Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

Mac5758/0461 - Introdução ao Escalonamento e Aplicações

Heurística Shifting Bottleneck

Alunos:
Leandro de Moraes
Otávio Moura do Nascimento

Índice

1 Introdução

Escalonamentos do tipo *job shop* pertencem a classe de problemas difíceis de otimização combinatória. Além de serem NP-completo(Garey e Johnson 1979), também estão entre os piores na prática, pois pode-se obter soluções exatas de problemas do tipo *Caixeiro Viajante* gerados aleatoriamente com cerca de 300-400 cidades, ou pode-se resolver problemas de cobertura com centenas de restrições e milhares de variáveis, mas, em geral, não se consegue obter um escalonamento ótimo para um problema do tipo *job shop* com cerca de dez tarefas e dez máquinas.

Muitos problemas práticos e importantes do dia-a-dia são resolvidos através de um escalonamento *job shop.* Como não se conhece nenhum algoritmo que resolva este tipo de problema em um tempo razoável, torna-se necessário o desenvolvimento de métodos de aproximação que produzam um escalonamento aceitável em um tempo razoável.

Existem algumas abordagens para resolver este tipo de problema que utilizam heurísticas como SPT(shortest processing time), MWKR(most work remaining), FCFS(fisrt come, first served), etc. Em geral, esses métodos produzem soluções que não são ruins em um curto espaço de tempo.

Este trabalho, que é baseado no artigo *The Shifting Bottleneck Procedure For Job Shop Scheduling*(Adams, Balas e Zawack 1988), tem o objetivo de apresentar a heurística *Shifiting Bottleneck,* que foi desenvolvida para ser aplicada em problemas do tipo *job shop*, com a premissa de se obter uma solução melhor em comparação com os métodos de aproximação existem até então, mesmo que acarretando em um consumo maior de tempo de processamento.

2 O Problema

Um probelma do tipo job shop pode ser caracterizado da seguinte forma:

- n tarefas devem ser processadas em m máquinas.
- O processamento de uma tarefa numa máquina é chamado de operação.
- Uma operação não pode ser interrompida.
- Uma tarefa consiste em um número fixo e finito de operações.
- Cada máquina pode executar somente uma operação por vez.
- As operações devem seguir um grafo de precedência pré-definido.
- O tempo de produção de cada operação é fixo e previamente conhecido.
- Objetivo: Minimizar C_{max} (tempo de término da última tarefa).

Podemos modelar esse problema da seguinte maneira: Seja $N = \{0, 1, \ldots, n\}$ o conjunto de operações (sendo 0 e n operações de controle, que não possuem influência no problema. São usadas somente para representar, respectivamente, o início e o fim das operações), M o conjunto de máquinas, A o conjunto de pares de operações restringidas pela relação de precedência de cada tarefa e E_k o conjunto dos pares das operações a serem executadas na máquina k, as quais não podem se sobrepor no tempo, dado que cada máquina pode executar somente uma tarefa de cada vez. Considere também p_i sendo a duração (tempo de produção, fixo) e S_i o tempo de início da operação i. Dessa forma, podemos caracterizar o problema como:

$$\min S_n$$

$$\begin{aligned} & S_j - S_i \geq p_i, && (i, j) \in A, \\ & S_i \geq 0, && i \in N, \\ & S_j - S_i \geq p_i \vee S_i - S_j \geq p_j, && (i, j) \in E_k, k \in M. \end{aligned} \qquad (P)$$

Desse modo, qualquer solução viável de (P) fornecerá uma sequência de operações a serem executadas em cada máquina, gerando assim um escalonamento.

É muito útil representar este problema em um grafo disjunto $G = (N, A, E)$, onde:

- N, o conjunto de vértices que é representado pelo conjunto de operações (incluindo as operações de controle, *0* e *n*. Cada vértice possui um rótulo (i, j), que significa a tarefa j processada na máquina i.
- A, conjunto de conjunções representadas pela relação de precedência de cada tarefa. Assim, temos arcos do tipo $i \rightarrow j$ para cada operação i. Se i é a última operação da tarefa, então j representa a operação de controle $n(\ i \rightarrow n)$. Adicionalmente, tem-se arcos $0 \rightarrow i$ para cada primeira operação i de uma tarefa. O significado de um arco $i \rightarrow j$ é que qualquer escalonamento deve respeitar a condição $S_j - S_i \geq p_i$.
- E, disjunções, representa os pares de arcos dirigidos entre os pares de operações que serão executadas na mesma máquina. Assim, se em uma determinada máquina devem ser executadas as operações i e j, este conjunto conterá os arcos $i \rightarrow j$, e $i \leftarrow j$. Mas, em um escalonamento, no máximo um arco desse par será representado. Estas disjunções representam as restrições do tipo $S_j - S_i \geq p_i \vee S_i - S_j \geq p_j$.

Para ilustrar, considere o seguinte problema, onde se tem três tarefas e três máquinas. A tabela 1 mostra o tempo de produção de cada operação. A tabela 2, mostra a relação de precedência de cada operação.

		Máquinas		
		1	2	3
Tarefas	1	1	2	4
	2	3		3
	3	2	4	1

Tabela 1 – Tempo de produção das tarefas em cada máquina

		Máquinas		
		1	2	3
Tarefas	1	3ª	2ª	1ª
	2	1ª		2ª

	3	2ª	1ª	3ª

Tabela 2 – Ordem de precedência de cada operação

Assim, obtemos o grafo ilustrado na figura 1. Os números nos arcos representam o tempo de produção da operação, assim, se o rótulo do arco $i \rightarrow j$ for k, significa que o tempo de produção da operação i vale k.

