Victor Ferreira Ferrari - 187890
Vinícius Couto Espindola - 188115
TP1 - MC658 - Grupo 3

1. Problema e Definição de Termos

O problema a ser resolvido consiste na ordenação de N tarefas em 2 máquinas, sendo que cada tarefa passa por duas máquinas e tem duração diferente em cada máquina, de forma a minimizar a soma do tempo de término das tarefas na máquina 2. Esse problema é conhecido como “Flow Shop Scheduling”, pertencente à classe NP-Difícil.

Para resolução do problema, foi implementado um algoritmo utilizando *branch and bound* com a estratégia de *best bound*. O algoritmo consiste em descer por uma **árvore de estados**, calculando estimativas inferiores (limitantes **duais**) e superiores (limitantes **primais**) para cada caminho, e assim conseguir descartar (“**podar**”) a maior quantidade possível de caminhos, selecionando sempre o menor dual para continuar a exploração.

Cada **nó** da árvore, que é composto por uma ordenação parcial das tarefas, possui $n - h$ filhos, sendo n o número de tarefas e h o nível do nó na árvore. Um nó é **explorado** assim que é inserido na fila de prioridades.

2. Implementações

a. Base

O algoritmo foi implementado na linguagem C. Após a leitura dos arquivos de entrada, é criado um nó inicial, inicializado com zeros, que serve de raiz da árvore de estados, que é inserido na fila de prioridades.

O algoritmo implementado utiliza a estratégia *best bound*, onde o próximo nó a ser explorado é definido por uma função classificadora, cujo valor retornado é usado como limitante dual do nó. O próximo nó explorado é o que possui menor dual.

Para a fila de prioridades, foi utilizado um heap de mínimo, dinâmico. O heap começa a execução com tamanho igual a $2 \times n_{tarefas}$, e tem seu tamanho dobrado quando completamente preenchido.

Ao ser retirado do heap, é feita uma verificação e possível atualização do melhor limitante dual global visto até então, se o limitante dual do nó atual for maior que a antiga melhor estimativa. Isso é feito antes da expansão dos filhos pois, pela estratégia *best bound*, sempre é explorado o nó com menor limitante dual. Isso evita que um limitante dual maior que a solução ótima seja considerado o melhor.

Depois, o nó tem seus filhos expandidos. Isso é feito ao calcular os limitantes primal e dual para aquele nó, além de seus momentos de término em cada máquina e da soma dos tempos de término das tarefas já alocadas na máquina 2.

O cálculo do limitante dual é feito por meio de duas funções classificadoras, soluções de relaxações do problema. Em uma, supõe-se que toda tarefa na máquina 2 inicia logo após o término da mesma na máquina 1. A segunda função tem a suposição de que toda tarefa na máquina 2 inicia logo após o término da tarefa precedente na máquina 2. Calcula-se então a

soma dos tempos de término de cada tarefa para ambas relaxações, e a maior é retornada como limitante dual do nó.

O cálculo do limitante primal se dá pela soma dos tempos de término de uma alocação das tarefas restantes. Essa alocação pode ser feita em ordem crescente de duração na máquina 1 ou da máquina 2. O tempo de término de uma determinada tarefa é dado pela soma do maior valor entre o tempo de término da tarefa precedente na máquina 2 ou o tempo de término da tarefa atual na máquina 1, de modo a não haver sobreposição de tarefas em qualquer máquina. Assim, a solução é viável.

Feito isso, cada filho é testado para podas por limitante e por otimalidade. A poda por limitante é feita quando o limite inferior (dual) de um nó é maior do que a melhor solução viável encontrada, o melhor limite superior (primal) global. Todos os caminhos que começam nesse nó podem então ser descartados.

Entre as podas, é feita a verificação e possível atualização do melhor limitante primal encontrado, se o limitante primal do nó atual for menor que a antiga melhor solução. Se o nó não foi podado por limitante, ele pode possuir a melhor solução encontrada até então.

A poda por otimalidade é feita quando os limitantes dual e primal de um mesmo nó são iguais. Nesse caso, os filhos não precisam ser expandidos, pois a melhor solução para aquele caminho já foi encontrada. Se o nó não foi podado, é inserido no heap para ser explorado futuramente.

