Procura local e otimização

13 de dezembro de 2023 21:41

Procura fazendo local avaliando e modificando 1 ou mais estados atuais

→ identificação de soluções o mais próximas quanto possível da solução

- Exploration : exploração geral do espaço
- Exploitation : procura focada nos locais mais promissores

- *Exploration* sem *Exploitation* permite ter uma visão geral do espaço de procura, mas sem chegar muito próximo do valor ótimo;
- Partir para a *Exploitation* de uma zona numa fase inicial do processo de procura pode levar a que a procura fique presa num ótimo local;
- Este balanceamento é normalmente gerido com sucesso através de Meta-heurísticos.

⇒ Não garantem solução ótima

↳ não se conhecem algoritmos eficientes

Algoritmos de melhoria iterativa

- Algoritmos de Melhoria Iterativa:
  - Procura Subida da Colina (Hill-Climbing Search)
  - Arrefecimento Simulado (Simulated Annealing)
  - Procura Tabu (Tabu Search)
  - Algoritmos Genéticos (Genetic Algorithms)
  - Colónia de Formigas (Ant Colony Optimization)
  - Enxame de Partículas (Particle Swarm Optimization)

- **Procura local vs Procura global**
  - Algumas meta-heurísticas aplicam métodos de procura local, onde as novas soluções exploradas são "vizinhas" de soluções anteriores (e.g. *Simulated Annealing, Tabu Search*);
  - Outras meta-heurísticas distribuem o processo de procura por todo o espaço de procura (normalmente através de abordagens baseadas em populações).

- **Solução única vs *Population-based***
  - As abordagens de solução única, são iterativas, e orientam o processo de procura através da melhoria da solução anterior;
  - As abordagens baseadas em populações utilizam uma procura em paralelo por parte de vários membros da população, podendo, ou não, existir a troca de informação entre os indivíduos (e.g. *Particle Swarm optimization, Genetic Algorithms, Ant Colony optmization*).

12

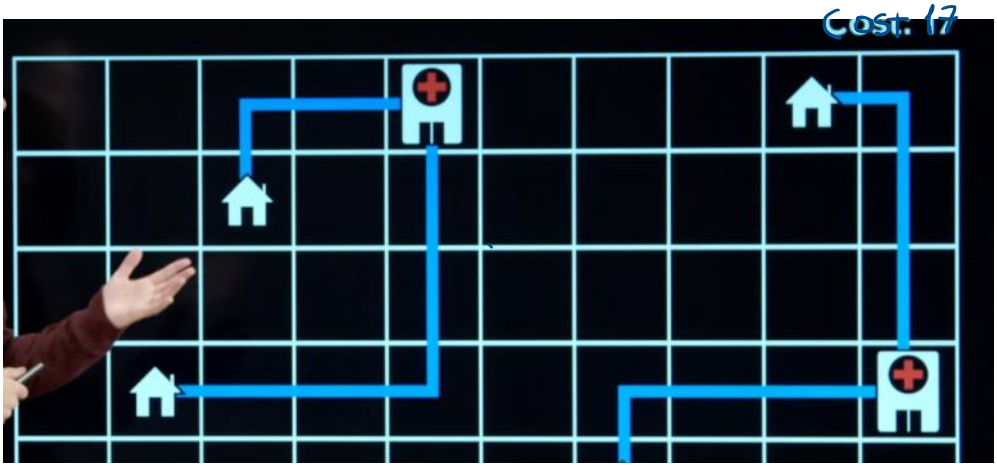
- Hill-Climbing Search:
  - Escolher um estado aleatoriamente do espaço de estados
  - Considerar todos os vizinhos desse estado
  - Escolher o melhor vizinho
  - Repetir o processo até não existirem vizinhos melhores
  - O estado atual é a solução

- Simulated Annealing:
  - Semelhante ao Hill-Climbing Search mas admite explorar vizinhos piores
  - Temperatura que vai sendo sucessivamente reduzida define a probabilidade de aceitar soluções piores

