ALGAV

2019-2020

RELATÓRIO
SPRINT 1

Carlos Moreira – 1161882
Marco Pinheiro – 1170483
Pedro Barbosa – 1150486
Pedro Mendes – 1161871

Índice

Introdução

Este relatório surge no âmbito da disciplina de Algoritmia Avançada (ALGAV), do terceiro ano da Licenciatura em Engenharia Informática do Instituto Superior de Engenharia do Porto. Este sprint tem como objetivo a concretização de conhecimentos da disciplina, nomeadamente ao nível de heurísticas, aplicação do Algoritmo A*, de forma a conseguir obter o melhor escalonamento possível, tendo em conta os tempos de ocupação ou tempos de atraso das operações.

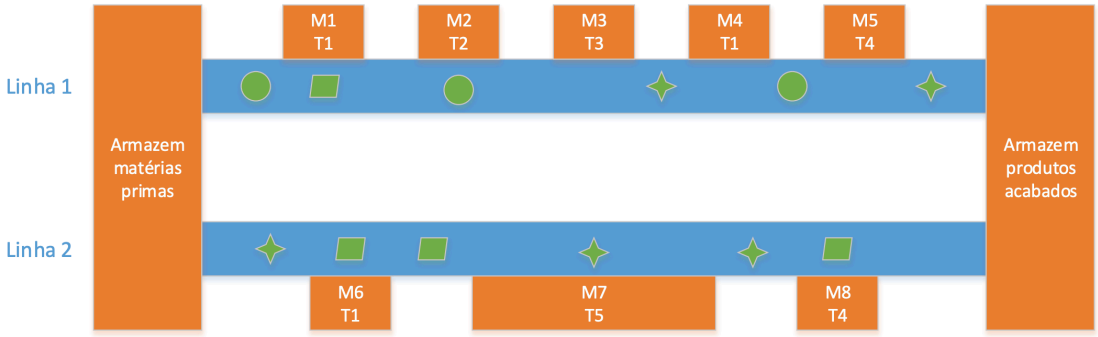
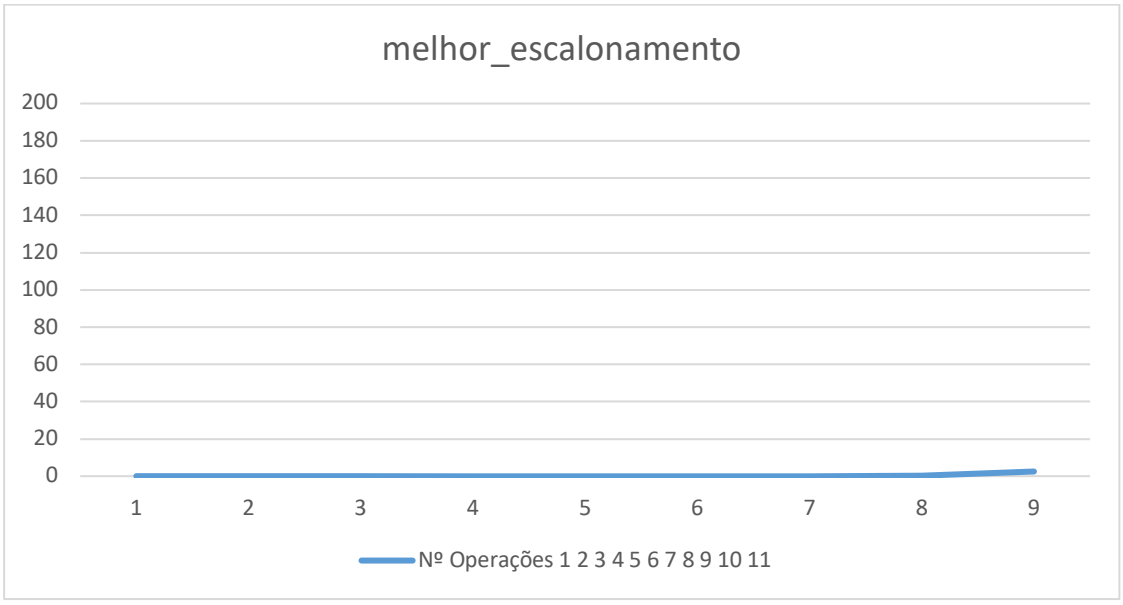
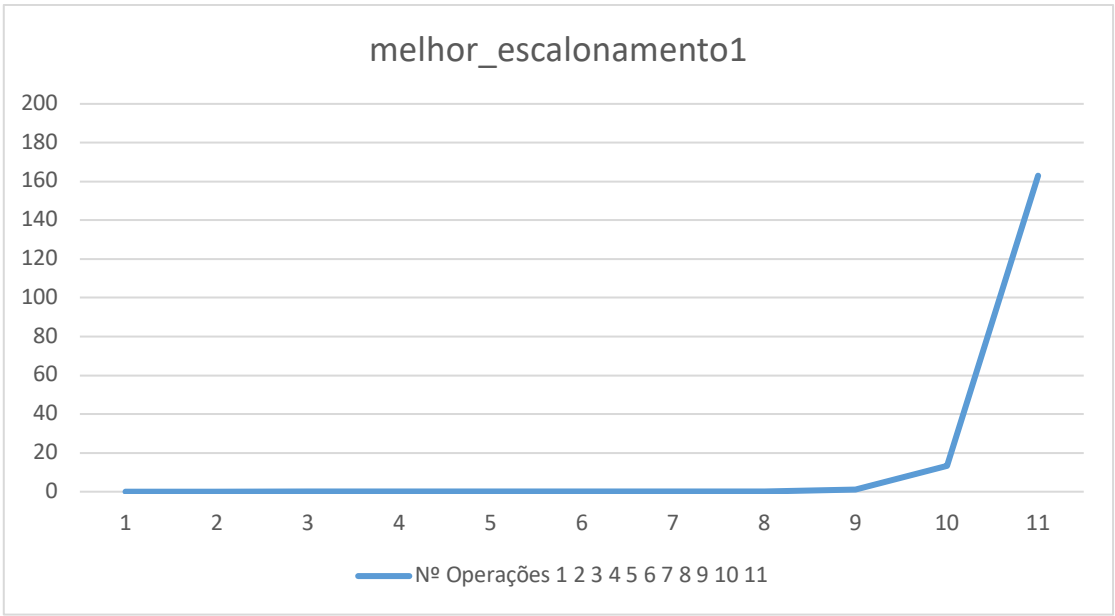


