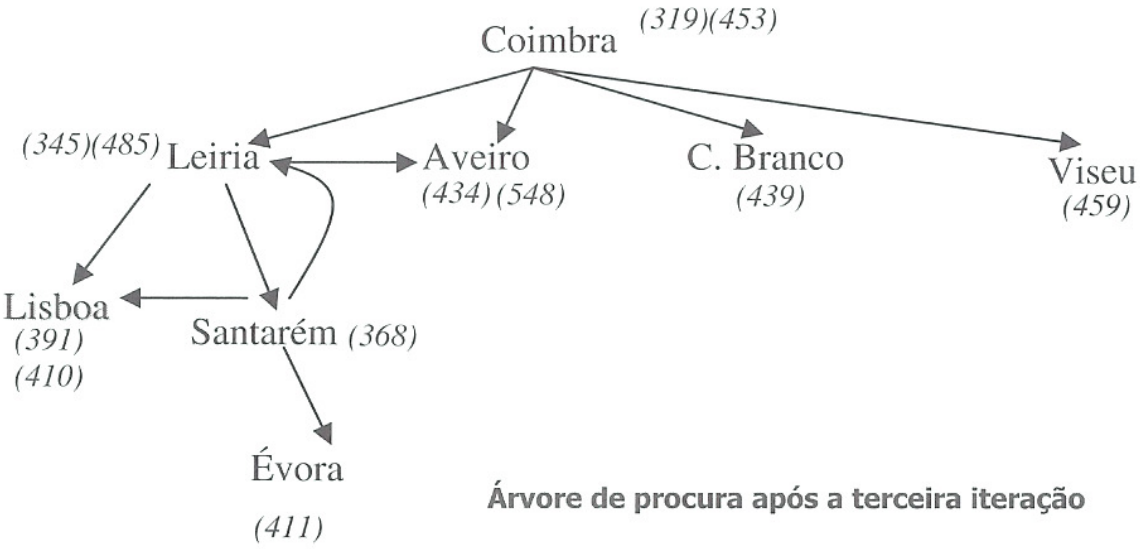
  Por exemplo, podemos ir de Coimbra diretamente até Aveiro ou indiretamente passando por Leiria

Procura A*

- Das diversas alternativas será escolhida primeiro a que tiver f(n) mais baixo

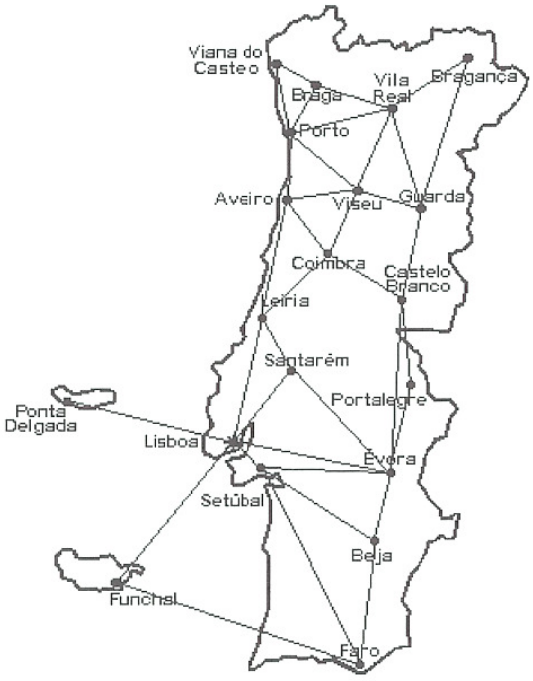
- Desta maneira, um primeiro caminho via Lisboa (Coimbra-Leiria-Lisboa) que tinha sido preterido face à alternativa Coimbra-Leiria-Santarém, é mais tarde recuperado

Procura A*



Árvore de procura após a terceira iteração

Procura A*



As estradas entre as capitais de distrito

	Faro
Aveiro	366
Braga	454
Bragança	487
Beja	99
C. Branco	280
Coimbra	319
Évora	157
Faro	0
Guarda	352
Leiria	278
Lisboa	195
Portalegre	228
Porto	418
Santarém	231
Setúbal	168
Viana	473
V. Real	429
Viseu	363

