

Trabalho de Otimização Combinatória

Flow Shop Permutacional com GRASP

Rafael Serafini - 155140
Frederico Betting - 158747

1. Introdução

Este relatório contém a formulação matemática do problema *Flow Shop* Permutacional e a descrição detalhada do algoritmo GRASP (*Greedy Randomized Adaptive Search Procedures*), heurística utilizada para resolver o problema proposto. Ao final, são apresentados os resultados obtidos após a execução da formulação no GLPK e da heurística escolhida.

2. Descrição do problema

O problema *Flow Shop* permutacional consiste em minimizar o tempo de fluxo de processamento a partir de um conjunto n tarefas e m máquinas ordenadas, considerando que a ordem de processamento de todas as tarefas serão iguais para todas as máquinas (*Fig. 01*).



Fig. 01: Processamento de tarefas no *Flow Shop* permutacional

Ao respeitar essa ordem de processamento, por exemplo, o processamento da tarefa T_2 só será iniciada pela máquina $M1$ após o término do processamento de T_1 em $M1$ (*Fig. 02*), uma vez que uma máquina só pode atender uma única tarefa por vez.

O processamento de tarefas é não preemptivo, ou seja, a partir do momento que uma máquina inicia o processamento de uma tarefa, tal processamento deve ser executado até a sua finalização, sem interrupções.

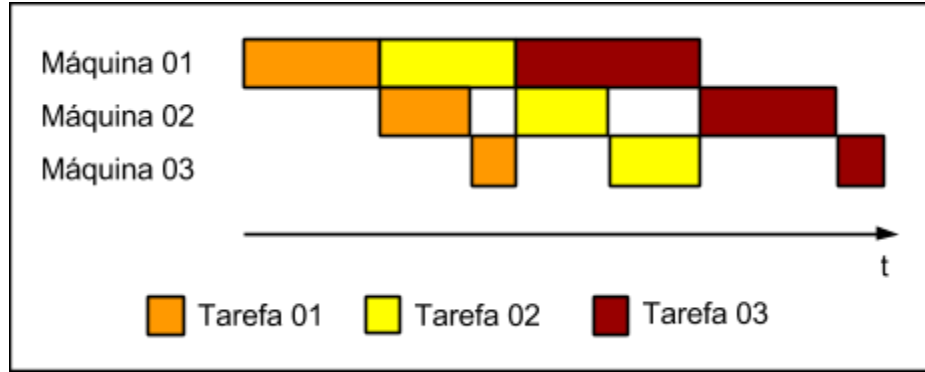


Fig. 02: Fluxo de processamento de tarefas

2.1. Formulação Matemática

Algumas formulações foram implementadas pelo grupo. Apesar de se ter conseguido resultados inteiros ótimos para pequenas instâncias (até duas máquinas), não foi possível obter qualquer resultado inteiro ótimo para as instâncias solicitadas na definição do trabalho.

A formulação matemática abaixo (Fig. 03), resolve o problema de Flow Shop. Nela, c_{max} representa o tempo total de processamento das tarefas na última máquina, t_{ij} é o tempo inicial de processamento do item i na máquina j e d_{ij} o tempo que uma máquina k leva para processar um determinado job.

$$\begin{aligned}
 & \text{Minimiza} \\
 & \cdot \quad c_{max} \\
 & \cdot \quad s.a. \\
 & \cdot \quad t_{i,j+1} \geq t_{i,j} + d_{i,j} \quad , i = \{1, \dots, n\}, j = \{1, \dots, m\} \quad (1) \\
 & \cdot \quad t_{i,j} \geq t_{k,j} + d_{k,j} \vee t_{k,j} \geq t_{i,j} + d_{i,j} \quad , i = \{1, \dots, n\}, j = \{1, \dots, m\} \quad (2) \\
 & \cdot \quad t_{i,m} + d_{i,m} \leq c_{max} \quad , i = \{1, \dots, n\} \quad (3)
 \end{aligned}$$

Fig. 03: Formulação matemática para *Flow Shop* permutacional

Na restrição (1), fica garantido a sequência de processamento dos itens nas máquinas, ou seja, o tempo inicial de processamento de um determinado item na máquina $j+1$ será maior que o tempo inicial deste mesmo item na máquina j . Em (2), é garantido que apenas um elemento será processado por vez em cada máquina. Por fim, na restrição (3), é avaliado o makespan, cálculo de processamento de todos os itens na última máquina.