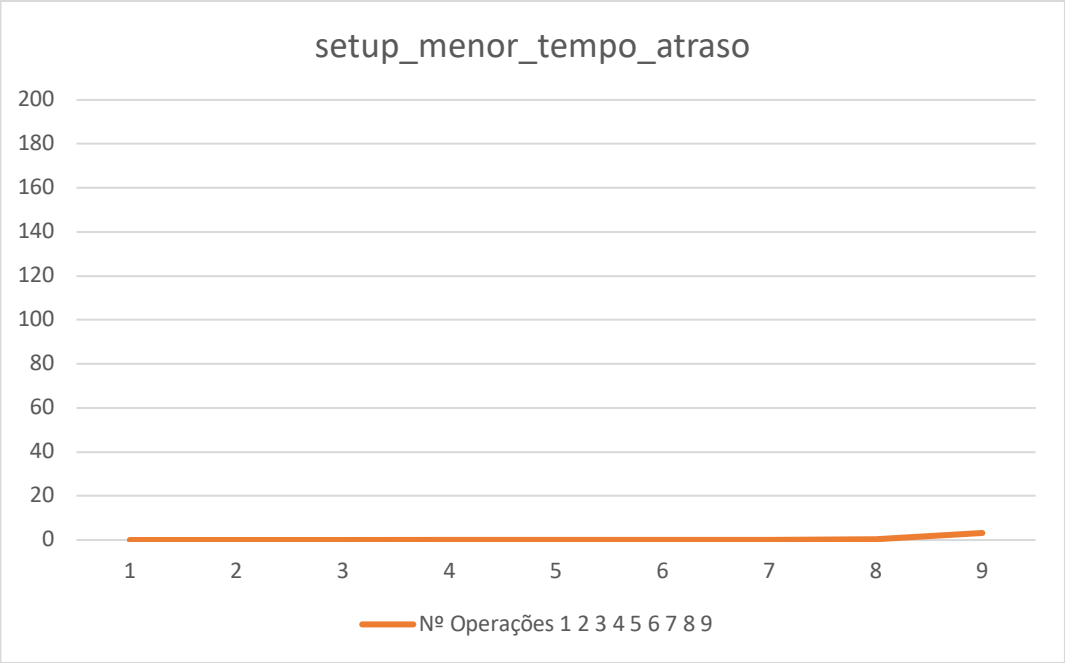
Figura 1 Exemplo de fábrica

Análise de complexidade

Predicado Nº Operações	melhor_escalonamento (usa findall)	melhor_escalonamento1 (sem findall)	setup_menor_tempo_atraso (usa findall)
1	-	0,000017	0
2	0,000007	0,00005	0
3	0,000009	0,00006	0
4	0,00023	0,0002	0,0002
5	0,0008	0,0006	0,0012
6	0,0002	0,003	0,0072
7	0,0008	0,02	0,038
8	0,2	0,13	0,31
9	2,5	1,23	3,2
10	FALHA (stack limit)	13,3	FALHA (stack limit)
11	FALHA (stack limit)	162,97	FALHA (stack limit)






setup_menor_tempo_atraso
200
180
160
140
120
100
80
60
40
20
0
1
2
3
4
5
6
7
8
9
Nº Operações 1 2 3 4 5 6 7 8 9

Heurísticas

Derivada da palavra grega "heuristiké", a heurística é a definição de regras e/ou estratégias, com o intuito de chegar à melhor solução possível e reduzir a quantidade de combinações que determinado problema nos providencie.
Não é garantido, porém que nos dê a melhor solução de todas, nem sequer que nos dê uma solução.

1. Heurística para minimização dos tempos de ocupação da máquina

O estudo feito para o problema de minimização dos tempos de ocupação da máquina, levou-nos à necessidade de ordenar as operações por tipo de operação, de forma a minimizar os tempos que a máquina demora a mudar de ferramenta (tempos de Setup). Tendo em conta essa necessidade foi usada a heurística de "Setup" para possível resolução do problema.

Figura 2 Exemplo de ordenação de operações por tipo de operação

a. Código implementado

```
heuristica_setup(M,List,Tempo):-
    operacoes_atrib_maq(M,L),
    setup(L,List),
    soma_tempos(semfer,M,List,Tempo),!.

setup(L,LR):-
    sort_by_type(L, LR).

sort_by_type(List,Sorted):-
    tsort(List,[],Sorted),!.

tsort([],Acc,Acc).
tsort([H|T],Acc,Sorted):-
    tinsert(H,Acc,NAcc),
    tsort(T,NAcc,Sorted).

tinsert(X,[Y|T],[Y|NT]):-
    \+mesmo_tipo(X,Y),
    tinsert(X,T,NT).
tinsert(X,[Y|T],[X,Y|T]):-
    mesmo_tipo(X,Y).
tinsert(X,[],[X]).

mesmo_tipo(X,Y):-
    classif_operacoes(X,R),
    classif_operacoes(Y,R).
```

Chamada ao predicado de entrada "**heuristica_setup**", onde se passa por parâmetro uma máquina e 2 variáveis não instanciadas "List" e "Tempo" que irão ser providenciadas pelo predicado "**setup**" e pelo predicado "**soma_tempos**".

O predicado "**setup**" chama o predicado sort_by_type, tendo L como a lista de operações da máquina e LR como o retorno esperado.

Chama o predicado "**tsort**", que irá enviar "List" com a lista de operações, uma lista vazia e "Sorted" será a lista final ordenada.

