Victor Ferreira Ferrari - 187890
Vinícius Couto Espindola - 188115
TP3 - MC658 - Grupo 2

1. Modelagem do Problema

Para o modelo, foi escolhido utilizar os jobs como nível de abstração principal, e modelar as restrições sobre tarefas relacionando a cada job, pois as tarefas são sequenciais e dependentes das ordens dos jobs.

a. Variáveis

O objetivo do modelo é minimizar o *makespan* (ou seja, o tempo de término da última tarefa), e por isso essa é uma das variáveis. Outra das principais variáveis é o vetor de tempos de início: para cada job i, start[i] é seu instante de início.

Para modelar as restrições de trabalhadores por instante de tempo, foram feitas duas matrizes bidimensionais nas quais cada linha corresponde a uma ordem, e cada coluna corresponde a um instante de tempo. A matriz final possui a quantidade de trabalhadores presentes em cada instante de tempo por cada ordem (linha). Para isso, é utilizada uma matriz flags binária (assume os valores 0 ou 1) que marca, para cada ordem, qual o instante de tempo que cada job começa.

A busca é feita pelos tempos de início. Exemplos visuais para essas estruturas podem ser vistas na seção 2.

b. Descrição Alto Nível do Funcionamento do Modelo

Dentro de cada ordem, o job "i+1" só deve começar após o término do job "i". Assim, a restrição segue exatamente essa lógica, para cada job j que tem a mesma ordem que j+1, $start[j+1] >= start[j] + duração[j]$. O mesmo vale para as relações arbitrárias de precedência dadas na entrada, mas ocorrendo par-a-par.

Para a tabela auxiliar de *flags*, a principal restrição envolve colocar como "1" as células equivalentes ao início de um job (onde a linha é a ordem e a coluna é o instante), $flag[ordem[j], start[j]] = 1$. Outra restrição (*exactly*) serve para restringir a quantidade de células verdadeiras por ordem, fazendo com que cada linha tenha exatamente o mesmo número de variáveis 1 e de jobs naquela ordem.

Para montar a matriz final (tabela *workload*), o modelo utiliza a matriz de flags para saber quando um job de uma certa ordem começa, e aplica o perfil de custos daquela ordem a partir daquele instante. Como as relações de precedência na mesma ordem já estão garantidas na tabela de flags, não haverá *overlap* em uma mesma ordem ou entre relações dadas de precedência.

Por fim, a soma de cada coluna da matriz final (trabalhadores para cada instante de tempo) tem que ser menor ou igual a L.

A restrição para a variável objetivo garante que esta assuma o maior instante de término, pegando o maior valor dentre a soma dos tempos de início de cada job com as respectivas durações.

c. Restrições Principais (não-redundantes)

Precedência entre Jobs

$start[prec[p,2]] >= start[prec[p,1]] + djj[prec[p,1]]$

$start[j+1] >= start[j] + djj[j] \ (se\ j\_ord[j] = j\_ord[j+1])$

O tempo de início de um Job de posterior na relação de precedência deve ser maior ou igual ao instante consecutivo ao término do Job predecessor. No caso de precedência entre ordens diferentes, utiliza-se a relação *prec* fornecida pela instância, no caso de mesma ordem, cria-se a restrição apenas se Jobs consecutivos são da mesma ordem.

Inicialização da tabela de flags

$flag[j\_ord[j], start[j]] = 1$

Se um Job (*j*) de ordem ***j_ord[j]***, tem tempo de início ***start[j]***, então deve-se sinalizar o agendamento de uma tarefa na linha $j\_ord[j]$ e coluna $start[j]$ na tabela de flags.

Inicialização da tabela de pesos

Se um Job de ordem *i* tem tempo de início *t*, então ***flag[i,t]=1***. Assim podemos aplicar o perfil de custos sobre a tabela *workload*. Supõe-se que o Job tem duração *3* e perfil de custos dado por ***[2,7,4]***, assim, temos que:

$workload[i,t] = flag[i,t] \cdot 2 + flag[i,t-1] \cdot 7 + flag[i,t-2] \cdot 4$

$workload[i,t+1] = flag[i,t] \cdot 2 + flag[i,t-1] \cdot 7 + flag[i,t-2] \cdot 4$

$workload[i,t+2] = flag[i,t] \cdot 2 + flag[i,t-1] \cdot 7 + flag[i,t-2] \cdot 4$

Com isso garantimos que: $workload[i,t] = 2,\ workload[i,t+1] = 7\ \&\ workload[i,t+2] = 4,\ se\ flag[i,t] = 1$.

Limite de trabalhadores por instante de tempo

Com a tabela de pesos por ordem e instante de tempo, temos que a somatória cada coluna corresponde ao total de trabalhadores em dado instante.

