

**PROJETO PROCURA E PLANEAMENTO**

**Campus Alameda**

**2018/19**

**GRUPO** 18

**Problema de Afectação em Empresas de Transporte**

Rafael Koener, ist176475

Kevin Batista Corrales, ist194131

Classificação:

Soma das horas gastas exclusivamente para fazer este trabalho: 36h

**Conteúdo**

[1. Introdução 3](#_Toc533162354)

[2. Estrutura de dados 3](#_Toc533162355)

[3. MODELAÇÕES DO PROBLEMA 4](#_Toc533162356)

[4. Algoritmos utilizados 6](#_Toc533162357)

[4.1 A\* 6](#_Toc533162358)

[4.2 IDA\* 6](#_Toc533162359)

[4.3 Profundidade Primeiro 6](#_Toc533162360)

[4.4 Largura primeiro 6](#_Toc533162361)

[4.5 Profundidade Iterativa 7](#_Toc533162362)

[4.6 Sondagem Iterativa 7](#_Toc533162363)

[4.7 ILDS (Improved Limited Discrepancy Search) 7](#_Toc533162364)

[4.8 Algoritmo Alternativo 7](#_Toc533162365)

[5. HEURÍSTICAS DESENVOLVIDAS 8](#_Toc533162366)

[6. ESTRATÉGIAS DE CORTE 10](#_Toc533162367)

[7. Resultados 12](#_Toc533162368)

[8. CONCLUSÃO 13](#_Toc533162369)

[9. BIBLIOGRAFIA 14](#_Toc533162370)

# 

# Introdução

Este projeto tem como objetivo desenvolver um programa que resolva um problema de afectação completa das tarefas de condução de veículos de transporte, a realizar pelos trabalhadores durante um dia de trabalho, a turnos de serviço, respeitando todas as restrições.

Foram utilizados para procura da solução ótima os seguintes algoritmos: A\*; IDA\*; Profundidade Primeiro; Largura Primeiro; Profundidade Iterativa; Sondagem Iterativa; ILDS; Algoritmo Alternativo.

Para os algoritmos A\* e IDA\* foram desenvolvidas heurísticas utilizadas na decisões de escolha na geração de nós.

Foi realizado a avaliação de implementações de cada algoritmo e cada heurística desenvolvida tanto de ponto de vista quantitativo como qualitativo, com o objetivo de encontrar a melhor solução possível.

# Estrutura de dados

Para a estruturação dos dados, na implementação do projeto, foi decidido estruturar o estado do problema (*state*):

No estado decidimos implementar informação como: *shifts* (lista de turnos, cada turno é uma lista de tarefas que foi atribuída dependendo das restrições, verificação da localização e verificação do tempo final e inicial de cada tarefa) e *unusedTasks* (lista de tarefas não utilizadas).

A lista de tarefas não utilizadas (*unusedTasks*) será utilizada para o processamento de atribuição e organização de turnos, será verificado em que turno pertence cada tarefa da lista.

A lista de turnos (*shifts*) será retornada após o problema estiver completamente resolvido.

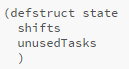


Figura 1 - G18.lisp - state

# MODELAÇÕES DO PROBLEMA

Na modelação do problema foi utilizado a função *cria-problema* (disponibilizada na biblioteca *procura.lisp*) em que recebe como argumento um estado-inicial, lista de operadores, o objetivo final, função de custo, heurística, o algoritmo de procura e profundidade máxima.

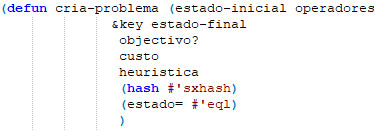


Figura 3 – procura.lisp – cria-problema

* *estado-inicial* será uma lista de tarefas, em que cada tarefas contém uma lista com informação, ordenada, sobre: local de partida; local de chegada; instante de partida; instante de chegada.

Exemplo:

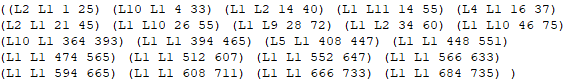


Figura 4 Exemplo estado-inicial

* *operadores* será uma lista de operadores possíveis para a alteração do estado do problema. Consideramos como operadores possíveis: adicionar turno (*addShift*) e adicionar tarefa (*addTask*).
  + *addShift* nesta operação é criado um novo turno na lista de turnos do estado (*shifts*)

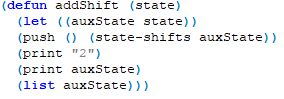


Figura 5 - G18.lisp - addShift

* + *addTask* nesta operação é adicionada uma tarefa ao um turno que pertence, que se encontra na lista de turnos do estado (*shifts*).

