UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE ESTATÍSTICA E MATEMÁTICA APLICADA

PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM E MÉTODOS QUANTITATIVOS

JUDECIR CAVALCANTE AGUIAR JÚNIOR

MODELO DE ARRANJO LINEAR E DESIGUALDADES VÁLIDAS PARA O PROBLEMA DO *JOB SHOP* COM MINIMIZAÇÃO DO *MAKESPAN*

FORTALEZA

2021

JUDECIR CAVALCANTE AGUIAR JÚNIOR

MODELO DE ARRANJO LINEAR E DESIGUALDADES VÁLIDAS PARA O PROBLEMA DO *JOB SHOP* COM MINIMIZAÇÃO DO *MAKESPAN*

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem e Métodos Quantitativos da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Modelagem e Métodos Quantitativos.
Área de Concentração: Modelagem e Métodos Quantitativos.

Orientador: Prof. Dr. Rafael Castro de Andrade
Coorientador: Prof. Dr. Christophe Didier Duhamel

FORTALEZA

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

A229m Aguiar Júnior, Judecir Cavalcante.
Modelo de arranjo linear e desigualdades válidas para o problema do job shop com minimização do makespan / Judecir Cavalcante Aguiar Júnior. – 2021.
84 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Modelagem e Métodos Quantitativos, Fortaleza, 2021.
Orientação: Prof. Dr. Rafael Castro de Andrade.
Coorientação: Prof. Dr. Christophe Didier Duhamel.

1. Otimização combinatória. 2. Planejamento da produção. 3. Desigualdades válidas. 4. Programação Inteira. I. Título.

CDD 510

JUDECIR CAVALCANTE AGUIAR JÚNIOR

MODELO DE ARRANJO LINEAR E DESIGUALDADES VÁLIDAS PARA O PROBLEMA DO *JOB SHOP* COM MINIMIZAÇÃO DO *MAKESPAN*

Dissertação apresentada ao Programa de Pós-Graduação em Modelagem e Métodos Quantitativos da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Modelagem e Métodos Quantitativos.
Área de Concentração: Modelagem e Métodos Quantitativos.

Aprovada em: 25 de fevereiro de 2021

BANCA EXAMINADORA

Prof. Dr. Rafael Castro de Andrade (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Christophe Didier Duhamel (Coorientador)
Université Le Havre

Prof. Dr. Anselmo Ramalho Pitombeira Neto
Universidade Federal do Ceará (UFC)

Prof. Dr. Bruno de Athayde Prata
Universidade Federal do Ceará (UFC)

Prof. Dr. Geraldo Robson Mateus
Universidade Federal de Minas Gerais (UFMG)

Prof. Dr. Anand Subramanian
Universidade Federal da Paraíba (UFPB)

À minha família, por sempre acreditar em mim e querer meu melhor.

AGRADECIMENTOS

Ao Prof. Dr. Rafael Castro de Andrade por me orientar em minha dissertação de mestrado, pela compreensão nos momentos difíceis, pela exigência de maneira assertiva e por me passar um pouco da sua vasta experiência.

Ao Prof. Dr. Christophe Didier Duhamel por me coorientar em minha dissertação de mestrado, pelas ideias incríveis sugeridas no trabalho e pela colaboração efetiva mesmo à distância.

Aos professores Dr. Anselmo Ramalho Pitombeira Neto, Dr. Bruno de Athayde Prata, Dr. Anand Subramanian e Dr. Geraldo Robson Mateus por aceitarem fazer parte da banca e pelas sugestões muito bem-vindas.

Ao corpo docente e servidores do Programa de Pós-Graduação em Modelagem e Métodos Quantitativos (MMQ), pelo empenho e dedicação em tornar o programa melhor a cada ano.

Ao meu pai Judecir Cavalcante Aguiar, minha mãe Natércia Girão Aguiar e minhas irmãs Nathalia Girão Aguiar e Nayra Girão Aguiar, pela apoio incondicional durante toda essa jornada.

À Giovanna Cristina pela motivação, carinho, e paciência que teve comigo durante todo esse tempo.

Ao colega Mardson da Silva pelas sugestões e apoio em diversas vezes, sua ajuda foi fundamental.

Aos meus colegas de trabalho da Secretaria de Tecnologia da Informação do Tribunal de Contas do Estado do Ceará pelo apoio, conversas e motivação.

Ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) e à CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) pelo apoio à ciência.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

“A vida é tipo uma estrada, por onde os dias vem e vão
E o plano é fazê-los não serem em vão.”

(Rashid)

RESUMO

O problema do sequeciamento de trabalhos, do inglês *job shop scheduling problem* (JSSP), consiste em agendar n trabalhos (*jobs*) em m máquinas. Cada trabalho tem uma ordem de processamento nas máquinas, podendo inclusive ser uma ordem específica para cada um, e um tempo de processamento em cada uma delas. Além disso, cada trabalho não pode ser processado ao mesmo tempo em duas máquinas distintas, cada máquina não pode processar dois trabalhos ao mesmo tempo e se um trabalho iniciou seu processamento em uma máquina, então tem que ser concluído sem interrupções. Uma das funções objetivo para esse problema é minimizar a conclusão do último trabalho a ser processado, também conhecido como *makespan*. Para essa função objetivo o problema é NP-Difícil para pelo menos três máquinas e dois trabalhos e, por isso, é importante desenvolver novos modelos para conseguir resultados mais eficientes. Para tanto, exploramos uma modelagem de Andrade *et al.* (2017) para o problema do arranjo linear mínimo, do inglês *minimum linear arrangement* (MinLA), e adaptamos essa abordagem para desenvolver um novo modelo para o JSSP e denominado de (MLJ). De forma complementar, exploramos a estrutura da solução do problema para desenvolver novas desigualdades válidas para o JSSP com base no tempo de processamento acumulado a *priori* e a *posteriori* da execução de um trabalho em uma dada máquina. Adicionamos essas desigualdades nos modelos de Manne (1960), Liao e You (1992) e (MLJ), denominados esses novos modelos como $(M2)$, $(L2)$ e $(MLJ2)$, respectivamente, e testamos em instâncias da literatura e para um novo conjunto de instâncias. Com base nos experimentos computacionais, quando comparamos os modelos com desigualdades válidas aos modelos originais, obtemos uma melhora na relaxação linear de $(M2)$ e $(L2)$ para as novas instâncias em cerca de 87% e conseguimos *gaps* tão bons quanto ou melhores em 50% delas para o modelo $(M2)$ e de cerca de 67% delas para o modelo de $(L2)$. Além disso, melhoramos a relaxação linear de $(M2)$ em cerca de 84% das instâncias da literatura e conseguimos *gaps* tão bons quanto ou melhores em 80% delas. Por fim, verificamos que o modelo ($MLJ2$) teve comportamento semelhante ao de ($M2$).

