

**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 2](#_Toc531701179)

[2. Estrutura de dados 2](#_Toc531701180)

[3. MODELAÇÕES DO PROBLEMA 3](#_Toc531701181)

[4. HEURÍSTICAS DESENVOLVIDAS 5](#_Toc531701182)

[5. ESTRATÉGIAS DE CORTE 7](#_Toc531701183)

[6. Resultados 8](#_Toc531701184)

[7. CONCLUSÃO 9](#_Toc531701185)

# 

# 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.

Neste relatório contém a avaliação de implementações alternativas tanto de ponto de vista quantitativo como qualitativo.

# Estrutura de dados

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

No estado decidimos implementar informação como: *shifts* (lista de turnos) 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.

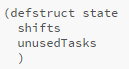


Figura 1 - G18.lisp - state

Na estrutura do turno decidimos implementar como componentes: lista de tarefas pertencentes ao turno (*tasks*), duração total do turno (*duration*), última estação em que serviço terminou (*lastPoint*) e um boolean que representa se foi realizado a pausa de refeição (*mealBreak*).

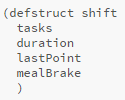


Figura 2 - G18.lisp - shift

# 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.

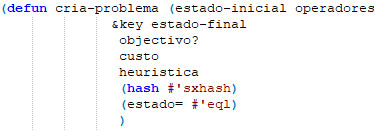


Figura 3 – procura.lisp – cria-problema

* *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:

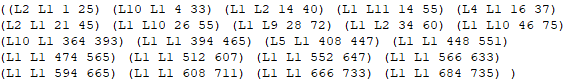


Figura 4 Exemplo estado-inicial

* *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*)

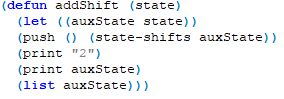


Figura 5 - G18.lisp - addShift

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

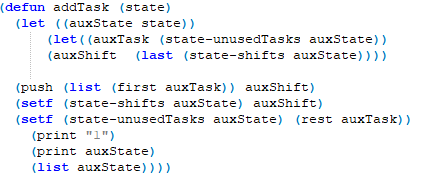


Figura 6 - G18.lisp - addTask

* *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.

# HEURÍSTICAS DESENVOLVIDAS

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

* Tempo total que sobrou dos turnos do estado

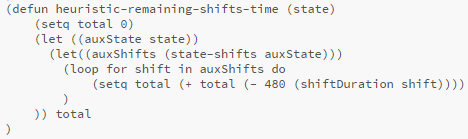


Figura 8 - G18.lisp - heuristic-remaining-shifts-time

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

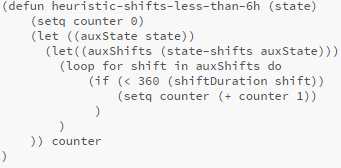


Figura 9 - G18.lisp - heuristic-shifts-less-than-6h

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

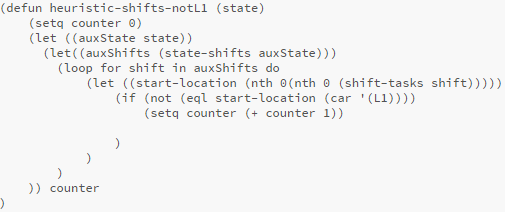


Figura 10 - G18.lisp - heuristic-shifts-notL1

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

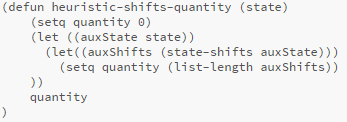


Figura 11 - G18.lisp - heuristic-shifts-quantity

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.

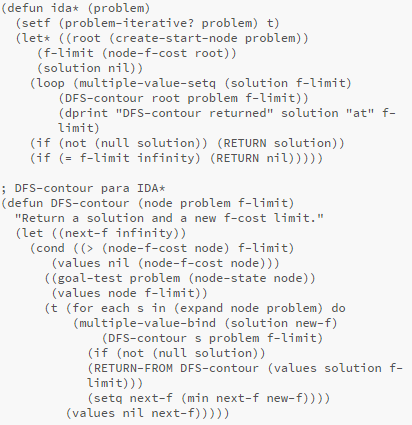


Figura 12 - G18.lisp - IDA\*

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