2. Exemplificação do Modelo

Utilizando a instância do enunciado como auxílio para a exemplificação do modelo, temos que o resultado ótimo obtido foi descrito pelos tempos de início descritos abaixo.

Tabela de tempos de início:

Ordens	1		2			3	
Jobs	1	2	3	4	5	6	7
Início	0	7	4	6	8	0	6

Para vincular a tabela de inícios a tabela de flags usamos a seguinte relação:

$flags[j\_ord[j], start[j]] = 1 \ \forall\ j \in JOB$

Assim temos a tabela de flags inicializada abaixo.

Tabela de flags:

		Instantes														
		0	1	2	3	4	5	6	7	8	9	10	11	12	....	19
Ordens	1	1	0	0	0	0	0	0	1	0	0	0	0	0	....	0
	2	0	0	0	0	1	0	1	0	1	0	0	0	0	....	0
	3	1	0	0	0	0	0	1	0	0	0	0	0	0	....	0

Para exemplificar a vinculação da tabela de flags com a tabela de custos, utilizaremos o Job 2 como exemplo. Com o perfil de custos [3,2,6,12] e as seguintes restrições sobre a tabela de custos:
Seja 'D' a duração dos Jobs da ordem 'i', 't' um instante qualquer e '*perfil*" um vetor com o perfil de custos da ordem 'i'. Além disso, define-se 'O' como o conjunto de ordens da instância e T como o conjunto de instantes. Assim, temos:

$$workload[i,t] = \sum_{k=0}^{D} flag[i,t-k] \cdot perfil[i,t+k] \quad \forall\ i \in O\ \&\ t \in T$$

$$workload[1,7] = flag[1,7] \cdot 3 + flag[1,6] \cdot 2 + flag[1,5] \cdot 6 + flag[1,4] \cdot 12$$
$$workload[1,8] = flag[1,8] \cdot 3 + flag[1,7] \cdot 2 + flag[1,6] \cdot 6 + flag[1,5] \cdot 12$$
$$workload[1,9] = flag[1,9] \cdot 3 + flag[1,8] \cdot 2 + flag[1,7] \cdot 6 + flag[1,6] \cdot 12$$
$$workload[1,10] = flag[1,10] \cdot 3 + flag[1,9] \cdot 2 + flag[1,8] \cdot 6 + flag[1,7] \cdot 12$$

Assim temos a tabela de custos por ordem e instante de tempo inicializada abaixo.

Tabela de custos:

		Instantes														
		0	1	2	3	4	5	6	7	8	9	10	11	12	....	19
Ordens	1	3	2	6	12	0	0	0	3	2	6	12	0	0	....	0
	2	0	0	0	0	6	3	6	3	6	3	0	0	0	....	0
	3	4	3	0	0	0	0	4	3	0	0	0	0	0	....	0

A partir da tabela de custos por ordem e instante de tempo, podemos garantir a restrição da mão de obra para todo instante de tempo.

3. Testes e Otimizações

a. Limites de Variáveis

Estabelecer limites inferiores e superiores para variáveis em PR é importante por restringir os domínios, assim diminuindo a quantidade de passos em *backtracking*.

No melhor caso, todas as ordens de jobs são executadas em paralelo, sem restrições de trabalhadores ou de relações extras de precedência. Assim, o limite inferior para o *makespan* é o valor máximo dentre os calculados pela multiplicação entre a duração dos jobs de cada ordem pelo número de jobs daquela ordem, ou seja, a duração máxima entre as ordens.

No pior caso, todos os jobs são executados sequencialmente, sem paralelismo algum. Assim, o limite superior para o *makespan* é a soma das multiplicações das durações dos jobs de cada ordem pelo número de jobs de cada ordem, ou seja, a soma das durações de todos os jobs.

Para tempos de início, o limite inferior deve ser 0, pois há sempre um job que inicia no instante 0. No pior caso para essa variável, o último job que termina também é o último que começa, e a sua duração é a menor dentre as ordens. Assim, o limite superior é encontrado ao tirar a menor duração dentre as ordens do limite superior para *makespan*.

b. Modelo Alternativo - BigV

Outro possível modelo envolve utilizar mais o nível de abstração de tarefas, considerando um job como uma coleção de tarefas.

Visando utilizar a restrição global *cumulative*, criou-se três vetores: um que contivesse os tempos de inícios de todas as tarefas, um que indicasse a respectiva duração de cada tarefa (como todas as tarefas são unitárias, é um vetor de elementos "1") e um que indicasse o custo de trabalhadores para cada tarefa. Com isso, garante-se a restrição do número de trabalhadores por instante de tempo através do *cumulative* (as restrições de precedência foram feitas a parte, como no modelo final).

A busca então seria feita sobre o vetor de tempos de início de tarefas. Essa estratégia pode ser usada em conjunto com a estrutura de *flags*, com esta se tornando uma estrutura redundante, ou sem estruturas auxiliares. Nos poucos testes feitos entre ambas estratégias, foi escolhida a primeira.