O predicado "**tsort**" irá chamar o predicado "**tinsert**" que verifica se a operação em "H" tem o mesmo tipo que a operação em "Y" que estará guardada na lista passada no 2º parâmetro do predicado "**tinsert**". Caso seja de um tipo diferente, continuará recursivamente á procura de um tipo igual até que a lista de operações passada por parâmetro estiver vazia.

Verifica se ambas as operações passadas por parâmetro são do mesmo tipo.

```
?- heuristica_setup(ma,Lista,Tempo)
Lista = [op3, op1, op4, op2, op5],
Tempo = 239.
```

Figura 3 Operações ordenadas por tipo de operação e tempo de ocupação da máquina "ma"

2. Heurística para o somatório dos tempos de atraso das operações

Com o estudo feito para o problema de minimização do somatório dos tempos de atraso das operações, assumimos que se conseguíssemos garantir o valor mínimo local de tempos de atraso para todas os pares de operações consecutivas numa lista ordenada por tempo de conclusão teríamos um escalonamento mais próximo do cenário ideal.

a. Código implementado

```
h_atraso_setup(L,R,T):-
    get_time(Ti),
    findall(p(Tmp,O),(op_prod_client(O,_,_,_,_,_,Tmp,_,_
    ),member(O,L)),LO),
    sort(LO,LS),
    h_atraso(LS,R,T),!,
    get_time(Tf),Tcomp is Tf-Ti,
    write('GERADO EM '),write(Tcomp),
    write(' SEGUNDOS'),nl.

h_atraso([p(_,H)|[p(_,T)|Tail]],Res,TMP):-
    op_prod_client(H,_,_,_,_,_,TConc,TSetup,TExec),!,
    op_prod_client(T,_,_,_,_,_,TConc2,TSetup2,TExec2),!,
    ((\+mesmo_tipo(T,H), RealTSetup is TSetup,RealTSetup2 is TSetup2);
    (mesmo_tipo(T,H), RealTSetup is 0,RealTSetup2 is 0)),
    atraso(TConc,TSetup,TExec,ThisAtraso,0),
    Acc1 is TSetup + TExec,
    atraso(TConc2,RealTSetup2,TExec2,ThisAtraso2,Acc1),
    atraso(TConc2,TSetup2,TExec2,ThisAtraso3,0),
    Acc2 is TSetup2 + TExec2,
    atraso(TConc,RealTSetup,TExec,ThisAtraso4,Acc2),
    AccAtraso1 is ThisAtraso + ThisAtraso2,
    AccAtraso2 is ThisAtraso3+ThisAtraso4,
    ((AccAtraso1=<AccAtraso2, NTA is Acc1,
    h_atraso([p(_,T)|Tail],H,NTA,
    ThisAtraso,RT,TMP
    ), append([H],RT,Res));
    (AccAtraso1>AccAtraso2, NTA is Acc2,
    h_atraso([p(_,H)|Tail],T,NTA,
    ThisAtraso3,RT,TMP), append([T],RT,Res))).

h_atraso([p(_,H)|[]],_,_,TMP,[H],TMP).
h_atraso([p(_,H)|[p(_,T)|Tail]],Prev,TA,AA,Res,TMP):-
    op_prod_client(H,_,_,_,_,_,TConc,TSetup,TExec),!,
    op_prod_client(T,_,_,_,_,_,TConc2,TSetup2,TExec2),!,
    ((\+mesmo_tipo(T,H), RealTSetup is TSetup,RealTSetup2 is TSetup2);
    (mesmo_tipo(T,H), RealTSetup is 0,RealTSetup2 is 0)),
    ((\+mesmo_tipo(Prev,H), StartTSetup is TSetup);
    (mesmo_tipo(Prev,H), StartTSetup is 0)),
    ((\+mesmo_tipo(T,Prev), StartTSetup2 is TSetup2);
    (mesmo_tipo(T,Prev), StartTSetup2 is 0)),
    atraso(TConc,StartTSetup,TExec,ThisAtraso,TA),
    Acc1 is StartTSetup + TExec+TA,
    atraso(TConc2,RealTSetup2,TExec2,ThisAtraso2,Acc1),
    atraso(TConc2,StartTSetup2,TExec2,ThisAtraso3,TA),
    Acc2 is StartTSetup2 + TExec2+TA,
    atraso(TConc,RealTSetup,TExec,ThisAtraso4,Acc2),
    AccAtraso1 is ThisAtraso + ThisAtraso2,
    AccAtraso2 is ThisAtraso3+ThisAtraso4,
    ((AccAtraso1=<AccAtraso2, NTA is Acc1,NAA is AA+ThisAtraso,
h_atraso([p(_,T)|Tail],H,NTA,NAA,RT,TMP), append([H],RT,Res));
    (AccAtraso1>AccAtraso2, NTA is Acc2,NAA is
AA+ThisAtraso3,h_atraso([p(_,H)|Tail],T,NTA,NAA,RT,TMP),
append([T],RT,Res))).
```

