Escalonamento de Tarefas com Restrições de Precedência e Custos de Execução e Comunicação Unitários

Eliezer Tomé de Paula Neto
Departamento de Computação – Universidade Federal do Ceará
Fortaleza – CE – Brazil
eliezer@lia.ufc.br

Manoel Bezerra Campêlo Neto
Departamento de Estatística e Matemática Aplicada – Universidade Federal do Ceará
Fortaleza – CE – Brazil
mcampelo@lia.ufc.br

Carlos Diego Rodrigues
Departamento de Estatística e Matemática Aplicada – Universidade Federal do Ceará
Fortaleza – CE – Brazil
diego@lia.ufc.br

Ricardo Cordeiro Corrêa
Departamento de Computação – Universidade Federal do Ceará
Fortaleza – CE – Brazil
correa@lia.ufc.br

RESUMO

Tratamos o problema de escalonamento de tarefas em um número limitado de processadores sob restrições de precedência e comunicação. Supomos que cada tarefa possui tempo de execução unitário, sua execução está condicionada à finalização de suas predecessoras e, caso duas tarefas dependentes sejam executadas em processadores diferentes, temos um tempo, também unitário, de comunicação. Este problema é NP-Difícil. Propomos uma heurística para o problema, um melhor limite inferior baseado em programação dinâmica e critérios de ramificação para um método de *Branch-and-Bound*. Apresentamos resultados computacionais com instâncias da literatura.
PALAVRAS CHAVE. Escalonamento de tarefas, Programação Dinâmica, Branch-and-bound, Programação Matemática.

ABSTRACT

We approach the problem of scheduling tasks in a bounded number of processors under precedence and communication constraints. We assume that each task has unit execution time, that its execution can only occur after its predecessors have ended and that, if two dependent tasks run on different processors, an unit communication cost is incurred. This is an NP-hard problem. We propose a heuristic for the problem, a better lower bound based on dynamic programming and branching rules for a branch-and-bound method. We report computacional results with test instances from the literature.
KEYWORDS. Task Scheduling, Dynamic Programming, Branch-and-bound, Mathematical Programming.

1. Introdução

Problemas de escalonamento ocupam, sem dúvida, um local de destaque no conjunto de problemas clássicos de otimização. Isto se deve sobretudo a suas aplicações diretas e extensas. Na indústria, por exemplo, onde o tempo e os custos de produção são determinantes para a obtenção de maiores lucros, existem grandes aplicações para problemas de escalonamento em suas diversas variações.

Apesar de abordarmos, neste trabalho, um problema de escalonamento restrito, é evidente a importância de seu estudo, sobretudo pela sua estreita relação com processamento paralelo, um paradigma computacional cada vez mais utilizado e que, em comparação ao paradigma sequencial, pode ser consideralvelmente mais eficiente em alguns casos. Nosso objeto de estudo é uma das bases para este paradigma. Funcionando como suporte básico para generalizações deste problema, as quais se aproximam propriamente da realidade. Consideramos o Problema de Escalonamento de Tarefas com Restrições de Precedência, Custos de Execução e Comunicação Unitários.

Genericamente, o problema consiste em agendar um conjunto de tarefa (possivelmente originárias do particionamento de um programa paralelo) em um conjunto de processadores disponíveis. Adotam-se as seguintes suposições: cada tarefa exige apenas uma única operação, que pode ser feita por qualquer máquina. Assim, basta que uma tarefa seja delegada a uma máquina. Todas as máquinas possuem as mesmas características de eficiência. Supomos que não há preempção e que possuímos em tempo de compilação todas as informações necessárias ao escalonamento, como precedência, tempos de execução e número de processadores (escalonamento estático). Os processadores não compartilham memória, a comunicação entre eles é feita por passagem de mensagens e não existem limitações nesse sentido, ou seja, um número arbitrário de mensagens pode ser trocado simultaneamente na rede.

Devemos indicar um agendamento para a execução das tarefas no conjunto de processadores, obedecendo duas restrições. Uma tarefa i só pode ser executada após suas antecessoras terem finalizado suas execuções e ainda transmitido os resultados de seus processamentos para o processador que executará i, respeitando os tempos unitários de execução e comunicação entre os processadores das tarefas. O tempo de comunicação dos resultados entre duas tarefas i e j é considerado zero, quando a antecessora j é executada no mesmo processador que executará i. O objetivo do problema é escalonar as tarefas no conjunto de processadores, respeitando as restrições impostas e minimizando o *makespan*, ou seja, minimizando o tempo total de execução do conjunto de tarefas. Este problema é NP-Difícil (Ullman, 1975).

Na literatura são conhecidos alguns métodos heurísticos (por exemplo, (Cheng and Sin, 1990; Kwok and Ahmad, 1999)) e também algoritmos aproximativos (Hanen and Munier, 2001), porém há ainda carência de bons métodos exatos de resolução do problema. Os métodos exatos existentes não se mostram eficientes para determinados conjuntos de instâncias. Existem ainda trabalhos que tratam de versões mais genéricas, porém não temos como objetivo, tratá-las. Por exemplo, em (Davidović et al., 2007) e (Coll et al., 2006), onde os tempos de processamento e comunicação não são necessariamente unitários. Neste trabalho, temos como objetivo diminuir o tempo da resolução exata do problema, que tem se mostrado impraticável até mesmo para instâncias com um número reduzido de tarefas. Apresentaremos formas de incrementar um método de resolução exata, baseado em uma formulação matemática existente na literatura (Campêlo et al., 2001). Para isso, mostramos uma generalização do limite inferior apresentado em (Campêlo et al., 2002). Este, ao

nosso conhecimento, apresentou o melhor limite inferior para o problema com tempos de execução e comunicação unitários.