A execução termina quando o heap está vazio (ou seja, a exploração da árvore foi finalizada e a melhor solução encontrada é a ótima), ou quando um dos limites impostos pela entrada foi atingido: número de nós explorados ou duração máxima.

Diferentemente das podas por limitante e por dominância, a poda por otimalidade pode apresentar um primal e um dual que sejam ótimos. Como o primal é atualizado mesmo que um nó seja podado por otimalidade, e o próximo nó explorado é sempre o aberto que possui menor limitante dual (*best bound*), pode-se dizer que quando é encontrado um dual no heap igual ou maior que o melhor limitante primal, foi encontrada a melhor solução, e esta solução deriva de um primal podado por otimalidade. Devido a isso, quando o menor limitante dual do nó retirado da fila de prioridades é maior que o melhor primal já encontrado, pode-se dizer que o melhor dual é igual ao melhor primal e, portanto, a solução ótima.

b. Otimizações

Algumas pequenas otimizações puderam ser feitas ao longo da implementação do algoritmo base, para melhor eficiência.

Nos casos de poda, os nós não são alocados dinamicamente, tendo apenas seus valores calculados temporariamente e guardados em uma variável auxiliar, que é alocada apenas uma vez. Apenas caso não haja poda do nó e ele tenha que ser inserido no heap, o nó é alocado dinamicamente, reduzindo a alocação de memória e o tempo gasto.

O cálculo de dois limitantes superiores, baseados na ordenação das tarefas pelas máquinas 1 ou 2, é desnecessário na maioria dos casos, afetando o tempo total de execução do programa e, em alguns casos, o

resultado retornado. Por isso, foi retirado o cálculo de um dos limitantes, deixando apenas o limitante a partir da ordenação das tarefas pela duração na máquina 1.

c. Outras Estratégias.

Uma estratégia para aumentar o número de podas da árvore de estados é melhorar a qualidade dos limitantes calculados. Um bom candidato para foco é o limitante primal, pois é responsável pelas podas por limitante, tipo mais comum de poda.

Na versão implementada, as tarefas restantes são ordenadas por duração na máquina 1. O mesmo cálculo pode ser feito a partir da duração na máquina 2, porém essa mudança não necessariamente melhora a qualidade do limitante. Para garantir que, das duas opções, o melhor valor será o primal, basta calcular ambos e selecionar o menor deles.

Outra estratégia para aumentar o número de nós podados foi a implementação de regra de dominância, explicada na próxima seção.

d. Regra de Dominância

A relação de dominância implementada foi a descrita na referência indicada no enunciado do problema[1]. A relação de dominância consiste em eliminar nós que, apesar de apresentarem soluções válidas, oferecem um resultado tão bom quanto algum outro nó.

Considerando dois nós A e B que possuem diferentes permutações do mesmo conjunto de tarefas já alocadas, se o tempo de término na segunda máquina do nó A for menor ou igual ao tempo de término da segunda máquina do nó B, e a soma do tempos de término da máquina 2 do nó A for menor ou igual que a do nó B, temos que A apresenta uma solução ótima tão boa quanto ou melhor que a solução ótima de B. Então A **domina** B/B **é dominado por** A.

A implementação da regra de dominância foi feita através de três componentes principais: a *struct node*, a função *remove_heap* e a função *check_dominance*. A *struct* armazena a permutação das tarefas até o nó atual (*char result[]*), o tempo de término na máquina 2 dada a permutação das tarefas do nó (*int f2tr*) e a soma dos tempos de término na máquina 2 das tarefas alocadas (*int sumf2*).

Depois que um nó é aberto e testado para poda por limitante e atualização do *primal bound*, ele é repassado para a função *check_dominance* que será responsável por comparar o nó com todos os já alocados no *min_heap*. Caso o nó ao ser expandido possua o mesmo conjunto de tarefas já alocadas, compara-se *f2tr* e *sumf2*. Se o conjunto de tarefas for o mesmo e o nó aberto domina sobre o nó alocado no heap, a função *remove_heap* é chamada para remover o nó dominado da estrutura. Se o conjunto de tarefas for o mesmo e o nó aberto é dominado, este nó não é inserido no heap e a checagem por dominância é finalizada.