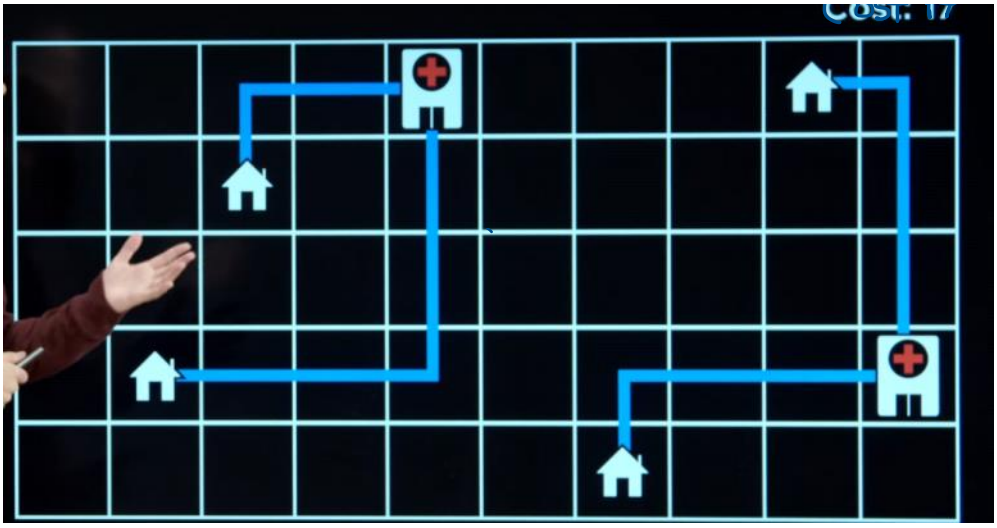
- Tabu Search:
  - Semelhante ao Hill-Climbing Search, explora os estados vizinhos mas elimina os piores (vizinhos tabu)
  - Algoritmo determinístico

local search
↳ a single node

⇒ bad for ages
⇒ don't care about the solution



Add another hospital?
↓
where?



Add another hospital?

↓

where?

How to minimize?

→ objective function

max ⇒ obj-func (state) → value

min ⇒ cost-function (state) → value

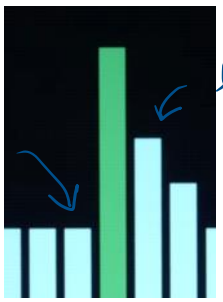
Hill climbing

↳ maximize the value

⇒ consider the neighbours

↳ move to the highest neighbour

until →



lower

lower

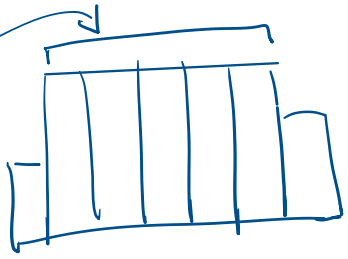
(hopefully the maximum)

```
function HILL-CLIMB(problem):
    current = initial state of problem
    repeat:
        neighbor = highest valued neighbor of current
        if neighbor not better than current:
            return current
        current = neighbor
```

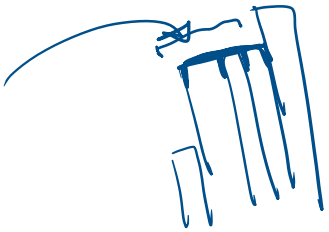
⇒ could stop at a local maximum

⇒ NAIVE

↳ OR at a flat local maximum



shoulder



Variant	Definition
steepest-ascent	choose the highest-valued neighbor
stochastic	choose randomly from higher-valued neighbors
first-choice	choose the first higher-valued neighbor
random-restart	conduct hill climbing multiple times

→

↳ But none of the variations choose at any time a state that is worse than the current

⇒ even tho it is sometimes necessary

higher temperature ⇒ more likely to move to a worst neighbour

energy: how worse/better?

