4

Resultados computacionais

4.1

Sistema usado para execução e simulação das técnicas

Todos os testes foram conduzidos num computador com processador Intel Pentium 4 de 1,7GHz com 224 MB de memória RAM DDR e 40GB de disco rígido, utilizando Windows XP® como sistema operacional. Todas as técnicas foram executadas num ambiente monousuário com a mais alta prioridade de escalonamento, garantindo uma elevada utilização do processador (normalmente em torno de 99% de uso da CPU). Como base para implementação das técnicas, foi desenvolvida toda uma hierarquia de classes em C++. Dentre as principais classes, pode-se citar:

classe link: Caracteriza um enlace de comunicação entre dois processadores P_0 e P_i, possuindo uma latência g_i, uma taxa de envio de dados $(1/G_i)$ e duas referências aos objetos da classe *host* de suas extremidades.

classe host: Caracteriza um processador P_i com tempo de processamento de uma unidade de informação igual a w_i.

classe grid_system: Classe que representa todo um *grid* computacional, contendo vários objetos das classes host e link.

classe simulation: Classe que inclui a descrição de um *grid*, bem como uma tarefa divisível a ser executada e uma referência a um objeto da classe escalonador que é utilizado durante a simulação.

classe task: Define uma tarefa genérica, podendo ser computacional ou de comunicação.

classe data_transfer: Especialização da classe task para representar transmissão de dados.

classe divisible_task: Especialização da classe task para representar tarefas computacionais divisíveis.

classe scheduler: Classe base para o desenvolvimento dos escalonadores descritos neste trabalho.

No Apêndice B é apresentado um trecho de código como exemplo de utilização das classes desenvolvidas. Para compilação e otimização dos códigos implementados, foi utilizado o compilador do ambiente integrado de desenvolvimento Microsoft Visual C++ ® 6.0 configurado para otimização máxima visando velocidade.

Para efetuar as análises tanto dos modelos de programação linear inteira mista, quanto das heurísticas desenvolvidas neste trabalho, não seria necessário o uso de simuladores, uma vez que o *makespan* gerado pelo escalonamento já é encontrado pela própria técnica. Existem porém técnicas encontradas na literatura e estudadas neste trabalho que necessitam ser analisadas com o uso de um simulador, devido ao seu comportamento pouco estruturado. Mesmo para as técnicas que não necessitariam ser processadas por simuladores, optou-se por utilizar a simulação do sistema computacional (estado dos processadores e enlaces, garantindo suas propriedades descritas na Seção 1.1), visando um método uniforme de análise e coleta de dados.

Dois sistemas foram utilizados para a simulação das técnicas, o simulador SimGRID [10, 18] e o simulador *SimStar* desenvolvido durante este trabalho. Inicialmente optou-se pelo simulador SimGRID, uma vez que é o mais utilizado na literatura [3, 22, 23]. Foi necessário portá-lo para o sistema computacional utilizado (Windows e Microsoft Visual C++ ® 6.0). Verificou-se porém que sua utilização implicaria em uma sobrecarga desnecessária de processamento, que tornaria a realização dos experimentos mais onerosa. Observando-se a simplicidade do sistema a ser simulado, foi desenvolvido o simulador *SimStar* descrito no Algoritmo 6, que utiliza-se apenas de algumas variáveis de estado para executar a simulação, exigindo um custo computacional muito baixo.

O simulador desenvolvido inicia a simulação no tempo $t = 0$ com uma carga total W a ser computada em um *grid* computacional com $n + 1$ processadores (o processador mestre e os demais n processadores escravos) e n enlaces de comunicação. Para cada processador $P_i, 1 \leq i \leq n$ é guardado um valor M_i que indica o próximo momento em que o processador estará disponível para processar dados. É também guardado um valor m_i que indica o próximo momento que o enlace de comunicação com P_i estará disponível para

ser utilizado. Por fim, o simulador também mantém uma variável d que indica o próximo momento em que o processador mestre estará disponível para enviar dados. Estas informações são necessárias, pois não é permitida concorrência de comunicação num mesmo enlace, nem concorrência de processamento num mesmo processador.

O simulador executará enquanto houver carga W a ser processada. Antes do início de cada período, na linha 8 é invocado o escalonador que irá definir uma ordem *order* de envio a ser seguida, bem como as quantidades de dados α_i que cada processador P_i deve receber. Nas linhas 9 a 11, caso o tempo t de simulação seja superior ao momento d em que o processador mestre estará disponível para enviar dados, este é então atualizado com o valor t. Nas linhas 12 a 21 é simulado o envio e o processamento dos dados por cada processador $P_{order[j]}, 1 \leq j \leq n$, seguindo desta forma a ordem *order* de envio.

Para cada processador P_i na ordem de envio, é verificado na linha 14 se ele deve receber dados. Caso deva receber, é atualizado na linha 15 o momento m_i que este processador P_i estará disponível para receber dados novamente, que será o tempo do início do envio mais o tempo de comunicação $(g_i + G_i\alpha_i)$. O tempo de início de envio será o maior entre o momento d em que o processador mestre está pronto para enviar dados e o momento m_i que indica quando o processador P_i esta pronto para recebê-los. Na linha 16 o instante d é atualizado para refletir o momento que o processador mestre terminou de enviar dados para P_i. Na linha 17 é atualizado o momento em que P_i estará pronto para processar novos dados, ou seja, o momento em que P_i terminará o processamento dos α_i dados. Este momento é igual ao tempo de início de processamento destes α_i dados mais a duração $w_i\alpha_i$. O tempo de início de processamento será o maior entre o momento que P_i terminou de receber α_i dados e o momento que P_i estava pronto para iniciar o processamento (após terminar o processamento dos dados do período anterior). Na linha 18 é então atualizado o *makespan* para refletir sempre o maior M_i dentre todos os processadores em todos os períodos. Por fim, na linha 19 é atualizado o total W de dados ainda a serem processados.

Após o envio de todos os dados de um certo período, na linha 22 o tempo de simulação é alterado conforme a duração do período estipulada pelo escalonador. Salienta-se que a duração de cada período é obtida para qualquer uma das técnicas apresentadas.

Algoritmo 6 SimStar: Simulador

Retorna: *makespan*

1: $makespan \leftarrow 0$
2: $d \leftarrow t \leftarrow 0$
3: **para** $i \leftarrow 1$ até n **faça**
4: $M_i \leftarrow 0$
5: $m_i \leftarrow 0$
6: **fim para**
7: **enquanto** $W \neq 0$ **faça**
8: Chame o escalonador passando W e a descrição do sistema atual como parâmetros. Guarde os valores da ordem de envio $order$, $\alpha_i, 1 \leq i \leq n$ e a duração ℓ do período retornados por ele.
9: **se** $d < t$ **então**
10: $d \leftarrow t$
11: **fim se**
12: **para** $j \leftarrow 1$ até n **faça**
13: $i \leftarrow order[j]$
14: **se** $alpha_i > 0$ **então**
15: $m_i \leftarrow \max\{d, m_i\} + g_i + G_i\alpha_i$
16: $d \leftarrow m_i$
17: $M_i \leftarrow \max\{m_i, M_i\} + w_i\alpha_i$
18: $makespan \leftarrow \max\{makespan, M_i\}$
19: $W \leftarrow W - \alpha_i$
20: **fim se**
21: **fim para**
22: $t \leftarrow t + \ell$
23: **fim enquanto**

4.2

Implementações das técnicas de escalonamento

Cada técnica foi implementada como uma subclasse da classe *scheduler*, herdando desta forma métodos para teste das decisões de escalonamento, bem como a medição de tempo de processamento. Cada escalonador herda também métodos que permitem a criação automática de um arquivo de saída que pode ser processado pelo visualizador de soluções descrito no Apêndice A.