Dado que tal checagem já foi realizada pelos nós que estão atualmente no heap, quando um nó do heap domina o nó aberto, isso implica

[1] “Combinatorial Optimization”, Papadimitriou e Steiglitz, Prentice-Hall INC., 1982.

que o nó alocado no heap já dominou todos que poderiam ser dominados pelo nó aberto. Caso as condições não sejam cumpridas (o nó não domina nem é dominado), nada é feito e a execução segue para a poda por otimalidade.

3. Resultados

Os testes foram realizados com limite superior de nós explorados igual a 150000000 e tempo máximo igual a 600s (10 minutos), em um computador pessoal cujas especificações estão disponíveis na Tabela 1.

Tabela 1 - Especificação do Hardware

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

O programa teve como saída os melhores limitantes primal e dual, a quantidade de nós explorados, o tempo necessário para encontrar os limitantes, e a duração total da execução. Esses dados podem ser encontrados na Tabela 2. As melhores soluções viáveis encontradas se encontram na Tabela 3. Quando o melhor limitante primal é igual ao limitante dual, a solução encontrada é ótima.

O código foi compilado usando o compilador *GCC*, utilizando a *flag* de otimização “O3”, para otimização máxima por parte do compilador.

Tabela 2 - Condições de Execução

Instância	Melhor Primal	Melhor Dual	Nós Explorados	Tempo Primal (s)	Tempo Dual (s)	Tempo Total (s)
i08	212	212	87	0,00	0,00	0,00
i08b	186	186	219	0,00	0,00	0,00
i09	197	197	367	0,00	0,00	0,00
i10	2177	2177	576	0,00	0,00	0,00
i10b	156	156	1777	0,01	0,01	0,01
i11	2965	2965	49	0,00	0,00	0,00
i12	3017	3017	230	0,00	0,00	0,00
i12b	302	302	11316	0,26	0,29	0,29
i13	24619	24619	7047	0,16	0,19	0,19
i13b	692	692	21728	1,56	1,64	1,64
i14	458	458	68300	8,97	9,77	9,77
i15	266	266	351720	2,68	74,17	74,17

i16	729	729	8743	0,26	0,32	0,32
i17	834	834	36044	4,27	4,27	4,27
i18	974	974	56343	12,36	12,37	12,37
i19	1009	1009	19043	1,89	1,89	1,89
i20	1162	1087	278651	403,46	587,82	600,01
i21	1089	1089	314271	451,23	525,43	525,43
i22	1214	1147	282827	513,08	596,61	600,00
i23	1310	1223	257276	387,42	556,02	600,04
i24	1171	1118	298154	431,06	573,50	600,00
i25	1818	1790	271454	584,40	581,36	600,04
i26	1525	1525	249950	416,08	416,72	416,72
i27	2065	2020	271034	387,21	599,53	600,01
i28	2067	1873	262451	420,63	587,82	600,01
i29	2333	2164	267930	368,08	566,21	600,05
i30	2304	2019	247390	177,88	594,70	600,02

Tabela 3 - Soluções Viáveis

Instância	Melhor Solução Viável Encontrada
i08	[4, 3, 2, 5, 7, 8, 6, 1]
i08b	[1, 7, 3, 5, 4, 6, 2, 8]
i09	[5, 7, 2, 3, 1, 4, 6, 9, 8]
i10	[6, 7, 5, 4, 1, 10, 3, 2, 9, 8]
i10b	[4, 7, 10, 8, 6, 9, 2, 3, 5, 1]
i11	[1, 8, 10, 2, 3, 4, 6, 5, 11, 9, 7]
i12	[11, 10, 9, 8, 12, 5, 7, 6, 4, 3, 2, 1]
i12b	[6, 1, 7, 11, 9, 3, 4, 10, 8, 2, 5, 12]
i13	[2, 12, 4, 6, 7, 11, 3, 9, 8, 10, 5, 13, 1]
i13b	[13, 2, 11, 4, 9, 6, 7, 8, 5, 10, 3, 12, 1]
i14	[8, 2, 10, 4, 11, 3, 5, 12, 9, 14, 7, 1, 6, 13]
i15	[11, 9, 7, 5, 15, 1, 3, 2, 4, 6, 8, 10, 12, 14, 13]