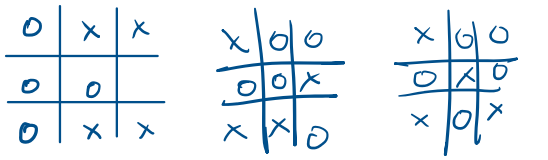
↳ less likely to move to a very bad neighbour

```
function SIMULATED-ANNEALING(problem, max):
    current = initial state of problem
    for t = 1 to max:
        T = TEMPERATURE(t)
        neighbor = random neighbor of current
        ΔE = how much better neighbor is than current
        if ΔE > 0:
            current = neighbor
        with probability e^(ΔE/T) set current = neighbor
```

$e^{\Delta E/T}$

Adversarial type of search

→ Minimax: values to the possible outcomes



{ -1 , 0 , 1 }

- S_0 : initial state
- PLAYER(s) : returns which player to move in state s
- ACTIONS(s) : returns legal moves in state s
- RESULT(s, a) : returns state after action a taken in state s
- TERMINAL(s) : checks if state s is a terminal state
- UTILITY(s) : final numerical value for terminal state s

↳ Utility

Procura Tabu

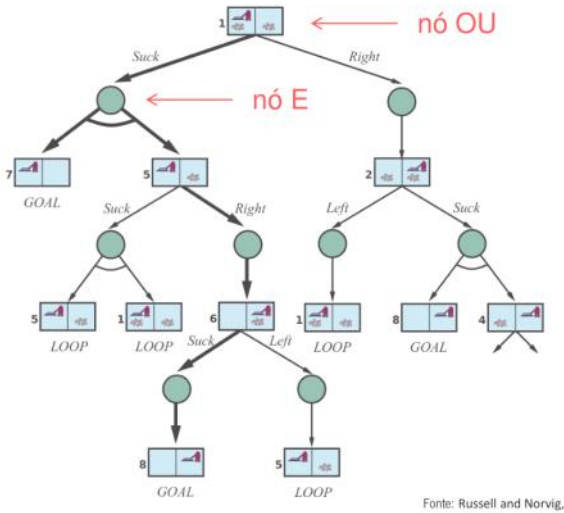
↳ Penalizar movimentos que levam a solução para espaços de procura visitados anteriormente.

- A Procura Tabu, no entanto, aceita de forma determinística soluções que não melhoram para evitar ficar presa em mínimos locais.
- **Estratégia:**
  - Ideia chave: manter a sequência de nós já visitados (Lista tabu);
  - Partindo de uma solução inicial, a procura move-se, a cada iteração, para a melhor solução na vizinhança, não aceitando movimentos que levem a soluções já visitadas, esses movimentos conhecidos ficam armazenados numa lista tabu;
  - A lista permanece na memória guardando as soluções já visitadas (tabu) durante um determinado espaço de tempo ou um certo número de iterações (prazo tabu). Como resultado final é esperado que se encontre um valor ótimo global ou próximo do ótimo global.

```java
public Solution run(Solution initialSolution) {
        Solution bestSolution = initialSolution;
        Solution currentSolution = initialSolution;

        Integer currentIteration = 0;
        while (!stopCondition.mustStop(++currentIteration, bestSolution)) {

                List<Solution> candidateNeighbors = currentSolution.getNeighbors();
                List<Solution> solutionsInTabu = IteratorUtils.toList(tabuList.iterator());

                Solution bestNeighborFound = solutionLocator.findBestNeighbor(candidateNeighbors, solutionsInTabu);
                if (bestNeighborFound.getValue() < bestSolution.getValue()) {
                        bestSolution = bestNeighborFound;
                }

                tabuList.add(currentSolution);
                currentSolution = bestNeighborFound;

                tabuList.updateSize(currentIteration, bestSolution);
        }

        return bestSolution;
}
```

```java
@Override
public Solution findBestNeighbor(List<Solution> neighborsSolutions, final List<Solution> solutionsInTabu) {
        //remove any neighbor that is in tabu list
        CollectionUtils.filterInverse(neighborsSolutions, new Predicate<Solution>() {
                @Override
                public boolean evaluate(Solution neighbor) {
                        return solutionsInTabu.contains(neighbor);
                }
        });

        //sort the neighbors
        Collections.sort(neighborsSolutions, new Comparator<Solution>() {
                @Override
                public int compare(Solution a, Solution b) {
                        return a.getValue().compareTo(b.getValue());
                }
        });

        //get the neighbor with lowest value
        return neighborsSolutions.get(0);
}
```

