

Relatório ALGAV

**Grupo 46**

José Mota (1161263)

Pedro Real (1170689)

João Flores (1171409)

Patrick Timas (1171352)

1. Introdução incluindo identificação dos objetivos atingidos no sprint1.

Para esta iteração, foi-nos lançada a proposta de obter a melhor sequência de operações tendo como critérios o menor tempo de ocupação e o menor somatório de tempos de atraso. Assim sendo, para este trabalho tivemos que criar duas heurísticas, uma para os tempos de ocupação das operações e outro para o tempo de atraso das mesmas. Foram adaptados quatro predicados que geram todas as soluções de sequenciamentos, sendo dois predicados para o critério do tempo de ocupação, um usando findall e outro não usando, e sendo os outros dois predicados para o somatório de tempos de atraso, um com findall e outro sem findall, sendo que estes aplicavam e utilizam as heurísticas criadas previamente. Por fim, foi pedido para criarmos o predicao A\*, um para cada heurística.

1. Análise da complexidade na abordagem da Geração de Todas as Soluções e escolha da melhor para o problema de escalonamento/sequenciamento de n operações numa máquina segundo os critérios de minimização do tempo de ocupação da máquina e minimização do somatório dos tempos de atraso das operações. Incluir tabelas e/ou gráficos.

|  |  |  |
| --- | --- | --- |
| Tempo de Ocupação |  |  |
| **Numero Operacoes** | **Tempo de Ocupacao Com FindAll** | **Tempo de Ocupacao Sem FindAll** |
| 5 | 0.0021398067474365 | 0.000920057296752 |
| 6 | 0.009619951248168945 | 0.011480093002 |
| 7 | 0.059319972991943 | 0.04891991615 |
| 8 | 0.5003600120544434 | 0.39574003219 |
| 9 | 4.9656801223754 | 3.88128018379 |
| 10 | stack limit | 41.746780157 |
| 11 | stack limit | 254,46906 |
| 12 | stack limit | 2470,905238 |
| 13 | stack limit | 32230,28975 |

|  |  |  |
| --- | --- | --- |
| Tempo Atraso |  |  |
| **Numero Operacoes** | **Tempo de Atraso Com FindAll** | **Tempo de Atraso Sem FindAll** |
| 5 | 0.0027201175689697266 | 0.002439975738525 |
| 6 | 0.014120101928710938 | 0.0024399757385256 |
| 7 | 0.07252001762390137 | 0.0675599575042 |
| 8 | 0.6466999053955078 | 0.552779912948 |
| 9 | 6.4278199672698975 | 5.431979894 |
| 10 | stack limit | 59.53699994 |
| 11 | stack limit | 254,46906 |
| 12 | stack limit | 2988,525452 |
| 13 | stack limit | 38619,35306 |

Analisando as tabelas e os gráficos resultantes, podemos concluir que a utilização do predicado sem findAll nem sempre é a solução mais eficiente, ou seja, a que demora menos tempo. No entanto, quando o número de operações aumenta, o tempo também aumenta. Todavia existem problemas de maior escala a quando da utilização do método findAll, sendo que o predicado não suporta o número de operações, dando erro stack limit.

1. Heurísticas para escalonamento/sequenciamento de n operações numa máquina segundo os critérios de minimização do tempo de ocupação da máquina e minimização do somatório dos tempos de atraso das operações. Explicar código desenvolvido para implementar as heurísticas.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Neste predicado, começamos por ordenar as operações por ferramentas. Primeiramente vamos obter todas as operações da máquina passada por parâmetro. Queremos ordenar as operações por ferramentas ­, assim criamos um par Ferramenta-Operação e passamos para um maplist todos esses pairs. É feita uma ordenação desse maplist pela key(ferramenta) e por fim passamos os values(operações) para uma nova lista. Com a lista já ordenada, vamos calcular o tempo de ocupação de cada uma. Vamos buscar o tempo de Setup e o tempo de execução de cada operação dessa mesma lista já ordenada, vamos comparar: caso a operação seguinte tenha a mesma ferramenta somamos apenas o tempo de execução da operação, caso contrário somamos também o tempo de Setup. A o tempo de ocupação da primeira operação da lista será obrigatoriamente composta por tempo de Setup mais o tempo de execução. A variável T irá conter a soma desses tempos.