2. Definições e notação

Podemos definir o Problema de Escalonamento de Tarefas com Restrições de Precedência, Custos de Execução e Comunicação Unitários com base em três elementos: um conjunto de tarefas $N = \{1, \ldots, n\}$, uma relação binária $\prec$ que define uma ordem parcial em N e um conjunto $Q = \{1, \ldots, m\}$ de processadores nos quais as tarefas devem ser executadas.

Comumente a ordem parcial $\prec$ é representada por um grafo acíclico e direcionado $G = (N, \prec)$ ou, de forma simplificada, pelo grafo $R = (N, \vdash)$, onde $\vdash$ é a redução transitiva da ordem parcial $\prec$. Escrever $i \prec j$ ($i \vdash j$) implica que $(i, j) \in \prec$ ($(i, j) \in \vdash$) e indica que a tarefa j necessita dos resultados de i para começar sua execução. Caso isso não ocorra escrevemos $i \not\prec j$. Se $i \vdash j$, i é uma predecessora imediata de j e j é uma sucessora imediata de i. Observe que se $i \not\prec j$ e $j \not\prec i$, as duas tarefas são incomparáveis, implicando que uma pode ser escalonada independentemente da outra. Escrevemos $i \parallel j$, quando isso ocorre. Caso $j \not\prec i$ ($i \not\prec j$) para todo $j \in N$, então i é uma tarefa minimal (maximal) de $\prec$. Por simplicidade de apresentação, consideramos que 1 e n são as únicas tarefas minimal e maximal, respectivamente, de $\prec$.

Para $N' \subseteq N$, $\prec_{N'}$ é o subconjunto de $\prec$ restrito a pares de elementos pertencentes a N'; esta relação também é transitiva, não-reflexiva e antissimétrica. Se $\prec_{N'}$ não contém tarefas incomparáveis, então a relação é denominada uma *cadeia* de $\prec$. Uma *extensão* $\prec'$ de $\prec$ é uma relação transitiva, não-reflexiva e antissimétrica em N tal que $\prec \subseteq \prec'$. Defina *anticadeia* de $\prec$ como um conjunto de tarefas incomparáveis duas a duas e a *largura* de $\prec$, denotada $\omega(\prec)$, como a cardinalidade da maior anticadeia de $\prec$. Dados $i, j \in N$, defina $N_{i,j}$ o conjunto de tarefas que são simultaneamente sucessoras de i e antecessoras de j, incluindo i e j. A *rede* $G_{i,j}$ é o subgrafo de G induzido por $N_{i,j}$. Por simplicidade, a relação $\prec_{N_{i,j}}$ será denotada por $\prec_{i,j}$.

Definimos o agendamento de uma tarefa $i \in N$ pelo par (x_i, p_i), onde $x_i \in \mathbb{N}$ é o tempo de início da tarefa e $p_i \in Q$ o processador em que ela será executada. De forma geral, um escalonamento é definido pelo par (X, P), onde $X = \{x_1, \ldots, x_n\}$ e $P = \{p_1, \ldots, p_n\}$. Para que um escalonamento (X, P) seja viável para o problema, deve obedecer as restrições estruturais, derivadas do grafo de tarefas, e de recurso, derivadas do número de processadores. Dividiremos as restrições de acordo com o relacionamento entre duas tarefas i e j, com $i, j \in N$.

Suponha que a tarefa j depende da tarefa i, portanto $i \prec j$. Claramente, $x_j > x_i$. Se as duas tarefas são executadas no mesmo processador, a diferença dos tempos de início das duas deve ser pelo menos um, pois este é exatamente o tempo de processamento da tarefa i. Caso sejam executadas em processadores diferentes, o tempo de execução deve ser acrescido do tempo de comunicação (unitário), portanto a diferença entre os tempos de início é pelo menos dois.

Se duas tarefas i, j são concorrentes, apesar de elas poderem ser executadas de forma independente, se não existem processadores disponíveis esta execução simultânea não poderá acontecer. Ou seja, mesmo que as tarefas sejam independentes os recursos podem não ser suficientes para que elas sejam efetivamente executadas concorrentemente. Portanto, se $i \parallel j$, temos dois casos: se $p_i \neq p_j$, os tempos de início são quaisquer, senão ($p_i = p_j$) temos que criar uma relação de precedência artificial entre as tarefas, fazendo com que $x_i \geq x_j + 1$ ou $x_j \geq x_i + 1$.

Em resumo, o escalonamento (X, P) é viável se satisfaz as seguintes restrições para qualquer par de tarefa $i, j \in N$:

- Para $(i \prec j)$: $x_j \geq x_i + 1$, mas se $p_i \neq p_j$, então $x_j \geq x_i + 2$;
- Para $(i \parallel j)$: se $p_i = p_j$, então $x_j \geq x_i + 1$ ou $x_i \geq x_j + 1$; se $p_i \neq p_j$, então os tempos de início são quaisquer;

3. Solução viável heurística

Apresentaremos agora uma heurística gulosa para o problema, baseada no caminho crítico de cada tarefa. Este tipo de abordagem é recorrente em problemas de escalonamento. Em particular, essa heurística já foi usada nos experimentos computacionais realizados em (Campêlo et al., 2002), porém agora a descrevemos em maiores detalhes. Definimos *caminho crítico de uma tarefa* como a maior distância entre ela e a tarefa maximal. A cada iteração, a heurística CPMISF *(Critical path, most immediate successors first)* agenda uma tarefa cujos predecessores já foram agendados, respeitando os custos de comunicação e dando preferência àquela tarefa com maior caminho crítico. Adicionalmente, se as duas tarefas possuem caminhos críticos de mesmo tamanho, a heurística fará opção pela tarefa que possui menos sucessores imediatos.