Distâncias quilométricas em linha recta

Procura A*

AVEIRO	Porto (68)	Viseu (95)	Coimbra (68)	Leiria (115)
BRAGA	Viana C.(48)	V. Real (106)	Porto (53)	
BRAGANÇA	V. Real (137)	Guarda (202)		
BEJA	Évora (78)	Faro (152)	Setúbal (142)	
C. BRANCO	Coimbra (159)	Guarda (106)	Portalegre (80)	Évora (203)
COIMBRA	Viseu (96)	Leiria (67)		
ÉVORA	Lisboa (150)	Santarém (117)	Portalegre (131)	Setúbal (103)
FARO	Setúbal (249)			
GUARDA	V. Real (157)	Viseu (85)		
LEIRIA	Lisboa (129)	Santarém (70)		
LISBOA	Santarém (78)	Setúbal (50)		
PORTO	V. Castelo (71)	V. Real (116)	Viseu (133)	
V. REAL	Viseu (110)			

Distâncias quilométricas entre cidades portuguesas

Procura A*

Algumas observações:

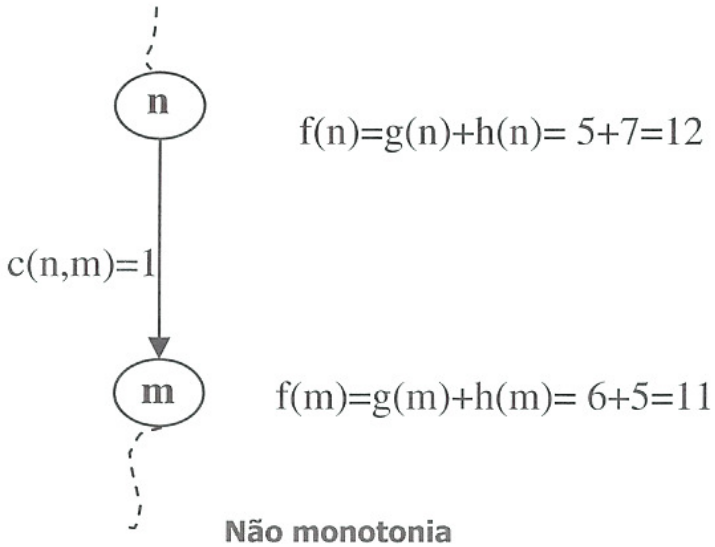
- O custo dado pela função f(n) ao longo de um caminho em geral nunca decresce
- Se tal for sempre o caso, diz-se que f(n) é monótona

Procura A*

Algumas observações:

- As heurísticas admissíveis são, na grande maioria dos casos, monótonas
- Mas e se não forem ?

Procura A*



Procura A*

- O valor de f decresce ao longo do caminho

- No entanto, o facto de f(m)=11 é irrelevante, pois, dado o significado da função h, já sabemos que o custo verdadeiro é pelo menos 12

Procura A*

- Assim, cada vez que um nó sucessor m, de n, for gerado podemos usar uma variante para calcular o seu f(m):

$$f(m) = \max (f(n) , g(m)+h(m))$$

- Com esta transformação o valor de f ao longo de um caminho nunca decresce

Procura A*

Caracterização do algoritmo

Considerações iniciais:

- h(n) é admissível para todo o n

  A função heurística nunca sobrestima o valor real do custo
- Vamos admitir que o fator de ramificação associado a cada nó é finito
- Cada arco tem um custo positivo

Procura A*

- O algoritmo A* é completo
- O algoritmo vai expandindo e analisando os nós por valores crescentes de f
- A única possibilidade de, desta forma, não encontrar a solução, seria existir uma infinidade de nós com um valor de f inferior ao valor de f da solução

Procura A*

- Ora, tal só poderia acontecer se, ou o fator de ramificação de um dado nó fosse infinito, ou se existisse um caminho com um número infinito de nós mas com custo finito