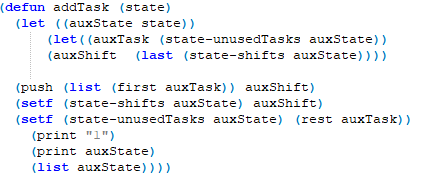


Figura 6 - G18.lisp - addTask

* *objetivo?* será uma função que representa o objetivo final do problema, como objetivo final será considerado o esvaziamento total de *unusedTasks* do estado (lista de tarefas não utilizadas). Este facto significa que todas as tarefas do estado inicial foram atríbuidas em turnos e o problema foi resolvido.

https://i.gyazo.com/522b994dc5808a1c9e051c10066da886.png

Figura 7 - g18.lisp - objetivo?

* *custo* será uma função de cálculo de custo de cada operação realizada.
* *heurística* é um argumento opcional dependendo do algoritmo de procura escolhido.
* *algoritmo/estratégia de procura*, tal como o nome indica, este argumento recebe o nome do algoritmo de procura a utilizar.
* *profundidade máxima*, este argumento é opcional, indica o valor da profundidade máxima. Este argumento depende do algoritmo de procura utilizado.

# Algoritmos utilizados

## A\*

O algoritmo a\* utiliza uma combinação de aproximações heurísticas para buscar um caminho até a solução num grafo desde o nó inicial até ao final. Este algoritmo tem uma complexidade espacial exponencial.

Este algoritmo é muito utilizado em resolução de problemas, resolução de quebra-cabeças, jogos e encontrar rotas de deslocamento.

O algoritmo a\* inicialmente expande o nó que pertence ao caminho com menor custo associado, este custo é calculado a partir da função: f(n) = g(n) + h(n), onde g(n) dá o valor do custo do caminho percorrido desde a raiz até o nó “n”. De seguida o algoritmo tenta primeiro os nós com menor custo até encontrar caminho.

Na implementação do a\* é utilizado uma lista de nós abertos e fechados, em que a lista de nós abertos é de inserção ordenada e a busca é feita por estados já visitados.

## IDA\*

O algoritmo ida\* é semelhante ao a\*, porém no algoritmo ida\* são definidos limites na profundidade de procura com o objetivo de economizar memória. Utilizando a mesma função usado no a\* para calcular o custo: f(n) = g(n) + h(n).

A desvantagem do algoritmo ida\* em relação ao a\* são os estados repetidos o que causa um maior custo de CPU.

A implementação do ida\* já difere do a\*, em que não será necessária uma lista porque a função é recursiva e o número máximo de nós alocados é a profundidade da solução.

## Profundidade Primeiro

O algoritmo em profundidade primeiro começa num nó raiz, explorando a partir

dele o quanto possível em cada ramo que exista, antes de realizar *backtracking*.

No caso de realizar uma procura numa árvore, esta é percorrida sistematicamente de cima para baixo e da esquerda para direita. Quando essa pesquisa falha, ou é encontrado uma solução, ou entra em funcionamento o mecanismo de *backtracking*.

*Backtracking* é um procedimento em que o sistema retorna pelo mesmo caminho percorrido com a finalidade de encontrar soluções alternativas.

## Largura primeiro

O algoritmo em largura primeiro começa num nó raiz, explorando inicialmente todos os vértices vizinhos, para cada vértice mais próximo é explorado os seus vizinhos não visitados e assim sucessivamente, efetuando uma busca exaustiva.

O algoritmo tem que garantir que nenhum vértice será visitado mais que uma vez. Utilizando uma estrutura de fila, em que garante a ordem de chegada dos vértices, será realizada, através da ordem de chegada, visitas aos vértices. Um vértice que já foi marcado não poderá entrar novamente na fila.

## Profundidade Iterativa

O algoritmo em profundidade iterativa combina as vantagens do algoritmo em profundidade primeiro e o algoritmo em largura primeiro (os dois algoritmos explicados anteriormente). O que tem origina uma capacidade de examinar todo o espaço de estados encontrando a solução ótima com pouco uso de memória.

Na primeira iteração, a árvore é gerada utilizando uma busca em profundidade limitada com limite de valor 1. Caso a solução não for encontrada, é iniciada uma segunda iteração, descartando toda a árvore anterior e construindo uma nova com a procura em profundidade limitada novamente, mas com o limite igual a 2. Continuando este processo sucessivamente, incrementando o valor do limite até que seja encontrada uma solução.