Na prática, foi visto que os resultados a princípio foram bem similares ao modelo final, porém à medida que as instâncias aumentam de tamanho, alguns resultados se mostraram um pouco piores que os obtidos anteriormente. Uma instância que não teve solução viável encontrada no modelo base foi resolvida neste, mas uma já resolvida não atingiu solução com o modelo.

Ainda nas maiores instâncias, alguns resultados foram melhores que os vistos pelo modelo original (por exemplo, a "*Ins_10o_87j_A*", cujo resultado foi 34 unidades de tempo melhor para este modelo) enquanto outros foram piores.

O modelo é promissor, e com a implementação de algumas ideias que não foram testadas por falta de tempo, poderia superar o desempenho do modelo *flags*. Porém, no âmbito do projeto, foi escolhido descartar este modelo.

c. Estruturas Redundantes

start_times - Visa restringir de forma redundante a tabela de flags com o vetor de tempos de início, de forma que cada linha representa uma ordem e, as colunas, tempos de início. Então, se um Job de ordem 'o' começa no instante 't', *start_times[o,t] = t*.

A tabela é relaciona com as flags restrição $start\_time[o,t] = flag[o,t] \cdot t$, garantindo a atribuição de valores a tabela *start_times*.

Por fim, a restrição global *member* restringe que o tempo de início de um Job de ordem 'o' esteja contido na linha 'o' da tabela start_times, ou seja: $start[j] \in start\_times[j\_ord[j], TIME]$, fechando a redundância.

d. Restrições Globais

exactly*(njo[i], flag[i,TIME], 1)* - cada ordem (*i*) deve ter exatamente o mesmo número de jobs (*njo[i]*) e flags levantadas (*flag[i,TIME]*), garantindo que as flags que não apontam nenhum tempo de início sejam zeradas.

count_geq*(flag[i, j..(j+djo[i]-1)], 1, 1)* - (redundante) garante que a tabela de flags não apresente jobs que interseccionam os tempos de execução. Se a flag do instante 'j' e ordem 'i' estiver levantada, e a ordem 'i' possui *Jobs* de duração *djo[i]*, então o intervalo de flag[i,j] até flag[i,j+k-1], deve conter apenas uma única flag levantada (aplica-se repetidas vezes nas linha/ordens da tabela de flags,).

maximum*(end, [start[i]+djj[i] | i in JOB])* - garante que a variável *end* assuma o valor do maior tempo de término entre todos os jobs.

member*(start_times[j_ord[j],TIME], start[j])* - (redundante) garante que o tempo de início do Job *j* (*start[j]*) de ordem *j_ord[j]*, esteja contido no conjunto de valores da linha *j_ord[j]* da tabela de tempos. Esta restrição vincula o vetor de tempos de início *start* com a tabela de flags, visando restringir o domínio de busca.

increasing*(start[ord_split[o]..(ord_split[o+1]-1)])* - (redundante) garante que os tempos de início dos Jobs de uma mesma ordem estejam em ordem crescente. O primeiro argumento define o *range* de Jobs que fazem parte desta restrição. No caso, *ord_split* é utilizado para identificar o índice do primeiro Job de uma ordem, assim, *ord_split[o] até (ord_split[o]-1)* indica o range de jobs que fazem parte da ordem *o*.

disjunctive*(start[ord_split[o]..(ord_split[o+1]-1)],djj[ord_split[o]..(ord_split[o+1]-1)])* - (redundante) garante que jobs de uma mesma ordem não tenham "*overlap*".

e. Testes de Parâmetros de Busca

Os parâmetros de busca utilizados foram "*most_constrained*" para seleção de variáveis e "*indomain_min*" para seleção de valores. A variável escolhida para busca foi o vetor de tempos de início. Esses parâmetros foram escolhidos empiricamente, a partir de testes do mesmo modelo modificando esses parâmetros.

Os testes mais extensos feitos foram com os parâmetros "*dom_w_deg"* e "*first_fail*" para seleção de variáveis. Em relação ao "*most_constrained*", o último é similar, mas teve um desempenho levemente pior pelo critério de desempate, que é a diferença entre os dois (o resultado de *makespan* foi o mesmo). O "*dom_w_deg*" forneceu resultados piores. Os outros testados ofereceram resultados notavelmente piores em testes mais restritos (*input_order, largest, smallest*).

Dentre os parâmetros de seleção de valores testados, o melhor foi o "*indomain_min*", o que faz sentido dentro do contexto de problema de

minimização (pegar os menores valores pode chegar a resultados melhores). Foram testados os valores “*indomain_max*”, “*indomain*”, “*indomain_median*” e “*indomain_middle*”.