3. Heurística GRASP

O Greedy Randomized Adaptive Search Procedure consiste em iterações feitas por sucessivas construções de uma solução randômica gulosa e subseqüentes melhorias através de uma busca local. As soluções randômicas gulosas são geradas adicionando elementos à solução do problema de uma lista de elementos ordenados por uma função gulosa, de acordo com a qualidade da solução. Para obter variedade à lista de candidatos de soluções gulosas, elementos são colocados em uma Lista Restrita de Candidatos (RCL) e selecionadas randomicamente, com probabilidades de acordo com a ordenação de qualidade.

Estruturas e algoritmo

Para cada instância é lida e armazenada uma matriz de acordo com o padrão usado no gerador de instâncias (tailard.c, ex. ao lado).

'5 5' significam 5 máquinas e 5 tarefas respectivamente. Cada tarefa é composta por 5 operações. Cada linha nos arquivo de entrada é uma tarefa e sua lista de operações.

5	5			
1	1	1	1	
2	4	6	8	10
3	2	4	5	7
4	3	1	4	5
3	4	3	8	4
3	6	5	7	3

Fase de Construção

1. Calcula o makespan (tempo total de execução) de cada operação.
 - a. Armazena-se em um vetor, no exemplo a primeira operação (ignorando sempre a linhas de 1's) teria 30 de makespan.
2. Acha o máximo e mínimo makespan
3. Calcula o Range (max - min)
4. Dado um α
 - a. Greedy Value, controla quão grande vai ser o Range para se colocar elementos e escolhe-os aleatoriamente com probabilidades
5. Acha Width (Range x α)
6. Insere na RCL (Restricted Candidate List) cada elemento dentro do Width
 - a. O RCL consiste nos elementos em que o makespan se encontra dentro do intervalo: {mínimo makespan, (mínimo makespan + width)}
7. Ordena as operações dentro da RCL
8. Calcula a probabilidade de cada elemento dando mais para os primeiros (menores makespans)
 - a. O cálculo é:
 - i. $\text{fitness} = 1 / (\text{ordem da operação na RCL})$
 - ii. $\text{probabilidade da op.} = (\text{fitness da op.}) / (\text{soma de todos os fitness})$

9. Coloca cada operação em uma faixa de valores entre 0 e 1, com o tamanho variando de acordo com a sua probabilidade
10. Escolhe aleatoriamente um número entre 0 e 1 e verifica em qual faixa ele corresponde
11. A operação correspondente à faixa é escolhida para ser adicionada à solução
12. Atualiza solução

Fase de Busca Local

1. Acha soluções com distância 1
 - a. Uma matriz é populada com vizinhos da solução encontrada na fase de construção onde cada tarefa é permutada uma vez em relação à solução original
2. Seleciona a melhor
3. Atualiza solução

A seleção da melhor solução é baseada no calculo do makespan total da execução de cada solução, isso é feito verificando quando cada máquina para de executar para começar a operação de outra tarefa, por exemplo caso uma tarefa termine de executar na máquina 1 ela deve esperar a máquina 2 terminar de executar, até que a última operação da última tarefa seja executada pela última máquina.

Os parâmetros principais usados são o valor do alpha (Greedy Value) e o número de rodadas. Com alphas menores cada iteração produz menos candidatos para a RCL, priorizando os de menor makespan, e fazendo com que as soluções encontradas em cada rodada sejam mais próximas (não há muita variação nos resultados), e com alphas maiores cada o RCL tem mais candidatos, permitindo mais candidatos e soluções mais variadas. O número de rodadas permite uma população maior para calculo das medias.

Na pratica a variação desses 2 parâmetros geraram resultados diferentes mas muitos próximos de qualquer outra variação.

Um exemplo

Supondo a matriz ao abaixo calcula-se o valor acumulado de cada operação em cada máquina, gerando a matriz ao lado. Ela representa os índices em que se encontra disponível cada máquina depois de processar as operações.