Algoritmo 1: Heurística CPMISF *(Critical path, most immediate successors first)*

Chamada: CPMISF(i, j, m)
Entrada: Tarefas: i, j; Número de processadores: m
Resultado: Solução viável: (X,P) e *makespan*

```
1  heapTask.add(i);
2  para cada k ∈ N faça
3      npred[k] ← |{z|z ≺ij k}| ;
4  para cada q ∈ Q faça
5      heapProc.add(q); tproc[q] ← 0;
6  enquanto heapTask ≠ ∅ faça
7      elected ← heapTask[0];
8      heapTask.remove(elected);
9      sortHeap(heapTask);
10     tmaxpred ← 0; nmaxpred ← 0;
11     para cada k : k ≺ij elected faça
12         se (taskTime[k] == tmaxpred) então
13             nmaxpred ← nmaxpred + 1; procpred ← proc[k];
14         se (taskTime[k] > tmaxpred) então
15             tmaxpred ← taskTime[k]; nmaxpred ← 1; procpred ← proc[k];
16     se (nmaxpred == 0) então
17         taskTime[elected] ← tproc[heapProc[0]]; proc[elected] ← heapProc[0];
18     senão se (nmaxpred == 1) e (tproc[procpred] ≤ tmaxpred + 1) então
19         taskTime[elected] ← tmaxpred + 1; proc[elected] ← procpred;
20     senão
21         se (tproc[heapProc[0]] > tmaxpred + 2) então
22             taskTime[elected] ← tproc[heapProc[0]];
23         senão
24             taskTime[elected] ← tmaxpred + 2;
25         proc[elected] ← heapProc[0];
26     tproc[proc[elected]] ← taskTime[elected] + 1 ;
27     sortHeap(heapProc);
28     para cada k : elected ≺ij k faça
29         npred[k] ← npred[k] − 1;
30         se (npred[k] == 0) então
31             heapTask.add(k);
32             sortHeap(heapTask);
33 retorne ((taskTime, proc), max_{q∈Q}{tproc[q]})
```

A ideia central da heurística é organizar as tarefas em um heap sob os critérios mencionados anteriormente (maior caminho crítico e menor número de sucessores) e a cada passo selecionar uma delas para ser executada. O processador em que ela será executada depende do tempo de término de suas antecessoras, especificamente das predecessoras que executam no tempo mais tarde. Os processadores também são organizados em um heap, de acordo com o tempo de término da última tarefa alocada a ele, de forma crescente. Portanto, preferencialmente são escolhidos processadores que possuem menor tempo de término da última tarefa executada nele. Os critérios de escolha do processador podem ser facilmente observados no Algoritmo 1.

A entrada do algoritmo são os nós $i, j \in N$, que induzem o subgrafo $G_{i,j}$ do qual devemos calcular o *makespan* ótimo, e ainda o número de processadores, m. Nas linhas $1-3$, adicionamos ao heap de tarefas a tarefa minimal i em $G_{i,j}$, pois inicialmente apenas ela pode ser executada, e ainda calculamos o número de predecessores de cada tarefa no subgrafo. A variável $tproc[q]$, para $q \in Q$, indica o tempo de término da última tarefa alocada a q até o momento. Portanto, inicialmente (linhas $4-6$) adicionamos cada processador ao heap de processadores e zeramos a variável $tproc$ associada.

No laço principal da heurística, entre as linhas $8-19$, elegemos a tarefa que atende os critérios mencionados e determinamos seu predecessor que executa no tempo mais tarde (*tmaxpred*). O número de predecessores (*nmaxpred*) agendados nesse tempo máximo é também calculado. Se não existirem predecessores, alocamos a tarefa ao processador obtido do heap de processadores. Se apenas um predecessor é executado no instante máximo, alocamos a tarefa eleita ao mesmo processador de seu antecessor. Caso contrário, a tarefa eleita será alocada ao processador obtido do heap de processadores, tendo o cuidado de adequar seu tempo de início de acordo com seus predecessores e ainda a variável $tproc$ do processador ao qual ela foi destinada (linhas $20-33$).

Como agendamos a tarefa eleita, é possível que alguns de seus sucessores agora estejam livres para serem executados. Esta verificação é feita nas linhas $34-38$. Se existir um sucessor que possa ser executado ele será adicionado ao heap de tarefas. O *makespan* $UB_{i,j}$ é obtido pelo valor máximo do instante de término das tarefas, ou seja, pelo valor máximo das variáveis $tproc[q]$, $q \in Q$. Note que $UB_{1,n}$ é um limite superior para para o makespan ótimo do problema em G.

4. Limite inferior

Nesta seção mostraremos um método para a obtenção de um limite inferior para o problema abordado. A base para esse método é a decomposição do grafo de tarefas em redes (Campêlo et al., 2002). Chrétienne desenvolveu método similar, também baseado em decomposição do grafo de tarefas e programação dinâmica, quando a relação de precedência define uma árvore e número de processadores é ilimitado (Chretienne, 1989). Nosso limite inferior melhora aquele obtido em (Campêlo et al., 2002). Para a sua apresentação, é necessária a introdução de novos conceitos e notações específicas.

4.1. Decomposição em cadeias

Um *corte-cadeia* C de $G_{i,j}$ é uma cadeia da relação $\prec_{i,j}$, tal que se $l \in C$ e $k \parallel l$ então $k \notin N_{i,j}$. Veja que a cadeia C é um corte de vértice em $G_{i,j}$, portanto cada tarefa desta cadeia ou é predecessora ou é sucessora de todas as outras tarefas de $N_{i,j}$. Um *pedaço* de $G_{i,j}$ relativo a C é uma rede induzida pelas tarefas de uma componente B de $G_{i,j} - C$ e ainda uma tarefa s pertencente ao corte-cadeia C. Se as tarefas de C são sucessoras de todas as de B, então s é minimal de C, caso contrário s será maximal em C.