Para a resolução dos problemas de programação linear e inteira foi utilizado o pacote ILOG CPLEX versão 8.0 configurado para realizar pré-processamento dos modelos e solucioná-los com ênfase em viabilidade de solução, seleção dos nodos conforme melhor limite e geração automática de cortes.

Tabela 4.1: Combinação de características de sistema testados

g_i	G_i	w_i	instâncias
LO	LO	LO	3
LO	LO	HI	3
LO	HI	LO	3
LO	HI	HI	3
HI	LO	LO	3
HI	LO	HI	3
HI	HI	LO	3
HI	HI	HI	3

4.3

Base de testes

Foram geradas 120 instâncias de *grids* com diferentes características. Cada instância é caracterizada pelo número de processadores e pela faixa de valores de suas características de processamento e comunicação. Foram criados grupos de *grids* com 10, 20, 40, 80 e 160 processadores. Cada grupo possui 24 instâncias divididas segundo segundo a faixa de valores das velocidades de processamento (w_i), do inverso das taxas de transmissão (G_i) e das latências (g_i), assim como mostrado na Tabela 4.1. São utilizadas duas faixas de valores, baixos (LO) e altos (HI). Valores baixos são gerados aleatoriamente entre 1 e 100, enquanto valores altos são gerados aleatoriamente entre 1000 e 100.000, formando-se desta maneira *grids* com características heterogêneas de processamento e comunicação. Para um melhor estudo dos resultados, foram executados testes com cargas de tamanho W igual a 100, 200, 400, 800, 1600 e 3200 para cada instância, o que gerou uma massa total de 720 casos teste (120 instâncias com seis tamanhos de carga distintos).

4.4

Técnicas de escalonamento em período único

Nesta seção inicialmente serão apresentados os resultados obtidos pela resolução dos modelos de programação linear inteira mista e suas relaxações lineares. Utilizando-se destes resultados, serão analisadas as soluções geradas pela heurística *HeuRet* e pelas buscas locais.

4.4.1

Modelos de programação linear inteira mista

Serão apresentados resultados referentes a dois modelos de programação, o Modelo 4 com limites melhorados (doravante referenciado como MIP1) e o Modelo aprimorado 5 com desigualdades válidas (doravante referenciado como MIP1E). Também serão analisados resultados obtidos para suas relaxações lineares, doravante referenciadas como MIP1R e MIP1ER respectivamente.

Relaxação linear

Solucionou-se MIP1R e MIP1ER para todos os casos teste com 10, 20 e 40 processadores, uma vez que o tempo de resolução de MIP1ER foi muito prolongado (mais de uma hora de processamento) para instâncias com mais de 40 processadores. Salienta-se porém que foi possível resolver MIP1R para todas as instância com tempos inferiores a uma hora.

Para cada caso teste foi verificada a diferença percentual do valor da solução obtida por MIP1ER e MIP1R (valor da solução de MIP1ER dividido pelo valor da solução de MIP1R menos um). Calculou-se então a média dessas diferenças percentuais para casos teste com o mesmo número de processadores. Com estes valores, é possível verificar no gráfico da Figura 4.1 o crescimento da diferença percentual dos valores das soluções de MIP1ER e MIP1R para instâncias com 10, 20 e 40 processadores. Mostra-se desta forma quanto em média os valores das soluções de MIP1ER são superiores aos valores das soluções de MIP1R. Conforme pode ser visto neste gráfico, com 10 processadores, a melhoria do limite inferior é de cerca de 3,6% em relação à MIP1R. A medida que o número de processadores aumenta, o ganho aumenta ainda mais, chegando a quase 10% com 40 processadores. Mostra-se desta forma que as desigualdades encontradas causam uma maior aproximação do modelo relaxado ao modelo inteiro, resultando em um limite inferior mais justo.

Salienta-se porém que a introdução das desigualdades em MIP1ER, além de melhorarem os resultados encontrados por MIP1R, causaram também um maior tempo de resolução do modelo. A fim de analisá-los por este prisma, obteve-se o tempo de resolução de MIP1R e MIP1ER para todas as instâncias com 10, 20 e 40 processadores. Calculou-se então o tempo médio de resolução de cada modelo relaxado para instâncias com o mesmo número de processadores,

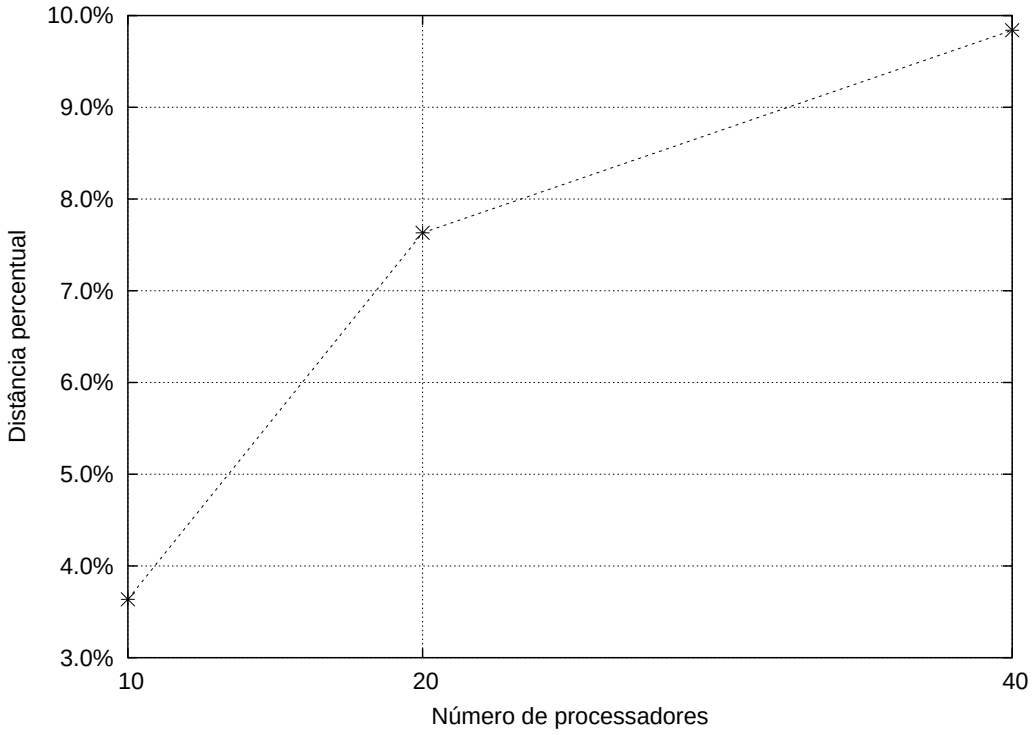

Figura 4.1: Diferença percentual dos valores das soluções de MIP1ER em relação aos valores das soluções de MIP1R

o que pode ser visto no gráfico da Figura 4.2. Nota-se que a medida que aumenta-se o número de processadores, o tempo de resolução de MIP1ER aumenta substancialmente, chegando a ficar cerca de quarenta vezes maior que o de resolução de MIP1R.