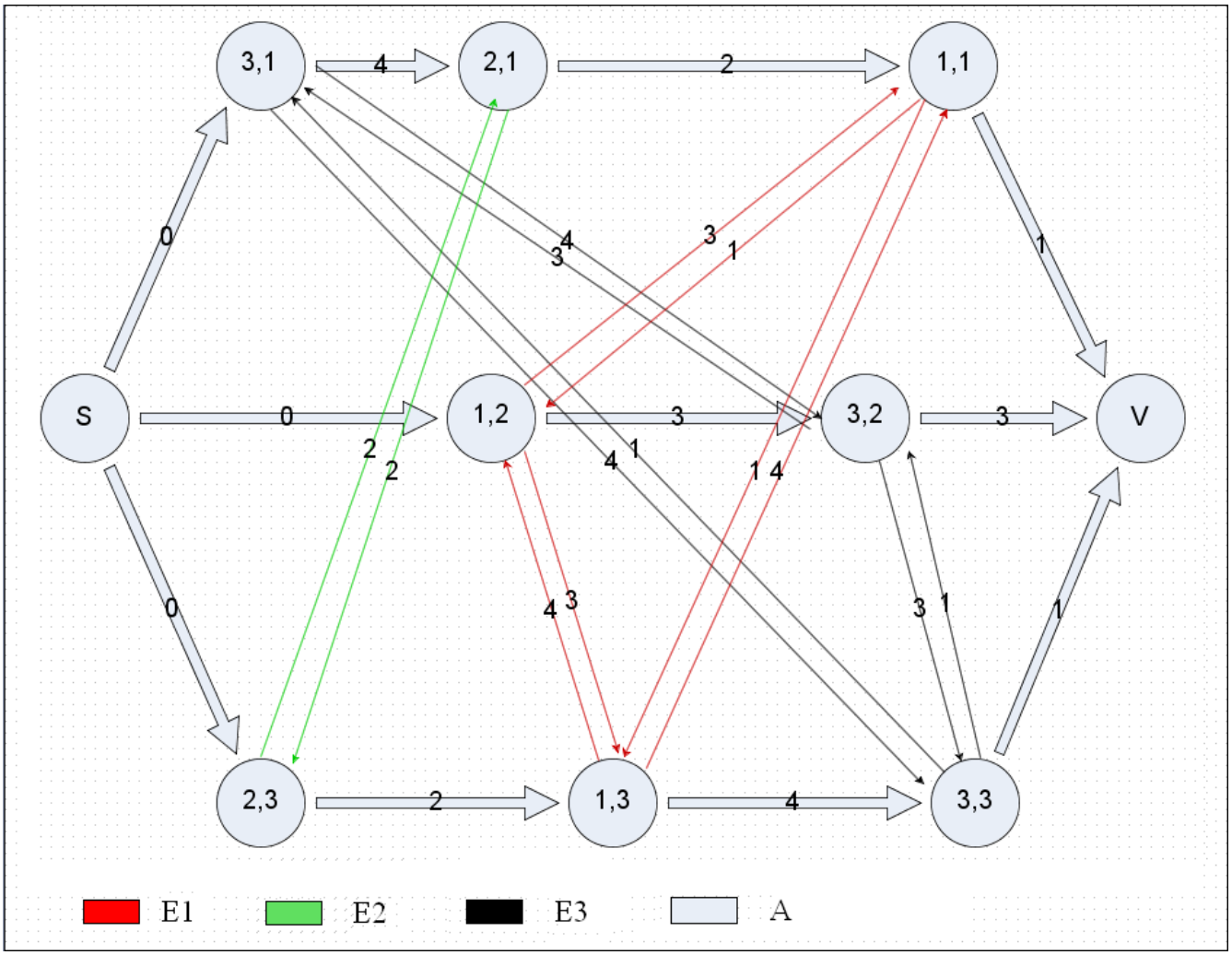


Figura 1 – Grafo representando o conjunto A e os conjuntos E_k's

Iremos denotar $D = (N, A)$ o digrafo obtido de G removendo todos os arcos disjuntos. Seja também $E = \bigcup(E_k : k \in M)$. Uma seleção S_k de E_k contem somente um membro de cada par de arcos disjuntos. Uma seleção é acíclica se ela não contém nenhum ciclo dirigido. Cada seleção acíclica S_k corresponde a uma sequência única de operações pertencentes a máquina k. Dessa

forma, sequenciar a máquina k significa escolher uma seleção acíclica em E_k. Uma seleção completa S consiste da união das seleções S_k, uma em cada E_k, $k \in M$. Pegando-se uma seleção completa S, ou seja, substituindo os arcos do conjunto E pelos arcos do conjunto S, obtemos o digrafo $D_s = (N, A \bigcup S)$. Uma seleção completa S é acíclica se o digrafo D_s é acíclico. Toda seleção completa acíclica S define uma família de escalonamento, e cada escalonamento pertence exatamente a uma única família. Assim, o *makespan* de um escalonamento que é ótimo para S é igual ao tamanho do maior caminho em D_s. Assim, o problema pode ser visto com o de encontrar uma seleção completa e acíclica $S \in E$, tal que minimize o tamanho do maior caminho no digrafo D_s.

3 Abordagem

Problemas do tipo *job shop* são NP-completos e a maior parte das heurísticas descritas na literatura para serem aplicadas neste tipo de problema são baseadas em *priority dispatching rules*, que são regras para escolher a próxima operação a ser escalonada dentre um conjunto de operações passíveis de serem escalonadas. Na maior parte das vezes, critérios como *SPT* (*shortest processing time*), *MWKR (most work remaining), FCFS (first come, first served)* são utilizados. Estes são procedimentos de um passo, do tipo guloso, onde a escolha de cada operação se dá levando em consideração somente o cenário no momento da tomada da decisão, fazendo-se escolhas que parecem localmente boas e uma vez tomadas, são finais. Esses tipos de procedimentos, em geral, são rápidos e encontram escalonamentos que não são tão ruins. Então, dependendo da necessidade, seu uso pode ser empregado. Entretanto, quando se procura escalonamentos mais eficientes, se faz necessário a aplicação de algum método de aproximação que mesmo podendo ter um custo computacional maior, retorne soluções melhores. A heurística *Shifiting Bottleneck* foi concebida para suprir esta necessidade.

Ao invés de escalonar uma operação por vez, sequencia-se uma máquina de cada vez, consecutivamente. Para fazer isto, para cada máquina ainda não sequenciada, deve-se resolver o problema de escalonamento em uma máquina, que é uma relaxação do problema original. A solução deste problema é utilizado tanto para ranquear a máquina para a identificação da máquina gargalo, quanto para sequenciá-la. Toda vez que uma nova máquina é sequenciada, deve-se reotimizar a sequência de cada máquina sequenciada anteriormente, resolvendo, novamente, para cada uma o problema do escalonamento em uma máquina. Este é um dos grandes diferenciais dessa abordagem, pois decisões tomadas anteriormente são melhoradas de acordo com as novas modificações causadas pelo sequenciamento de novas máquinas, fazendo com que decisões tomadas anteriormente não sejam finais, mas que se adaptem segundo novas condições.

O método utilizado para solucionar o problema de escalonamento em uma máquina não é

inovador, mas o diferencial esta em utilizá-lo para identificar em qual ordem as máquinas devem ser sequenciadas, pois é através da solução deste problema que as máquinas são classificadas para a identificação da máquina gargalo, a qual será sequenciada. Isto é baseado na clássica ideia de dar prioridade às máquinas gargalos.

Assim, o procedimento para identificação da máquina gargalo é descrito como: Seja $M_0 \subset M$ o conjunto das máquinas já sequenciadas, escolhendo-se uma seleção S_p, $p \in M_0$, e para qualquer $k \in M \setminus M_0$, seja $(P(k, M_0))$ o problema obtido de (P) substituindo cada conjunto de arcos disjuntos E_p, $p \in M_0$, pela correspondente seleção S_p, e apagando cada conjunto de arcos disjuntos E_q, $q \in M \setminus M_0$, $q \neq k$. Este problema é equivalente a minimizar o maior atraso no problema de escalonamento de uma máquina(para máquina k). A máquina m é denominada gargalo entre as máquinas de $M \setminus M_0$, se $v(m, M_0) = max \{ v(k, M_0) : k \in M \setminus M_0 \}$, onde $v(k, M_0)$ é o valor da solução ótima do problema $(P(k, M_0))$. Ou seja, para cada máquina ainda não sequenciada, resolve-se o problema do escalonamento em uma máquina. A máquina que obtiver o maior valor como solução deste problema, será a máquina gargalo.

Uma breve descrição do procedimento *Shifiting Bottleneck* é dado como: Seja M_0 o conjunto de máquinas já sequenciadas ($M_0 = \emptyset$ no início).

- *Passo 1.* Identificar a máquina gargalo m entre as máquinas ainda não sequenciadas (conjunto $M \setminus M_0$) e sequenciá-la otimamente. Faça $M_0 \leftarrow M_0 \cup \{m\}$ e vá para o *passo 2*.
- *Passo 2.* Reotimize a sequência de cada máquina $k \in M_0$ de cada vez, mantendo a sequência das outras máquinas fixas, isto é, faça $\in M'_0 := M_0 - \{k\}$ e resolva o problema $P(k, M'_0)$. Então, se $M_0 = M$, pare. Caso contrário, vá para o *passo 1*.

Para identificar a máquina gargalo a ser sequenciada no *passo 1*, para cada $k \in M \setminus M_0$, deve-se resolver o problema:

$$\min S_n$$

$$S_j - S_i \geq p_i, \qquad (i, j) \in \bigcup (S_p : p \in M_0) \cup A,$$

$$S_i \geq 0, \qquad i \in N,$$

$$S_j - S_i \geq p_i \vee S_i - S_j \geq p_j, \qquad (i, j) \in E_k. \qquad (P(k, M_0))$$

Para reotimizar a sequência de cada máquina k no *passo 2,* deve-se resolver o problema da forma $(P(k,M'_0))$ para algum subconjunto $M'_0 \subset M_0$.