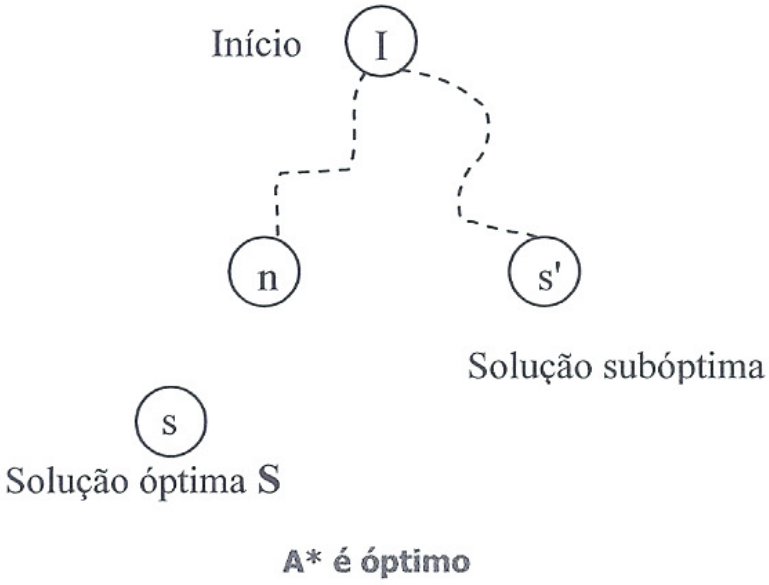
- O facto de não poderem existir nós com fator de ramificação infinito e ainda o facto de a aplicação de cada operador ter um custo positivo impedem as duas condições de se verificar

Procura A*

- O algoritmo A* é um algoritmo ótimo, isto é, discrimina entre as várias soluções possíveis encontrando sempre primeiro a melhor
- Prova por contradição:

  Admitamos, por hipótese, que o algoritmo pode escolher um caminho que o leve para uma solução de custo superior à do melhor custo, ou seja, que se prepara para expandir para s', que é uma solução subóptima, em vez de expandir n, nó na fronteira e que pertence ao caminho da solução óptima s

Procura A*



Procura A*

- Sabendo que h é admissível, temos que $f_{opt} \geq f_n$
- Por outro lado, como n não foi escolhido é porque $f(n) \geq f(s')$
- Daqui resulta que $f_{opt} \geq f(s')$
- Como s' é uma solução ($h(s')=0$), temos $f_{opt} \geq g(s')$, o que implica que s' não é uma solução subóptima, o que contradiz a hipótese inicial

Procura A*

- Conclui-se assim que o algoritmo nunca opta por uma solução de qualidade inferior

Procura A*

Análise de complexidade: em que medida o algoritmo é económico ?

- A complexidade do A* depende em parte da qualidade da função heurística
- Em geral, no entanto, o número de nós na fronteira definida pelo valor de f cresce exponencialmente
- Por outro lado, todos esses nós são conservados em memória

Procura A*

- Daqui se infere que o comportamento genérico do algoritmo, quer em espaço, quer em tempo, é exponencial

Procura IDA*

- Um dos maiores problemas das estratégias de procura prende-se com a elevada complexidade espacial

- O algoritmo A* não foge a esta regra: para muitos problemas reais este algoritmo não é viável pois esgota a memória disponível

Procura IDA*

- Será possível desenvolver um algoritmo que, sem perder a qualidade de encontrar a solução ótima (discriminador), possa ser executado sem problemas de memória?

Procura IDA*

- A solução passa por usar estratégias que autolimitem a memória que pode ser usada em cada momento: a estratégia IDA* pertence a esta categoria

- IDA* - *Iterative Deepening A**

Procura IDA*

- Trata-se de uma estratégia semelhante à estratégia de aprofundamento progressivo. As diferenças são as seguintes:
- É usada a função heurística

$$f(n) = g(n) + h(n)$$

- O aprofundamento é controlado não pelo nível mas pelos valores da função f(n)

Procura IDA*

- Torna-se necessário definir uma política para o valor inicial de f, bem como para os seus sucessivos incrementos

- Uma solução simples seria incrementar f de uma constante em cada iteração

Procura IDA*

- No exemplo das cidades, incrementos de 75Km podem ser considerados razoáveis
- Assim, por cada iteração seria definido um contorno f_i, iniciado a 75 e que tomaria sucessivamente os valores de 150, 225, 300, etc.
- Para cada contorno o algoritmo efetuará uma procura em profundidade

