**CAPITULO 3**

* Agente solucionador de problema
* Formulação do objetivo: Encontrar uma sequencia de ações que o leve ao objetivo.
* O Ambiente é Estático, Acessível, Discreto, Determinístico.
* Formulação do problema: Algumas característica do ambiente são desconsiderada ou seja, é realizado uma Abstração: Processo de remover detalhes, ex.: Entre as rodas de cidade, existir (Barreiras policiais, tempo, clima, etc.).
* Componentes do problema: Estado inicial (Começa o processo de busca, Possíveis ações do agente (Função sucessor)), Teste de meta (determinar se um dado estado é final ou não), custo do caminho.
* Solução ótima é a solução com o menor custo do caminho.
* Soluções por busca:

**Arvore de busca:** A cada passo, o algoritmo de busca deve escolher um nodo folha para expandir.

* Podemos dizer que o nó esta em uma fila e são inseridos de diferentes maneiras pelo algoritmo.
* **Critérios de avaliação das estratégias de busca:**
* Completo: Sempre encontra a solução? / Tempo: Traz no menor tempo / Espaço de Memoria
* B : Numero de sucessores por no / d: profundidade da solução / m:profundidade máxima
* **Busca Cega:** Não tem informação sobre o numero de passos e custo do estado atual para o final
* **Busca Heurística:** Tem informações suficiente para perceber que um estado não final é mais promissor que o outro.

**Busca por Amplitude(LARGURA):** Expande todos os nós do primeiro Nível, depois do segundo ,e assim por diante, pode usar fila FIFO. (Completo, Tempo Exponencial, nodos ultimo nível na memória, Otimo se o custo fixo por expansão).

**Busca por custo Uniforme:** Explora primeiro o nó de menor custo, obtendo assim a solução de menor custo.

**Busca por profundidade:** Explora o nó mais a esquerda e depois o filho deste nó.(Pilha).

* *Completo*: Não pode falhar se a arvore ter ramos infinitos.
* *Tempo*: Em problemas com várias soluções ele pode ser mais rápido, porque vai ate o nodo folho antes de analisar os demais..
* *Ótimo*: Não

**Busca por Aprofundidade Iterativo:** Busca em profundidade para encontrar o melhor limite, - incrementando gradativamente – Primeiro 0, depois 1 e então 2...

* Requisitos de memória modesto (Busca Profundidade) – Completo se fator ramificação b for finito(Busca Largura). Ótimo quando o custo do caminho é crescente em função da profundidade do nodo.
* A busca por aprofundamento iterativo pode ser considera dispendiosa visto que explora os mesmos nodos repedidamente. No entando, os nodos do d-ésimo nível são explorado somente uma vez, os nodos do (d-1)-ésimo nível são explorados duas vezes, os do (d-2)-ésimo nível são explorados 3 vezes, e assim sucessivamente.

**Busca com informação Parcial**

* Múltiplos estados - O Agente pode estar em qualquer um dos possíveis estados iniciais, e cada ação pode leva-lo a qualquer um dos vários estados sucessores possíveis.
* Problema de contingência – Após cada ação os agentes fornecem novas informações e define novas contingência que deve ser planejada.
* Problema de exploração – Quando estados e ações são desconhecidos, o agente deve agir pra descobri-los. (Estar perdido em uma cidade)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CAPITULO 4 – MÉTODOS DE BUSCA INFORMADA**

* Encontra a solução, utilizando conhecimento específico do problema

**Algoritmo “Best-first search” (O Melhor primeiro)**

- É realizada uma avaliação f(n) que retorna qual o nodo deseja expandir, ou seja, o nodo com a menor avaliação ao objetivo é expandido primeiro.

- O “best-first search” é respeitável mas incorreto ou impreciso, pois na verdade escolhemos o nodo que **PARECE** ser o melhor de acordo com a avaliação.

- Função Heurística: Calcula uma estimativa de custo h(n) e estima o custo do caminho de um estado n ate o objetivo.

**-> Busca Gulosa: Minimiza o custo estimado para alcançar o objetivo**

**-** Sempre expande o primeiro nodo que for julgado mais próximo da meta.

- Busca uma solução rápida, mas nem sempre ótima.

- Utiliza a função heurística h para determinar o próximo nó a ser expandido.

Exemplo: Função Heurística para encontrar a menor distancia, é traçar uma linha reta.