Palavras-chave: Otimização combinatória. Planejamento da produção. Desigualdades válidas. Programação Inteira

ABSTRACT

The job-shop scheduling problem (JSSP) consists in scheduling n jobs on m machines. Each job has a processing order on the machines, it can even be a specific order for each one, and a processing time on each one. In addition, each job cannot be processed at the same time on two different machines, each machine cannot process two jobs at the same time, and if a job started processing on one machine, then this processing will be completed without interruptions. One of the objective functions for this problem is to minimize the completion of the last job processed, also known as *makespan*. Thus, for this objective function, the problem is NP-Hard for at least three machines and two jobs and, for this problem, the computational effort can be huge to solve the problem in real instances. Therefore, it is important to develop new models to achieve more efficient results, both in terms of the execution time and the quality of the problem solutions. For that, we explored the model of Andrade *et al.* (2017) for the minimal linear arrangement (MinLA) problem and we adapted the model for MinLA to develop a new model for JSSP, called (MLJ). In a complementary way, we explore the structure of the problem solution to develop new valid inequalities for JSSP based on the accumulated processing time a *priori* and a *posteriori* of the execution of a job on a given machine. We apply these inequalities for the models of Manne (1960), Liao and You (1992) and (MLJ), named of $(M2), (L2)$ e $(MLJ2)$, respectively, and tested them on benchmark instances from the literature and for a new set of instances. Based on the computational experiments, when we compare the models with valid inequalities to the original models, we obtain an improvement in the linear relaxation of $(M2)$ and $(L2)$ for the new instances by about 87% and we obtained gaps as good as or better in 50% of them for the model $(M2)$ and about 67% of them for the model $(L2)$. In addition, we improved the linear relaxation of $(M2)$ in about 84% of the instances from the literature and achieved gaps as good as or better in 80% of them. Finally, we found that the model $(MLJ2)$ had a similar behavior of model $(M2)$.

Keywords: Combinatorial optimization. Production planning. Valid inequalities. Integer programming.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ELS	*Evolutionary Local Search*
GRASP	*Greedy Randomized Adaptive Search Procedure*
JSSP	*job shop scheduling problem*
LB	*lower bound*
LC	lista de candidatos
LRC	lista retrista de candidatos
MinLA	*minimum linear arrangement*
MIP	*Mixed Integer Programming*

LISTA DE SÍMBOLOS

π_v	Para o modelo Andrade *et al.* (2017) é o rótulo do vértice v
σ_j	Ordem de execução das máquinas do trabalho j
$\sigma^{j^{-1}}(i)$	Função que retorna a posição da máquina i na ordem de execução do trabalho j
C_{max}	*Makespan*
$G = (V, E)$	Grafo de conjunto de vértices V e de arestas E
h_{ijk}	Para o modelo Wagner (1959) é o tempo de início do k-ésimo trabalho da máquina i
H	Para o modelo de Kondili *et al.* (1988) é a discretização do tempo
J	Conjunto de trabalhos
M	Conjunto de máquinas
m	Número de máquinas
n	Número de trabalhos
P	*Big-M* utilizado para o problema do sequenciamento de trabalhos
P_{ij}^{-}	Parâmetro da soma dos tempos a *priori* do trabalho j na máquina i
P_{ij}^{+}	Parâmetro da soma dos tempos a *posteriori* do trabalho j na máquina i
p_{ij}	Para todos os modelos do JSSP representa o tempo de processamento do trabalho j na máquina i
P_j	Parâmetro do tempo total de processamento do trabalho $j \in J$ em todas as máquinas;
q_{ijk}	Para os modelos de Liao e You (1992) e (L2) é a variável de folga das restrições disjuntivas
r_{ijk}	Para o modelo de Wagner (1959) é um parâmetro que indica se o k-ésimo trabalho j requer a máquina i
w_{uv}	Para o modelo Andrade *et al.* (2017) é a diferença de rótulos de v e u, se u precede v

x_{ij} Para os modelos Manne (1960), Liao e You (1992), $(M2)$, $(L2)$, (MLJ) e $(MLJ2)$ é o tempo de início do trabalho j na máquina i

x_{ijk} Para o modelo de Wagner (1959) é 1 se o trabalho j é o k-ésimo trabalho da máquina i

x_{ijt} Para o modelo de Kondili *et al.* (1988) é 1 se o trabalho j inicia o processamento no tempo t na máquina i

x_{uv} Para o modelo Andrade *et al.* (2017) é 1 se o vértice u precede o vértice v

z_{ijk} Para os modelos Manne (1960), Liao e You (1992), $(M2)$, $(L2)$, (MLJ) e $(MLJ2)$ é 1 se o trabalho j precede k na máquina i

SUMÁRIO

1 INTRODUÇÃO

O JSSP consiste em agendar, organizar ou sequenciar um conjunto J de n trabalhos (*jobs*) em um conjunto M de m máquinas, em que cada trabalho tem um tempo de processamento para cada máquina. Além disso, na versão clássica do problema, esse sequenciamento segue quatro regras: cada trabalho tem sua ordem de processamento nas máquinas, cada trabalho não pode ser processado ao mesmo tempo em duas máquinas distintas, cada máquina não pode processar dois trabalhos ao mesmo tempo e considera a preempção, ou seja, se um trabalho iniciou o processamento em uma máquina, então esse processamento será concluído sem interrupção. Uma das funções objetivo para esse problema é minimizar a conclusão do último trabalho processado, também conhecido como *makespan*.

O objetivo geral deste trabalho foi desenvolver tanto uma nova modelagem matemática para o JSSP quanto propor novas desigualdades para fortalecer a relaxação linear de modelos da literatura [Manne (1960), Liao e You (1992)] para o problema, com a função objetivo sendo a minimização do *makespan*. Para tanto, tomamos como base um novo modelo proposto para o problema do arranjo linear mínimo de Andrade *et al.* (2017) e propriedades estruturais do JSSP para o desenvolvimento de novas desigualdades válidas. Apresentamos uma revisão sucinta dos principais trabalhos existentes para o problema do sequenciamento de trabalhos e abordamos em detalhes alguns dos melhores modelos de programação inteira para resolvê-lo. Mostramos também o mais recente trabalho sobre meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP) para o JSSP, desenvolvida em Bissoli (2018). Com base nas ideias desse trabalho, empregamos um procedimento heurístico para fornecer valores de solução viável para o JSSP usados como valor de corte no *Mixed Integer Programming* (MIP) solver do CPLEX.

Quanto ao JSSP, os principais modelos de programação inteira podem ser encontrados no *survey* Ku e Beck (2016). Nele estão o modelo disjuntivo de Manne (1960), disjuntivo de Liao e You (1992), o modelo indexado por variáveis de tempo de Kondili *et al.* (1988) e um modelo baseado na ordem de Wagner (1959).