Foram feitos testes que consideravam a tabela de *flags* como variáveis de busca no lugar dos tempos de início, por terem um domínio menor, e ainda restrições sobre elas. Porém, essa mudança gerou um aumento muito grande na duração de execução das menores instâncias, e respostas muito piores para instâncias maiores. Isso possivelmente ocorre por uma maior dificuldade de propagação.

4. Especificações do Ambiente e Resultados

As especificações de hardware do computador no qual foram feitas as execuções estão na Tabela 1.

Modelo da CPU	Intel(R) Core 17-3630QM (4C/8T)
Frequência do *Clock* da CPU	2.40 GHz
RAM Disponível	12 GB/1600 MHz

Tabela 1 - Especificações de Hardware.

Foi executado o modelo com limite de 5 minutos no *Minizinc*. As melhores soluções encontradas estão na Tabela 2.

Instância	Makespan	Tempo (s)	Nós Explorados
Ins_3o_7j_A	10	0.000223	7
Ins_4o_21j_A	82	0.075917	647
Ins_4o_23j_A	58	0.012534	97
Ins_4o_24j_A	69	300.000	2919846
Ins_4o_24j_B	74	300.000	4912008
Ins_4o_27j_A	67	300.001	2665541
Ins_6o_41j_A	167	300.001	1324031
Ins_6o_41j_B	128	300.001	1401135
Ins_6o_41j_C	136	300.001	1247413
Ins_6o_44j_A	121	300.001	1495310
Ins_6o_44j_B	155	300.005	1204232
Ins_8o_63j_A	304	300.006	444117

Ins_8o_63j_B	390	300.007	323374
Ins_8o_63j_C	344	300.007	218982
Ins_8o_65j_A	UNKNOWN	300.015	119261
Ins_8o_65j_B	UNKNOWN	300.007	141086
Ins_10o_84j_A	UNKNOWN	300.016	45823
Ins_10o_84j_B	UNKNOWN	300.013	50626
Ins_10o_85j_A	1075	300.016	292483
Ins_10o_87j_A	708	300.013	166716
Ins_10o_88j_A	UNKNOWN	UNKNOWN	UNKNOWN
Ins_10o_100j_A	UNKNOWN	UNKNOWN	UNKNOWN
Ins_10o_102j_A	UNKNOWN	UNKNOWN	UNKNOWN
Ins_10o_106j_A	UNKNOWN	UNKNOWN	UNKNOWN
Ins_12o_108j_A	UNKNOWN	UNKNOWN	UNKNOWN
Ins_12o_109j_A	UNKNOWN	UNKNOWN	UNKNOWN

Tabela 2 - Resultados e Condições

Pela referência [1], as soluções ótimas de algumas instâncias estão disponíveis para comparação. Para as outras instâncias, a referência possui bons resultados viáveis para comparação.

Para a primeira instância, foi verificado que a solução encontrada é ótima, a partir da solução ótima dada para a instância de exemplo no enunciado, que é similar.

Segundo a referência e verificação própria, pôde-se verificar que as soluções encontradas para as três primeiras instâncias são **ótimas**, assim como a encontrada para "*Ins_4o_27j_A*". Para as duas outras instâncias com solução ótima na referência, temos que as respostas encontradas são boas, distanciando de 1 unidade de tempo (*Ins_4o_24j_A)* e 2 unidades de tempo (*Ins_4o_24j_B*).

A partir da instância seguinte às analisadas, a qualidade da solução tende a cair, não alcançando as melhores soluções encontradas na referência (que não necessariamente são ótimas). A maior distância absoluta é de 245, na instância "*Ins_10o_85j_A*".

Em várias das maiores instâncias, o modelo não encontrou solução viável em 5 minutos, retornando o código "*UNKNOWN*" visto na tabela. Nas últimas instâncias, o *Minizinc* não retornou as estatísticas de execução também (tempo, nós explorados e outras informações). Ainda assim, o tempo gasto foi o limite imposto.

5. Conclusão

Dos resultados, se conclui que o modelo utilizado foi ineficiente em escalabilidade. Um motivo para tal pode ser o uso de restrições globais para redundância em restrições que englobam poucas variáveis, não tirando proveito da principal função de otimização delas.

O uso de restrições globais pode melhorar o desempenho do modelo, contanto que seja feito de forma adequada. Idealmente, deve-se englobar o maior número de variáveis dentro de um restrição global altamente otimizada para que seja realizada a redução do domínio das variáveis de forma eficiente. O uso de restrições globais em redundância melhora o desempenho gradualmente, mas em outro modelo poderia ter um ganho em ordens de grandeza.

Referências:

[1] Cristina C.B. Cavalcante, Cid Carvalho de Souza. *Scheduling Problem under Labour Constraints -- Benchmarks*. Acesso em: 29/05/2019. Disponível em: <http://www.ic.unicamp.br/~cid/SPLC/SPLC.html>