Procura IDA*

- O problema desta abordagem é o de não garantir que a solução ótima seja encontrada
- De qualquer modo, o erro seria sempre inferior à constante utilizada

Procura IDA*

- Para evitar esse problema, terá de ser usada a política seguinte:

- O primeiro valor de f será igual ao valor da distância estimada do nó inicial à solução, ou seja,

h (estado_inicial)

Procura IDA*

- O próximo valor de f será igual ao valor mais pequeno de f(n) da etapa anterior, ou seja, dos nós pertencentes à fronteira da árvore de procura, ainda não visitados por ultrapassarem o limite de f

- Dessa forma, fica garantido que se encontra a solução ótima

Procura IDA*

- No entanto, existe um preço a pagar, tanto maior quanto maior for o número de nós com valores distintos de f

- No limite, podemos ter todos os valores distintos, pelo que em cada iteração apenas mais um nó será analisado

Procura IDA*

- Este algoritmo pode ser descrito do seguinte modo:

```
Função IDA* (problema): solução ou falha
    1. f_limite ← f(EstadoInicial(problema))
    2. Repete
          2.1. f_limite_ant ← f_limite
          2.2. f_limite ← PppLimite(EstadoInicial(problema), f_limite)
       Até  sucesso ou f_limite sem alteração
    3. Se f_limite = sucesso Então
          3.1. Devolve sucesso
       Senão
          3.2. Devolve falha
       Fim_de_Se
Fim_de_Função

Função PppLimite(nó, limite): sucesso ou valor de limite
    1. Se f(nó) > limite Então
          1.1. Devolve f(nó)
       Fim_de_Se
    2. Se TesteObjectivo(nó) Então
          2.1. Devolve sucesso
       Senão
          2.2. Devolve Mínimo(PppLimite(n, limite), para n ∈ Sucessores(nó))
       Fim_de_Se
Fim_de_Função
```

Algoritmo de pesquisa IDA*

Procura IDA*

Características deste algoritmo:

- Uma vez que as sucessivas iterações são controladas pelos valores da função heurística f(n) e admitindo que h(n) é admissível, então o algoritmo IDA*:
  - É completo
  - Encontrará a melhor solução

Procura IDA*

Características deste algoritmo:

- É completo, uma vez que vão sendo sucessivamente explorados nós para valores crescentes de f(n)
- A solução encontrada é ótima, atendendo ao modo como é definido o primeiro valor para o limite e à forma como é atualizado (valor mais pequeno de f(n))

Procura IDA*

Com efeito,

- O valor inicial para o limite coincide com o valor de h para o estado inicial. Sendo a heurística admissível, não poderá existir uma solução de custo inferior

Procura IDA*

- Como os valores sucessivos para o limite são os valores mais pequenos para a função f aplicada aos sucessores e como h é admissível, então não poderá haver nenhum caminho cujo custo esteja no intervalo entre dois valores de f
- Ou seja, temos sempre que todos os nós entre os limites f_i e f_{i+1} terão $f(n) = f_{i+1}$

Procura IDA*

- Consideremos agora a sequência ótima: como a heurística é admissível, nenhum nó nessa sequência pode tomar um valor superior ao valor ótimo, pelo que o algoritmo irá encontrar essa sequência quando o limite tomar o valor ótimo
- Finalmente, é impossível encontrar um caminho para a solução com um valor superior, visto que em cada iteração é escolhido para limite o menor valor encontrado

Procura IDA*

Carácter económico:

- Numa estratégia de procura em profundidade primeiro, o número de nós que é necessário manter em memória num dado instante é reduzido
- Admitamos que f_{opt} é o custo do caminho ótimo e que δ é o custo mais baixo dos operadores

Procura IDA*

Carácter económico:

- Então f_{opt}/δ representa a profundidade máxima a que se poderá encontrar a solução
- O espaço necessário será da ordem de $O(r \times (f_{opt}/\delta))$, ou seja, linear

Procura IDA*

Carácter económico:

- Relativamente ao tempo, tudo dependerá do número de valores diferentes dados por f, ou seja, tudo depende da heurística e do problema

Procura IDA*

Carácter económico:

- Se admitirmos que esses valores são todos distintos, então se o algoritmo A* visitar k nós, o algoritmo IDA* visitará

$$1 + 2 + 3 + \ldots + k = O(k^2) \text{ nós}$$

- O que se poupa em espaço pode ser perdido em tempo

Procura SMA*

- Simplified Memory-Bounded A*
- Este algoritmo foi desenvolvido para responder à mesma questão da complexidade espacial do A*
- Para ganhar maior eficiência do ponto de vista da memória, o algoritmo tem de abandonar o requisito de ser ótimo

Procura SMA*

Consiste no seguinte:

- Guardar o maior número possível de nós promissores (baixo custo dado por f(n))
- Guardar informação sobre a qualidade dos nós abandonados, para evitar expandir repetidas vezes

Procura SMA*

Funcionamento do algoritmo:

- Mantém numa fila a fronteira da árvore de procura, ordenando os nós pelos valores de f(n)
- A dimensão da fila é fixa e é um parâmetro do algoritmo

Procura SMA*

Funcionamento do algoritmo:

- Iterativamente, seleciona o elemento de menor custo, n, que, no caso de ser a solução, faz terminar com sucesso a procura
- Não sendo solução, calcula um dos seus sucessores, s_n, e determina o seu custo dado por f

Procura SMA*

Funcionamento do algoritmo:

- Atualiza, eventualmente, o custo de f(n) e dos seus antecessores, para guardar informação sobre a qualidade do caminho passando por n, no caso de todos os sucessores de n terem sido gerados
- Se a memória estiver cheia, retira da fila o nó de maior custo e insere s_n na fila

Procura SMA*

```
Função SMA* (problema): solução ou falha
    1. l_nós ← FazFilaOrdenada(EstadoInicial(problema))
    2. Repete
        2.1. Se VaziaFila(l_nós) Então
                2.1.1. Devolve falha
            Fim_de_Se
        2.2. nó ← RetiraFilaMelhor(l_nós)
        2.3. Se TesteObjectivo(nó) Então
                2.3.1. Devolve solução
            Fim_de_Se
        2.4. s_n ← PróximoSucessor(nó)
        2.5. f(s_n) ← max(f(nó), g(s_n)+h(s_n)) ou infinito se estiver à profundidade máxima
        2.6. Se TodosSucessoresGerados(nó) Então
                2.6.1. Actualiza(nó)
            Fim_de_Se
        2.7. Se TodosSucessoresMemória(nó) Então
                2.7.1. RetiraFila(nó)
            Fim_de_Se
        2.8. Se MemóriaCheia(l_nós) Então
                2.8.1. pior ← RetiraFilaPior(l_nós)
                2.8.2. RetiraListaFilhos(pior)
                2.8.3. InsereFila(Pai(pior))
            Fim_de_Se
        2.9. InsereFila(s_n)
    Fim_de_Repete
Fim_de_Função
```

Algoritmo de pesquisa SMA*

Procura SMA*

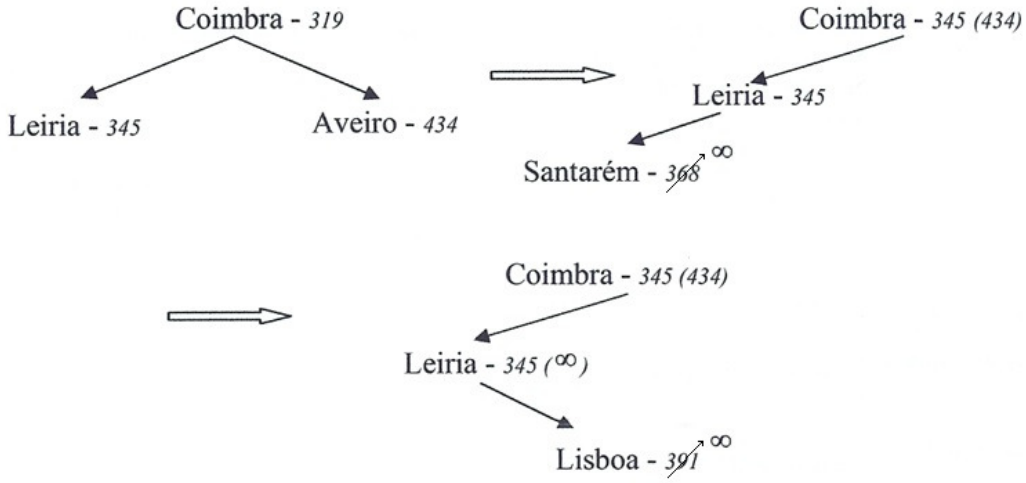
- A atualização do custo f(n) é feita por uma função recursiva simples:

```
Função Actualiza(nó): estrutura actualizada
    1. Se TodosSucessoresGerados(nó) e TemPai(nó) Então
            1.1. f(nó) ← min (f(s_n), para todo o sucessor s_n de nó)
            1.2. Se Mudou(f(nó)) Então
                    1.2.1. Actualiza(Pai(nó))
                Fim_de_Se
        Fim_de_Se
Fim_de_Função
```