Explanamos de maneira sintetizada como combinar as meta-heurísticas GRASP e *Evolutionary Local Search* (ELS) para fornecer um *upper bound* inicial para o *makespan* passado como valor de corte no *solver* CPLEX.

Além disso, como referência de base do novo modelo proposto para o JSSP, empregamos o modelo de Andrade *et al.* (2017) para o problema do arranjo linear mínimo. Precisamente, desenvolvemos um novo modelo (MLJ) adaptando restrição de Andrade *et al.* (2017) para o modelo disjuntivo de Manne (1960).

Propusemos novas desigualdades válidas explorando a estrutura do problema em relação ao tempo de processamento acumulado antes e depois da realização de um trabalho em uma dada máquina. Por fim, realizamos testes computacionais com instâncias *benchmark* e adaptações do JSSP para mostrar a efetividade das ideias desenvolvidas neste trabalho.

Como resultados alcançados, em relação às desigualdades propostas, mostramos o impacto delas nos modelos de Manne (1960) e Liao e You (1992) na melhora de limites inferiores da relaxação linear em instâncias adaptadas da literatura. Com base nesses experimentos, melhoramos a relaxação linear do modelo de Manne (1960) e Liao e You (1992) nas novas instâncias em cerca de 87% e conseguimos *gaps* tão bons quanto ou melhores em 50% dessas instâncias para o modelo de Manne (1960) e de cerca de 67% para o modelo de Liao e You (1992). Esses resultados foram objeto de um artigo completo apresentado no LII Simpósio Brasileiro de Pesquisa Operacional, 2020 (ver em Cavalcante *et al.* (2020)).

Além disso, estudamos o impacto das desigualdades nos modelos de Manne (1960) e (MLJ) com instâncias da literatura e observamos que melhoramos a relaxação linear de Manne (1960) em cerca 84% das instâncias da literatura com o acréscimo das desigualdades válidas e também conseguimos *gaps* tão bons quanto ou melhores em 80% dessas instâncias. Por fim, o modelo (MLJ) com as desigualdades válidas teve o comportamento semelhante ao modelo de Manne (1960) com as novas desigualdades.

O restante do trabalho é organizado da seguinte forma. No Capítulo 2 definimos e exemplificamos a versão usada neste trabalho do problema do sequenciamento de trabalhos. No Capítulo 3 discutimos uma revisão bibliográfica das principais abordagens da resolução do JSSP e detalhamos mais os principais modelos de programação linear inteira da literatura, bem como a mais recente meta-heurística para o problema e um procedimento evolucionário de busca local, assim como uma combinação destes para obter uma meta-heurística usada para fornecer *upper bounds* para o problema. No Capítulo 4 explicamos o problema do arranjo linear mínimo, a modelagem de Andrade

et al. (2017) e por fim apresentamos um novo modelo para o JSSP baseado nesse modelo. No Capítulo 5 apresentamos desigualdades válidas para o JSSP. No Capítulo 6 discutimos os resultados obtidos a partir de instâncias da literatura e de um conjunto novo de instâncias. Por fim, no Capítulo 7 apresentamos a conclusão dos resultados alcançados neste trabalho, além de projetar novas direções de pesquisa que podem ser implementadas para o problema como trabalhos futuros.

2 PROBLEMA DO SEQUENCIAMENTO DE TRABALHOS COM MINIMIZAÇÃO DO *MAKESPAN*

De maneira geral, segundo Pinedo e Hadavi (1992), um sequenciamento, do inglês *scheduling*, é um processo de tomada de decisão que trata da alocação de trabalhos em máquinas, cujo objetivo é otimizar uma ou mais funções objetivos. A interpretação de trabalho varia bastante de acordo com cada contexto, podemos entender como tarefa, atividade, etc. O mesmo ocorre para o termo máquina, podemos interpretar também como recurso, responsável, etc. Como exemplos reais de sequenciamento podemos pensar na alocação de tarefas de projeto para os responsáveis, alocação de processamentos na CPU de um computador, alocação dos portas de embarque e desembarque dos voos de um aeroporto, etc.

Acerca das funções objetivos também são várias possibilidades. Podemos minimizar do tempo de término do processamento do último trabalho em sua última máquina; o prazo de término de processamento de cada trabalho, o atraso total, se associarmos um peso de importância em cada trabalho podemos minimizar o atraso ponderado, etc.

Assumimos que as quantidades de trabalhos e de máquinas são finitas, e denotamos por J o conjunto de trabalhos (*jobs*), com $|J| = n$, e M o conjunto de máquinas, com $|M| = m$. Além disso, podemos chamar de O o conjunto de operações, de modo que uma operação é descrita como um par (i, j), com $i \in M$ e $j \in J$. A cardinalidade de O pode variar dependendo do tipo de problema definido. Pinedo e Hadavi (1992) descrevem os problemas de sequenciamento com a tupla $\alpha|\beta|\gamma$, de modo que α representa o ambiente das máquinas, descreveremos posteriormente quais cenários são possíveis, β as características de processamento e restrições, podendo ser várias configurações diferentes e, por fim, γ a função objetivo associada.

Alguns tipos de ambientes possíveis representados por α são:

a) *Single machine* (1): o caso mais simples de sequenciamento, onde há apenas uma máquina e todos os trabalhos devem ser processados nessa máquina;
b) *Flow shop* (Fm): são consideradas m máquinas e cada trabalho deve ser processado em todas elas uma vez. Além disso, todos os trabalhos devem seguir a mesma ordem de processamento nas máquinas, ou seja, sem perda de generalidade podemos dizer que todos os trabalhos são executados primeiro na máquina

1, depois na máquina 2 e assim por diante, até a máquina m;

c) *Job shop* (Jm): são consideradas m máquinas e cada trabalho deve ser processado em todas elas uma vez. Além disso, cada trabalho tem uma ordem pré-definida de máquinas em que deve ser processado;

d) *Open shop* (Om): são consideradas m máquinas e cada trabalho deve ser processado em todas elas uma vez. Além disso, não há restrição de precedência dos trabalhos nas máquinas, ou seja, os trabalhos podem ser processados em qualquer ordem nas máquinas.

Já na perspectiva de características de processamento, temos algumas possibilidades de β citadas abaixo:

a) Preempções ($prmp$): significa que o processamento pode ter interrupção. Caso não seja incluído em β, então significa que se um trabalho iniciou o processamento em uma máquina, então será concluído sem interrupção;

b) Restrições de precedência ($prec$): representa as precedências de um ou mais trabalhos, ou seja, para inciar o processamento de um trabalho é necessário o processamento de um ou mais trabalhos. Se $prec$ não aparece, então significa que os trabalhos são independentes entre si;

c) Recirculação ($rcrc$): significa que um trabalho pode ser processado em uma máquina mais de uma vez.