Podemos decompor o grafo $G_{i,j}$ em pedaços relativos a cortes-cadeia de $G_{i,j}$. Seja $\mathcal{C}$ um conjunto de cadeias disjuntas de G e $\mathcal{C}_{i,j}$, com $i \prec j$ e $i, j \in N$, um subconjunto maximal de $\mathcal{C}$ contendo apenas cortes-cadeia de $G_{i,j}$. Um pedaço de $G_{i,j}$ relativo a $\mathcal{C}_{i,j}$ é um pedaço de $G_{i,j}$ relativo a algum corte-cadeia de $\mathcal{C}_{i,j}$.

Uma decomposição em cadeias do grafo $G = (N, \prec)$, relativa a $\mathcal{C}$, é um grafo direcionado, com coloração nos arcos, $D = (V, E)$. A definição dos conjuntos de vértices e arestas é dada recursivamente da seguinte forma:

1. Definimos trivialmente que $[1, n] \in V$ é a raiz de D.
2. Seja $[k, l]$ um par de tarefas, com $k \neq l$, e $[i, j] \in V$. Existem três possibilidades para que este par pertença a V e para que $a = ([i, j], [k, l]) \in E$:
   - $\mathcal{C}_{i,j} \neq \emptyset$ **e** $G_{k,l}$ **é um pedaço de** $G_{i,j}$**:** Damos a cor **branca** ao arco a.
   - $\mathcal{C}_{i,j} = \emptyset$ **e** $(i = k$ **e** $l \vdash j)$**:** O arco a receberá a cor **azul**.
   - $\mathcal{C}_{i,j} = \emptyset$ **e** $(j = l$ **e** $i \vdash k)$**:** Neste caso, o arco a receberá a cor **vermelha**.

Um vértice $[i, j]$ de D é denominado folha se possui vizinhança positiva vazia. Veja que as folhas do grafo são cadeias e, portanto, o tempo mínimo de execução das folhas é o tamanho das cadeias.

4.2. Limite inferior a partir de D

Descreveremos agora como o grafo de decomposição D de G pode ser usado para a obtenção de um limite inferior para o escalonamento de $(N, \prec, Q)$. Seja a ordem $\{v_1 = [i_1, j_1], \ldots, v_v = [i_{|V|}, j_{|V|}]\}$, com $v_i \in V$, obtida pela inversão de uma ordem topológica de V. O grafo D não possui ciclos orientados e, portanto, tal ordem existe, podendo ser obtida por uma busca em profundidade. Através de um processo de programação dinâmica o limite inferior para o par $[i, j]$ (ou melhor, para o *makespan* de $(N_{i,j}, \prec_{i,j}, Q)$) pode ser obtido em função do limite dos pares de sua vizinhança positiva em D, que pela ordem anterior já devem ter sido calculados ao serem requisitados.

Vamos denotar por $LB_{i,j} + 1$ o limite inferior para $(N_{i,j}, \prec_{i,j}, Q)$, de modo que $LB_{i,j}$ será a diferença mínima entre os tempos de início das tarefas j e i em $(N, \prec, Q)$. O cálculo de $LB_{i,j}$ segue a definição de D acima. O caso mais simples ocorre quando $G_{i,j}$ é uma cadeia, ou seja, quando $[i, j]$ é uma folha do grafo D. O limite inferior é o tamanho da própria cadeia, portanto definiremos que $LB_{i,j}^{folha} = |N_{i,j}| - 1$.

Considere agora um vértice $[i, j] \in V$ que não é uma folha do grafo D. Utilizaremos a classificação apresentada anteriormente que se baseia no conjunto de cortes-cadeia, $\mathcal{C}_{i,j}$. Iniciamos com o caso em que $\mathcal{C}_{i,j} \neq \emptyset$. Portanto, sejam $[i'_1, j'_1], [i'_2, j'_2], \ldots, [i'_r, j'_r]$, $r \geq 1$, os pedaços de $G_{i,j}$, relativos a $\mathcal{C}_{i,j} = \{C_2^1, C_3^2 \ldots, C_r^{r-1}\}$, onde j'_t e i'_{t+1} são os vértices minimal e maximal, respectivamente, do corte-cadeia C_{t+1}^t, para $t \in \{1, \ldots, r-1\}$. Observe que o corte-cadeia C_{t+1}^t conecta os pedaços $[i'_t, j'_t]$ e $[i'_{t+1}, j'_{t+1}]$ e ainda $i'_1 = i$ e $j'_r = j$. Dessa forma, percebemos que $G_{i,j}$ é uma sequência de pedaços e cadeias (cortes-cadeia) intercalados e essa deve ser a exata ordem de execução no escalonamento, ou seja, a ordem de execução sempre será um pedaço $[i'_t, j'_t]$, o corte-cadeia C_{t+1}^t e o pedaço $[i'_{t+1}, j'_{t+1}]$. Isto posto, um limite inferior para o subgrafo $G_{i,j}$ é:

$$LB_{i,j}^{branco} = \sum_{t=1}^{r} LB_{i'_t, j'_t} + \sum_{t=1}^{r-1} (|C_{t+1}^t| - 1)$$

Na primeira parcela somamos os limites de cada pedaço de $G_{i,j}$, relativos a $\mathcal{C}_{i,j}$, enquanto na segunda parcela somamos os limites das cadeias (cortes-cadeia), porém não

adicionamos ao limite de cada cadeia C^t_{t+1} o vértice maximal i'_{t+1}, pois este já está incluído no limite $LB_{i'_{t+1},j'_{t+1}}$.