		Machines				
		1	2	3	4	5
Jobs	1	2	4	6	8	10
	2	3	2	4	5	7
	3	4	3	1	4	5
	4	3	4	3	8	4
	5	3	6	5	7	3

Job 1						
		2	6	12	20	30
Job 2						
		3	5	9	14	21
Job 3						
		4	7	8	12	17
Job 4						
		3	7	10	18	22
Job 5						
		3	9	14	21	24

Em seguida calcula-se o máximo, mínimo, range e width para então se obter o intervalo dos elementos que entrarão na Restricted Candidate List:

Max = 30
 Min = 17
 Range = 30 - 17 = 13
 alpha = 0.5
 Width = Range × alpha = 6.5
 RCL = {Min, Min + width} = {17, 23.5}

Neste caso as tarefas 2, 3 e 4 são escolhidas e ordenadas em um rank, calculados sua fitness e sua probabilidade de ser escolhida como solução:

Rank:

R[2] = 2

R[3] = 1

R[4] = 3

Fitness (1/Rank) :

Tarefa 2 = 0.5

Tarefa 3 = 1

Tarefa 4 = 0.3

Probabilidades:

Tarefa 2 = 0.5/1.8 = 0.28

Tarefa 3 = 1/1.8 = 0.55

Tarefa 4 = 0.3/1.8 = 0.17

Job(21)				
5	9	15	23	33

Job (23)				
7	10	11	15	20

Job (24)				
6	10	13	21	25

Job (25)				
6	12	17	24	27

Então ordena-se as tarefas em uma faixa:

0 - 0.17 = tarefa 2

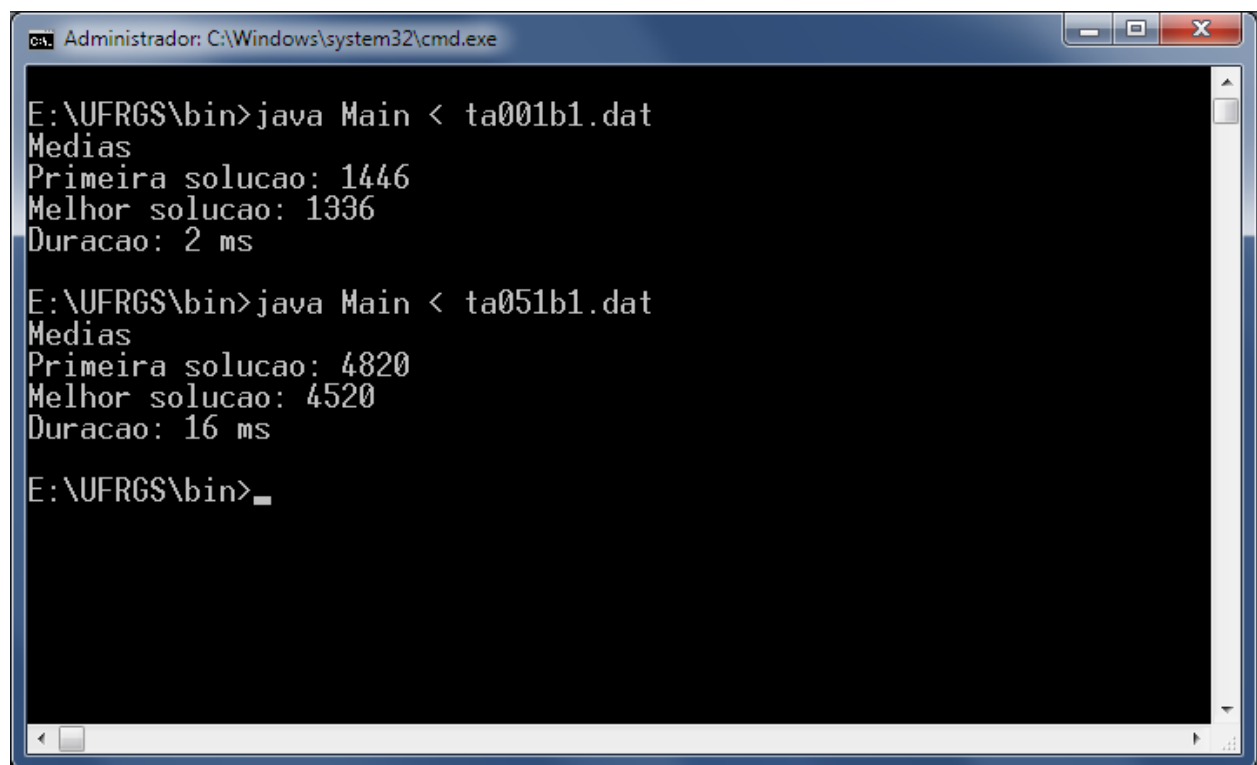
0.18 - 0.45 = tarefa 3

0.45 - 1 = tarefa 4

Gera-se um número aleatório, supondo ser 0.15 ele se encontra na faixa da tarefa 2 que é escolhida. Atualiza-se então a matriz de valores acumulados das operações nas máquinas (acima) e calcula-se assim a cada iteração as probabilidades para a escolha da próxima tarefa para a solução.