Por fim, acerca das funções objetivos, geralmente consideramos a minimização de uma ou mais características. Logo, podemos representar γ como:

a) *Makespan* (C_{max}): o tempo de término do último trabalho processado na última máquina. Considerando C_{ij} o tempo de término do trabalho j na máquina i, então $C_{max} = \max_{(i,j)\in O} C_{ij}$;

b) *Atraso máximo* (L_{max}): sendo $C_j = \max_{i\in M|(i,j)\in O} C_{ij}$, o término de processamento do trabalho j e d_j o prazo para o processamento da atividade j, então o atraso máximo pode ser caracterizado como $L_{max} = \max_{j\in J}(C_j - d_j)$;

c) Tempo total de conclusão ponderado ($\sum_{j\in J} w_j C_j$): sendo C_j definido anteriormente e w_j um peso associado ao trabalho $j \in J$, então minimizar $\sum_{j\in J} w_j C_j$ significa priorizar a conclusão de trabalho com peso maior.

Desse modo, o problema que abordaremos neste trabalho é $Jm||C_{max}$, conhecido como *job shop* clássico. Dessa forma, consideramos que há n trabalhos

$J_j(j = 1, \cdots, n)$ que devem ser processados em m máquinas $M_i(i = 1, \cdots, m)$, ou seja $|O| = mn$. Cada máquina pode processar um único trabalho por vez, cada trabalho pode ser processado em apenas uma máquina por vez e não há preempção, então se o processamento de um trabalho iniciou em uma máquina, então será concluído sem interrupção. O objetivo é sequenciar todos os trabalhos em todas as máquinas de modo que o tempo de conclusão do processamento do último trabalho na sua última máquina seja o menor possível.

Vale notar que quanto mais máquinas e trabalho tivermos, mais combinações de sequenciamento são possíveis. Então, surge a necessidade de métodos eficientes para a resolução desse problema do sequenciamento de trabalhos. O JSSP com função objetivo de minimização do *makespan*, como pode ser visto em Ku e Beck (2016), é NP-Difícil para pelo menos três máquinas e dois trabalhos. Assim, o esforço computacional pode ser elevado quando da resolução de instâncias reais do problema.

Desse modo, precisamos introduzir alguns parâmetros para facilitar a compreensão dos modelos mostrados futuramente.

a) $(\sigma_1^j, \cdots, \sigma_h^j, \cdots, \sigma_m^j)$ representa a ordem de execução de cada trabalho $j \in J$ nas máquinas, sendo σ_h^j a h-ésima máquina em que o trabalho j deve ser executado, $1 \leq h \leq m$;

b) p_{ij} tempo de execução do trabalho $j \in J$ na máquina $i \in M$;

c) $P_j = \sum_{i \in M} p_{ij}$ é o tempo total de processamento do trabalho $j \in J$ em todas as máquinas;

Para facilitar o entendimento, apresentamos um exemplo para um dado conjunto de parâmetros de entrada de uma instância do JSSP que pode ser encontrado em Piroozfard *et al.* (2016).

Trabalho	Tempo de processamento, máquina		
j_1	1, m_1	3, m_2	2, m_3
j_2	8, m_2	5, m_1	10, m_3
j_3	5, m_1	4, m_3	8, m_2
j_4	4, m_3	10, m_1	6, m_2

Tabela 1 – Parâmetros para um exemplo de instância do JSSP.

Em que:

a) $J = \{j_1, j_2, j_3, j_4\}$;

b) $M = \{m_1, m_2, m_3\}$;

c) $P_1 = 1+3+2 = 6$, $P_2 = 5+8+10 = 23$, $P_3 = 5+8+4 = 17$ e $P_4 = 10+6+4 = 20$;

d) $\sigma^1 = (m_1, m_2, m_3)$, ou seja, a ordem de execução do trabalho j_1 no conjunto das máquinas M é (m_1, m_2, m_3). Isso quer dizer que o trabalho j_1 deve primeiro ser executado na máquina m_1, depois em m_2 e, por fim, em m_3. A ordem dos trabalhos restantes é $\sigma^2 = (m_2, m_1, m_3)$, $\sigma^3 = (m_1, m_3, m_2)$ e $\sigma^4 = (m_3, m_1, m_2)$.

Um sequenciamento de execução para esses trabalhos é apresentado na Figura 1.

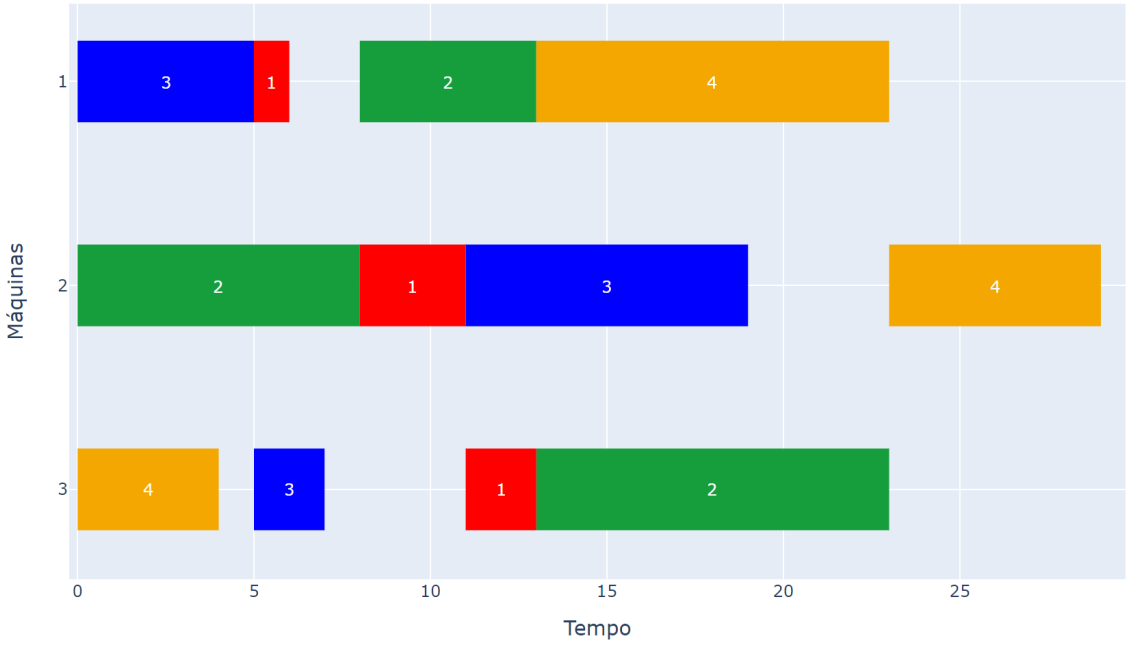

Figura 1 – Exemplo de sequenciamento de 4 trabalhos em 3 máquinas. Fonte: Adaptado de Piroozfard *et al.* (2016).

