

**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: 72h

**Conteúdo**

[1. Introdução 3](#_Toc533879907)

[2. Estrutura de dados 3](#_Toc533879908)

[3. MODELAÇÕES DO PROBLEMA 4](#_Toc533879909)

[4. Algoritmos utilizados 6](#_Toc533879910)

[4.1 A\* 6](#_Toc533879911)

[4.2 IDA\* 6](#_Toc533879912)

[4.3 Profundidade Primeiro 6](#_Toc533879913)

[4.4 Largura primeiro 6](#_Toc533879914)

[4.5 Profundidade Iterativa 7](#_Toc533879915)

[4.6 Sondagem Iterativa 7](#_Toc533879916)

[4.7 ILDS (Improved Limited Discrepancy Search) 7](#_Toc533879917)

[4.8 Algoritmo Alternativo 7](#_Toc533879918)

[5. HEURÍSTICAS DESENVOLVIDAS 8](#_Toc533879919)

[6. ESTRATÉGIAS DE CORTE 10](#_Toc533879920)

[7. Resultados 12](#_Toc533879921)

[7.1 Problema 1 12](#_Toc533879922)

[7.2 Problema 2 14](#_Toc533879923)

[7.3 Problema 3 16](#_Toc533879924)

[7.4 Problema 4 18](#_Toc533879925)

[7.5 Problema 5 20](#_Toc533879926)

[8. CONCLUSÃO 22](#_Toc533879927)

[9. BIBLIOGRAFIA 23](#_Toc533879928)

# 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.

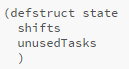


Figura 1 - G18.lisp - state

# 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.

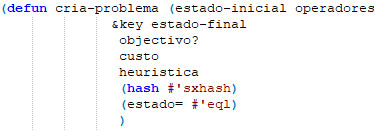


Figura 2 – procura.lisp – cria-problema

* *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:

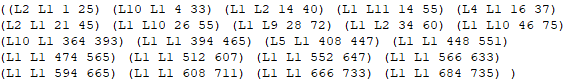


Figura 3 Exemplo estado-inicial

* *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*)

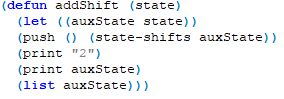


Figura 4 - G18.lisp - addShift

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

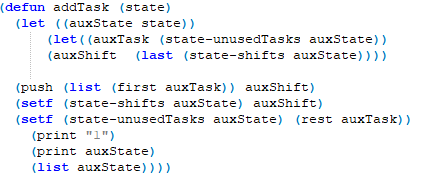


Figura 5 - G18.lisp - addTask

* *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 6 - 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

Foi utilizado uma versão alternativa do algoritmo de procura em profundidade primeiro. Apenas serão gerados estados em que apenas adiciona uma tarefa a um turno (se possível) ou adiciona um turno com essa tarefa.

Este algoritmo foi desenvoldido com o objetivo de ter resultados mais rápidos.

# HEURÍSTICAS DESENVOLVIDAS

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

* Tempo total que sobrou dos turnos do estado

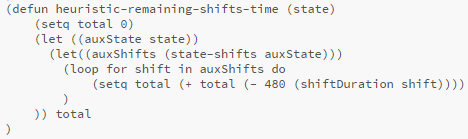


Figura 7 - G18.lisp - heuristic-remaining-shifts-time

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

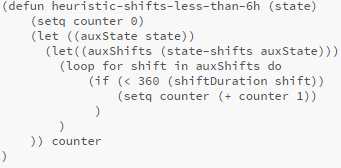


Figura 8 - G18.lisp - heuristic-shifts-less-than-6h

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

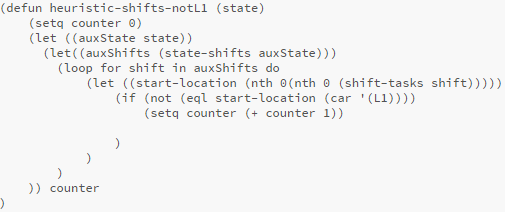


Figura 9 - G18.lisp - heuristic-shifts-notL1

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

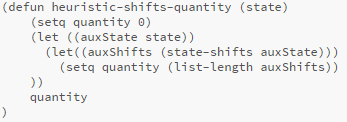


Figura 10 - G18.lisp - heuristic-shifts-quantity

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.