Trataremos agora o caso em que $\mathcal{C}_{i,j} = \emptyset$, considerando a vizinhança positiva de $[i,j]$ em D conforme sejam ligados por arcos azuis ou vermelhos. Separaremos estes vizinhos de acordo com a cor do arco e determinaremos um limite inferior para os dois grupos de vizinhos ($LB^{azul}_{i,j}$ e $LB^{vermelho}_{i,j}$). O limite inferior de $[i,j]$ será obtido como o melhor dos dois limites. Os cálculos de $LB^{azul}_{i,j}$ e $LB^{vermelho}_{i,j}$ são análogos, portanto mostraremos apenas a obtenção do limite para a cor azul.

Inicialmente, tome os diferentes valores de limites inferiores dos vizinhos positivos de $[i,j]$ que possuem arco de cor azul, $lb_1 < lb_2 < \ldots < lb_z$. Perceba que mais de um vizinho pode possuir o mesmo valor de limite inferior. Denotaremos o conjunto Λ_t, com $t \in \{1,\ldots,z\}$, como os vizinhos positivos azuis de $[i,j]$ que possuem limite maior ou igual a lb_t. Formalmente:

- $lb_1 = min\{LB_{k,l} : ([i,j],[k,l]) \in E^{azul}\}$
- $lb_{t+1} = min\{LB_{k,l} : ([i,j],[k,l]) \in E^{azul}, LB_{k,l} > lb_t\}$, com $t = \{1,\ldots,z-1\}$
- $\Lambda_t = \{[k,l] : ([i,j],[k,l]) \in E^{azul}, LB_{k,l} \geq lb_t\}$, com $t = \{1,\ldots,z\}$

Observação 1. *Se $[k,l] \in \Lambda_t$ e $[a,b] \in \Lambda_t$, então $k = a = i$ e $l \neq b$, com $l \vdash j$ e $b \vdash j$, pois estamos tratando os arcos de cor azul.*

Observação 2. *Se $[k,l] \in \Lambda_t$, então a tarefa l deve ser executada pelo menos lb_t unidades de tempo depois do início de $k = i$.*

A partir da definição do conjunto Λ_t e das observações acima, podemos concluir que pelo menos $|\Lambda_t|$ tarefas devem ser executadas lb_t unidades de tempos após o início da tarefa i. Sabemos que a tarefa j de $[i,j]$ deve executar após a execução de todas as tarefas l de cada vizinho positivo $[k,l] \in \Lambda_t$. Portanto, o tempo de início de j é pelo menos $lb_t + \delta(\Lambda_t)$, sendo a função δ uma função que associa o número de tarefas que podem ser executadas, de acordo com o limite de processadores e as restrições de precedência, ao mínimo de unidades de tempo que essas tarefas levarão para executar. Por exemplo, se $|\Lambda_t| = 1$, o tempo adicional será de apenas uma unidade, pois basta executarmos essa tarefa no mesmo processador de sua antecessora. Se $2 \leq |\Lambda_t| \leq m+1$, precisamos de pelo menos duas unidades de tempo. Pelas restrições de precedência e comunicação, apenas uma tarefa sucessora pode executar exatamente uma unidade de tempo após o início da tarefa antecessora. No tempo seguinte, podemos alocar as outras m tarefas. Assim, a definição da função segue segundo a tabela

$\lvert\Lambda_t\rvert$ tarefas	$\delta(\Lambda_t)$
$\lvert\Lambda_t\rvert = 1$	1
$2 \leq \lvert\Lambda_t\rvert \leq m+1$	2
$m+2 \leq \lvert\Lambda_t\rvert \leq 2m+1$	3
$2m+2 \leq \lvert\Lambda_t\rvert \leq 3m+1$	4
$\vdots$	$\vdots$
$(\alpha-1)m+2 \leq \lvert\Lambda_t\rvert \leq \alpha m+1$	$\alpha+1$
$\vdots$	$\vdots$
$\lvert\Lambda_t\rvert = \beta$	$\left\lceil \frac{\beta-1}{m} \right\rceil + 1$

Table 1. Definição da função δ

Portanto, o limite inferior para $[i,j]$ baseado nos vizinhos positivos azuis é:

$$LB_{i,j}^{azul} = \max_{t=1,\ldots,z} \left\{ lb_t + \left\lceil \frac{|\Lambda_t| - 1}{m} \right\rceil + 1 \right\}$$

Como mencionado, o limite inferior baseado nos vizinhos positivos de $[i,j]$ ligados por arcos vermelho pode ser definido de forma análoga. Esta definição de limite pode ser considerada uma generalização do limite apresentado em (Campêlo et al., 2002). Lembre que lb_z é maior valor de um limite inferior de um vizinho positivo de $[i,j]$. Na referência citada, $LB_{i,j}^{c} = lb_z + \Delta$ com $\Delta = 1$, se $|\Lambda_z| = 1$, ou $\Delta = 2$, se $|\Lambda_z| > 1$, e ainda $c \in \{azul, vermelho\}$. Observe que obviamente o novo limite apresentado generaliza o anterior, pois leva em consideração cada valor de limite lb_t, com $t \in \{1,\ldots,z\}$ e sempre $\Delta \leq \left\lceil \frac{|\Lambda_t|-1}{m} \right\rceil + 1$, portanto o novo limite inferior é sempre melhor ou igual ao anterior. Reunindo as diferentes formas de calcular o limite $LB_{i,j}$, podemos formular a seguinte proposição:

Proposição 1. *Seja D o grafo de decomposição do grafo de tarefas $G = (N, \prec)$ e $[i,j] = G_{i,j} = (N_{i,j}, \prec_{i,j}) \in V(D)$. Se*

$$LB_{i,j} = \begin{cases} LB_{i,j}^{folha}, & [i,j] \text{ é uma folha} \\ LB_{i,j}^{branco}, & \mathcal{C}_{i,j} \neq \emptyset \\ max\{LB_{i,j}^{azul}, LB_{i,j}^{vermelho}, \left\lceil \frac{|N_{i,j}|}{m} \right\rceil - 1\}, & \mathcal{C}_{i,j} = \emptyset \end{cases}$$

então $LB_{i,j} + 1$ é um limite inferior para o makespan ótimo de $(N_{i,j}, \prec_{i,j}, Q)$.