Propriedades:

Não é completa (pode gerar busca infinita);

Não é ótima (Nem sempre encontra a melhor solução);

Tempo (Profundidade máxima do espaço de busca (pior caso)).

**-> Busca A\*: Minimiza o custo total do caminho**

- Minimiza o custo do caminho, é ótimo e completo, mas pode ser muito ineficiente.

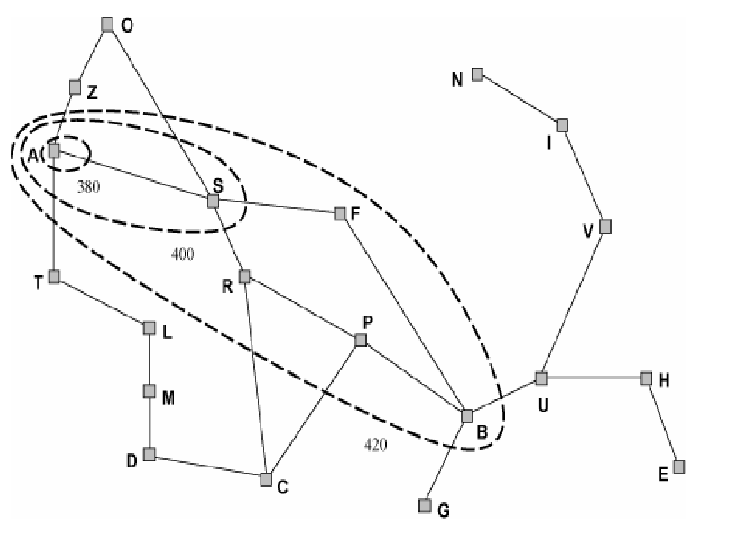
- combinar duas estratégias (Gulosa e custo uniforme) *h(n)* e *g(n): f(n) = h(n) + g(n),* onde:

*h(n)* é a estimativa de custo de *n* até o objetivo;

*g(n)* é o custo acumulado do estado inicial até *n*;

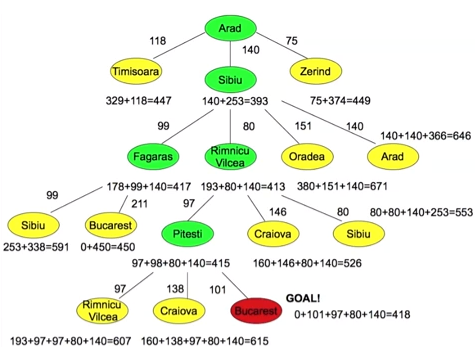
*f(n)* é o custo estimado da solução que passa por *n.*

**OBS.: Busca por custo Uniforme:** Explora primeiro o nó de menor custo, obtendo assim a solução de menor custo.



Semelhante a busca em largura, mas ao invés de geração de nodos da árvore por

profundidade, o que é considerado é o contorno gerado por *f*. Para a busca com custo uniforme (A\* usando *h(n)* = 0) os contornos são circulares ao redor do estado inicial.



**INVENTANDO FUNÇÕES HEURÍSTICAS**

Existem algumas estratégias para definir *h*:

1. **Relaxar restrições do problema**

Problemas com menos restrições nas ações/operadores são chamados de ***Problemas***

***relaxados*** (*"relaxed problem"*), em outras palavras, problema relaxado é uma versão

simplificada do problema original, onde os operadores são menos restritivos.

**2. Usar informações estatísticas - executar a busca sobre um número aleatório de problemas (configurações aleatórias).**

Ex.: Se em 90% dos casos, quando *h(n)* = 14, a distância real da solução é 18, então

quando o algoritmo encontrar 14 para o resultado da função, substitui-se esse valor por 18.

1. **Identificar os atributos mais relevantes do problema (exige aprendizagem).**

**TÉCNICAS DE BUSCA COM MEMÓRIA LIMITADA**

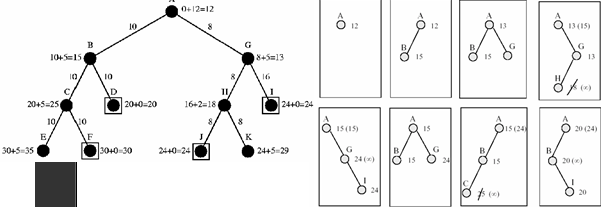
Problemas do A\* é a utilização de memória. Dois algoritmos com limitação de memória:

**IDA\*** *(Iterative Deepening A\*):* Igual ao aprofundamento iterativo, porém com limite

na função de avaliação *f* ao invés da profundidade *d*.