Resolução do problema inteiro

Como descrito no manual do CPLEX 8.0, o valor padrão do *gap* utilizado para resolução dos problemas inteiros é de 10^{-4}, que garante que o valor da solução obtida encontra-se a uma diferença máxima de 0,01% do valor da solução ótima. Nas Tabelas 4.2 e 4.3 são apresentados os tempos de resolução médios dos modelos inteiros. Cada célula representa a média de tempo de 18 casos teste (três instâncias de *grids* com características semelhantes testadas com seis cargas diferentes cada). As células não especificadas indicam que o tempo médio de execução superou uma hora de processamento. Pode-se notar nestas tabelas que o tempo de resolução depende não somente do número de processadores da instância, mas também depende muito das características do sistema. Nota-se também que os tempos de resolução de MIP1E são em geral bem maiores que os tempos de resolução de MIP1, o que leva à conclusão que

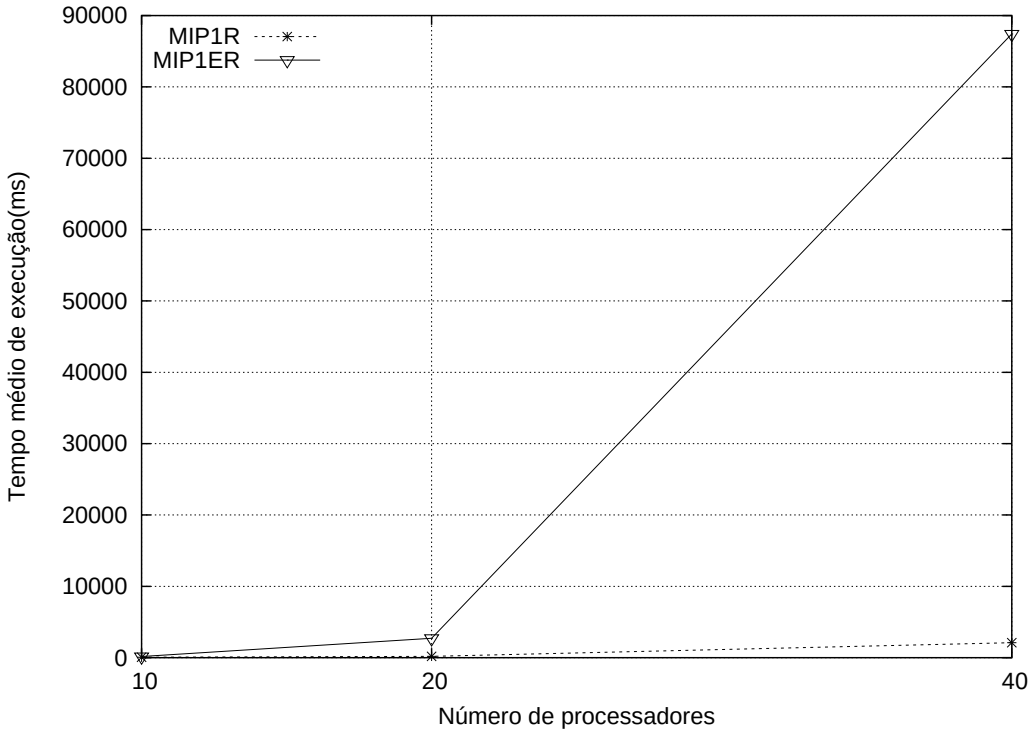

Figura 4.2: Tempo de resolução de MIP1R e MIP1ER

as desigualdades inseridas introduzem uma maior complexidade no modelo causando uma grande degradação do tempo de resolução. Salienta-se que, mesmo tendo sido possível a resolução de MIP1 para todas as instâncias com dez processadores, quando este número passa para 20, já encontra-se um grande número de instâncias cujo tempo de resolução é muito prolongado (superior a uma hora). Por fim, conclui-se que a utilização destes métodos exatos para o cálculo do escalonamento ótimo não é interessante do ponto de vista prático devido ao alto custo computacional associado. Por outro lado, os resultados obtidos para os problemas resolvidos são importantes parâmetros de avaliação de métodos não-exatos de escalonamento, permitindo avaliar a diferença entre os *makespans* das soluções propostas e os *makespans* ótimos.

4.4.2

Heurística $HeuRet$ e buscas locais

A partir dos resultados da heurística $HeuRet$, foram aplicados os algoritmos de busca local com melhoria iterativa (doravante referenciada como $H + MI$) e com descida rápida (doravante referenciada como $H + DR$). De todos os 720 casos teste, em apenas 17 foram obtidas soluções diferentes por

Tabela 4.2: Tempo médio (*segundos*) de resolução de MIP1 para diferentes sistemas

características			número de processadores				
w_i	g_i	G_i	10	20	40	80	160
LO	LO	LO	0,74	2,91	–	–	–
LO	LO	HI	0,13	0,27	0,60	2,55	24,70
LO	HI	LO	0,17	0,42	–	–	–
LO	HI	HI	0,14	0,27	1,16	4,23	21,07
HI	LO	LO	54,27	–	–	–	–
HI	LO	HI	0,23	–	–	–	–
HI	HI	LO	10,16	–	–	–	–
HI	HI	HI	0,40	1,77	–	–	–

Tabela 4.3: Tempo médio (*segundos*) de resolução de MIP1E para diferentes sistemas

características			número de processadores				
w_i	g_i	G_i	10	20	40	80	160
LO	LO	LO	1,77	34,87	–	–	–
LO	LO	HI	0,18	0,64	–	–	–
LO	HI	LO	0,23	1,01	–	–	–
LO	HI	HI	0,19	0,66	–	–	–
HI	LO	LO	1128,40	–	–	–	–
HI	LO	HI	0,41	–	–	–	–
HI	HI	LO	9,05	–	–	–	–
HI	HI	HI	0,84	–	–	–	–

$H + MI$ e $H + DR$. As diferenças encontradas foram ínfimas, atingindo uma diferença máxima de *makespan* de 0,03%. Para cada caso teste foi calculada a diferença percentual dos *makespans* obtidos por $H + MI$ e $H + DR$ em relação a $HeuRet$, ou seja, em quanto a busca local melhorou os resultados encontrados por $HeuRet$ em cada caso. Calculando-se então a média dessas diferenças percentuais para casos teste com o mesmo número de processadores, gerou-se o gráfico da Figura 4.3. Nota-se que as melhorias conseguidas por ambas as técnicas, além de muito similares (os gráficos praticamente se sobrepõem), são em média muito pequenas, levando à conclusão que a heurística $HeuRet$ encontrou soluções sempre muito próximas de ótimos locais. Apesar das melhorias introduzidas serem pequenas, somente com elas foi possível alcançar algumas soluções ótimas, como será descrito mais adiante.