5. Formulação matemática

Utilizaremos como formulação básica a introduzida em (Campêlo et al., 2001). Esta formulação é baseada na partição do grafo em cadeias e determinação dos tempos de início das tarefas.

Em uma solução viável, a cada processador está associada uma sequência de tarefas que são executadas nele. Nas sequências formadas, estão presentes relações de precedência que não existiam antes no grafo de entrada original. São tarefas que são incomparáveis originalmente, mas que passam a ter uma precedência artificial na sequência a que pertence. Portanto as relações de precedência devem ser respeitadas também entre esses nós. Observado isso, podemos entender um escalonamento viável como sendo uma extensão da relação original que pode ser particionada em até m cadeias, onde m é o número de processadores. Desta forma, cada tarefa pertenceria à uma única cadeia e cada cadeia poderia ser alocada em um processador diferente.

Matematicamente, podemos descrever as cadeias através de uma variável binária, $w_{i,j}$, tal que $w_{i,j} = 1$ se, e somente se, a tarefa j for a próxima a ser escalonada depois de i em um mesmo processador, com $i \prec j$ ou $i \parallel j$.

Como mencionado, para que um escalonamento seja viável, basta particionarmos em cadeias uma extensão da relação de precedência original e, a partir destas cadeias, determinar os tempos de execução de cada tarefa em seus respectivos processadores, respeitando as restrições impostas. Vamos então definir o conjunto de variáveis que será utilizado para formular matematicamente o problema. O tempo de início de cada tarefa i é determinado pela variável x_i. A variável $w_{0,i}$ terá valor 1 se i for a primeira tarefa a ser executada por algum processador. Da mesma forma $w_{i,n+1}$ será 1 se i for a última tarefa

executada em algum processador. Finalmente, a variável $w_{i,j}$, definida para todo par de tarefas i, j, tal que $i \prec j$ ou $i \parallel j$, assumirá valor 1 se j for a próxima tarefa a ser executada após i em um mesmo processador. Devemos minimizar o tempo total de processamento (*makespan*), que deve ser o tempo de início da tarefa maximal n mais 1.

Formulação

$$\begin{aligned}
\min \quad & x_n + 1 \\
s.a \quad & w_{0,j} + \sum_{i:i\prec j} w_{i,j} + \sum_{i:i\parallel j} w_{i,j} = 1 && j \in N && (1)\\
& w_{i,n+1} + \sum_{j:i\prec j} w_{i,j} + \sum_{j:i\parallel j} w_{i,j} = 1 && i \in N && (2)\\
& \sum_{i\in N} w_{i,n+1} \leq m && && (3)\\
& w_{i,j} \in \{0,1\} && i,j \in N, i \prec j \quad \text{ou} \quad i \parallel j && (4)\\
& w_{0,i}, w_{i,n+1} \in \{0,1\} && i \in N && (5)\\
& x_j - x_i \geq 2 - w_{i,j} && i,j \in N, \quad i \vdash j && (6)\\
& x_j - x_i \geq 1 - \alpha_{ij}(1 - w_{i,j}) && i,j \in N, \quad i \parallel j && (7)\\
& x_i \geq 0 && i \in N &&
\end{aligned}$$

Apresentaremos as restrições da formulação em dois grupos. Iniciamos com as restrições que definem as cadeias a partir do grafo de entrada (1-3).

Uma cadeia de tarefas pode ser caracterizada como um conjunto de tarefas tal que toda tarefa possui exatamente um antecessor e exatamente um sucessor, com exceção da primeira e da última tarefas que, respectivamente, não possuem antecessor e sucessor. Assim as restrições (1-3) definem m cadeias no grafo. A primeira restrição afirma que toda tarefa j ou é a primeira tarefa executada em algum processador $(w_{0,j} = 1)$ ou possui uma tarefa antecessora, seja ela concorrente ou não à j. Por outro lado, a segunda restrição afirma que toda tarefa i ou é a última tarefa a ser executada em algum processador $(w_{i,n+1} = 1)$ ou possui uma tarefa sucessora e esta pode ser dependente ou não de i. Finalmente a última restrição garante que o número de cadeias formadas é menor ou igual ao número de processadores, m.

Com as cadeias formadas, devemos adequar os tempos de início para que eles respeitem as restrições de precedência e comunicação. Considere as restrições (6-7). Claramente, as restrições de precedência e comunicação são dependentes da relação que existe entre duas tarefas i e j. Portanto, temos uma restrição para cada caso. Pela restrição (6), se j depende de i, então a diferença entre seus tempos de início deve ser maior ou igual a 1, sendo as duas executadas em um mesmo processador $(w_{i,j} = 1)$, e maior ou igual a 2, se elas são executadas em processadores diferentes. Por outro lado, pela restrição (7), se i e j são concorrentes devemos garantir que, se elas são executadas em um mesmo processador, o tempo de início de j é pelo menos o tempo de início de i mais um. Observe que, se as tarefas são concorrentes e não são executadas no mesmo processador, a restrição deve se tornar redundante. Isto é feito através de um Big-M, logo $\alpha_{i,j}$ deve ser grande o suficiente para tornar esta restrição redundante. Não é difícil ver que $\alpha_{i,j}$ é um limite superior para $x_i - x_j + 1$ em uma solução ótima do problema. Isto pode ser observado fazendo $(w_{i,j} = 0)$ nesta restrição. A constante $\alpha_{i,j}$ pode ser estimada satisfatoriamente utilizando

os limites superior viável ($UB_{1,n}$), baseado na heurística CPMISF, e inferior $LB_{i,j}$ apresentado anteriormente. Neste caso, $\alpha_{i,j} = UB_{1,n} - LB_{1,j} - LB_{i,n}$, pois $LB_{1,j} \leq x_j$ e $x_i \leq UB_{1,n} - (LB_{i,n} + 1)$.