No exemplo acima, temos que o término de processamento do último trabalho executado é 29 unidades de tempo do início das operações, ou seja, $C_{max} = 29$. Notamos que, de fato, a ordem das máquinas de cada trabalho é respeitada e que existe uma permutação π^m de trabalhos para cada máquina.

a) $\pi^{m_1} = (j_3, j_1, j_2, j_4)$;

b) $\pi^{m_2} = (j_2, j_1, j_3, j_4)$;

c) $\pi^{m_3} = (j_4, j_3, j_2, j_1)$.

Vale ressaltar que a ordem de execução $(\sigma_1^j, \cdots, \sigma_m^j)$ nas m máquinas de um trabalho j é diferente da ordem dos trabalhos de uma máquina. As listas σ^j para todo $j \in J$ são parâmetros iniciais do problema. São ordens pré-estabelecidas de máquinas para cada trabalho. Já π^i para todo $i \in M$ são permutações de trabalhos encontradas para cada máquina, dada uma solução do problema. Por exemplo, o

trabalho j_1 deve ser executado primeiramente em m_1, porém isso não significa que j_1 deva ser o primeiro trabalho de m_1. Logo, podemos notar que existe diferença entre a ordem dos trabalhos em uma máquina m dada uma solução, representada por π^m, e a ordem pré-estabelecida das máquinas do trabalho j, representada por σ^j.

Tendo introduzido o JSSP, apresentamos a seguir o aparato metodológico que usamos como base para nosso trabalho. Iniciamos por apresentar uma revisão bibliográfica sobre algumas referências relativas ao JSSP que nos foram úteis, incluindo uma meta-heurística (Bissoli (2018)) recente para o JSSP e uma meta-heurística ELS (Merz e Wolf (2006)) que são combinadas para a construção de uma nova meta-heurística.

3 REVISÃO BIBLIOGRÁFICA

Neste capítulo serão explanados alguns trabalhos referentes aos problemas de sequenciamento. Discutiremos em mais detalhes sobre modelos de programação linear inteira na seção 3.1. Além disso, na seção 3.2 citamos o trabalho heurístico mais recente da literatura para o JSSP de Bissoli (2018) e um método heurístico ELS de Merz e Wolf (2006) que será adaptada para o JSSP.

Os métodos de resolução de problemas de sequenciamento, segundo Zhang *et al.* (2019), podem ser divididos de maneira mais ampla em basicamente dois tipos de abordagem: métodos de otimização exatos e métodos não exatos.

Os métodos de otimização exatos, como o próprio nome já sugere, são métodos que, com o tempo e capacidade computacional necessários, têm garantias de encontrar a solução ótima. Contudo, na prática, essas características podem ser grandes o suficiente para tornarem a busca por essa solução muito complexa de ser realizada. De qualquer modo, esses métodos podem ainda ser divididos em abordagem de regra eficiente, programação matemática e métodos de *branch and bound*.

Algumas abordagens de programação matemática serão descritas em detalhes na seção 3.1 deste Capítulo. Descreveremos quatro modelos para o JSSP abordados em Ku e Beck (2016): disjuntivo de Manne (1960), disjuntivo de Liao e You (1992), indexado por variáveis de tempo de Kondili *et al.* (1988) e baseado na ordem de Wagner (1959). Todos são modelos de programação linear inteira, porém abordam o problema por perspectivas diferentes.

Contudo, ainda sobre programação matemática, vale citar o estudo de Oliveira e Carravilla (2017). O trabalho apresentou um estudo da complexidade prática e analisou as características das instâncias que mais impactaram na resolução. Além disso, comparou os modelos de programação linear inteira e programação por restrições.

Para abordagens de regra eficiente, em geral, limita-se o problema para encontrar características que auxiliem a resolução. Em Johnson (1954) tratou-se do problema $F_2||C_{max}$, ou seja, o *flow shop* com duas máquinas e minimização do *makespan*. O artigo mostrou que existe um sequenciamento ótimo em que o trabalho j precede k se $\min\{p_{1j}, p_{2k}\} \leq \min\{p_{2j}, p_{1k}\}$. Além disso, o problema pode ser resolvendo em tempo de $O(n \log n)$.

Da perspectiva de métodos de *branch and bound* podemos citar Lomnicki (1965), que basicamente aborda o problema $J_3||C_{max}$. O artigo utiliza uma técnica de *branch and bound* baseada no artigo Little *et al.* (1963), que aborda o algoritmo da perspectiva do problema do caixeiro viajante.

Com a abordagem utilizando métodos não exatos, não temos garantias de que encontraremos o ótimo, contudo, geralmente, eles apresentam um comportamento de execução mais rápido do que em comparação aos métodos exatos. Como exemplo temos os métodos heurísticos, métodos de busca local e métodos construtivos.

Os métodos heurísticos geralmente são baseados em algum processo empírico ou lógica adaptada de algum processo intuitivo. Zhang *et al.* (2019) cita que algoritmo genético é um dos métodos heurísticos mais populares para problemas em geral. É baseado nos mecanismos de evolução genética da biologia. O primeiro artigo a utilizar essa técnica para buscar soluções para o JSSP foi Davis (1985), que indicou que a sequência de trabalhos para cada máquina pode ser resolvido com um algoritmo genético.

Outro método heurístico bastante popular é o algoritmo de colônia de formigas. É baseado no comportamento das formigas ao sair da colônia para procurar comida e deixar o rastro de feromônio. Colorni *et al.* (1994) utilizou pela primeira vez essa abordagem para resolver o JSSP.

Um método de busca local, segundo Zhang *et al.* (2019), geralmente consiste em aplicar uma estratégia de busca, baseada em uma ou mais funções, em um conjunto finito de soluções. Podemos citar o método do *simulated annealing*, que também é um método heurístico. Basicamente o método é baseado no processo de *annealing*, que consiste em superaquecer um material (metal, vidro, etc) e em seguida resfriar gradualmente para que o material torne-se rígido. Laarhoven *et al.* (1992) propôs a resolução do JSSP por este método e, de maneira geral, utilizou um grafo direcionado a partir das máquinas e trabalhos. Aplicou também o conceito de caminho crítico na definição da estrutura de vizinhança.

Por fim, segundo Zhang *et al.* (2019) os métodos baseados em heurísticas como *Shifting Bottleneck Process* são abordagens sofisticadas para encontrar o equilíbrio entre bons resultados em um bom tempo computacional. Esses métodos estão classificados no Zhang *et al.* (2019) como métodos aproximativos. Essa abordagem

foi aplicada para os problemas de sequenciamento pela primeira vez em Adams *et al.* (1988). No artigo os autores resolvem o problema $J_m||C_{max}$. Basicamente o problema é relaxado e decomposto em subproblemas para cada máquina. A cada iteração escolhe-se a ordem do processamento dos trabalhos na máquina escolhida. Assim, cada vez que uma nova máquina é sequenciada, então é realizada uma re-otimização.