Arvores E-OU (*AND-OR*)



- ações do agente – nós OU
- result. no mundo – nós E

- Plano genérico
  - –um objetivo em cada folha
  - –uma ação em cada nó OU
  - –todos os resultados em nós E

Fonte: Russell and Norvig, (2009) Artificial Intelligence - A Modern Approach.

21

Ambiente acessível VS inacessível

- Um ambiente acessível é aquele em que o agente pode obter informações completas, exatas e atualizadas sobre o estado do ambiente;
- Os ambientes mais moderadamente complexos são inacessíveis.

- Um ambiente determinístico é aquele em que qualquer ação tem um único efeito garantido;
- O mundo real é, para nós ***humanos***, não determinístico.

? um pouco mais complicado do que isso, mas acho tendo em conta o contexto

- Num ambiente episódico, o tempo de execução do agente pode ser dividido numa série de intervalos (episódios) que são independentes uns dos outros, no sentido em que o que acontece num episódio não tem qualquer influência sobre os outros episódios

→ Estratégia e eficiência

Tipos de jogos

↳ Informação perfeita ↳ xadrez
vs imperfeita
↳ poker

→ norte, determinístico

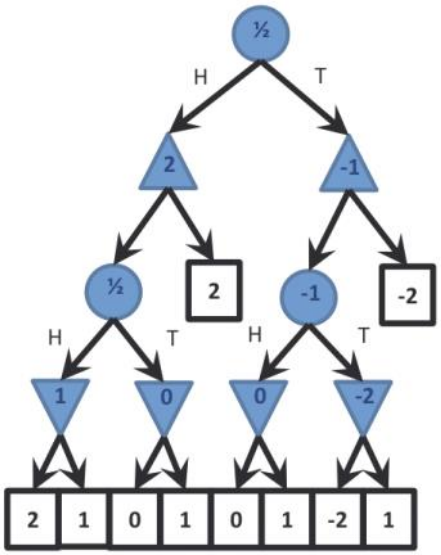
Características
→ adversário
→ oponente imprevisível
→ tempo limite mas necessária na aproximação

36 biliões de avaliações/s
⇒ 18 jogadas

Jogos estocásticos
↳ incerteza/aleatoriedade
⇒ probabilidade e sorte
⇒ nós de probabilidade
⇒ ExpectMiniMax
↳ expected reWard

$$Value(node) = \max_{\text{my moves}}\left(\min_{\text{min's moves}}\left(\bullet\right)\right)$$

$$\bullet = \mathbb{E}[Reward]$$
$$= \sum_{\text{outcomes}} Probability(outcome) \times Reward(outcome)$$



Neste caso em vez de se utilizar apenas o valor atribuido aos nós da árvore também se utiliza a probabilidade do acontecimento.

$$E_{P(X)}[f(X)] = \sum_x f(x)P(x)$$

A função tem em conta tanto a probabilidade de um acontecimento como o seu valor dando uma estimativa do valor médio dos nós filho daquele ramo.

Exemplo:

Se pensarmos nos nós "hipótese" como nós Min, neste caso iriamos escolher o nó da direita.