Ao comparar-se o tempo médio de execução de cada técnica para casos teste com o mesmo número de processadores, obtém-se o gráfico da Figura 4.4. Nele é possível verificar que $H + MI$ tende a ser mais rápida que $H + DR$ para instâncias maiores (em média cerda de duas vezes mais rápida com 160 processadores). Enquanto as buscas locais consumiram um tempo de

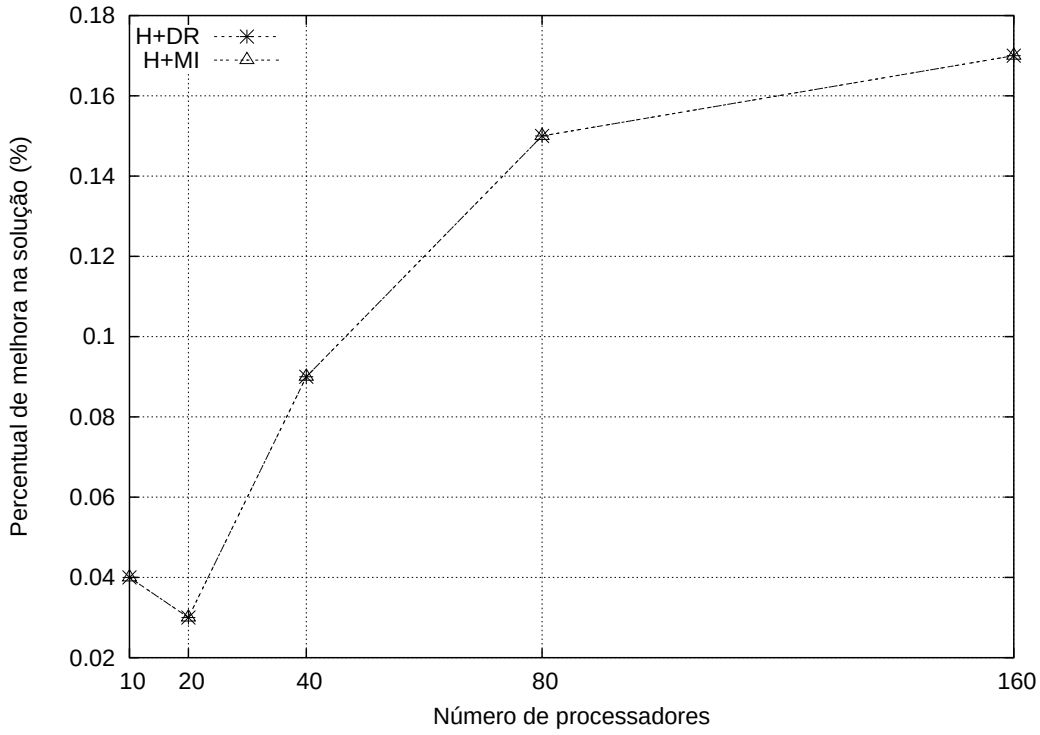

Figura 4.3: Melhoria percentual média conseguida pelas buscas locais a partir da heurística *HeuRet*.

processamento grande, a execução da heurística *HeuRet* em nenhum caso teste exigiu mais do que 40ms de processamento.

Para cada caso teste foi executada a heurística *HeuRet* e calculada a diferença percentual com relação ao valor da solução encontrada por MIP1R. Da mesma forma, também foram calculadas as diferenças percentuais dos valores das soluções de MIP1ER e dos *makespans* ótimos encontrados por MIP1 em relação aos valores das soluções de MIP1R. Utilizando-se das médias destas diferenças percentuais encontradas para instâncias com o mesmo número de processadores, elaborou-se o gráfico da Figura 4.5. Nele observa-se que os *makespans* obtidos pela heurística *HeuRet* na média nunca se distanciam mais do que 30% de MIP1R. Também pode ser visto que para instâncias de até 40 processadores, os *makespans* obtidos por *HeuRet* na média não diferem mais do que 10% dos valores das soluções de MIP1ER. Estas diferenças para as relaxações lineares implicam em diferenças iguais ou inferiores para os *makespans* ótimos, o que mostra que, mesmo para instâncias com muitos processadores, os resultados da heurística continuam sendo bons. Saliente-se também que das 160 soluções ótimas encontradas, apenas oito eram diferentes das soluções encontradas por *HeuRet*. Nessas oito, a diferença percentual para o *makespan* da solução ótima foi menor que 3,6% (graficamente, os pontos

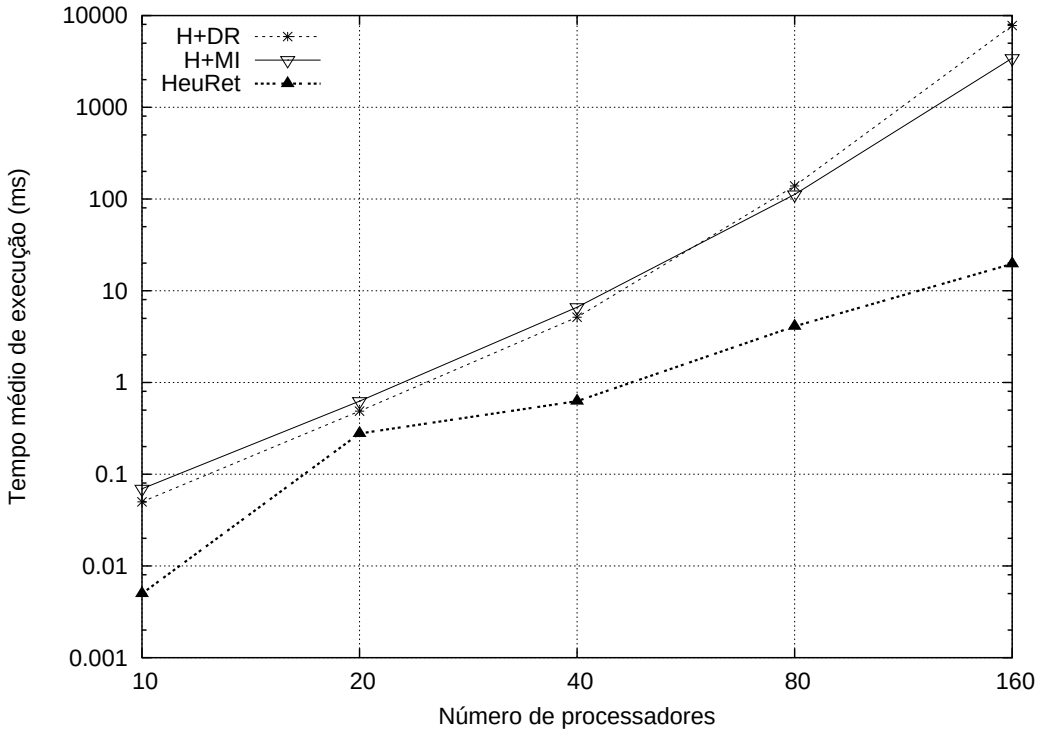

Figura 4.4: Tempo de execução da heurística *HeuRet* e das buscas locais.

referentes às soluções ótimas e às soluções de *HeuRet* se sobrepõem). Com a busca local de melhoria iterativa foi possível alcançar mais cinco soluções ótimas, restando apenas três casos teste onde não foi alcançada a solução ótima. Como não foi possível obter algumas soluções ótimas para sistemas com 20 ou mais processadores, no gráfico da Figura 4.5 são apresentados apenas os resultados obtidos para sistemas com 10 processadores.