Com isso, na próxima serão descreveremos em mais detalhes sobre os principais modelos de programação linear inteira para o JSSP.

.

3.1 Modelos matemáticos da literatura para o JSSP

Segundo Ku e Beck (2016), no ano de 2014, para resolver problemas de sequenciamento em geral, a programação inteira era o principal tema dos artigos do *Journal of Scheduling*, com 35% das publicações fazendo uso dessa técnica. Assim, mostra a relevância da abordagem usando métodos de programação inteira para a resolução desse tipo de problema. Com isso, tomamos como base os modelos apresentados no *survey* Ku e Beck (2016) para implementar e comparar com as ideias aqui propostas.

Existem diversos modelos de programação linear inteira para resolver o JSSP. Em Ku e Beck (2016) podemos encontrar os modelos citados neste trabalho. Para introduzir os modelos do JSSP, consideramos os parâmetros definidos no capítulo anterior e definimos como *big-M*, um valor suficientemente grande, a constante $P = \sum_{j \in J} \sum_{i \in M} p_{ij}$, pois no pior caso todos os trabalhos serão executados uma após o outro, sem nenhum paralelismo entre as máquinas.

3.1.1 Modelo disjuntivo de Manne (1960)

A principal característica do modelo de Manne (1960) é a utilização de variáveis binárias para representar a precedência de dois trabalhos. Tendo em vista essas variáveis, foram criadas restrições para garantir que se um trabalho k precede o trabalho j na máquina i, então o início de j deve ser pelo menos o término de k na mesma máquina. O mesmo ocorre caso j preceda k na máquina i. Vale ressaltar que essas restrições são para toda máquina e todo par de trabalhos j e k distintos, ou

seja são $mn(n-1)$ variáveis binárias. Além disso, o modelo usa variáveis contínuas para determinar o valor de início de um trabalho em uma máquina e uma variável para representar o *makespan*. De antemão, vale citar que a abordagem deAndrade *et al.* (2017) é semelhante à abordagem de Manne (1960) , pois ambos os modelos utilizam restrições disjuntivas para garantir a precedência.

As variáveis de decisão são:

- $x_{ij} \in \mathbb{R}_+$ variável contínua que representa o tempo de início do trabalho $j \in J$ na máquina $i \in M$;
- $z_{ijk} \in \{0,1\}$ variável binária que é 1 se o trabalho j preceder o trabalho k na máquina i e 0 caso contrário, para todo $i \in M$ e todo $j,k \in J, j < k$.

Com isso, pode-se apresentar o modelo a seguir:

$$(MANNE) \quad \min C_{max} \tag{3.1}$$

$$s.a \quad x_{\sigma_h^j j} \geq x_{\sigma_{h-1}^j j} + p_{\sigma_{h-1}^j j}, \quad \forall j \in J, h = 2, \cdots, m \tag{3.2}$$

$$x_{ij} \geq x_{ik} + p_{ik} - Pz_{ijk}, \quad \forall j,k \in J, j < k, \forall i \in M \tag{3.3}$$

$$x_{ik} \geq x_{ij} + p_{ij} - P(1 - z_{ijk}), \quad \forall j,k \in J, j < k, \forall i \in M \tag{3.4}$$

$$C_{max} \geq x_{\sigma_m^j j} + p_{\sigma_m^j j}, \quad \forall j \in J \tag{3.5}$$

$$x_{ij} \geq 0, \ \forall j \in J, \forall i \in M \tag{3.6}$$

$$z_{ijk} \in \{0,1\}, \ \forall j,k \in J, \forall i \in M \tag{3.7}$$

$$C_{max} \geq \max_{j \in J} P_j \tag{3.8}$$

Em que (3.1) é a função objetivo que indica minimizar o *makespan*. O conjunto de restrições (3.2) representa que a precedência de cada trabalho seja respeitada. Assim, um trabalho j iniciará em σ_h^j pelo menos quando terminar em σ_{h-1}^j. Os conjuntos de restrições (3.3) e (3.4) são as restrições disjuntivas que garantem que dois trabalhos não estejam na mesma máquina no mesmo período de execução. Ou seja, se $z_{ijk} = 0$, então sabemos que o trabalho k foi processado antes de j na máquina i. Assim, o tempo de início de j em i será pelo menos o término de processamento de k em i. Caso $z_{ijk} = 1$, então sabemos que j foi processado antes de k na máquina i, então o tempo de início de k em i será pelo menos o término de processamento de j em i. O conjunto de restrições (3.5) garante que o *makespan* seja pelo menos o tempo de término do último trabalho executado em sua última máquina. Assim, o tempo final de processamento total será maior que o tempo final de processamento de todos os

trabalhos em suas máquinas. O conjunto de restrições (3.6) garante que o tempo de início de todos os trabalhos seja sempre positivo. O domínio das variáveis z é garantido por (3.7). Por fim, (3.8) é um *upper bound* trivial para o problema, visto que, no melhor cenário nenhum trabalho ficará ocioso entre o processamento de uma máquina e outra, ou seja, não haverá nenhuma espera entre o processamento de um trabalho entre suas máquinas. Logo, como não é possível paralelizar o processamento de um trabalho em suas máquinas, pela definição do problema, então o *makespan* será pelo menos a soma dos tempos de processamento de cada trabalho.

3.1.2 Modelo disjuntivo de Liao e You (1992)

O segundo modelo disjuntivo é o de Liao e You (1992), que é basicamente uma adaptação do modelo disjuntivo de Manne (1960). Adicionam-se variáveis de folga q_{ijk} no conjunto de restrições (3.3) e isso reduz o número de restrições lineares. Assim, trocam-se as restrições disjuntivas (3.3) e (3.4) por uma restrição (3.9). A variável de folga $q_{ijk} \geq 0$ é contínua e tem o *upper bound* definido na restrição (3.10). Essa alteração foi diminui a quantidade de restrições, visto que para cada par de trabalhos em uma máquina é reduzida uma restrição em relação ao modelo de Manne (1960). A expressão (3.10) não contém as variáveis de decisão e P refere-se ao *big-M* citado no início do capítulo. Então, o modelo é representado como:

$$
\begin{aligned}
(LIAO) \quad & \min C_{max} \\
s.a \quad & (3.2), (3.5) - (3.8) \text{ e} \\
& Pz_{ijk} + (x_{ij} - x_{ik}) - p_{ik} = q_{ijk}, \quad \forall j,k \in J, j < k, \forall i \in M && (3.9) \\
& q_{ijk} \leq P - p_{ij} - p_{ik}, \quad \forall j,k \in J, j < k, \forall i \in M && (3.10)
\end{aligned}
$$

Notamos que se $z_{ijk} = 0$, ou seja, o trabalho k precede o trabalho j na máquina i, então o lado esquerdo das restrições do tipo (3.9) representa o tempo de espera entre o término do processamento de k na máquina i e o início de j na mesma máquina. Pela definição dos limites de q_{ijk}, essa expressão será maior que zero e menor do que um número suficientemente grande P. Portanto, garante que os trabalhos não serão executados ao mesmo tempo na máquina i. Caso $z_{ijk} = 1$, então podemos manipular matematicamente o conjunto de restrições (3.9) e o *upper bound* (3.10) e

concluiremos que o início de k na máquina i será pelo menos o término de j na máquina i. Isso também que os trabalhos não serão executados ao mesmo tempo.