6. Incrementos à formulação original

A resolução desta formulação diretamente no *solver* CPLEX mostra-se impraticável para certas instâncias do problema. Mesmo para instâncias que conseguiríamos resolver facilmente com um algoritmo ingênuo, o tempo de resolução da formulação pode não ser razoável. Para diminuir esse tempo de resolução, como tentativa inicial, buscamos o fortalecimento da formulação através da inclusão de restrições derivadas dos limites inferiores apresentados anteriormente e ainda a adoção de novas estratégias de ramificação.

6.1. Limite inferior $LB_{i,j}$

Detalhamos anteriormente a obtenção de um limite inferior para o problema. O valor $LB_{i,j} + 1$ provê um limite inferior para o *makespan* do subgrafo induzido pelo conjunto de vértices $N_{i,j}$. Assim, podemos utilizar esse limite para gerar restrições válidas para o problema e acrescentá-las à formulação matemática. Estas restrições são do tipo:

$$x_j - x_i \geq LB_{i,j} \qquad i, j \in N, i \prec j \tag{8}$$

Para efeito de comparação, usamos o CPLEX com sua parametrização padrão para resolver a formulação sem as restrições e, posteriormente, acrescentamos as novas restrições baseadas no limite inferior $LB_{i,j}$.

DAG	n	$\omega(\prec)$	$UB_{1,n}$	$LB_{1,n}$	Makespan ótimo	IP (s)	IP c/sol. inicial (s)	IP c/ (8) (s)	IP c/sol. inicial e (8) (s)
tree7	**255**	**128**	**15**	**14**	**15**	**-**	**-**	**166**	**171**
bin8	255	128	15	15	15	4	3	3	3
bin9	511	256	17	17	17	31	32	27	31
di100	100	10	36	28	28	4	3	8	9
di144	144	12	44	34	34	28	29	34	33
di225	225	15	58	43	43	7205	7205	7248	7200
di256	**256**	**16**	**60**	**46**	**46**	**366**	**364**	**230**	**210**
divconq-m	**382**	**128**	**25**	**22**	**22**	**7201**	**291**	**293**	**301**
fft2-b-m	**194**	**32**	**19**	**14**	**16**	**7209**	**1813**	**1230**	**1235**
iterative2-b-m	262	26	26	13	24	-	7202	-	-
prolog-m	**214**	**126**	**15**	**14**	**15**	**7200**	**-**	**953**	**953**
ran3	153	48	16	13	14	8	7	10	11
ran6	223	110	9	8	9	1	2	2	1
ran7	298	119	12	10	11	28	50	28	28
ran8	256	118	12	11	11	2	23	1	20
ran10	286	83	19	17	18	241	239	272	272
ran13	357	167	12	10	10	36	36	39	40
ran14	364	121	16	15	15	5	28	5	64
ran15	152	47	15	14	15	1	1	1	1
ran16	186	76	11	10	11	1	1	2	2
ran17	**546**	**158**	**22**	**21**	**21**	**1014**	**1012**	**657**	**756**
ran18	234	74	15	14	15	3	3	2	3
ssc4	128	48	15	14	14	3	3	3	3
ssc5	200	75	16	14	15	10	16	16	16
ssc6	288	108	17	16	17	22	19	19	19
ssc7	392	147	18	16	17	599	688	695	709

Table 2. Resultados computacionais com a adição das restrições (8)**, com** $m = \omega(\prec)/2$

A tabela 2 apresenta uma comparação dos resultados. Para cada instância (DAG), apresentamos o número de tarefas (n), a largura do grafo de tarefas ($\omega(\prec)$), o limite superior dado pela heurística CPMISF ($UB_{1,n}$), o limite inferior dado pelo processo de programação dinâmica ($LB_{1,n}$), o valor do *makespan* ótimo, calculado através da resolução exata da formulação com o solver CPLEX, e os tempos (em segundos) das quatro versões analisadas. As colunas IP (s), IP c/sol. inicial (s), IP c/ (8) (s) e IP c/sol. inicial e (8) (s) representam os tempos respectivamente de resolução exata da formulação, de resolução exata informando inicialmente uma solução viável (CPMISF), de resolução exata com a adição das restrições (8) e de resolução exata passando uma solução inicial (CPMISF) e

acrecentando as restrições (8). Células marcadas com "-" indicam casos que não puderem ser resolvidos em um limite de tempo de 2 horas. As instâncias consideradas são as mesmas utilizadas em (Campêlo et al., 2002), onde são descritas em detalhes. Os experimentos foram feitos em uma máquina com 2 processadores quad-core (8 núcleos) Intel(R) Xeon(R) CPU X5450, 3.00GHz.

Analisando os resultados, percebemos que a inclusão das restrições quase sempre melhora o desempenho, em particular para as instâncias destacadas em negrito. Algumas instâncias que não puderam ser resolvidas apenas com as restrições originais da formulação dentro do limite de tempo, passaram a ser resolvidas após a inclusão das novas restrições. Por outro lado, o fornecimento de uma solução viável para o solver não levou a um melhor desempenho para os casos avaliados. Comparativamente, a versão que foi melhor que a resolução exata pura (IP-Integer Programming) em um maior número de instâncias, 11, foi a coluna IP c/ (8), enquanto as colunas IP c/sol. inicial e IP c/sol. inicial e (8), superaram a resolução inteira, respectivamente em 9 e 8 intâncias.