i16	[2, 4, 13, 14, 1, 3, 10, 12, 5, 6, 16, 15, 11, 7, 9, 8]
i17	[2, 8, 13, 5, 17, 11, 16, 10, 3, 9, 15, 14, 1, 12, 6, 7, 4]
i18	[17, 5, 4, 6, 1, 15, 12, 16, 13, 14, 10, 9, 11, 7, 2, 3, 8, 18]
i19	[4, 8, 13, 3, 2, 9, 6, 5, 11, 18, 19, 14, 7, 17, 16, 10, 1, 12, 15]
i20	[2, 15, 16, 11, 18, 17, 5, 13, 19, 10, 3, 4, 9, 1, 12, 14, 6, 20, 7, 8]
i21	[15, 10, 16, 2, 14, 11, 3, 9, 17, 6, 7, 20, 19, 5, 1, 4, 8, 18, 21, 12, 13]
i22	[9, 7, 11, 17, 13, 22, 15, 19, 12, 3, 4, 21, 5, 6, 14, 8, 10, 20, 1, 2, 16, 18]
i23	[6, 18, 19, 8, 12, 14, 17, 16, 2, 22, 9, 23, 21, 1, 13, 3, 11, 15, 20, 5, 7, 4, 10]
i24	[5, 4, 24, 17, 12, 19, 22, 21, 14, 16, 2, 3, 13, 18, 20, 10, 23, 11, 6, 9, 15, 1, 8, 7]
i25	[18, 9, 10, 14, 4, 7, 20, 12, 5, 11, 21, 25, 24, 22, 15, 2, 3, 6, 8, 13, 17, 19, 23, 1, 16]
i26	[24, 18, 26, 5, 11, 23, 3, 6, 25, 4, 8, 10, 12, 13, 17, 22, 14, 16, 1, 9, 20, 2, 15, 21, 7, 19]
i27	[9, 20, 7, 21, 22, 12, 3, 2, 4, 27, 15, 8, 1, 17, 10, 13, 24, 25, 11, 23, 18, 19, 5, 6, 14, 16, 26]
i28	[22, 10, 11, 19, 17, 1, 3, 7, 21, 4, 28, 9, 8, 24, 20, 14, 5, 6, 26, 2, 18, 23, 12, 13, 15, 16, 25, 27]
i29	[16, 18, 29, 28, 20, 1, 2, 14, 10, 25, 23, 4, 9, 11, 22, 21, 3, 5, 7, 26, 8, 19, 6, 15, 17, 27, 12, 13, 24]
i30	[28, 21, 15, 9, 14, 24, 13, 7, 3, 1, 5, 17, 4, 16, 25, 22, 27, 2, 6, 19, 30, 10, 8, 11, 12, 23, 18, 20, 26, 29]

Dada a Tabela 2, observa-se que o aumento dos tempos não apresentam um padrão claro com o tamanho da entrada. Isso pode ser exemplificado comparando as instâncias i18, i19, i20 e i21 que aparentam ter uma variação quase arbitrária do tempo de execução. Justifica-se essas discrepâncias a partir das características intrínsecas do algoritmo utilizado e do problema em questão: existem diferentes entradas de mesmo tamanho que apresentam uma performance muito diferente (*e.g* i13/i13b). O tamanho da entrada não define o quão eficiente são as podas, mas sim os valores dela.

4. Comparação

a. Parâmetros Apertados

A implementação que utiliza regra de dominância possui um *tradeoff* com a base: explora uma quantidade bem menor de nós da árvore de estados, fazendo uma maior quantidade de podas, porém para isso gasta muito mais tempo por nó explorado, uma vez que compara o nó aberto com todos os já alocados no heap. Isso foi visto pela soma do tempo total de execução das instâncias para os dois casos.

A implementação base, por sua vez, explora muitos nós, e por isso memória se torna um limite. Assim, ambas implementações possuem aspectos que não escalam bem, o que dificulta dizer qual a melhor.