O problema $(P(k,M_0))$ é equivalente a encontrar um escalonamento para a máquina *k* que minimize o maior atraso , ou seja, o problema pode ser definido como: $1 \mid pj, rj, dj \mid L_{max}$.

É necessário notar que o artigo (Adams, Balas e Zawack, 1988) sugere uma solução diferente da apresentada aqui para este problema. Decidimos utilizar uma solução mais simples que encontramos em outros artigos, para facilitar o entendimento da simulação (Apêndice A).

Para resolver este problema, precisamos definir, para cada operação da máquina, seu tempo de disponibilidade (r_j) e sua data devida (d_j). Estes valores serão calculados utilizando o grafo *G*.

No caso da identificação da máquina gargalo, o grafo *G* deve ser alterado de forma a remover todas as arestas disjuntas das máquinas ainda não sequenciadas, mantendo as seleções das máquinas já sequenciadas.

No caso do resequenciamento de uma máquina, retiramos a sua seleção do grafo, bem como as arestas disjuntas das máquinas ainda não sequenciadas.

Definimos *G'* como o grafo *G* após a atualização descrita acima e definimos:
p_{ij}= tempo de produção da tarefa *j* na máquina *i* (dado pelo problema inicial)
r_{ij} = comprimento do maior caminho entre *S* e o nó (*i,j*), sem contar p_{ij}, no grafo *G*'
d_{ij} = comprimento do maior caminho entre o nó (*i,j*) e *V,* sem contar p_{ij}, no grafo *G*'

É possível solucionar este problema resolvendo-se um problema equivalente, chamado de "*head-body-tail*". Este problema tem este nome pois, para cada operação *j* da máquina *i,* definimos:
head= r_{ij}
body= p_{ij}

tail= d_{ij}

Além disso, definimos as variáveis:

tempo de início da tarefa: $S_{ij} \geq r_{ij}$

tempo de término da tarefa: $C_{ij} = S_{ij} + p_{ij}$

O objetivo deste problema é minimizar *max* ($j = 1$ a n) $C_{ij} + d_{ij}$

Podemos definir $q_{ij} = - d_{ij}$ (temos, neste caso, datas devidas negativas!) e, portanto:

min max ($j = 1$ a n) $C_{ij} + d_{ij}$ = *min max* ($j = 1$ a n) $C_{ij} - q_{ij}$ = *min max* ($j = 1$ a n) $L_j = L_{max}$

Logo, solucionar o problema head-body-tail definido acima é equivalente a solucionar o problema $1 \mid p_j, r_j, d_j \mid L_{max}$

Chamaremos de *f(n)* o valor da solução encontrada quando resolvemos o problema *head-body-tail* para a máquina *n*.

Quando estivermos interessados em encontrar a máquina gargalo de uma iteração, ela será a máquina que tiver o maior valor de *f* entre as máquinas ainda não escalonadas.

Quando estivermos interessados em reotimizar uma máquina, simplesmente resolvemos o problema *head-body-tail* para ela e a sequenciamos com o escalonamento que resultou na solução do problema.

Uma outra abordagem para a resolução do problema *(P(k,M₀))* descrito no anteriormente, que é equivalente a encontrar um escalonamento para a máquina *k* que minimize o maior atraso, pode ser descrita como: Seja $r_i = L(0,i)$, $f_i = L(0, n) - L(i, n) + d_i$ e $q_i := L(0, n) - f_i$, com *L(i, j)* sendo o tamanho do maior caminho de *i* a *j* em D_t, e $T := \bigcup(S_p : p \in M_0)$. Assim, podemos representar o problema do escalonamento em uma máquina como:

min S_n

$S_n - S_i \geq p_i + q_i$,

$$p_i \geq r_i, \qquad i \in N^*,$$

$$S_j - S_i \geq p_j \vee S_i - S_j \geq p_j, \qquad (i, j) \in E_k, \qquad (P^*(k,M_0))$$

onde N^* é o conjunto de operações a serem a serem processadas na máquina k.

Dado que o problema $(P^*(k,M_0))$ é NP-completo, utiliza-se o algoritmo de Carlier(1982), que é do tipo *branch and bound*, para resolvê-lo. Como o objetivo deste trabalho não é discutir este algoritmo, será mostrado a seguir, em linhas gerais, seu funcionamento.

Seja inicialmente $t = min \{ r_j : j \in N^* \}$, $Q = N^*$, então executa-se repetidamente o seguinte passo

Passo iterativo. Entre as operações não escalonadas e que estão prontas para serem (isto é, aquelas $j \in Q$ *tal que* $r_j \leq t$), escolha uma , j, com o maior q_i (caso ocorra empate, escolha a com maior p_i), e escalone-a, fazendo as seguintes alterações, $t_j := t$, $Q := Q \backslash \{j\}$. Então, se $Q = \varnothing$, pare, caso contrário faça $t := max \{ t_j + d_j, min \{ r_i : i \in Q \} \}$ e repita o passo novamente.

Gerando o escalonamento utilizando esta heurística, iremos obter um caminho crítico no digrafo referente a este escalonamento. Se houver mais de um caminho crítico, deve-se pegar o que contiver maior número de arcos. Assim, seja $j(1), \ldots, j(p)$ os vértices do caminho crítico a menos dos vértices de controle, S e V. Seja k o maior inteiro em $\{ 1, \ldots, p \}$ tal que $q_{j(k)} < q_{j(p)}$ (se não existir tal k, o escalonamento é ótimo), e seja $J = \{ j_{(k+1)}, \ldots, j_{(p)} \}$. Assim, podemos definir como

$$h(J) := min \{ r_i : i \in J \} + \sum (p_i : i \in J) + min \{ q_i : i \in J \}$$

o *lower bound* da minimização do *makespan.* Pode-se mostrar que em um escalonamento ótimo, a operação $j_{(k)}$ é escalonada antes ou depois de todas as operações pertencentes a J.

Enquanto podemos utilizar o *lower bound* para descartar ramos da árvore cujo ramo tenha atingido o *upper bound,* a separação causada pela operação $j_{(k)}$ servirá de base para a regra de *branching.* Assim, se o nó atual não puder ser descartado pela comparação dos limitantes, ele pode ser substituído por dois outros nós sucessores, um aonde $j_{(k)}$ terá uma nova *tail* $q'_{j(k)}$, grande o suficiente para forçar a heurística a escalonar a tarefa $j_{(k)}$ antes de todas $i \in J$, e um segundo no qual $j_{(k)}$ tem uma

nova cabeça, $r'_{j(k)}$, grande o suficiente para ter $j_{(k)}$ escalonado depois de todos $i \in J$.

A altura da árvore gerada é tipicamente menor que n, e o número de nós raramente excede $2n$.

4 Resultados e Conclusão

Uma implementação(SBI) em FORTRAN da heurística *Shifiting Bottleneck* foi testada em uma *VAX 780/11*, utilizando pequenos problemas e problemas com mais de 500 operações.

Os problemas indicados na tabela 3, são problemas que foram tanto gerados aleatoriamente quanto retirados da literatura. Em todos os problemas, com exceção do problema 1, todas as tarefas são processadas em todas as máquinas.

Como os resultados mostram, SBI levou cerca de um a dois minutos para resolver problemas grandes, embora tenha resolvido centenas de *micro-runs,* ou seja, problemas de escalonamento em uma máquina.