3.1.3 Modelo indexado por variáveis de tempo de Kondili et al. (1988)

Kondili *et al.* (1988) adaptou a ideia original de Bowman (1959) para propor um modelo baseado na discretização do tempo com variáveis binárias para representar o processamento de um trabalho em uma máquina i no tempo t. Assim, para cada máquina e cada trabalho temos uma discretização do tempo. Para o exemplo da Figura 1 com 4 trabalhos e 3 máquinas o modelo de Kondili *et al.* (1988) poderia precisar de uma discretização de no mínimo 29 unidades de tempo. Notamos que esse modelo pode ter uma quantidade de variáveis bastante grande em comparação com outros.

A abordagem indexada por variáveis de tempo de Kondili *et al.* (1988) discretiza o tempo e verifica em que instante de cada máquina será executado cada trabalho. A discretização do tempo é dividida em intervalos com a mesma duração e podemos enumerá-los. Então, seja a discretização do tempo H, podemos fazer $H = \{0, 1, \cdots, P\}$, ou seja discretizar de modo que o tamanho de cada intervalo sendo igual a uma unidade de tempo. Obviamente, a discretização terá a duração de um *upper bound* conhecido, no exemplo o P, mais uma unidade, pois o instante zero é considerado. Porém, na prática, segundo Kondili *et al.* (1988), esse tamanho do intervalo será o maior divisor comum dos tempos de processamento, considerando que os tempos de processamento dos trabalhos nas máquinas são racionais. Por exemplo, se os tempos de processamento são $15, 5, 10$ e 20, então $P = 50$, e uma discretização com tamanho de cada intervalo em uma unidade seria $H = \{0, 1, 2, \cdots, 50\}$. Por outro lado, como o maior divisor comum é 5, então uma discretização com tamanho de cada intervalo sendo 5 seria $H' = \{0, 5, 10, \cdots, 50\}$. Ou seja, ao invés de uma discretização em uma unidade, em que $|H| = 51$, teríamos uma discretização de 5 em 5 unidades e $|H'| = 11$. Podemos chamar esse tamanho do intervalo de δ.

Além disso, podemos definir uma discretização $T_{ijt} = \{t - p_{ij} + \delta, \cdots, t\}$ que contém os tempos anteriores da execução do trabalho j na máquina i iniciando no tempo t, considerando um tamanho de intervalo δ. Por exemplo, se $t = 10$, $\delta = 1$ e o tempo de processamento do trabalho j na máquina i for $p_{ij} = 5$, então $T_{ij10} = \{6, 7, 8, 9, 10\}$.

Assim, definimos:

- $x_{ijt} \in \{0,1\}$ variável binária que tem valor igual a 1 quando o trabalho $j \in J$ inicia o processamento no tempo t na máquina $i \in M$ e 0 caso contrário.

Assim, podemos apresentar o modelo:

$$(KONDILI) \quad \min C_{max}$$

$$s.a \quad \sum_{t \in H} x_{ijt} = 1, \quad \forall j \in J, \forall i \in M \tag{3.11}$$

$$\sum_{t \in H} (t + p_{\sigma_{mj}^j}) x_{\sigma_{mt}^j} \leq C_{max}, \quad \forall j \in J \tag{3.12}$$

$$\sum_{j \in J} \sum_{t' \in T_{ijt}} x_{ijt'} \leq 1, \quad \forall i \in M, \forall t \in H \tag{3.13}$$

$$\sum_{t \in H} (t + p_{\sigma_{h-1}^j j}) x_{\sigma_{h-1}^j jt} \leq \sum_{t \in H} t x_{\sigma_h^j jt}, \quad \forall j \in J, h = 2, \cdots, m \tag{3.14}$$

$$x_{ijt} \in \{0,1\}, \quad \forall j \in J, \forall i \in M, \forall t \in H \tag{3.15}$$

O conjunto de restrições (3.11) garante que cada trabalho terá apenas um tempo de início t de processamento em cada máquina. O conjunto de restrições (3.12) garante que o *makespan* seja maior que o tempo de início do processamento t, quando $x_{\sigma_{mjt}^j} = 1$, mais o tempo de execução $p_{\sigma_{mj}^j}$ para todos os trabalhos e máquinas, ou seja, o *makespan* será no mínimo o término dos trabalhos em suas últimas máquinas. O conjunto de restrições (3.13) garante que cada máquina não inicie a mesmo trabalho j em mais de uma posição do intervalo T_{ijt}; ou seja, no intervalo da discretização, no máximo uma variável do conjunto $x_{ijt'}$ terá valor igual a um, logo não terá mais de um trabalho executando paralelamente na mesma máquina. O conjunto de restrições (3.14) garante a ordem de precedência em que cada trabalho é executado nas máquinas. Ou seja, o tempo de início de um trabalho j mais o tempo de execução dele deve ser menor que o tempo de início desse trabalho j na sua próxima máquina. Por fim, o conjunto de restrições (3.15) representa o domínio das variáveis. Vale ressaltar que esse modelo é puramente inteiro e que sua dimensão depende da discretização do tempo, o que pode gerar bastante variáveis.

3.1.4 Modelo baseado na ordem de Wagner (1959)

Wagner (1959) propôs uma modelo baseado na atribuição dos trabalhos nas posições das máquinas. Assim, cada máquina tem n posições em que devem ser alocadas os n trabalhos. O modelo contém: variáveis binárias para representar

se um trabalho j foi atribuído na posição k de uma máquina i; restrições comuns para problemas de atribuição, por exemplo restrições para garantir que o trabalho j seja alocado em uma posição da máquina; restrições para garantir a ordem das máquinas de um trabalho e para calcular o *makespan*. Assim, considere as variáveis e parâmetros:

- $x_{ijk} \in \{0,1\}$ variável binária que tem valor 1 quando o trabalho $j \in J$ for sequenciado na ordem k, com $k = 1, \cdots, n$, na máquina $i \in M$;
- $h_{ik} \in \mathbb{R}_+$ variável que indica o tempo de início do k-ésimo trabalho da máquina i, para $k = 1, \cdots, n$ e $i \in M$;
- $r_{ijk} \in \{0,1\}$ parâmetro que indica se a k-ésima operação do trabalho j requer a máquina i, para $i \in M$, $j \in J$ e $k = 1, \cdots, n$.