Desse modo, foram escolhidos parâmetros apertados para comparar desempenho: 10000000 nós e 1 minuto. Analisando os resultados, tem-se que em média o *gap de otimalidade*, definido por $g = \frac{p-d}{p}$, é menor para a implementação com regra de dominância. Logo, pode-se dizer que essa implementação possui melhor desempenho.

A implementação com cálculo duplo de primais nesses parâmetros gerou resultados levemente melhores que a base *(gap* médio menor), porém teve um acréscimo de tempo mais significativo, mostrando que com parâmetros mais relaxados essa vantagem pode ser anulada pela desvantagem na duração.

Ao unir a estratégia da regra de dominância e do cálculo duplo de primais, o resultado foi melhor que a implementação base, mas foi ainda pior que o resultado da implementação com relação de dominância. Isso se deve ao acréscimo de tempo que a estratégia do cálculo duplo de primais fornece, piorando ainda mais o tempo de execução, que já é um limite para a implementação de dominância.

Uma instância que teve considerável diferença no resultado (com parâmetros apertados) entre implementações é a "i14". Nas implementações sem regra de dominância o limite de nós é alcançado em poucos segundos e a solução ótima não é encontrada, enquanto com a regra de dominância a solução ótima é encontrada antes do limite de tempo ser alcançado. Assim, o *gap* foi reduzido a zero naquela instância. As condições de saída para cada implementação na instância i14 podem ser vistas na Tabela 4.

Tabela 4 - Instância i14 em diferentes estratégias

	Primal	Dual	Nós	T. Primal	T. Dual	T. Total
Base	459	447	10000000	5.77	5.37	6.43
2 Primais	459	447	10000000	5.89	5.48	6.55
Dominância	458	458	68300	9.07	9.93	9.93

Feita essa análise de desempenho com parâmetros apertados, foram feitos testes com parâmetros relaxados, 150000000 nós e 10 minutos, que permitem a redução do impacto dos limites na execução.

Na análise dos resultados com alto limite de nós, foi visto que o *gap de otimalidade* médio da implementação base foi menor que o visto pela implementação com cálculo duplo de primais, porque o adicional de tempo no último afeta mais com mais nós explorados, e a execução termina antes.

Comparando as estratégias base e regra de dominância, cada implementação atingiu um limite diferente. A implementação base chegou ao limite de nós em todos os casos em que a solução ótima não foi encontrada (o limite de tempo não foi atingido em instância alguma). A implementação de dominância atingiu o limite de tempo em todos os casos em que a solução

ótima não foi encontrada. Como visto na Tabela 2, nenhuma instância explorou mais que 352000 nós.

O *gap de otimalidade* médio de ambas implementações foi similar, porém com uma pequena vantagem à estratégia de dominância.

b. Parâmetros Relaxados

Feita essa análise de desempenho com parâmetros apertados, foram feitos testes com parâmetros relaxados, 150000000 nós e 10 minutos, que permitem a redução do impacto dos limites na execução.

Na análise dos resultados com alto limite de nós, foi visto que o *gap de otimalidade* médio da implementação base foi menor que o visto pela implementação com cálculo duplo de primais, porque o adicional de tempo no último afeta mais com mais nós explorados, e a execução termina antes.

Comparando as estratégias base e regra de dominância, cada implementação atingiu um limite diferente. A implementação base chegou ao limite de nós em todos os casos em que a solução ótima não foi encontrada (o limite de tempo não foi atingido em instância alguma). A implementação de dominância atingiu o limite de tempo em todos os casos em que a solução ótima não foi encontrada. Como visto na Tabela 2, nenhuma instância explorou mais que 352000 nós.

O *gap de otimalidade* médio de ambas implementações foi similar, porém com uma pequena vantagem à estratégia de dominância.

5. Conclusão

Durante a análise dos resultados das diferentes implementações, a principal conclusão foi a verificação do *tradeoff* entre memória gasta e tempo de execução. É possível fazer um programa que resolve o problema de forma mais rápida, utilizando mais memória, assim como pode-se implementar uma solução que não gasta tanta memória, mas é bem mais lenta. A solução ideal depende de qual é o fator limitante no ambiente de execução, mas um bom objetivo é encontrar um meio-termo entre esses extremos.