	Número de			SBI		
Problema	Máquinas	Tarefas	Operações	Makespan	CPU Seg	Micro-runs
1	5	4	20	13*	0.5	21
2	6	6	36	55*	1.5	82
3	10	10	100	1015	10.1	249
4	5	20	100	1290	3.5	71
5	10	10	100	1306	5.7	181
6	10	10	100	962	12.67	235
7	15	20	300	730	118.87	1057
8	15	20	300	774	125.02	1105
9	15	20	300	751	94.32	845
10	10	15	150	1172	21.89	343
11	10	15	150	1040	19.24	293
12	10	20	200	1304	48.54	525
13	10	20	200	1325	45.54	434
14	10	30	300	1784*	38.26	212
15	10	30	300	1850*	29.06	164
16	10	40	400	2553*	11.05	61
17	10	40	400	2228*	75.03	226
18	10	50	500	2864*	53.42	98
19	10	50	500	2985*	27.47	75

Tabela 3 – Desempenho do procedimento da heurística Shifiting Bottleneck(SBI).
Makespan: Makespan do melhor escalonamento obtido
Micro-runs: número de problemas de escalonamento de uma máquina resolvidos
CPU Seg: Segundos até achar a solução final
*: Valor conhecido por ser ótimo.

Pode-se notar também que o aumento no número de máquinas aumentou a dificuldade do problema. Mas, é interessante notar que, o aumento do número de tarefas contribuiu menos para a dificuldade do problemas, mas contribuiu para a melhora na qualidade da solução encontrada.

Com o objetivo de comparar esta implementação com abordagens existentes, foram resolvidos 40 problemas gerados por Lawrence (1984) que também foram resolvidos por ele utilizando dez procedimentos baseados em *priority dispatching rule(p.d.r.),* de modo direto e aleatório. Nestes problemas, cada tarefa deve ser processada em todas as máquinas.

As dez *p.d.r* usadas foram: FCFS (First Come, First Served), LST (Late Start Time), EFT (Early Finish Time), LFT (Late Finish Time), MINSLK (Minimum Slack), SPT (Shortest Processing Time), LPT (Longest Processing Time), MIS (Most Immediate Successors), FA (First Available) e RANDOM.

Cada *p.d.r* foi aplicada de modo direto e de modo aleatorizado. O modo aleatorizado consiste em selecionar aleatoriamente uma operação de acordo com uma distribuição de probabilidade que faz a chance de ser selecionado proporcional a prioridade de cada operação dada pela *p.d.r.*.

Foram executados os procedimentos das dez *p.d.r* para cada problema*.,* onde se extraiu a melhor solução e o tempo de execução foi a média do tempo das oito melhores *p.d.r.* para cada problema, ou seja, as duas mais ineficientes foram descartadas. A tabela 4 ilustra essa comparação.

Problemas	Priority Dispatching Rule		SBI		Melhoria
	Direto	Aleatorizado			

	Valor	CPU Seg	Valor	CPU seg	Valor	CPU seg.	SBI %
5 máquinas, 10 tarefas							
1	679	4.11	679	157	666*	1.26	1.91
2	792	4.03	727	125	720	1.69	1
3	673	4.22	634	113	623	2.46	1.74
4	670	4.33	621	139	597	2.79	3.86
5	594	3.58	594	100	593*	0.52	0.2
5 máquinas, 15 tarefas							
6	927	8.2	927	233	926*	1.28	0
7	947	8.57	920	194	890*	1.51	3.26
8	880	8.28	866	280	868	2.41	-0.2
9	952	8.31	952	260	951*	0.85	0.11
10	959*	8.4	959*	217	959*	0.81	0
5 máquinas, 20 tarefas							
11	1223	15.24	1223	364	1222*	2.03	0
12	1041	12.68	1040	291	1039*	0.87	0
13	1151	14.17	1151	409	1150*	1.23	0
14	1293	14.77	1293	379	1292*	0.94	0
15	1320	15.74	1314	327	1207*	3.09	8.14
10 máquinas, 10 tarefas							
16	1036	7.66	1036	240	1021	6.48	1.45
17	857	6.85	857	192	796	4.58	7.12
18	897	6.55	897	225	891	10.2	0.67
19	926	7.45	898	240	875	7.4	2.56
20	1001	7.89	942	289	924	10.2	1.91
10 máquinas, 15 tarefas							
21	1208	14.71	1198	392	1172	21.9	2.17
22	1085	13.93	1038	414	1040	19.2	-0.2
23	1163	14.22	1108	417	1061	24.6	4.24
24	1142	14.33	1048	435	1000	25.5	4.58
25	1259	14.7	1160	430	1048	27.9	1.03
10 máqunas, 20 tarefas							

26	1373	24.62	1373	744	1304	48.5	5.03
27	1472	25.79	1417	837	1325	45.5	6.49
28	1475	25.5	1402	901	1256	28.5	10.41
29	1539	25.38	1382	892	1294	48	6.37
30	1604	26.7	1508	816	1403	37.8	6.96
10 Máquinas, 30 tarefas							
31	1935	55.42	1852	1786	1784*	38.3	3.67
32	1969	57.48	1916	1889	1850*	29.1	3.44
33	1871	54.13	1806	1313	1719*	25.6	4.82
34	1926	55.65	1844	1559	1721*	27.6	6.67
35	2097	56.61	1987	1537	1888*	21.3	4.98
15 máquinas, 15 tarefas							
36	1517	26.2	1385	735	1351	46.9	2.45
37	1670	26.95	1551	837	1485	61.4	4.26
38	1405	24.43	1388	1079	1280	57.7	7.78
39	1436	24.4	1341	669	1321	71.8	1.49
40	1477	24.71	1383	899	1326	76.7	4.12

Tabela 4 – Comparando o desempenho do SBI.
Valor: Makespan do melhor escalonamento obtido
*: Valor provado como ótimo.

Como podemos notar na tabela 4, o procedimento implementando a heurística *Shifiting Bottleneck (SBI)*, encontrou soluções, na maioria das vezes (38 dos 40 casos), melhores que os procedimentos baseados nas *p.d.r.,* tanto na versão direta como aleatorizada, com um custo computacional comparável aos procedimentos que implementaram as *p.d.r.* de modo direto, mas pelo menos uma ordem de grandeza mais rápida do que as versões aleatorizadas.

As soluções tradicionais para os problemas de escalonamento em job shop apresentam soluções razoáveis, apesar de não repensarem as decisões tomadas a cada passo. Entretanto, podemos obter resultados mais aproximados da solução ótima se utilizarmos uma heurística que reotimize partes da solução do problema, a um custo maior de processamento.

A heurística *Shifting Bottleneck* propõe a reotimização das decisões tomadas a cada passo de seu processo, e obtém resultados mais satisfatórios do que os algoritmos gulosos tradicionais. Apesar de não termos estudado as garantias de aproximação da heurística, pudemos observar pelos experimentos realizados pelos autores dos artigos que a sua aplicação realmente garante melhores resultados.