Algoritmo da função Actualiza

Procura SMA*

- Simulação do algoritmo, admitindo que a dimensão máxima da memória é 3:
  (Coimbra - Faro)



Procura SMA*

- Este algoritmo não é em geral nem completo nem discriminador

- Para problemas particulares pode, no entanto, ser possível provar que possui essas propriedades

Procura SMA*

- Se para um determinado problema se sabe que a solução pode ser encontrada no máximo de m aplicações dos operadores e que o fator de ramificação máximo é r, então basta ter memória para armazenar r^m nós (tratando-se de uma expressão exponencial, os valores de m e de r terão de ser baixos)

Procura SMA*

- Ao manter uma memória fixa, é claramente económico do ponto de vista da complexidade espacial
- Esta característica é a mais importante visto que, mesmo quando ambas as complexidades são exponenciais, os programas rebentam sempre por razões de espaço antes de terem problemas com o tempo de execução

Procura Trepa-colinas

- Para alguns problemas o mais importante é encontrar o caminho que conduz do estado inicial ao estado final

- Para outros problemas, é a solução em si que interessa e não tanto o modo como foi obtida

Procura Trepa-colinas

- O problema das N-Rainhas pertence a esta última categoria:

  Para resolver este tipo de problemas é possível partir de uma solução candidata e ir tentando melhorá-la passo a passo

Procura Trepa-colinas

- No caso das N-Rainhas, por exemplo, podemos partir de um tabuleiro com as n rainhas colocadas em posições aleatórias e procurar movimentar uma delas de cada vez, de tal modo que o número de ataques diminua

Procura Trepa-colinas

- Este tipo de abordagem tem uma natureza tipicamente local e envolve ir melhorando progressivamente a solução candidata
- O algoritmo trepa-colinas baseia-se nesta filosofia
- Para além disso, o algoritmo vai descartando todos os vizinhos menos o melhor

Procura Trepa-colinas

- O algoritmo é extremamente simples:

```
Função TrepaColinas(problema): solução
   1. nó_corrente ← EstadoInicial(problema)
   2. Repete
         2.1. nó_seguinte ← Melhor(h, Expansão(nó_corrente, Operadores(problema)))
         2.2. Se h(nó_seguinte) > h(nó_corrente) Então
               2.2.1. Devolve nó_corrente
            Fim_de_Se
      2.3. nó_corrente ← nó_seguinte
    Fim_de_Repete
Fim_de_Função
```

Algoritmo da procura trepa colinas

Procura Trepa-colinas

- No exemplo do problema de encontrar o caminho de Coimbra até Faro:

ITERAÇÃO	NÓ_CORRENTE
0	[(Coimbra, 319)]
1	[(Leiria, 278)]
2	[(Lisboa, 195)]
3	[(Évora, 157)]
4	[(Beja, 99)]
5	[(Faro, 0)]

Conteúdo da lista Nó_corrente

Procura Trepa-colinas

- A distância real desta solução é de 576Km

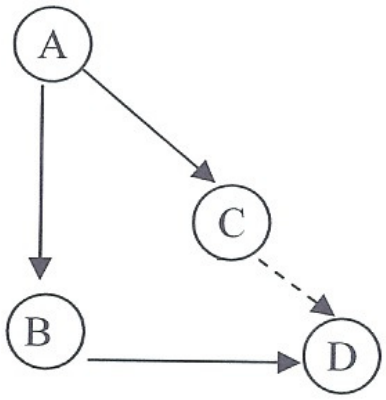
Procura Trepa-colinas

Características desta estratégia

- Não é uma estratégia completa
- Pode haver situações nas quais uma transição inicialmente promissora conduz para um estado que não é solução e cujos descendentes não melhoram o resultado dado pela função de avaliação