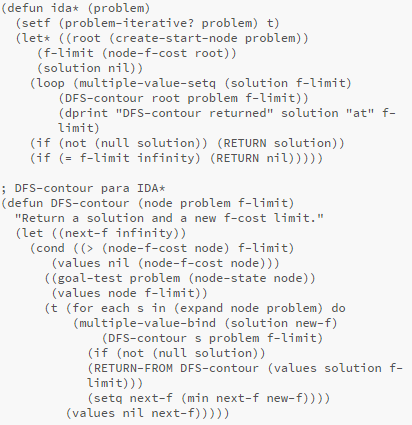


Figura 11 - G18.lisp - IDA\*

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

Todos os problemas seguintes foram testados com ILDS,Profundidade Primeiro e Largura Primeira com profundidade igual a 1000.

Os resultados com A\* e IDA\* seguintes foram iguais com todas as heurísticas criadas.

## 7.1 Problema 1

82 tarefas, há solução com 20 turnos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Profundidade Primeiro | Largura Primeiro | A\* | IDA\* | Profundidade Iterativa | ILDS | Sondagem Iterativa |
| Número de turnos | 25 | 25 | 25 | 25 | 25 | 25 | 25 |
| Número de nós gerados | 82 | 163 | 163 | 6806 | 6643 | 83 | 82 |
| Número de nós expandidos | 83 | 164 | 164 | 6889 | 6725 | 84 | 83 |
| Tempo | 0 | 0 | 0 | 14 | 12 | 0 | 0 |

* Procura em profundidade primeiro:
  + - Evaluation took:
    - 0.059 seconds of real time
    - 0.050989 seconds of total run time (0.050927 user, 0.000062 system)
    - 86.44% CPU
    - 142,415,196 processor cycles
    - 3,079,024 bytes bytes consed
* Procura em largura primeiro:
  + Evaluation took:
  + 0.057 seconds of real time
  + 0.052695 seconds of total run time (0.044945 user, 0.007750 system)
  + 92.98% CPU
  + 136,256,076 processor cycles
  + 3,111,520 bytes consed
* A\*:
  + - * Evaluation took:
      * 0.090 seconds of real time
      * 0.067353 seconds of total run time (0.060236 user, 0.007117 system)
      * 74.44% CPU
      * 214,753,112 processor cycles
      * 3,079,136 bytes consed
* IDA\*:
  + Evaluation took:
  + 0.104 seconds of real time
  + 0.066652 seconds of total run time (0.062370 user, 0.004282 system)
  + 64.42% CPU
  + 247,910,972 processor cycles
  + 5,307,248 bytes consed
* Profundidade Iterativa:
  + Evaluation took:
  + 0.061 seconds of real time
  + 0.058361 seconds of total run time (0.050126 user, 0.008235 system)
  + 95.08% CPU
  + 146,016,196 processor cycles
  + 5,339,824 bytes consed
* ILDS:
  + Evaluation took:
  + 0.001 seconds of real time
  + 0.001105 seconds of total run time (0.001105 user, 0.000000 system)
  + 100.00% CPU
  + 2,643,848 processor cycles
  + 98,272 bytes consed
* Sondagem Iterativa:
  + Evaluation took:
  + 0.001 seconds of real time
  + 0.000740 seconds of total run time (0.000740 user, 0.000000 system)
  + 100.00% CPU
  + 1,767,268 processor cycles
  + 65,536 bytes consed

## 7.2 Problema 2

157 tarefas, há solução com 40 turnos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Profundidade Primeiro | Largura Primeiro | A\* | IDA\* | Profundidade Iterativa | ILDS | Sondagem Iterativa |
| Número de turnos | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| Número de nós gerados | 162 | 323 | 323 | 26406 | 26083 | 26083 | 162 |
| Número de nós expandidos | 163 | 324 | 324 | 26569 | 26245 | 26245 | 163 |
| Tempo | 1 | 1 | 2 | 89 | 74 | 74 | 0 |

* Procura em profundidade primeiro:
  + - Evaluation took:
    - 0.398 seconds of real time
    - 0.210838 seconds of total run time (0.174875 user, 0.035963 system)
    - 53.02% CPU
    - 952,726,256 processor cycles
    - 3 page faults
    - 11,649,008 bytes consed