4.5

Técnicas de escalonamento em múltiplos períodos

Nesta seção inicialmente serão apresentados os resultados obtidos pela resolução do Modelo 7 com número máximo de períodos. Serão também apresentados os resultados obtidos pela nova heurística *HeuMul* e, por fim, serão analisados os resultados obtidos pelas heurísticas de período fixo (Seção 3.2.1) e adaptativo (Seção 3.2.2).

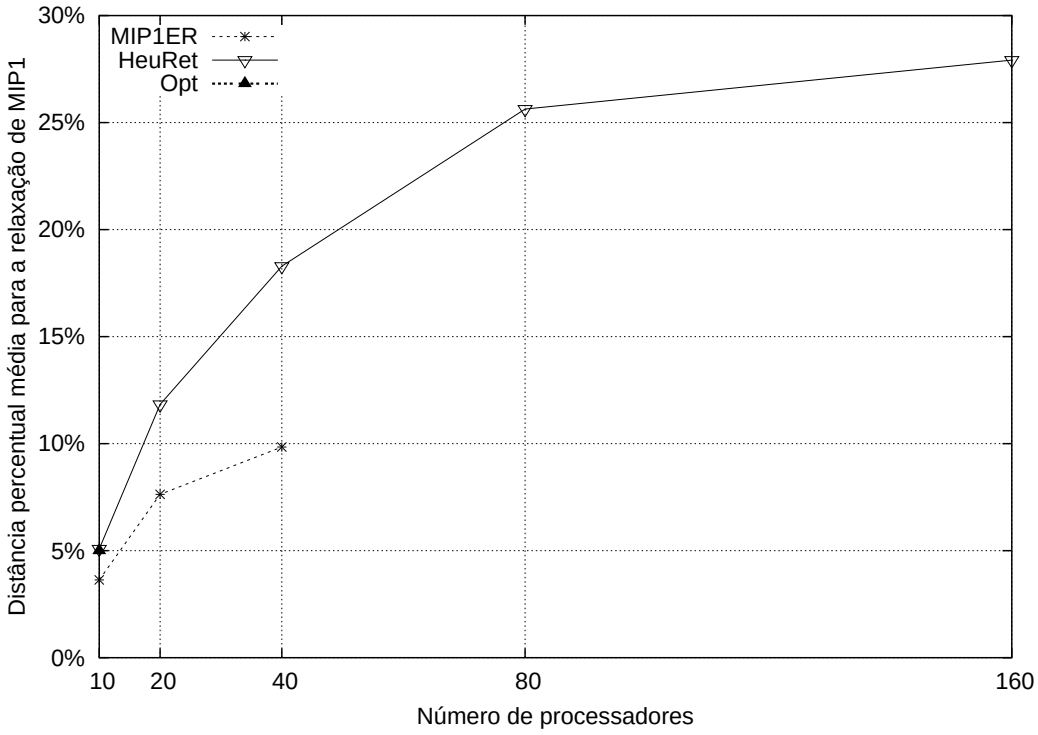

Figura 4.5: Diferença percentual dos *makespans* conseguidos por $HeuRet$, dos *makespans* ótimos (Opt) conseguidos por MIP1 e dos valores das soluções de MIP1ER em relação aos valores das soluções de MIP1R.

4.5.1

Modelos de programação linear inteira mista

Optou-se por analisar o Modelo 7 com número máximo de períodos (doravante chamado MIP2) pois, comparado aos demais modelos de escalonamento em múltiplos períodos, é o que permite encontrar melhores resultados. Devido à grande quantidade de variáveis que possui, bem como ao grande número de desigualdades, só foi possível resolver sua relaxação linear para algumas instâncias com dez processadores e algumas poucas com vinte. As demais exigiam tempo de processamento muito prolongado (acima de uma hora). Utilizando-se deste modelo foi possível encontrar a solução para apenas 14 instâncias com dez processadores, as demais exigiam tempo de processamento muito prolongado (acima de uma hora). Todos estes resultados serão apresentados na Seção 4.5.2 como base para análise das soluções obtidas pela Heurística $HeuMul$.

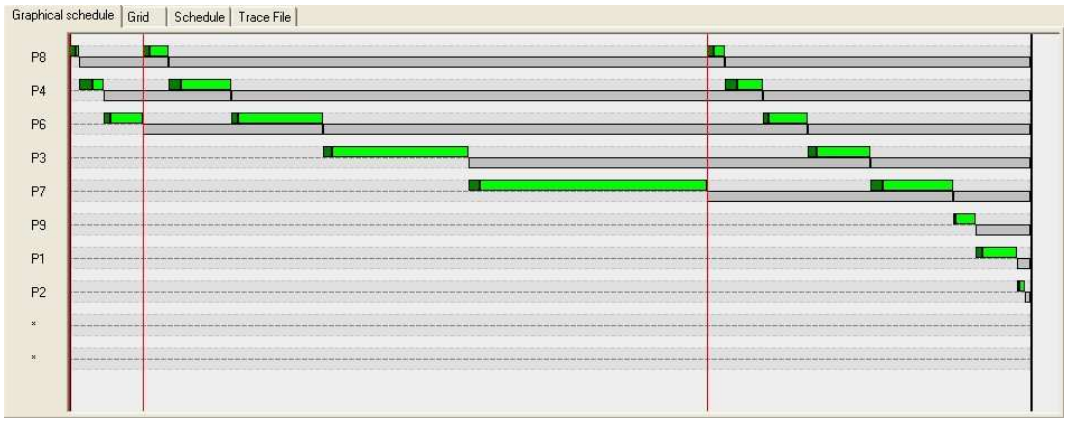

Figura 4.6: Resultado ótimo encontrado pelo modelo MIP2

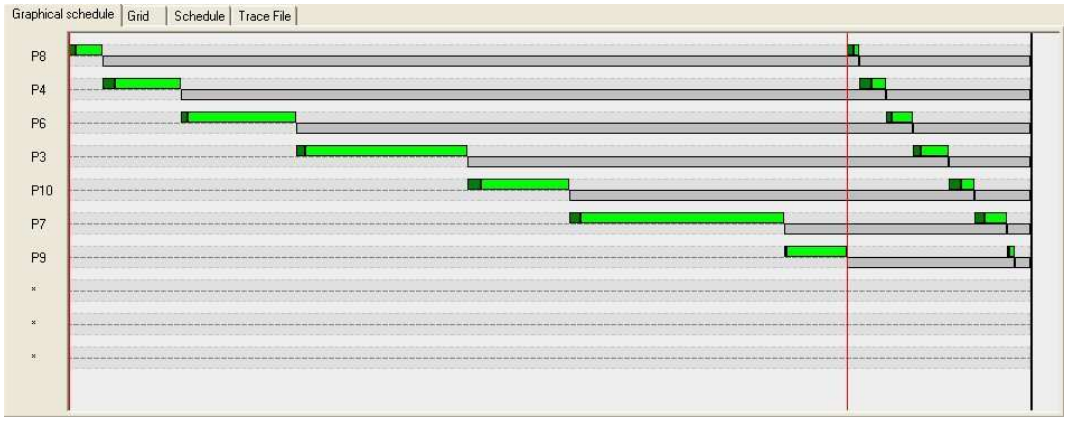

Figura 4.7: Resultado encontrado pela heurística $HeuMul$

4.5.2

Heurística $HeuMul$

Para a análise descrita neste trabalho, estipulou-se que a heurística $HeuMul$ utilizaria um número máximo $MaxP$ de períodos igual a cinco. Um estudo mais detalhado sobre o impacto da definição do número máximo de períodos não é abordado neste trabalho, sendo deixado para estudos futuros.