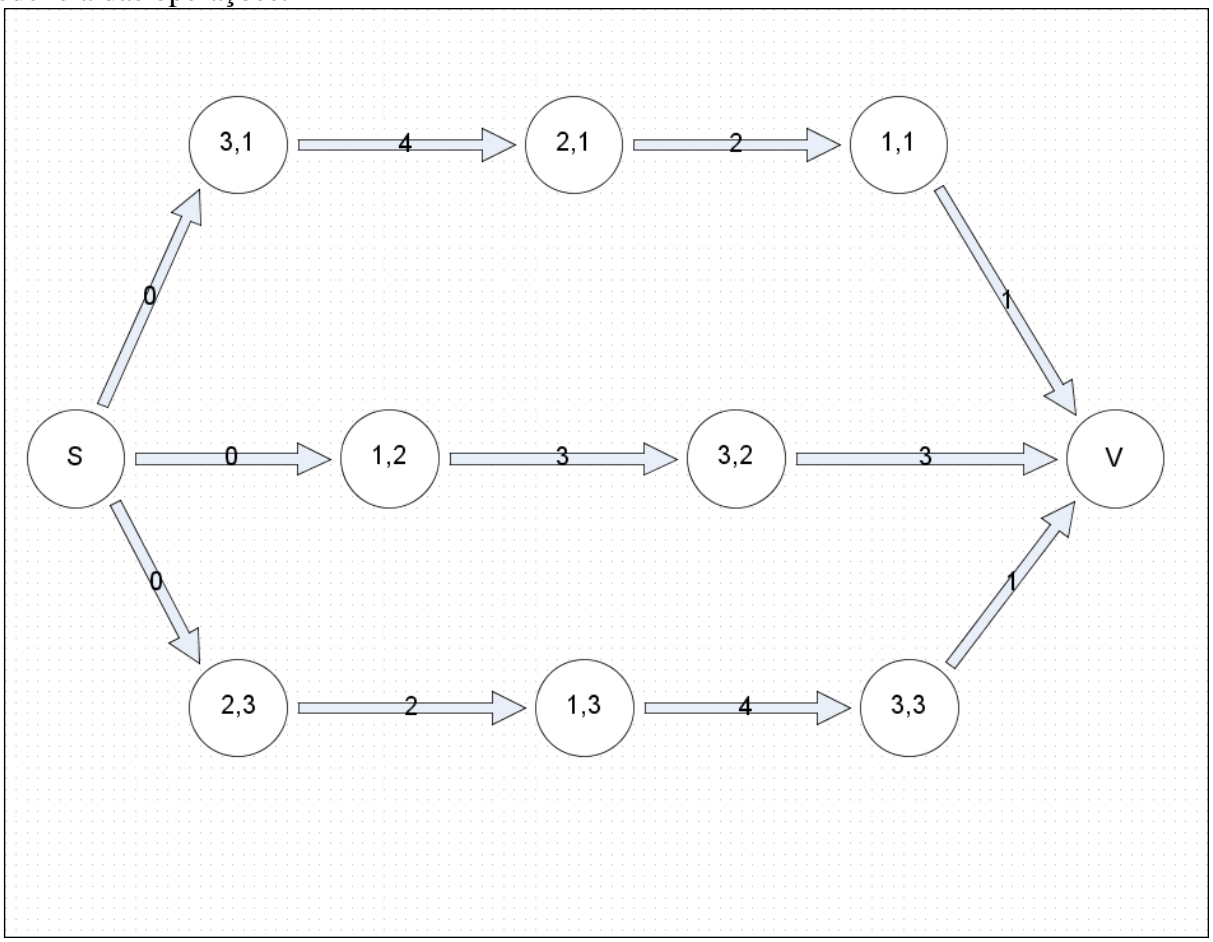
A idéia da heurística é bem intuitiva, e as justificativas do seu funcionamento fazem sentido. A modificação do caminho crítico do grafo, a cada passo, evidencia a melhora do *makespan* do escalonamento conforme as decisões são tomadas e o processo evolui.

O trabalho que desenvolvemos foi muito importante para o nosso aprendizado nesta disciplina. Estávamos curiosos a respeito de como eram solucionados os problemas difíceis, para os quais não havia uma solução intuitiva. Agora temos uma noção de como os problemas de escalonamento da vida real podem ser solucionados.

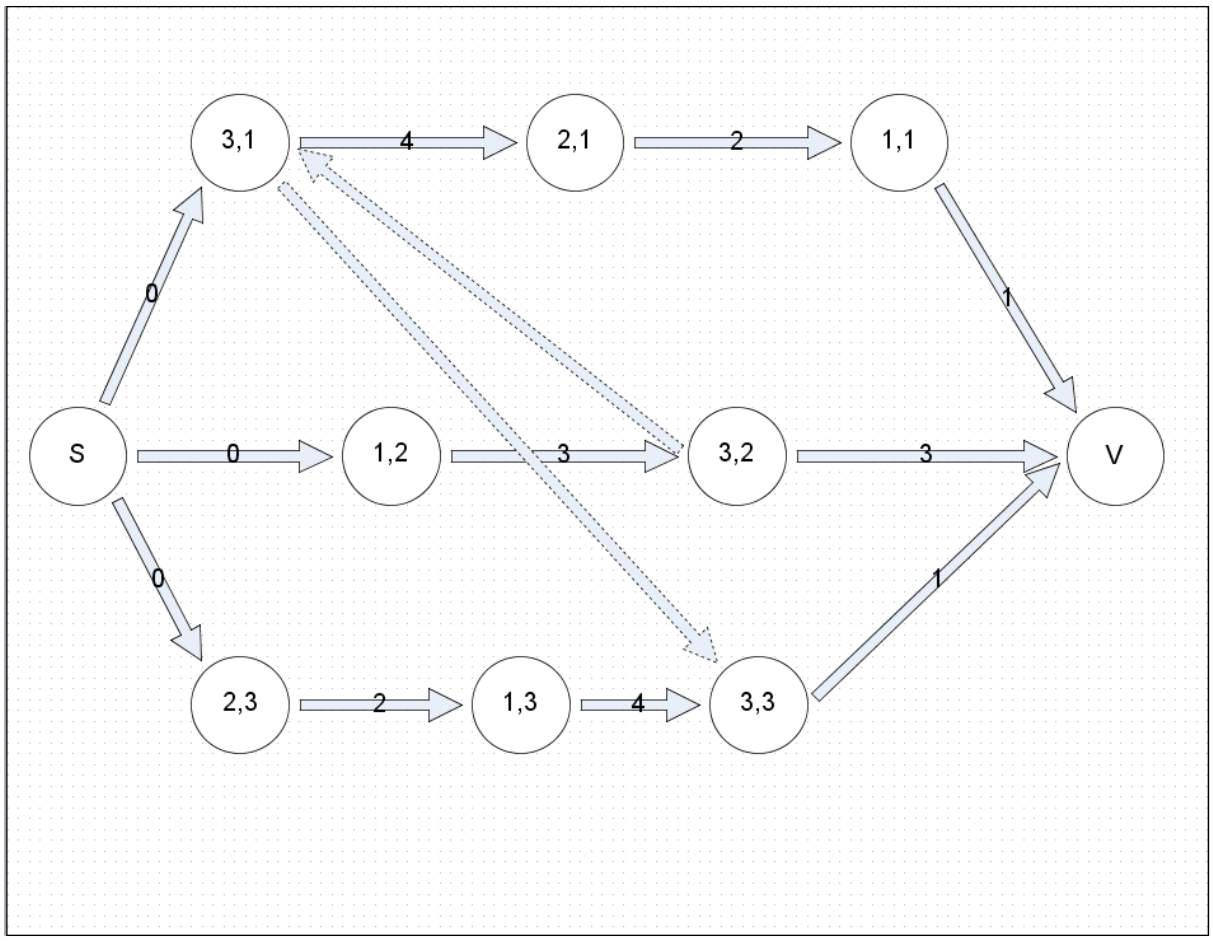
Apêndice A

Simulação

Abaixo segue uma simulação da execução da heurística Shifting Bottleneck em um problema de escalonamento em Job Shop. A instância do problema simulado é: 3 máquinas, 3 tarefas. Grafo de precedência das operações:



Pularemos o primeiro passo da simulação, que consiste em identificar a máquina 3 como a primeira máquina gargalo. O escalonamento que produz a solução do problema de otimização para a máquina 3 é adicionado ao grafo:

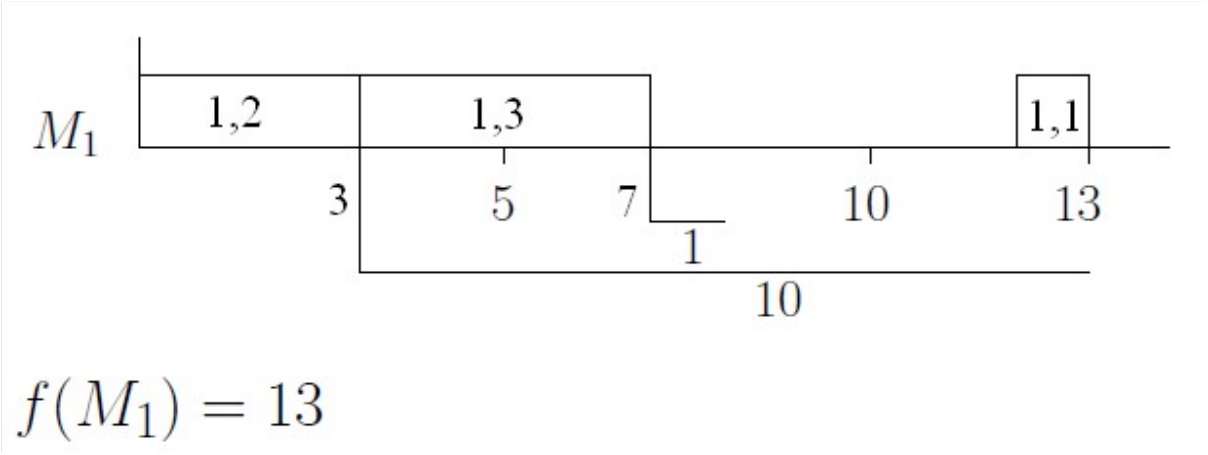


Cmax atual = 13

Temos M0 = {M3}. Devemos encontrar a máquina gargalo desta iteração. Para isso, resolvemos o problema head-body-tail para as máquinas pertencentes a M \ M0. No caso, estas máquinas são M1 e M2.

Escalonamento da máquina M1:

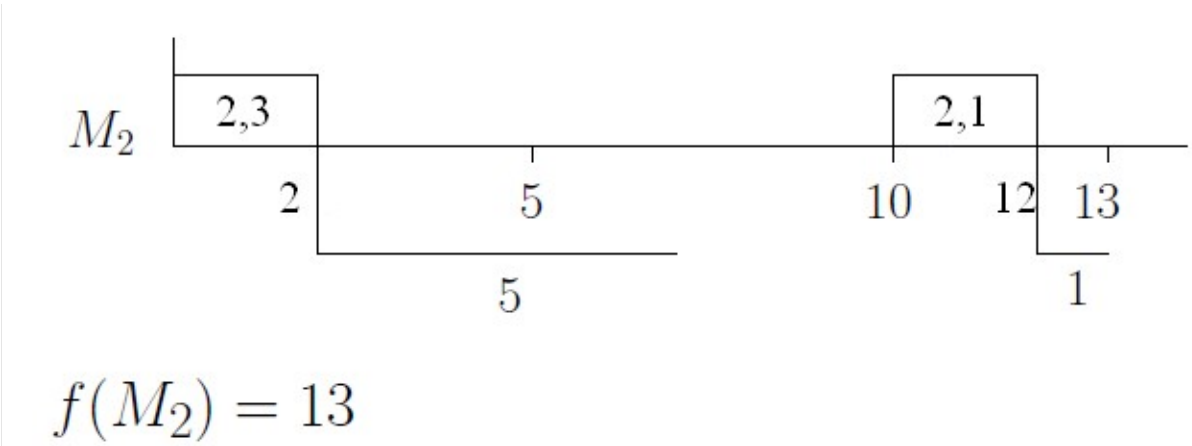
(i, j)	**(1, 1)**	**(1, 2)**	**(1, 3)**
rij	12	0	2
dij	0	10	1
pij	1	3	4



$$f(M_1) = 13$$

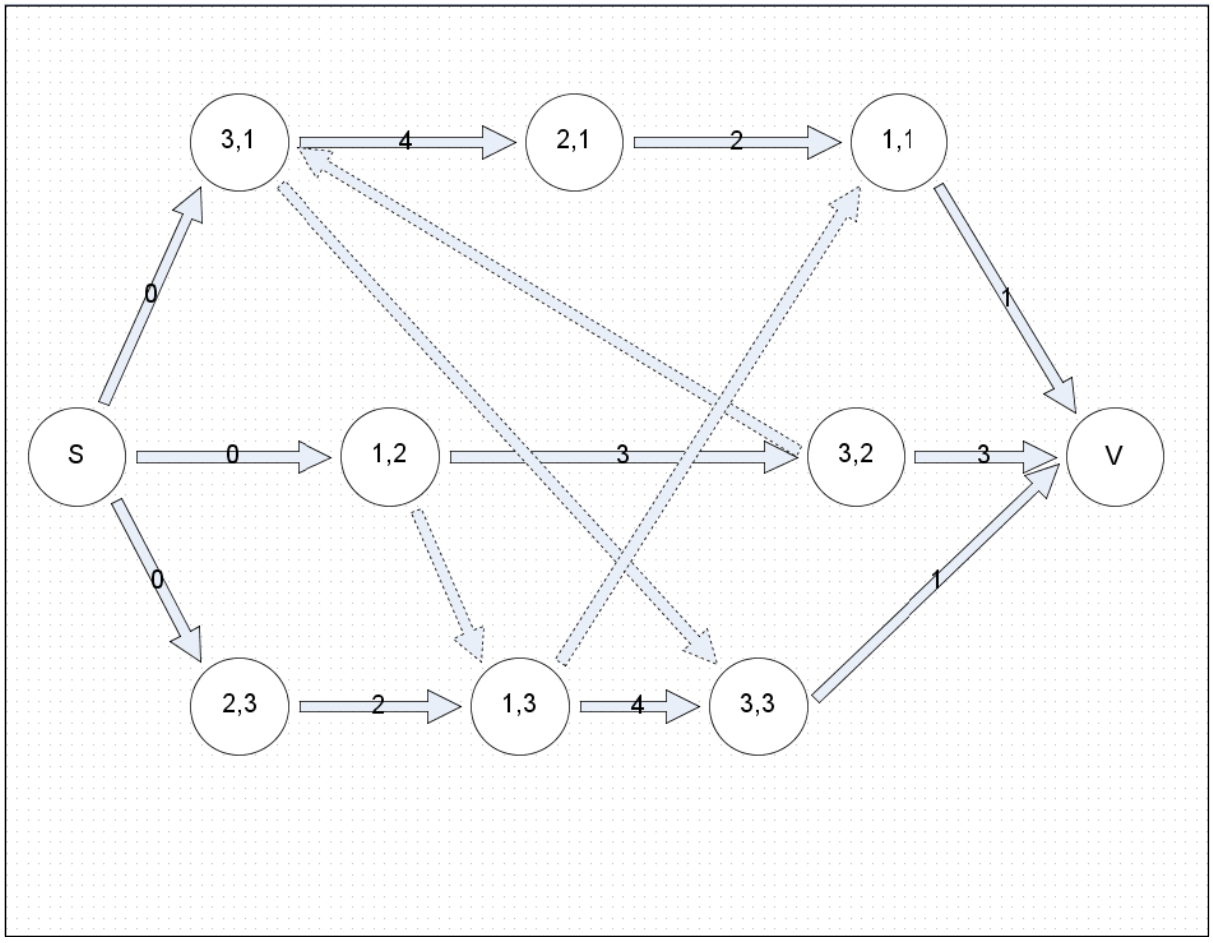
Escalonamento da máquina M2:

(i, j)	**(2, 1)**	**(2, 3)**
rij	10	0
dij	1	5
Pij	2	2



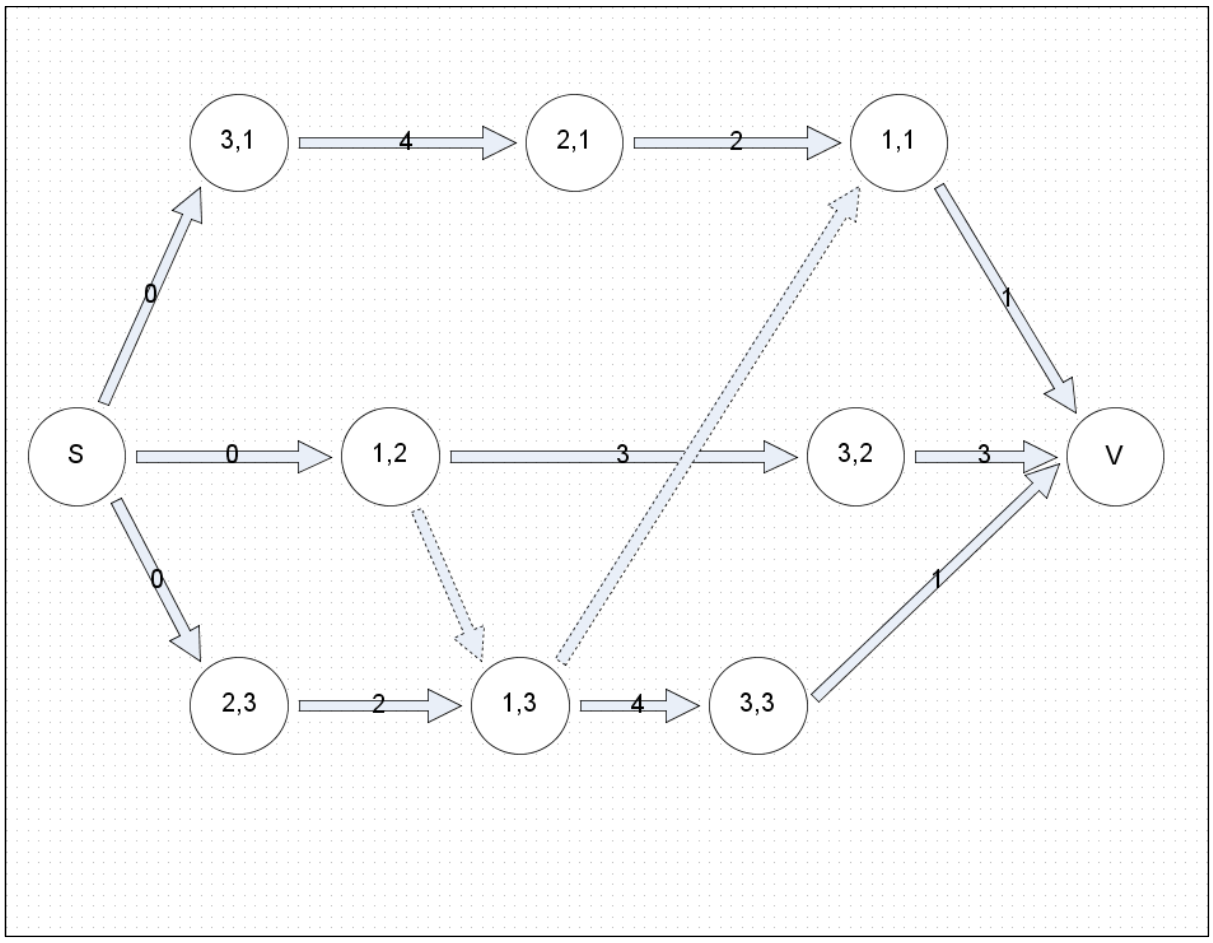
$$f(M_2) = 13$$

Ambas as máquinas obtiveram o mesmo valor de f. Escolhemos a máquina M1 como gargalo, por ter o menor índice. Adicionamos o seqüenciamento da solução do problema para a máquina M1 no grafo:



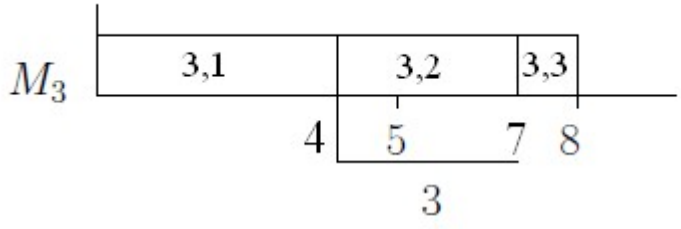
Cmax atual = 13

Devemos resequenciar a máquina M3. Para isso, removemos as arestas de M3 do grafo e minimizamos de novo o maior atraso para M3, levando em conta as arestas de M1.



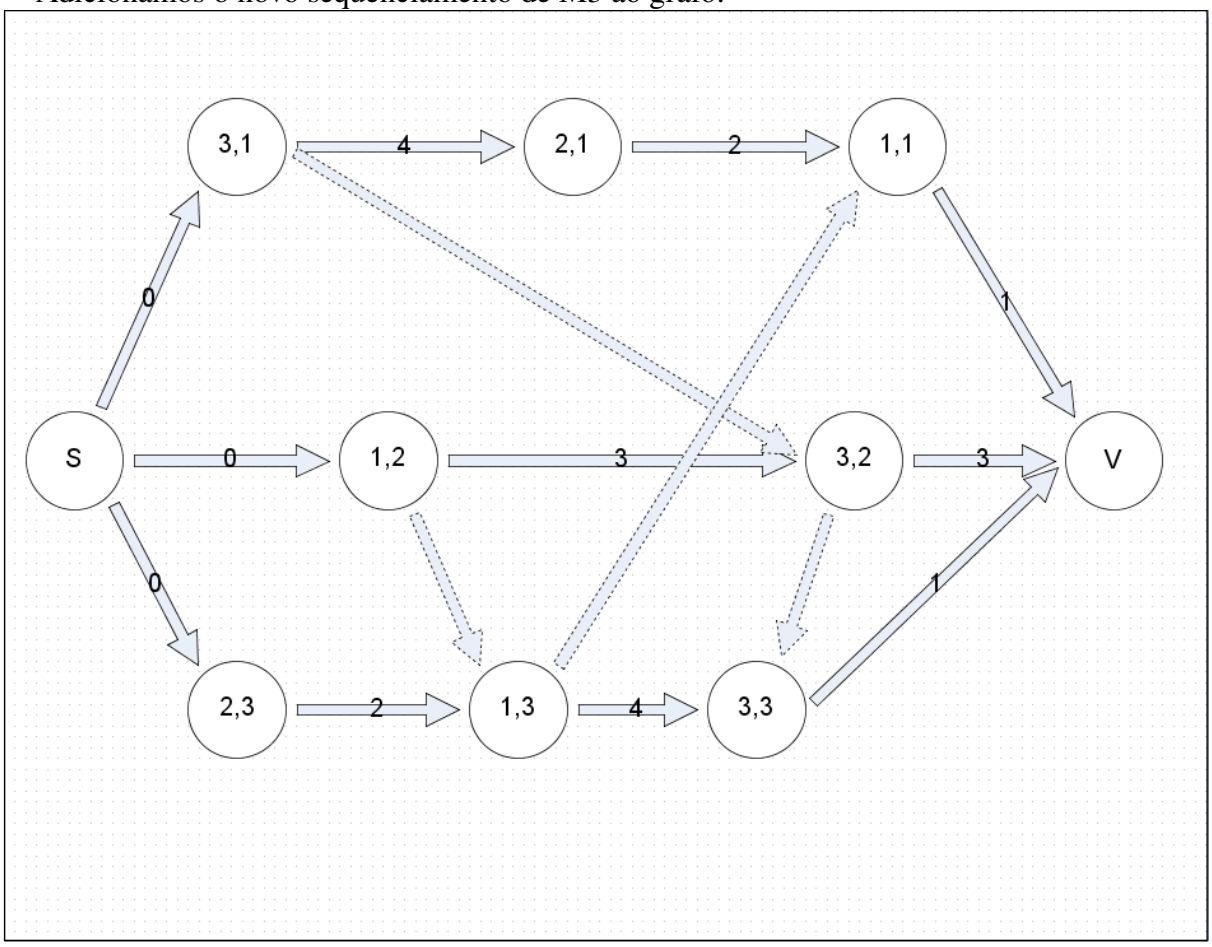
Escalonamento da máquina M3:

(i, j)	**(3, 1)**	**(3, 2)**	**(3, 3)**
rij	0	3	7
dij	3	0	0
pij	4	3	1



$$f(M_3) = 3$$

Adicionamos o novo sequenciamento de M3 ao grafo:

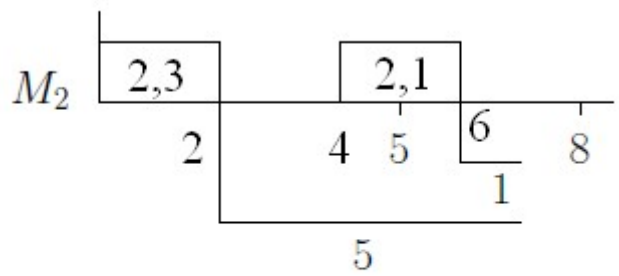


Cmax atual = 8

A única máquina restante é M2. Portanto, ela é o gargalo desta iteração.

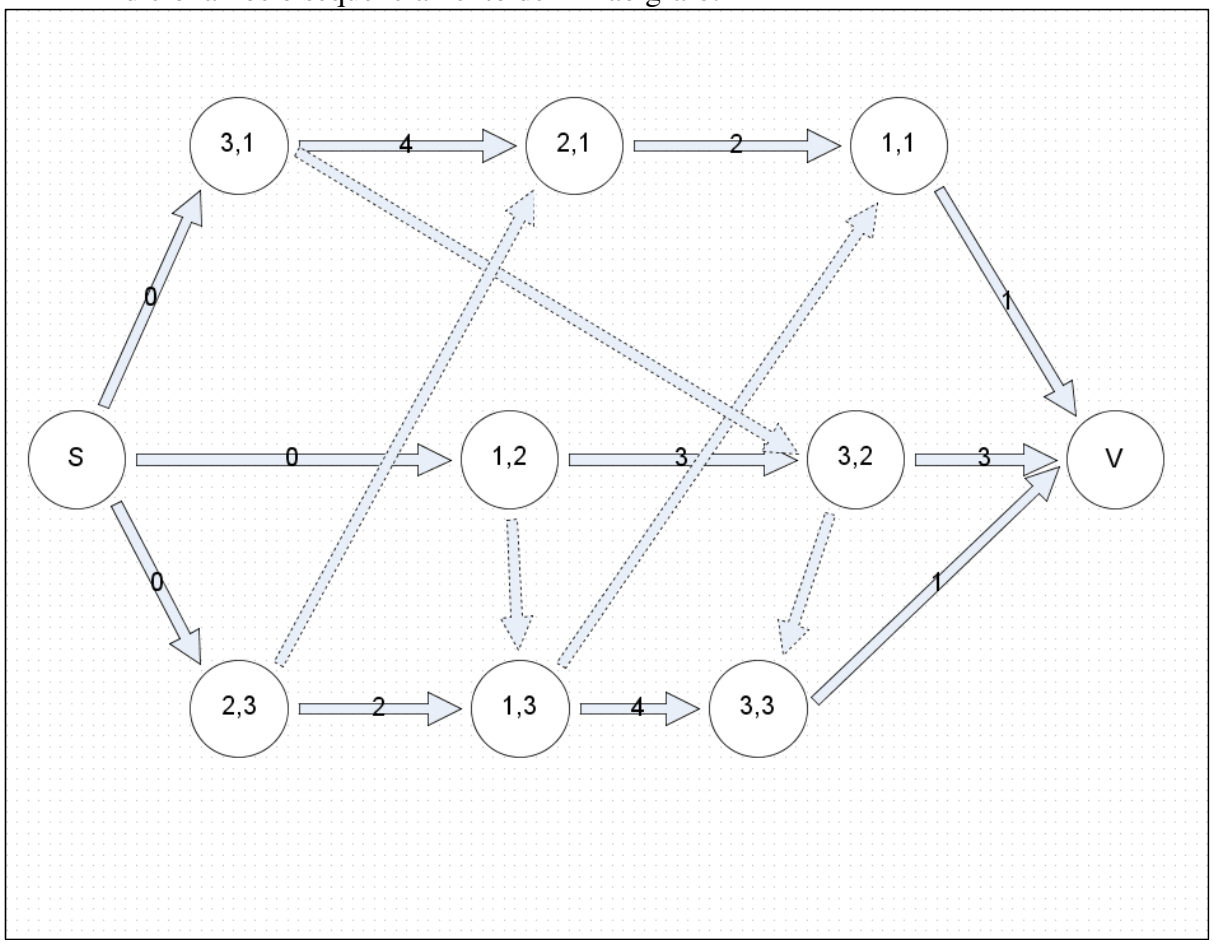
Escalonamento da máquina M2:

(i, j)	**(2, 1)**	**(2, 3)**
rij	4	0
dij	1	5
pij	2	2



$$f(M_2) = 7$$

Adicionamos o sequenciamento de M2 ao grafo:

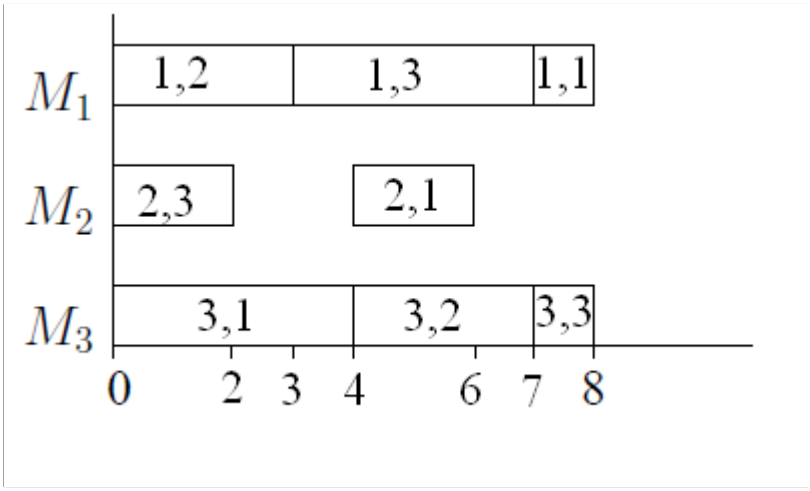


Cmax atual = 8

Resequenciamentos de M1 e M3 não mudam as arestas do grafo, portanto chegamos ao fim do processo.

Escalonamento final das tarefas:

24



Cmax final = 8

Referências

Adams, Joseph; Balas, Egon; Zawack, Daniel. TheshiftingBottleneckprocedurefor jobshop scheduling. Management Science; Mar 1988; 34, 3; ABI/INFORM Global, pg.391

Scheduling(LNMB MasterCourse), lecture3
http://wwwhome.math.utwente.nl/~hurinkjl/sched/sl3-handout.pdf

Scheduling(LNMB MasterCourse), lecture8
http://wwwhome.math.utwente.nl/~hurinkjl/sched/sl8-handout.pdf