**SMA\*** *(Simplified Memory Bounded A\*):* Similar ao A\*, mas restringe o tamanho da

fila para que caiba na memória. O número de nós guardados em memória é fixado previamente. (Nós esquecidos) são os nós retirados, para economizar memoria.

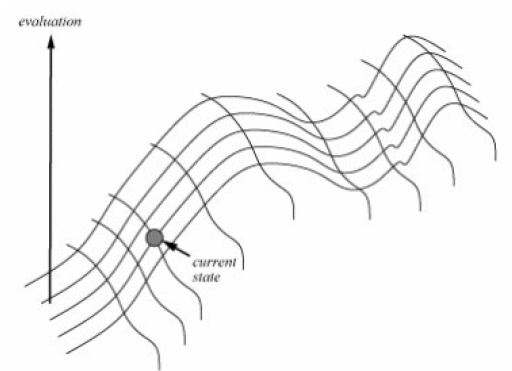
****

**ALGORITMO DE MELHORIAS ITERATIVAS (AMI)**

Começa em um estado inicial, e melhora iterativamente. Os estados são representados em uma superfíciel (função de avaliação) onde a altura de qualquer ponto na superfície corresponde a sua avaliação do estado naquele ponto.

O algoritmo se "move" pela superfície em busca de pontos mais altos/baixos (dependendo do objetivo): o ponto mais alto/baixo (máximo/mínimo global) corresponde a solução ótima.

Esses algoritmos guardam apenas o estado atual, e não vêem além dos vizinhos imediatos.



**HILL-CLIMBING (Subida da encosta) -** Só faz modificações q melhoram o estado atual.

Guarda o estado atual e sua avaliação. É um “loop” que fica se movendo na direção crescente (para maximizar) ou decrescente (para minimiza) da função de avaliação. O algoritmo move-se sempre na direção que apresenta a maior taxa de variação de f.

Pode gerar três problemas:

Máximos locais: Picos mais baixos que os pico mais alto, o algoritmo para porque considera o melhor, por não conhecer além dos seus vizinhos.

Planícies/Platôs e Picos.

**SIMULATED ANNEALING**

-Pode fazer modificações que pioram o estado temporariamente para possivelmente melhorá-lo no futuro (ideal para problema que encontram muitas planícies e muitos máximos locais).

- Semelhante a subida da encosta, porém oferece meios para se escapar de máximos

locais: ao invés de recomeçar em um local aleatório, retrocede para situações piores, no entanto não garante que a solução encontrada seja a melhor possível.

**BUSCA ONLINE X OFFLINE**

**Busca offline:** eles calculam uma solução completa antes de entrar no mundo real, e depois executam a solução sem recorrer a suas percepções. Em contraste, um agente de **Busca online**, opera pela intercalação de planejamento e ação: primeiro ele executa uma ação, observa o ambiente e calcula a próxima ação. Ideal para ambientes dinâmicos e/ou estocásticos! É aplicada a problemas de exploração, em que os estados e as ações são desconhecidos para os agente. Ex.robô implantado num edifício.

**AGENTES DE BUSCA ONLINE**

Depois de cada ação, um agente online recebe uma percepção informando-o de qual estado ele alcançou; a partir dessa informação, ele pode ampliar seu mapa

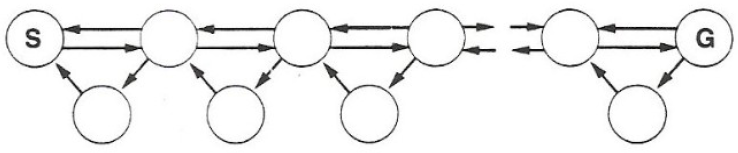
do ambiente.

**BUSCA LOCAL ONLINE**

Assim como a busca em profundidade, a busca da subida na encosta tem a propriedade da localidade em suas expansões de nós.

Problema:

Não é possível usar reinício aleatório => caminhamento aleatório (random walking)!!



**LRTA\* - LEARNING REAL TIME A\*** subida da encosta com memória!

Armazenar a melhor estimativa atual. H(s) começa sendo apenas a estimativa h(s) e é atualizada a medida que o agente ganha experiência.