Foram analisados inicialmente os resultados da heurística $HeuMul$ com relação aos resultados do Modelo 7 com número máximo de períodos. Dos 14 casos teste nos quais encontrou-se a solução exata, em 12 eram utilizados apenas um período e os *makespans* eram iguais aos obtidos por $HeuMul$. Nas duas restantes, a solução do modelo resultou nos escalonamentos apresentados nas Figuras 4.6 e 4.8, que possuíam *makespans* de 1 a 3% menores que os escalonamentos encontrados por $HeuMul$ apresentados nas Figuras 4.7 e 4.9.

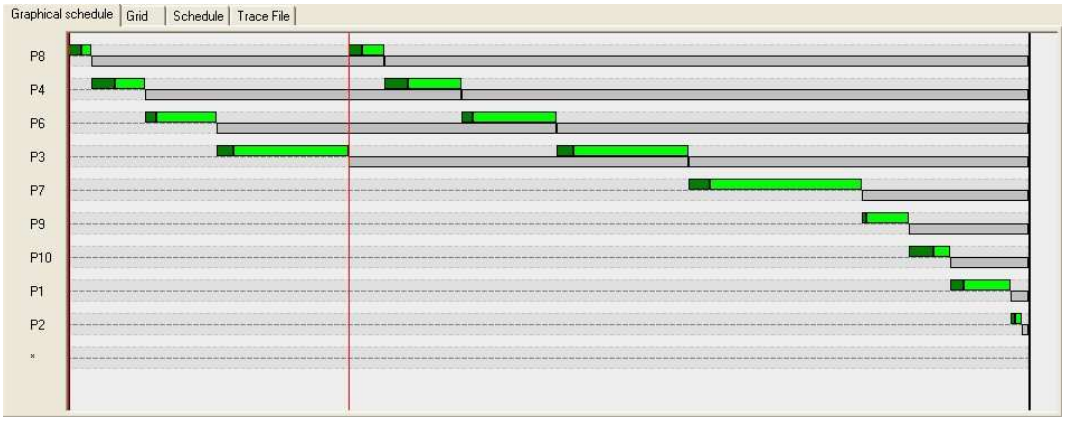

Figura 4.8: Resultado ótimo encontrado pelo modelo MIP2

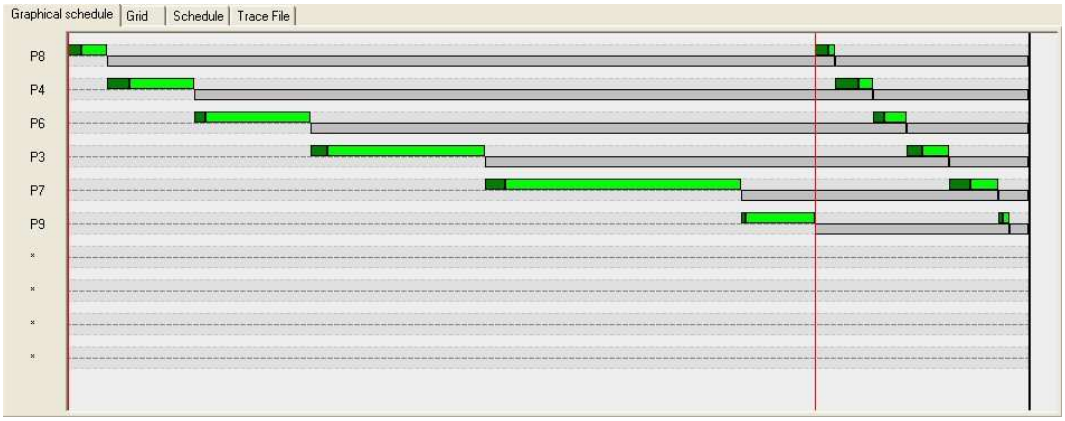

Figura 4.9: Resultado encontrado pela heurística *HeuMul*

Calculou-se, para todos os casos teste, a diferença percentual dos *makespans* obtidos por *HeuMul* em relação aos obtidos por *HeuRet*. No gráfico da Figura 4.10 pode ser vista a diferença percentual média encontrada para casos teste com o mesmo número de processadores. Como pode ser visto neste gráfico, a melhoria percentual conseguida utilizando-se *HeuMul* não parece apresentar uma relação direta com o número de processadores. Porém quando as médias são feitas agrupando-se os casos teste conforme a carga total processada, como no gráfico da Figura 4.11, pode-se notar uma relação direta entre o aumento de carga e a melhoria percentual do *makespan*. Ao analisar-se os resultados em função das características do sistema, como apresentado na Tabela 4.4, é possível verificar que as características do sistema exercem grande influência nos *makespans* obtidos por *HeuMul*. Pode ser verificado que os melhores resultados percentuais foram encontrados com casos teste com um número alto de processadores com latência de comunicação pequena em relação a w_i.