## Sondagem Iterativa

O algoritmo em sondagem iterativa é um algoritmo otimizado do algoritmo de busca melhor-primeiro, baseado em heurística que explora o nó mais promissor num conjunto limitado.

O algoritmo melhor-primeiro utiliza o algoritmo f(n)=h(n) de procura até o nó destino, procurando assim expandir o nó que é mais próximo ao objetivo, que resulta numa condução rápida até o nó destino. A heurística é aplicada globalmente, o nó aberto com a melhor nota é utilizado para a expansão.

O algoritmo em sondagem iterativa é considerado como um algoritmo *greedy.* Um algoritmo *greedy* é uma técnica que tem como objetivo encontrar a solução ótima global, fazendo a escolha localmente ótima em cada fase.

## ILDS (Improved Limited Discrepancy Search)

O algoritmo ILDS, tal como o nome indica, é uma versão melhorada (*improved*) do algoritmo de procura LDS (*Limited Discrepancy Search*). Para cada k iteração de discrepância é gerado todos os caminhos com k ou menos ramos localizados à direita. Cada iteração cria todos os caminhos de todas as iterações previamente geradas.

Dado um valor máximo para a profundidade, o algoritmo pode ser modificado para que cada iteração gere apenas os caminhos com exatamente k discrepâncias.

## Algoritmo Alternativo

# HEURÍSTICAS DESENVOLVIDAS

Foi considerado várias heurísticas mas chegamos a conclusão em utilizar as seguintes:

* Tempo total que sobrou dos turnos do estado

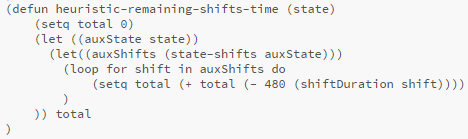


Figura 8 - G18.lisp - heuristic-remaining-shifts-time

Esta heurística tem como utilidade calcular: a percentagem de tempo que sobrou do turno.

Com a informação obtida a partir da heurística podemos concluir: se existe grande desperdício de tempo laboral contido em cada turno.

* Número total de turnos com menos de 6h de serviço contido no estado

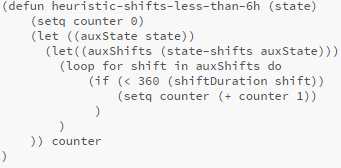


Figura 9 - G18.lisp - heuristic-shifts-less-than-6h

Esta heurística permite o cálculo de: percentagem de número de turnos inferior a 6h.

Com a informação obtida a partir da heurística podemos concluir: se a distribuição das tarefas entre os turnos foi bem realizada.

* Número total de turnos que não tem como ponto de partida a localização “L1” dentro do estado

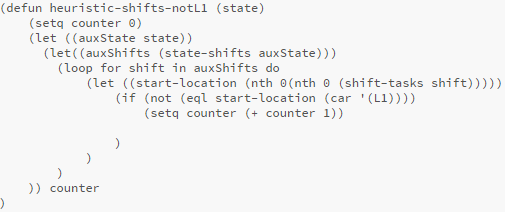


Figura 10 - G18.lisp - heuristic-shifts-notL1

Foi escolhido esta heurística devido a possibilidade de calcular: o estado com menor ou maior quantidade de turnos que tem ponto de partida a localização “L1”; a percentagem de estados que não iniciam na localização “L1”.

Com a informação obtida a partir da heurística podemos concluir: se existe um grande desperdício de tempo em transporte para ser possível iniciar o serviço.

* Número total de turnos do estado

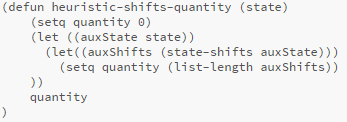


Figura 11 - G18.lisp - heuristic-shifts-quantity

Foi escolhido esta heurística devido a possibilidade de cálculo do estado com menor ou maior quantidade de turnos.

Com a informação obtida a partir da heurística podemos concluir: se a distribuição das tarefas entre os turnos foi bem realizada.

# ESTRATÉGIAS DE CORTE

As estratégias de corte são importantes em domínios ruidosos, nós com profundidade relativamente grande aumentam o tempo de processamento e uso de memória. Cortar um nó significa trocar nós profundos por folhas.

Os métodos de corte podem ser divididos em dois grupos:

* Pré-poda: verifica as regras de paragem e previnem a construção dos ramos que não melhorariam a predição do resultado.
* Pós-poda: método mais comum baseado no erro estático. Construir uma árvore é mais lento, porém mais confiável.