Procura Trepa-colinas

- O nó C encontra-se mais perto da solução D do que o nó B, pelo que será o nó selecionado e B descartado
- Se não houver ligação direta de C a D, o algoritmo terminará sem encontrar a solução D



Procura Trepa-colinas

Características desta estratégia

- Também não é uma estratégia ótima

- No exemplo das cidades, a solução

  Coimbra – Leiria – Lisboa – Setúbal – Faro

  tem uma distância real de apenas 495Km

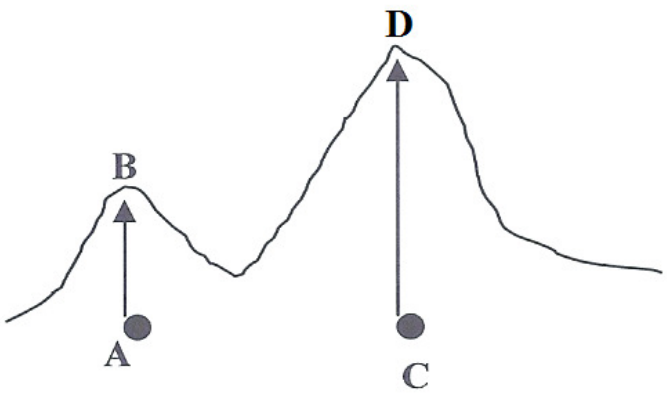
Procura Trepa-colinas

Problema dos máximos locais

- As duas situações anteriores podem ser descritas visualmente pelo que se costuma designar por problema dos máximos locais

Procura Trepa-colinas

Problema dos máximos locais



Máximos locais

Procura Trepa-colinas

- Admitamos que o problema se resume a alcançar o ponto mais alto numa montanha
- Partindo do ponto A, escolhido aleatoriamente, o algoritmo consegue ir melhorando progressivamente a solução até alcançar o ponto B (solução subóptima)

Procura Trepa-colinas

- Nessa altura é impossível melhorar a solução e chegar ao ponto D
- Se o ponto de partida tivesse sido o ponto C, a solução óptima seria encontrada

Procura Trepa-colinas

Caracterização económica

- Este algoritmo guarda em memória apenas um estado, pelo que a sua complexidade espacial é constante, ou seja, de ordem O(k), em que k é uma constante

Procura Trepa-colinas

Caracterização económica

- Do ponto de vista temporal a sua complexidade é semelhante à de uma procura em profundidade primeiro
- Se considerarmos um fator de ramificação r e a solução se encontrar no nível n, então a complexidade temporal será da ordem de O(r x n)

Procura Trepa-colinas

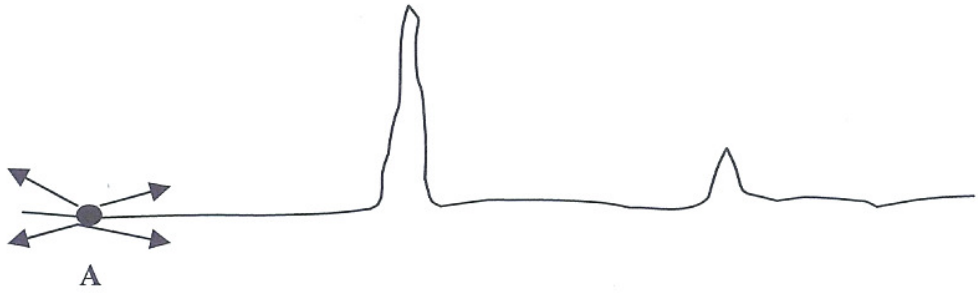
Problema dos Planaltos

- A utilização deste algoritmo depende bastante da "natureza do terreno", ou seja, do problema
- Para além do problema dos máximos locais, também a existência de planaltos pode dificultar o funcionamento do algoritmo