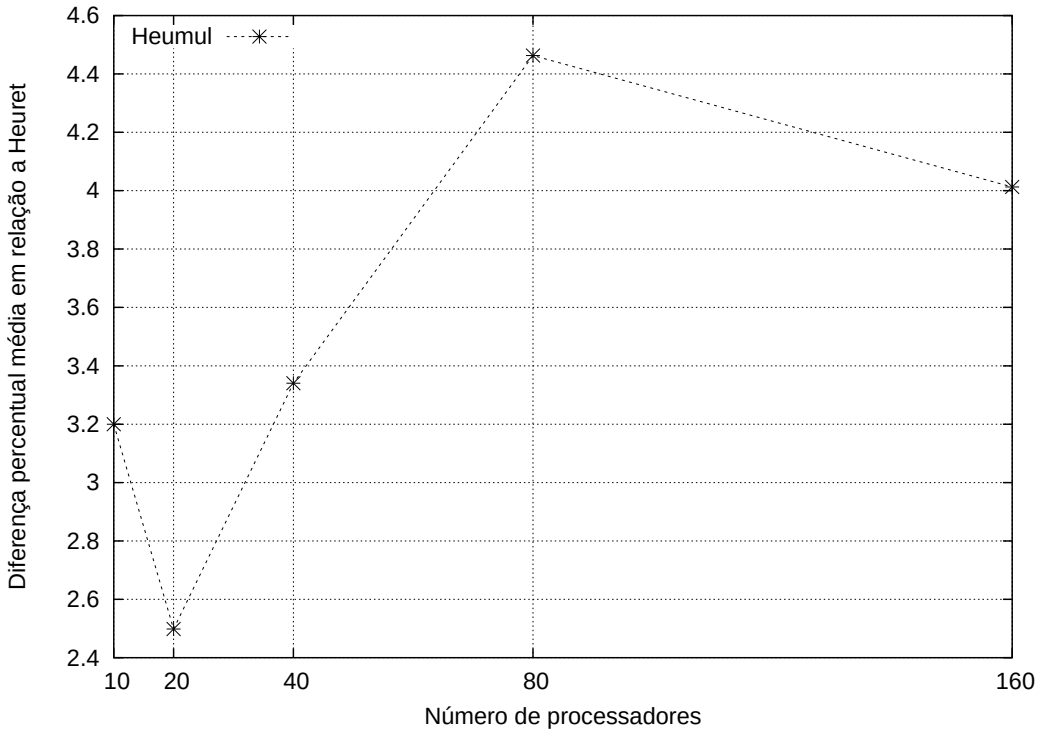

Figura 4.10: Diferença percentual dos *makespans* obtidos por $HeuMul$ em relação aos obtidos por $HeuRet$.

Além do *makespan* resultante da utilização das heurísticas, foi analisada também a quantidade média de processadores utilizados. Enquanto $HeuRet$ utilizou em média 16 processadores em todos os casos teste, $HeuMul$ utilizou 12, uma melhoria média de 25%. Acredita-se que esta melhoria pode acarretar melhores resultados quando do escalonamento de múltiplas tarefas divisíveis num mesmo sistema, o que não é tratado neste trabalho.

É importante notar porém que, enquanto $HeuRet$ exigiu em média cinco milisegundos de processamento (sem nunca ultrapassar 40 milisegundos), a heurística $HeuMul$ exigiu em média 12 segundos de processamento, chegando a consumir até 12 minutos em alguns casos. Esta grande diferença tem como motivo principal a forma de cálculo do escalonamento por parte de ambas as heurísticas. Enquanto $HeuRet$ a cada iteração se utiliza de um algoritmo de complexidade $O(n)$ para definir o escalonamento, $HeuMul$ a cada iteração utiliza-se de um algoritmo de complexidade $O(n^3)$. Além disso, $HeuMul$ realiza sempre $5 \times MelhorL^*$ iterações, enquanto que a heurística $HeuRet$, por possuir retroalimentação, realiza um número variável de iterações que nunca foi superior a quatro.

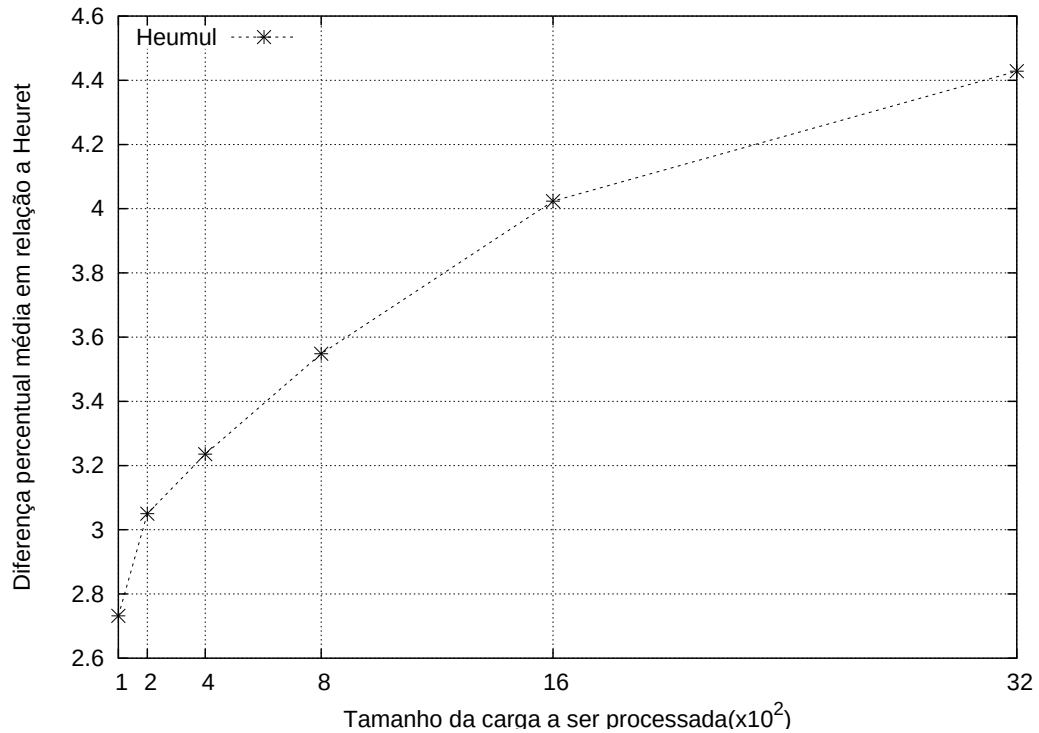

Figura 4.11: Diferença percentual dos *makespans* obtidos por $HeuMul$ em relação aos obtidos por $HeuRet$.

Tabela 4.4: Diferença percentual média dos *makespans* obtidos por $HeuMul$ em relação aos obtidos por $HeuRet$

sistema			número de processadores				
w_i	g_i	G_i	10	20	40	80	160
LO	LO	LO	4,8	4,4	3,5	6,9	4,5
LO	LO	HI	1,0	0,7	1,9	0,7	0,1
LO	HI	LO	3,6	1,3	2,5	1,9	1,3
LO	HI	HI	0,1	0,2	0,7	2,2	1,9
HI	LO	LO	1,9	1,7	4,0	7,6	9,1
HI	LO	HI	6,6	7,2	8,4	12,4	10,3
HI	HI	LO	0,4	0,3	0,1	0,1	0,7
HI	HI	HI	7,3	4,3	5,6	3,9	4,3

4.5.3

Heurísticas de período fixo e adaptativo

Ao analisar-se a média dos *makespans* obtidos pelas heurísticas de período fixo (heurística $HFix$, apresentada na Seção 3.2.1) e adaptativo (heurística $HAdapt$, apresentada na Seção 3.2.1) em todos os casos teste, verificou-se que obtiveram em média *makespans* 28 e 30 vezes superiores aos obtidos por $HeuRet$ respectivamente. Esta grande diferença é decorrente da forma de previsão de *makespan* utilizado por ambas as técnicas, onde latências de comunicação são desprezadas.

Calculou-se a média dos *makespans* gerados por $HFix$, $HAdapt$ e $HeuRet$ para casos teste com o mesmo número de processadores e com características semelhantes, permitindo a criação das Tabelas 4.5 e 4.6. Observou-se que a média dos *makespans* obtidos por $HFix$ e $HAdapt$ apresentaram valores quase sempre iguais, diferindo principalmente em sistemas com altos valores de w_i e baixos valores de g_i e G_i. Este comportamento é explicado pelo limite dado à duração dos períodos, não podendo ser reduzidos abaixo de determinado valor (conforme descrito na Seção 3.2.1). Como na maioria dos casos de teste esta limitação determinou a duração dos períodos, ambas as técnicas se comportaram da mesma forma.