Comandos

Para a execução do GRASP, é preciso um PC com interpretador Java instalado (JRE) e abrir uma janela de comando na pasta 'bin', onde se encontra o executável. O comando é como segue: `./.../ java Main < arquivo_de_entrada.dat`



```
Administrador: C:\Windows\system32\cmd.exe

E:\UFRGS\bin>java Main < ta001b1.dat
Medias
Primeira solucao: 1446
Melhor solucao: 1336
Duracao: 2 ms

E:\UFRGS\bin>java Main < ta051b1.dat
Medias
Primeira solucao: 4820
Melhor solucao: 4520
Duracao: 16 ms

E:\UFRGS\bin>_
```

4. Resultados

Instância	Melhor Solução Conhecida	Primeira Solução (GRASP) ¹	Melhor Solução (GRASP) ¹	Tempo (GRASP) (ms) ¹	Solução GLPK	Tempo (GLPK)	Desvio %
ta001	1278	1452	1336	2	----	1h	4,34
ta011	1582	2091	1879	3	----	1h	15,81
ta021	2297	2847	2525	2	----	1h	9,03
ta031	2724	3162	2988	9	----	1h	8,84
ta041	2991	3750	3553	10	----	1h	15,82
ta051	3771	4826	4517	14	----	1h	16,52
ta061	5493	6265	5953	45	----	1h	7,73
ta071	5770	6786	6488	66	----	1h	11,07
ta081	6106	7738	7485	89	----	1h	18,42
ta091	10862	12115	11756	464	----	1h	7,60
ta101	11152	13553	13131	719	----	1h	15,07
ta111	26040	30365	29824	12510	----	1h	12,69

¹ Resultado referente a média de 10 rodadas executadas

5. Conclusão

FlowShop scheduling é um problema interessante para ser solucionado com PL e Metaheurísticas. Obtivemos bons resultados com a heurística GRASP, ao contrário do GLPK em que apenas obtivemos resultados inteiros ótimos com instâncias pequenas.

Houve dificuldade, também, no desenvolvimento dos algoritmos para uma linguagem de programação, assim como, da formulação para a linguagem do solver GLPK. Apesar disso, na implementação do algoritmo GRASP por exemplo, os resultados foram muito satisfatórios, chegando bem próximos aos melhores valores encontrados [5]. A ideia do problema e de sua resolução é intuitiva em um nível de abstração maior mas transferindo essas ideias para o código e as estruturas usadas, como matrizes e vetores, percebeu-se uma grande complexidade.

6. Bibliografia

- [1] AHMADIZAR, F.; BARZINPOUR, F.; ARKAT, J. **Solving Permutation Flow Shop Sequencing using Ant Colony Optimization**. 2007 IEEE International Conference on Industrial Engineering and Engineering Management, pp 753 - 757
- [2] MODRÁK V.; PANDIAN, R. S. **Flow shop scheduling algorithm to minimize completion time for n-jobs m-machines problem**. Technical Gazette 17, 3(2010), pp 273-278
- [3] PRABHAHARAN, G.; KHAN, B.S.H.; RAKESH, L. **Implementation of grasp in flow shop scheduling**. The International Journal of Advanced Manufacturing Technology, Volume 30, Issue 11-12 , pp 1126-1131
- [4] RESENDE, M.G.C.; RIBEIRO, C.C. **GRASP: Greedy Randomized Adaptive Search Procedures**. Search Methodologies, Capítulo 11, Springer, 2013, 2a edição, pp 285-310.
- [5] Melhores resultados Flow-Shop achados em:
<http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html>
- [6] http://en.wikipedia.org/wiki/Greedy_randomized_adaptive_search_procedure