Procura Trepa-colinas

Problema dos Planaltos

- Uma vez que o algoritmo tem uma visão do terreno que é local, se a função de avaliação devolver valores idênticos para todos os vizinhos, a respetiva escolha será aleatória



O problema dos planaltos

Procura Trepa-colinas

- Existem soluções para estes problemas (mantendo a filosofia do trepa-colinas, pois se admitirmos a possibilidade de retrocesso teremos o algoritmo da procura sôfrega)

Procura Trepa-colinas

Uma possibilidade de tratar os máximos locais:

- Retomar o algoritmo a partir de uma nova posição
- Do ponto de vista da complexidade, o algoritmo deixaria de ser tão apelativo

Procura Trepa-colinas

Uma solução para o problema dos planaltos:

- Deixar o algoritmo ter uma vizinhança maior
- Ou seja, permitir a análise do que se passa depois de usar mais do que um operador em cadeia
- Deixaremos de ter um algoritmo tão simples e, ao mesmo tempo, não sabemos qual deve ser a dimensão do "olhar para a frente"

Procura Tabu

- A procura tabu é uma estratégia heurística que procura melhorar progressivamente uma solução através de uma pesquisa local
- O que distingue este algoritmo dos restantes, nomeadamente do trepa-colinas, é a existência de uma memória, geralmente designada por lista tabu, destinando-se:
  - A evitar ciclos,
  - A explorar zonas promissoras,
  - Ou a forçar a visita a novas zonas do espaço de procura

Procura Tabu

- Esse efeito é conseguido, uma vez que a lista tabu contém:
  - soluções que foram testadas ou
  - os operadores que foram utilizados
- O conteúdo da lista é usado fundamentalmente para
  - inibir o teste de soluções ou
  - evitar operadores

Procura Tabu

A memória pode ter várias dimensões:

- as que medem o momento da ocorrência de um dado evento (*recency-based memory*) – uso de um operador, por exemplo
- as que medem a frequência com que determinado evento ocorreu (*frequency-based memory*)

Procura Tabu

- Na 1.ª dimensão, porque são guardadas as ocorrências mais recentes, podemos dizer que estamos perante uma memória de curto termo
- Na 2.ª dimensão, como são guardadas situações mais antigas, podemos falar de memória de longo termo

Procura Tabu

Na procura tabu a memória pode ser utilizada segundo diferentes estratégias:

- para intensificar a busca na vizinhança de soluções de boa qualidade
- para diversificar a busca para regiões do espaço de procura ainda não visitadas

Procura Tabu

- O algoritmo é formado por 2 ciclos principais
- No ciclo externo controla-se o número de vezes que se tenta melhorar uma solução
- No ciclo interno estabelece-se o modo de produzir a nova solução candidata a partir da solução corrente, recorrendo aos vizinhos desta e à informação contida na memória tabu

Procura Tabu

```
Função ProcuraTabu(problema): nó_corrente
    1. memória_tabu ← ∅
    2.  nó_corrente ← GeraEstadoInicial(problema)
    3. Repete max_vezes
            3.2. Repete oper_vezes
                    1.2.1. nó_corrente_local ← nó_corrente
                    1.2.1. nó_escolhido ←MelhorVizinhoPossível(memória_tabu,
                                                    nó_corrente_local)
                    1.2.2. ActualizaMemóriaTabu(memória_tabu)
                    1.2.3. Se Melhor(nó_escolhido, nó_corrente_local) Então
                                1.2.3.1. nó_corrente_local ← nó_escolhido
                            Fim_de_Se
                Fim_de_Repete
            3.3. Se Melhor(nó_corrente_local, nó_corrente) Então
                    1.3.1. nó_corrente ← nó_corrente_local
                Fim_de_Se
        Fim_de_Repete
    4. Devolve nó_corrente
Fim_de_Função
```

Algoritmo de procura tabu

Procura Tabu

Características

- Este algoritmo apresenta características semelhantes às do algoritmo trepa-colinas
- Não é completo
- Não é discriminador

Procura Tabu

Características

- Tem complexidade espacial constante (embora superior ao trepa-colinas devido à existência da memória tabu)
- Tem complexidade temporal de ordem O(rxn), sendo r o fator de ramificação e n o valor máximo das tentativas de procura da solução