Observa-se também que $HFix$ e $HAdapt$ obtêm resultados ruins em sistemas onde os valores das latências de comunicação são elevados. Pode-se citar como exemplo os sistemas com 20 processadores com baixos valores de G_i e w_i e altos valores de g_i, onde constatou-se que a média dos *makespans* obtidos por $HFix$ e $HAdapt$ foram ambas 2.436.315,33, enquanto que $HeuRet$ alcançou uma média de 21.460,67. Por outro lado, bons resultados são encontrados em sistemas com baixos valores de g_i e altos valores de G_i. Pode-se citar como exemplo, os sistemas com 80 processadores com altos valores de G_i e w_i e baixos valores de g_i, onde constatou-se que a média dos *makespans* obtidos por $HFix$ e $HAdapt$ foram ambas 4.362.852,39, enquanto que $HeuRet$ alcançou uma média de 5.117.465,72.

Nota-se também que os *makespans* obtidos por $HFix$ e $HAdapt$ em relação aos obtidos por $HeuRet$ pioram em média conforme o número de processadores aumenta. Por exemplo, nas instâncias com baixos valores de g_i, G_i e w_i, com dez processadores o *makespan* médio obtido por $HFix$ foi cerca de duas vezes maior que o *makespan* médio obtido por $HeuRet$. Nas instâncias com as mesmas características porém com 160 processadores, o

makespan médio de $HFix$ foi cerca de oito vezes maior que o *makespan* médio de $HeuRet$. Este comportamento pode ser explicado pelo fato das previsões de *makespan* considerarem a utilização de todos os processadores desprezando-se as latências. No trabalho [3] foram apresentados resultados para sistemas sem latências (ou seja, g_i muito inferior a G_i), bem como para sistemas com latências utilizando-se poucos processadores, o que favorece o escalonamento feito pelas técnicas $HFix$ e $HAdapt$.

Tabela 4.5: Média dos *makespans* obtidos por *HFix* e, entre parênteses, média dos *makespans* obtidos por *HeuRet* para instâncias com 10, 20, 40, 80 e 160 processadores

sistema			número de processadores	
w_i	g_i	G_i	10	20
LO	LO	LO	36.918,87 (22.080,91)	31.484,69 (12.155,64)
LO	LO	HI	8.405.108,67 (8.434.309,92)	5.957.496,50 (5.986.780,09)
LO	HI	LO	4.743.049,33 (70.078,34)	1.944.246,33 (57.863,96)
LO	HI	HI	9.733.781,33 (5.668.659,35)	14.284.337,83 (8.155.164,24)
HI	LO	LO	2.376.573,16 (2.396.238,08)	1.949.368,59 (1.808.436,45)
HI	LO	HI	19.034.637,13 (21.562.724,80)	10.960.999,75 (12.457.967,70)
HI	HI	LO	59.607.026,98 (2.793.663,15)	37.549.121,75 (1.064.842,03)
HI	HI	HI	24.036.784,70 (21.242.853,61)	35.361.732,47 (13.503.633,73)
w_i	g_i	G_i	40	80
LO	LO	LO	25.617,99 (6.916,61)	39.628,95 (5.873,72)
LO	LO	HI	2.102.609,00 (2.127.701,64)	2.102.470,67 (2.115.052,93)
LO	HI	LO	2.436.315,33 (21.460,67)	8.203.108,00 (36.532,92)
LO	HI	HI	7.500.668,50 (2.870.850,49)	4.900.973,00 (2.184.346,95)
HI	LO	LO	1.760.856,20 (534.382,67)	2.713.009,43 (296.929,64)
HI	LO	HI	9.108.630,76 (10.498.666,85)	4.362.852,39 (5.117.465,72)
HI	HI	LO	47.211.703,14 (801.633,06)	22.193.585,13 (569.660,04)
HI	HI	HI	40.965.915,44 (8.474.013,36)	44.373.268,37 (7.388.954,58)
w_i	g_i	G_i	160	
LO	LO	LO	33.232,49 (4.891,01)	
LO	LO	HI	1.055.486,00 (1.050.514,77)	
LO	HI	LO	5.935.349,33 (23.756,41)	
LO	HI	HI	6.317.051,67 (1.085.022,53)	
HI	LO	LO	6.148.635,51 (146.919,33)	
HI	LO	HI	4.105.022,26 (4.806.512,84)	
HI	HI	LO	68.440.839,45 (383.940,54)	
HI	HI	HI	81.962.236,71 (4.254.029,77)	

Tabela 4.6: Média dos *makespans* obtidos por *HAdapt* e, entre parênteses, média dos *makespans* obtidos por *HeuRet* para instâncias com 10, 20, 40, 80 e 160 processadores

sistema			número de processadores	
w_i	g_i	G_i	10	20
LO	LO	LO	40.149,86 (22.080,91)	34.692,99 (12.155,64)
LO	LO	HI	8.405.108,67 (8.434.309,92)	5.957.496,50 (5.986.780,09)
LO	HI	LO	4.743.049,33 (70.078,34)	1.944.246,33 (57.863,96)
LO	HI	HI	9.733.781,33 (5.668.659,35)	14.284.337,83 (8.155.164,24)
HI	LO	LO	2.500.436,57 (2.396.238,08)	3.235.432,37 (1.808.436,45)
HI	LO	HI	19.034.637,13 (21.562.724,80)	10.960.999,75 (12.457.967,70)
HI	HI	LO	59.607.026,98 (2.793.663,15)	37.549.121,75 (1.064.842,03)
HI	HI	HI	24.036.784,70 (21.242.853,61)	35.361.732,47 (13.503.633,73)
w_i	g_i	G_i	40	80
LO	LO	LO	25.617,99 (6.916,61)	39.628,95 (5.873,72)
LO	LO	HI	2.102.609,00 (2.127.701,64)	2.102.470,67 (2.115.052,93)
LO	HI	LO	2.436.315,33 (21.460,67)	8.203.108,00 (36.532,92)
LO	HI	HI	7.500.668,50 (2.870.850,49)	4.900.973,00 (2.184.346,95)
HI	LO	LO	4.187.978,10 (534.382,67)	6.538.203,76 (296.929,64)
HI	LO	HI	9.108.630,76 (10.498.666,85)	4.362.852,39 (5.117.465,72)
HI	HI	LO	47.211.703,14 (801.633,06)	22.193.585,13 (569.660,04)
HI	HI	HI	40.965.915,44 (8.474.013,36)	44.373.268,37 (7.388.954,58)
w_i	g_i	G_i	160	
LO	LO	LO	33.232,49 (4.891,01)	
LO	LO	HI	1.055.486,00 (1.050.514,77)	
LO	HI	LO	5.935.349,33 (23.756,41)	
LO	HI	HI	6.317.051,67 (1.085.022,53)	
HI	LO	LO	13.735.077,58 (146.919,33)	
HI	LO	HI	4.105.022,26 (4.806.512,84)	
HI	HI	LO	68.440.839,45 (383.940,54)	
HI	HI	HI	81.962.236,71 (4.254.029,77)	