A estratégia de corte utilizada:

* IDA\* (Iterative Deepening A\*)
  + Versão iterativa em profundidade da procura A\*
  + Critério de corte:
    - F(n) = G(n) + H(n)
    - Em vez da profundidade
    - Inicializado com F(estado inicial)
  + Em cada iteração é feita uma procura em profundidade primeiro com seguinte corte:
* Quando um nó *n* é gerado, se F(n) > limite o nó cortado
  + Em cada nova iteração o valor limite é atualizado com menor valor de F(n) para os nós cortados na iteração anterior.

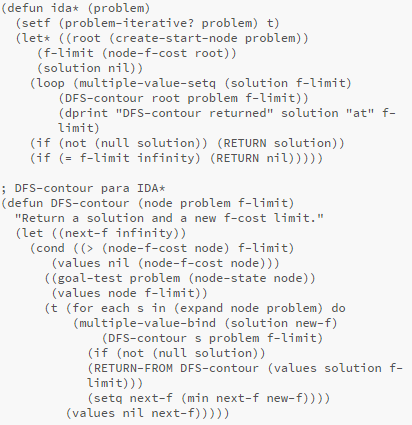


Figura 12 - G18.lisp - IDA\*

Explicação do código:

- O loop principal realiza uma série de cálculo do custo de f limitado (*f-cost-bounded*) com procura em profundidade primeiro até que a solução é encontrada, o custo de f limitado é aumentado até o menor valor de custo f que excede o limite anterior. A função *DFS-contour* retorna a solução de novo custo de f limitado.

# Resultados

Devido a limitações do código produzido apenas foi possível testar problemas com um número muito reduzido de tarefas:

'((L1 L2 34 60) (L5 L1 408 447) (L1 L1 448 551) (L1 L1 474 565))

foi ulitizada a função time para verificar tempo dos algoritmos (insignificante dado a dimensão do teste)

* Procura em profundidade primeiro:
  + Solução:(((L1 L2 34 60))((L5 L1 408 447) (L1 L1 474 565))((L1 L1 448 551)))
  + Evaluation took:
    - 0.000 seconds of real time
    - 0.000008 seconds of total run time (0.000008 user, 0.000000 system)
    - 100.00% CPU
    - 15,616 processor cycles
    - 0 bytes consed
* Procura em largura primeiro:
  + Solução: (((L1 L2 34 60))((L5 L1 408 447) (L1 L1 474 565))((L1 L1 448 551))
  + Evaluation took:
    - 0.000 seconds of real time
    - 0.000014 seconds of total run time (0.000014 user, 0.000000 system)
    - 100.00% CPU
    - 33,228 processor cycles
    - 0 bytes consed
* Profundidade iterativa:
  + Solução:(((L1 L2 34 60))((L5 L1 408 447) (L1 L1 474 565))((L1 L1 448 551))
  + Evaluation took:
    - 0.000 seconds of real time
    - 0.000009 seconds of total run time (0.000009 user, 0.000000 system)
    - 100.00% CPU
    - 19,392 processor cycles
    - 0 bytes consed
* A\*:
  + Não foi possível testar a procura a\*, devido a má codificação das heuristicas
* IDA\*:
  + Não foi possível testar a procura a\*, devido a má codificação das heuristicas

Não foi possível testar os algoritmos de sondagem iterativa, ilds, e 1-samp (abordagem alternativa), pois não conseguimos implementar os algoritmos.

# CONCLUSÃO

Através dos resultados podemos perceber que para problemas simples A\* é uma boa opção do que IDA\*, A\* é mais rapido porque não é necessário *iterative deepening loop* mas IDA\* poupa mais memória, porém tendo o custo de repetir grandes partes de procura várias vezes.

Não foi possivel obter resultados relevantes devido as limitações do código produzido.

# BIBLIOGRAFIA

<http://www.barbon.com.br/wp-content/uploads/2013/08/Aula3_IC.pdf>

[https://pt.wikipedia.org/wiki/Algoritmo\_A\*](https://pt.wikipedia.org/wiki/Algoritmo_A*)

<https://www.cos.ufrj.br/~ines/courses/cos740/leila/cos740/apres_ia.pdf>

<http://home.iscte-iul.pt/~luis/aulas/ia/Algoritmos%20de%20procura.pdf>

<https://pt.wikipedia.org/wiki/Backtracking>

<http://www.rafaeldiasribeiro.com.br/downloads/IC1_6.pdf>

<https://pt.slideshare.net/mcastrosouza/busca-em-largura-bfs>

<https://en.wikipedia.org/wiki/Beam_search>

<https://en.wikipedia.org/wiki/Best-first_search>

<https://en.wikipedia.org/wiki/Greedy_algorithm>

<https://www.aaai.org/Papers/AAAI/1996/AAAI96-043.pdf>