Chamada ao predicado h_atraso_setup onde passamos uma lista de operações. Este predicado vai fazer um findall de todas as operações e respetivos tempos de conclusão e ordenar a lista resultante por tempo de conclusão.

Este predicado é semelhante ao seguinte, porém foi necessário distinguir a primeira operação das restantes, pois esta não tinha nenhuma operação anterior para comparar.

Chamada ao predicado h_atraso, onde se passam um pares consecutivos de operações e restante lista ordenada.
O predicado calcula o atraso local para o par de operações e compara com o atraso do mesmo par em posições invertidas. O par com o menor atraso é adicionado à lista de operações final.

```
GERADO EM 0.00016188621520996094 SEGUNDOS
R = [op1, op2, op4, op3, op5],
T = 87.
```

Figura 4 Menor tempo de atraso e respetiva lista de operações

Método A*

Existem aplicações comuns a ambas as implementações dos métodos A*, nomeadamente o facto de não existir uma origem e um destino fixo, mas sim uma lista de operações a fazer e uma lista de operações feitas, que inicialmente está vazia, em cada nó da árvore.

a. Minimização dos tempos de ocupação

Esta minimização usa o predicado "estimativa" fornecido nos slides das aulas teórico-práticas com uma modificação que permite ter em conta o tempo de setup entre duas operações do mesmo tipo, bem como a utilização de ferramenta para o cálculo da soma dos setups. Esta alteração foi feita por forma a diminuir a complexidade e ser mais eficiente.

```
estimativa(X,LR,Estimativa):-
  findall(p(F,TS),(op_prod_client(Op,_,F,_,_,_,_,TS,_),member(Op,LR)),List),
  op_prod_client(X,_,FX,_,_,_,_,_,_),
  sort(List,L),
  soma_setups(FX,L,Estimativa).

soma_setups(_,[],0).
soma_setups(FX,[p(FX,_)|L],Ttotal):-
  soma_setups(FX,L,Ttotal),!.
soma_setups(FX,[p(_,Tsetup)|L],Ttotal):-
  soma_setups(FX,L,T1),
  Ttotal is Tsetup+T1.
```

b. Minimização do somatório dos tempos de atraso

Esta minimização usa a heurística criada anteriormente o que permite calcular o menor tempo de atraso.

```
estimativaAtrasos(L,Estimativa):-
    h_atraso_setup(L,_,Estimativa),!.
```

Conclusão

Este trabalho foi importante na consolidação de conhecimentos respeitantes a programação lógica, usando o Prolog como linguagem de programação.

A criação de heurísticas permitiu-nos pensar numa forma mais eficiente de resolver os problemas apresentados que embora possa não ser a solução correta, aproxima-se ao máximo da solução e reduz muito à complexidade que a quantidade de permutações nos traz.

O contacto com o método de pesquisa A* também foi um ponto positivo, no entanto verificamos que para o nosso problema torna-se menos eficiente dado que está otimizado para uma pesquisa entre dois pontos fixos e não com listas em variação livre.