Uma imagem com captura de ecrã

Descrição gerada automaticamente

Neste predicado, começamos por ordenar as operações por Prazos. Primeiramente vamos obter todas as operações da máquina passada por parâmetro. Queremos ordenar as operações por prazo ­, assim criamos um par Prazo-Operação e passamos para um maplist todos esses pairs. É feita uma ordenação desse maplist pela key(prazo)x e por fim passamos os values(operações) para uma nova lista. Com a lista já ordenada, vamos calcular o tempo de atraso de cada uma. Vamos buscar o prazo, o tempo de Setup e o tempo de execução de cada operação dessa mesma lista já ordenada, vamos comparar: caso a operação seguinte tenha o mesmo prazo, somamos apenas o tempo de ocupação e tempo de execução da operação, se for menor, somamos também o tempo de Setup, caso contrario vamos ao tempo de ocupação e subtraímos a soma do prazo com o atraso. A variável T irá conter a soma desses tempos de atraso.

1. Aplicação do método A\* no escalonamento/sequenciamento de n operações numa máquina segundo os critérios de minimização do tempo de ocupação da máquina e minimização do somatório dos tempos de atraso das operações. Explicar alterações face ao exemplo do A\* fornecido nas aulas TP e como quantifica os custos acumulados e as estimativas
   1. A \* para o tempo de Ocupação

Uma imagem com captura de ecrã

Descrição gerada automaticamente

O predicado apresentado acima foi alterado conforme o problema apresentado, dado que o exemplo apresentado nas aulas TP era inserido para problemas com nós e o a solução que pretendemos alcançar é para sequencias.

No exemplo apresentado nota-se que temos um inicio e um fim, mas para o nosso problema a solução consiste em fazer todas as operações necessarias, logo começamos por alterar os argumentos de nós para lista, uma lista contendo todos as operações necessarias e outra que vai as opercações realizadas conforme a melhor custo tendo em conta estimativas e custos. O criterio de paragem, tendo em conta as listas, ira ser cumprido quando a lista com as operacões inicias estiver vazia, a lista que começou vazia vai contem o caminho determinado.

Dado a nova implementação guardamos as opercações num lista conforme o resultado e a estimativa, a cada escolha volta a realizar os calculos com o custo atual, custo correspondente até ao da operação onde se encontra, e a estimativa do tempo de setup.

Esta estimativa é realizada tendo em conta a operação foi adicionada a lista na iteração anterior, o algoritmo faz verificações e modificações para garantir que as listas continuam como são necesarias, tendo como procura no metodo “obterTemposPorOperacao” obtem a ferramenta e tempo de setup, verifica se a ferramenta é igual a proxima ferramenta na lista, se for o tempo de setup ira ser 0, senão vai ser o obtido acima.

* 1. A \* para o tempo de Atraso

Uma imagem com captura de ecrã

Descrição gerada automaticamente

A funcionamento deste algoritmo é igual ao A\* acima apresentado, o que difere é o uso da heuristica de EDD-“EarlyDueDate” onde fazemos a diferença entre o tempo de ocupação e tempo Cx, correspondente ao tempo de setup e de ocupação ou tempo de ocupação conforme a maquina e ferramenta utilizada entre operações seguidas, a soma dos tempo é realizada com auxilio do “verificarFerramenta”.

Neste algoritmo a parte da estimativa é calculada de modo um pouco diferente, a parte disso o resto da implementação é praticamente igual.

1. Conclusões relativas ao Sprint 1

Nesta iteração, concluímos que em termos de minimização do tempo de ocupação, a melhor solução é o uso da heurística, uma vez que se obtém o melhor resultado possível da maneira mais rápida e eficiente. Se pretendermos obter a solução através do cálculo de todas as soluções possíveis, então é mais adequado será o uso do predicado sem findall, dado ser mais eficiente em termos de tempo que com findall e que suporte um maior número de operações. Para além disso, para o somatório dos tempos de atraso, concluímos que através da heurística não é garantido a melhor solução possível e, que, para obter a melhor solução possível é necessário o uso de um predicado que calcule todas as soluções, sendo, neste caso, preferível o uso do predicado sem findall visto ser mais rápido e que suporte um maior número de operações em relação ao predicado que usa findall. Em suma, o trabalho foi realizado todo com sucesso, apesar de algumas dificuldades na implementação dos A\*,mas com algum esforço e múltiplas tentativas, conseguimos executar todas as tarefas propostas.