* Procura em largura primeiro:
  + Evaluation took:
  + 0.284 seconds of real time
  + 0.217463 seconds of total run time (0.193459 user, 0.024004 system)
  + 76.41% CPU
  + 678,581,968 processor cycles
  + 11,791,584 bytes consed
* A\*:
  + - * Evaluation took:
      * 0.259 seconds of real time
      * 0.211448 seconds of total run time (0.195454 user, 0.015994 system)
      * 81.47% CPU
      * 621,645,704 processor cycles
      * 11,791,248 bytes consed
* IDA\*:
  + Evaluation took:
  + 0.322 seconds of real time
  + 0.279404 seconds of total run time (0.247389 user, 0.032015 system)
  + 86.65% CPU
  + 772,385,112 processor cycles
  + 1 page fault
  + 24,687,824 bytes consed
* Profundidade Iterativa:
  + Evaluation took:
  + 0.322 seconds of real time
  + 0.274049 seconds of total run time (0.246073 user, 0.027976 system)
  + 85.09% CPU
  + 771,300,792 processor cycles
  + 24,898,528 bytes consed
* ILDS:
  + Evaluation took:
  + 0.255 seconds of real time
  + 0.200459 seconds of total run time (0.176511 user, 0.023948 system)
  + [ Run times consist of 0.005 seconds GC time, and 0.196 seconds non-GC time. ]
  + 78.43% CPU
  + 610,344,292 processor cycles
  + 11,726,016 bytes consed
* Sondagem Iterativa:
  + Evaluation took:
  + 0.002 seconds of real time
  + 0.002391 seconds of total run time (0.002384 user, 0.000007 system)
  + 100.00% CPU
  + 5,718,452 processor cycles
  + 196,576 bytes consed

## 7.3 Problema 3

231 tarefas, há solução com 60 turnos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Profundidade Primeiro | Largura Primeiro | A\* | IDA\* | Profundidade Iterativa | ILDS | Sondagem Iterativa |
| Número de turnos | 84 | 84 | 84 | 84 | 84 | 84 | 84 |
| Número de nós gerados | 244 | 487 | 487 | 59780 | 59293 | 487 | 244 |
| Número de nós expandidos | 245 | 488 | 488 | 60025 | 59537 | 488 | 245 |
| Tempo | 1 | 2 | 4 | 280 | 240 | 2 | 0 |

* Procura em profundidade primeiro:
  + - Evaluation took:
    - 0.564 seconds of real time
    - 0.430478 seconds of total run time (0.371210 user, 0.059268 system)
    - 76.24% CPU
    - 1,351,786,296 processor cycles
    - 26,235,872 bytes consed
* Procura em largura primeiro:
  + Evaluation took:
  + 0.577 seconds of real time
  + 0.454771 seconds of total run time (0.377943 user, 0.076828 system)
  + 78.86% CPU
  + 1,381,909,376 processor cycles
  + 26,498,144 bytes consed
* A\*:
  + - * Evaluation took:
      * 0.588 seconds of real time
      * 0.466569 seconds of total run time (0.387535 user, 0.079034 system)
      * 79.42% CPU
      * 1,407,105,308 processor cycles
      * 26,529,968 bytes consed
* IDA\*:
  + Evaluation took:
  + 0.874 seconds of real time
  + 0.757362 seconds of total run time (0.668611 user, 0.088751 system)
  + 86.61% CPU
  + 2,094,267,280 processor cycles
  + 70,744,928 bytes consed
* Profundidade Iterativa:
  + Evaluation took:
  + 0.819 seconds of real time
  + 0.703635 seconds of total run time (0.643328 user, 0.060307 system)
  + 85.96% CPU
  + 1,960,330,688 processor cycles
  + 71,192,304 bytes consed
* ILDS:
  + Evaluation took:
  + 0.005 seconds of real time
  + 0.005128 seconds of total run time (0.005011 user, 0.000117 system)
  + 100.00% CPU
  + 12,277,164 processor cycles
  + 524,160 bytes consed
* Sondagem Iterativa:
  + Evaluation took:
  + 0.004 seconds of real time
  + 0.003695 seconds of total run time (0.000988 user, 0.002707 system)
  + 100.00% CPU
  + 8,839,528 processor cycles
  + 393,184 bytes conse