6.2. Critérios de ramificação

Além da adição de novas restrições, testamos a mudança no critério da ramificação no método Branch-and-Bound utilizado para fornecer a solução exata para o problema. Da formulação original possuímos dois conjuntos de variáveis, as variáveis X e as variáveis W. Durante o processo do Branch-and-Bound podemos priorizar um conjunto de variáveis e ainda dentro do conjunto de variáveis, podemos ordená-las por algum critério definido antecipadamente. Os seguintes critérios foram avaliados:

- Critério 1: Priorizar variáveis X sobre variáveis W.
- Critério 2: Priorizar variáveis W sobre variáveis X (em ordem decrescente do valor de $UB_i - LB_i$).
- Critério 3: Priorizar variáveis X (em ordem decrescente do valor de $UB_i - LB_i$) sobre variáveis W.
- Critério 4: Priorizar variáveis X (em ordem crescente do valor de $UB_i - LB_i$) sobre variáveis W.
- Critério 5: Priorizar a variável x_n sobre as demais.

Observe que em alguns critérios utilizamos o parâmetro $UB_i - LB_i$, que indica o intervalo de agendamento da tarefa i. Teoricamente, quanto maior esse intervalo, maior a indefinição a respeito do tempo de início da tarefa na solução ótima. Os limites superior e inferior que usamos nos critérios são $UB_i = UB_{1,n} - (LB_{i,n} + 1)$ e $LB_i = LB_{1,i}$.

Para cada critério, avaliamos o desempenho do CPLEX aplicado à formulação com ou sem as restrições (8) e usando ou não a solução viável inicial dado pela heurística CPMISF. Na tabela 3, apresentamos uma comparação entre o melhor resultado para cada critério, escolhemos as versões que obtiveram os melhores resultados dentre todas as combinações. Com os resultados, não foi possível obter uma versão que obtivesse o melhor resultado em todas as instâncias, ou seja, não existe uma versão unânime. Porém, observamos que os critérios 4 e 5 resultaram nos melhores desempenhos. Em todas as instâncias, a exceção de três delas, cada um desses critérios obteve o melhor desempenho ou desempenho comparável ao melhor, entre as versões concorrentes.

Não observamos outras formas de melhorar a resolução exata do problema através dessa formulação. Dados os resultados, buscaremos a concepção de uma nova formulação para o problema. Possivelmente, baseando as variáveis no tempo específico em que uma tarefa é executada.

DAG	n	Makespan ótimo	IP (s)	Critério 1 IP c/sol. inicial e (8) (s)	Critério 2 IP c/sol. inicial e (8) (s)	Critério 3 IP c/sol. inicial e (8) (s)	Critério 4 IP c/ (8) (s)	Critério 5 IP c/(8) (s)
tree7	255	15	-	164	186	176	183	170
bin8	255	15	4	3	2	3	3	3
bin9	511	17	31	32	31	32	28	27
di100	100	28	4	9	9	9	8	8
di144	144	34	28	33	33	34	35	31
di225	225	43	7205	7200	-	7205	7200	7201
di256	256	46	366	230	209	231	212	211
divconq-m	382	22	7201	292	293	294	292	293
fft2-b-m	194	16	7209	1211	7207	1521	1544	1826
iterative2-b-m	262	24	-	7200	-	-	-	7202
prolog-m	214	15	7200	841	-	56	57	7201
ran3	153	14	8	10	11	11	11	10
ran6	223	9	1	2	1	2	2	2
ran7	298	11	28	28	28	28	29	27
ran8	256	11	2	20	20	20	1	2
ran10	286	18	241	272	269	285	286	168
ran13	357	10	36	39	39	40	41	38
ran14	364	15	5	63	63	62	5	5
ran15	152	15	1	1	1	0	1	1
ran16	186	11	1	1	1	2	2	2
ran17	546	21	1014	805	660	811	809	806
ran18	234	15	3	3	3	2	3	2
ssc4	128	14	3	3	3	2	2	3
ssc5	200	15	10	16	16	17	17	16
ssc6	288	17	22	19	20	19	19	19
ssc7	392	17	599	705	829	641	640	700

Table 3. Resultados computacionais dos diversos critérios de ramificação. $m = \omega(\prec)/2$

References

Campêlo, M., Corrêa, R., Maculan, N., and Protti, F. (2001). Ilp formulations for scheduling ordered tasks on a bounded number of processors. *Electronic Notes in Discrete Mathematics*, 7:166 – 169.

Campêlo, M., Corrêa, R., Maculan, N., and Protti, F. (2002). Improved lower bounds for scheduling ordered tasks on a bounded number of processors. In *Proceedings of CLAIO - Congresso Latino-Ibero-Americano de Investigacion Operativa*.

Cheng, T. and Sin, C. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271 – 292.

Chretienne, P. (1989). A polynomial algorithm to optimally schedule tasks on a virtual distributed system under tree-like precedence constraints. *European Journal of Operational Research*, 43(2):225 – 230.

Coll, P. E., Ribeiro, C. C., and de Souza, C. C. (2006). Multiprocessor scheduling under precedence constraints: Polyhedral results. *Discrete Applied Mathematics*, 154(5):770 – 801.

Davidović, T., Liberti, L., Maculan, N., and Mladenović, N. (2007). Towards the optimal solution of the multiprocessor scheduling problem with communication delays. In *In MISTA Proceedings*.

Hanen, C. and Munier, A. (2001). An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. *Discrete Applied Mathematics*, 108(3):239 – 257.

Kwok, Y.-K. and Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471.

Ullman, J. D. (1975). Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3):384–393.