## 7.4 Problema 4

455 tarefas, há solução com 120 turnos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Profundidade Primeiro | Largura Primeiro | A\* | IDA\* | Profundidade Iterativa | ILDS | Sondagem Iterativa |
| Número de turnos | 159 | 159 | 159 | 159 | 159 | 159 | 159 |
| Número de nós gerados | 472 | 943 | 943 | 223256 | 222313 | 223256 | 473 |
| Número de nós expandidos | 473 | 944 | 944 | 223729 | 222785 | 223729 | 473 |
| Tempo | 4 | 7 | 11 | 1829 | 1573 | 1829 | 0 |

* Procura em profundidade primeiro:
  + - Evaluation took:
    - 2.218 seconds of real time
    - 1.740492 seconds of total run time (1.492723 user, 0.247769 system)
    - 78.45% CPU
    - 5,312,737,732 processor cycles
    - 97,414,288 bytes consed

* Procura em largura primeiro:
  + Evaluation took:
  + 2.179 seconds of real time
  + 1.682426 seconds of total run time (1.432264 user, 0.250162 system)
  + 77.19% CPU
  + 5,216,058,024 processor cycles
  + 98,352,352 bytes consed
* A\*:
  + - * Evaluation took:
      * 2.142 seconds of real time
      * 1.681578 seconds of total run time (1.480120 user, 0.201458 system)
      * 78.52% CPU
      * 5,129,416,972 processor cycles
      * 98,369,760 bytes consed
* IDA\*:
  + Evaluation took:
  + 3.978 seconds of real time
  + 3.504504 seconds of total run time (3.224935 user, 0.279569 system)
  + 88.11% CPU
  + 9,525,703,612 processor cycles
  + 397,023,760 bytes consed
* Profundidade Iterativa:
  + Evaluation took:
  + 3.696 seconds of real time
  + 3.212356 seconds of total run time (2.923120 user, 0.289236 system)
  + 86.90% CPU
  + 8,849,807,928 processor cycles
  + 398,779,424 bytes consed
* ILDS:
  + Evaluation took:
  + 2.183 seconds of real time
  + 1.689589 seconds of total run time (1.456958 user, 0.232631 system)
  + 77.42% CPU
  + 5,228,214,975 processor cycles
  + 97,633,280 bytes consed
* Sondagem Iterativa:
  + Evaluation took:
  + 0.009 seconds of real time
  + 0.009043 seconds of total run time (0.009029 user, 0.000014 system)
  + 100.00% CPU
  + 21,662,248 processor cycles
  + 1,114,016 bytes consed

## 7.5 Problema 5

693 tarefas, há solução com 180 turnos.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Profundidade Primeiro | Largura Primeiro | A\* | IDA\* | Profundidade Iterativa | ILDS | Sondagem Iterativa |
| Número de turnos | 221 | 221 | 221 | 221 | 221 | 221 | 221 |
| Número de nós gerados | 693 | 1385 | 1385 | 480942 | 479557 | 1385 | 693 |
| Número de nós expandidos | 694 | 1386 | 1386 | 481636 | 480250 | 1386 | 694 |
| Tempo | 10 | 18 | 25 | 5355 | 4684 | 47 | 0 |

* Procura em profundidade primeiro:
  + - Evaluation took:
    - 4.712 seconds of real time
    - 3.659262 seconds of total run time (3.071673 user, 0.587589 system)conds GC time, and 3.642 seconds non-GC time. ]
    - 77.65% CPU
    - 11,283,629,740 processor cycles
    - 209,429,296 bytes consed

* Procura em largura primeiro:
  + Evaluation took:
  + 4.691 seconds of real time
  + 3.686176 seconds of total run time (3.093261 user, 0.592915 system)
  + 78.58% CPU
  + 11,231,810,004 processor cycles
  + 211,293,232 bytes consed
* A\*:
  + - * Evaluation took:
      * 4.682 seconds of real time
      * 3.607760 seconds of total run time (3.101157 user, 0.506603 system)
      * 77.06% CPU
      * 11,210,802,212 processor cycles
      * 211,307,904 bytes consed
* IDA\*:
  + Evaluation took:
  + 10.132 seconds of real time
  + 9.001788 seconds of total run time (8.425502 user, 0.576286 system)
  + 88.85% CPU
  + 24,261,589,792 processor cycles
  + 1,046,968,832 bytes consed
* Profundidade Iterativa:
  + Evaluation took:
  + 9.495 seconds of real time
  + 8.419918 seconds of total run time (7.837197 user, 0.582721 system)
  + 88.68% CPU
  + 22,736,324,764 processor cycles
  + 1,051,077,824 bytes consed
* ILDS:
  + Evaluation took:
  + 4.469 seconds of real time
  + 3.418203 seconds of total run time (2.951780 user, 0.466423 system)
  + 76.48% CPU
  + 10,701,647,840 processor cycles
  + 227,529,008 bytes consed
* Sondagem Iterativa:
  + Evaluation took:
  + 0.015 seconds of real time
  + 0.015466 seconds of total run time (0.015329 user, 0.000137 system)
  + 100.00% CPU
  + 37,067,632 processor cycles
  + 2,097,104 bytes consed

# CONCLUSÃO

Apesar de ter sido testado todas as heurísticas desenvolvidas para os algoritmos **A\*** e **IDA\*,** foram obtidos os mesmos resultados com todas as heurísticas, com diferença muita ligeira no tempo e memória usada. Em todos os problemas (1,2,3,4 e 5) foi observado um valor constante no número de turnos criado, desconfiamos que este valor pode ter sido afetado pela função sucessores (*operator*).

Em questão de tempo de processamento, foi notável o desnivelamento dos algoritmos **IDA\*,** **Profundidade Iterativa** e **ILDS**. Dos quais tomaram valores muito altos no tempo, nós expandidos e nós gerados em comparação com os outros algoritmos.

Com os resultados obtidos foi possível concluir que para resolução de problemas com pequena quantidade de tarefas (problema 1 e 2, por exemplo) é melhor utilizar algoritmos como: procura em **Profundidade Primeiro** e procura em **Largura Primeiro**. A afirmação anterior foi escrita devido aos resultados obtidos em questão de tempo e memória utilizada, não será necessário utilizar um algoritmo mais complexo para a procura de uma solução de um problema pequeno.

Porém para problemas com quantidade de tarefas elevado (por exemplo: problema 3, 4 e 5) é melhor utilizar algoritmos como: **A\*** e **Sondagem Iterativa**.

**Sondagem Iterativa** é o que obtem melhores resultados para problemas grandes, porque utiliza menos memória, é mais rápido em questão de tempo e mais eficaz que os restantes algoritmos.

O algoritmo **A\*** é mais rapido que o **IDA\*** porque não é necessário *iterative deepening loop* mas **IDA\*** poupa mais memória, porém tendo o custo de repetir grandes partes de procura várias vezes, demora mais tempo.

Também é possível verificar que para a nossa solução o algorito de **ILDS** apenas demonstrou resultados aceitáveis para o problema 2 e 4, desconfiamos que foram obtidos estes resultados devido a profundidade máxima definida de valor 1000, do qual foi ultrapassada nos restantes problemas, desta forma não obtendo resultados.

# BIBLIOGRAFIA

1. http://www.barbon.com.br/wp-content/uploads/2013/08/Aula3\_IC.pdf
2. https://pt.wikipedia.org/wiki/Algoritmo\_A\*
3. https://www.cos.ufrj.br/~ines/courses/cos740/leila/cos740/apres\_ia.pdf
4. http://home.iscte-iul.pt/~luis/aulas/ia/Algoritmos%20de%20procura.pdf
5. https://pt.wikipedia.org/wiki/Backtracking
6. http://www.rafaeldiasribeiro.com.br/downloads/IC1\_6.pdf
7. https://pt.slideshare.net/mcastrosouza/busca-em-largura-bfs
8. https://en.wikipedia.org/wiki/Beam\_search
9. https://en.wikipedia.org/wiki/Best-first\_search
10. https://en.wikipedia.org/wiki/Greedy\_algorithm
11. https://www.aaai.org/Papers/AAAI/1996/AAAI96-043.pdfhttp://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>