Para exemplificar o parâmetro r_{ijk}, podemos recorrer na instância referente à Tabela 1. Temos que $r_{112} = 1$, pois para o trabalho j_1 ser executado em sua segunda máquina, deve ter sido executado anteriormente na máquina 1. Podemos escrever o modelo matemático da seguinte forma:

$$(WAGNER) \qquad \min C_{max}$$

$$s.a \qquad \sum_{j \in J} x_{ijk} = 1, \quad \forall i \in M, k = 1, \cdots, n \tag{3.16}$$

$$\sum_{k=1}^{n} x_{ijk} = 1, \quad \forall j \in J, \forall i \in M \tag{3.17}$$

$$h_{ik} + \sum_{j \in J} p_{ij} x_{ijk} \leq h_{i,k+1}, \quad \forall i \in M, k = 1, \cdots, n \tag{3.18}$$

$$\sum_{i \in M} r_{ijl} h_{ik} + \sum_{i \in M} r_{ijl} p_{ij} \leq P(1 - \sum_{i \in M} r_{ijl} x_{ijk}) + P(1 - \sum_{i \in M} r_{ijl+1} x_{ijk'}) + \sum_{i \in M} r_{ijl+1} h_{ik'}, \quad \forall j \in J, \ k, k' = 1, \cdots, n \quad l = 1, \cdots, m-1 \tag{3.19}$$

$$h_{in} + \sum_{j \in J} p_{ij} x_{ijk} \leq C_{max}, \quad \forall i \in M \tag{3.20}$$

$$h_{ik} \geq 0, \quad \forall i \in M, k = 1 \cdots, n \tag{3.21}$$

$$x_{ijk} \in \{0,1\}, \quad \forall j \in J, \forall i \in M, k = 1, \cdots, n \tag{3.22}$$

O conjunto de restrições (3.16) garante que em cada posição de uma máquina seja alocada apenas um trabalho. O conjunto de restrições (3.17) garante que cada trabalho seja alocado em uma única posição de cada máquina i. O conjunto de restrições (3.18) garante que o tempo de início do trabalho processado na k-ésima posição da máquina i, mais o tempo de processamento desse trabalho executado na mesma máquina, deverá ser menor que o tempo de início do trabalho que será processado na $(h+1)$-ésima

posição da máquina i. O conjunto de restrições (3.19) garante que cada trabalho será executado seguindo a ordem σ^j. Vale ressaltar que os índices k e k' representam a ordem de execução dos trabalhos nas máquinas, ou seja $x_{ijk'}$ é a variável binária que indica se o trabalho j for o k'-ésimo processamento da máquina i. O lado esquerdo da desigualdade (3.19) é a soma do tempo de início mais o tempo de execução em todas as máquinas que são necessárias para o l-ésimo processamento de j; ou seja, é o tempo até o término do processamento do trabalho j na l-ésima posição de qualquer máquina. Caso a l-ésima operação de j tenha sido executado na ordem k em alguma máquina e a $(l+1)$-ésima operação de j seja executado na ordem k' em alguma máquina, então esse tempo representado pelo lado esquerdo da desigualdade será menor que o tempo inicial de processamento do $(l+1)$-ésimo processamento de j na k'-ésima ordem de uma máquina. Caso contrário, a restrição será redundante. Ou seja, o trabalho j apenas será processado na k'-ésima ordem da máquina σ_h^j caso tenha sido alocado na k-ésima posição da máquina σ_{h-1}^j. Como as restrições valem para todas as ordens k e k' das máquinas, para toda ordem $l = 1, \cdots, m-1$ dos trabalhos e todo trabalho $j \in J$, logo garante que todas as ordens σ^j sejam respeitadas. O conjunto de restrições (3.20) garante que o *makespan* seja maior que o tempo de início do n-ésimo trabalho de uma máquina; ou seja, seu último trabalho mais o tempo de processá-lo. Por fim, (3.21) e (3.22) são restrições que definem o domínio das variáveis de decisão do modelo.

É interessante observar as dimensões dos modelos apresentados anteriormente. Na Tabela 2, n é o número de trabalhos, m o número de máquinas e h o número de pontos da discretização.

Modelo	**Número de restrições**	**Número de variáveis binárias**
Manne (1960)	mn^2	$\frac{mn(n-1)}{2}$
Liao e You (1992)	$\frac{m(n^2+2n-1)}{2}$	$\frac{mn(n-1)}{2}$
Kondili *et al.* (1988)	$m(2n+h)$	nmh
Wagner (1959)	$m(n^3+3n+1)-n^3$	n^2m

Tabela 2 – Dimensões dos modelos disjuntivo Manne (1960), time-indexed Kondili *et al.* (1988) e rank-based Wagner (1959) para o JSSP.

Vale ressaltar que o modelo de Kondili *et al.* (1988) pode ter dimensões extremas se a discretização do espaço for demasiadamente grande.

Tendo apresentado os modelos mais referenciados na literatura do JSSP, agora apresentamos a mais recente meta-heurística para o mesmo, além de uma meta-heurística ELS e um procedimento que é a combinação delas.

3.2 Meta-heurística para o JSSP

Heurísticas são procedimentos desenvolvidos para a busca de soluções viáveis, preferencialmente a ótima. Geralmente são inspiradas em algum conceito empírico ou lógica adaptada de algum processo intuitivo. Contudo, enquanto as resoluções de modelos de programação matemática fornecem métricas para quantificar o quão longe a solução obtida encontra-se da solução ótima ou conseguem provar se uma solução é ótima quando limites inferiores e superiores para o valor ótimo coincidem, a utilização de heurísticas não garante que o ótimo do problema será encontrado e raramente fornecem uma noção de o quão distante o ótimo está da solução encontrada. Com essa característica, um dos objetivos ao se desenvolver uma heurística é ter um procedimento que encontre uma solução viável. Geralmente buscamos uma solução que seja pelo menos um ótimo local e que seja obtida em um tempo aceitável para o usuário. Por isso, heurísticas são, geralmente, bem específicas para cada problema. Na verdade, quanto mais se agrega conhecimentos específicos do problema para incrementar a heurística, melhor tendem a ser os resultados. Além disso, procedimentos mais sofisticados que combinam heurísticas, como por exemplo uma heurística para construção de uma solução viável e depois a aplicação de outros procedimentos para realizar uma busca local nessa solução, são chamados de meta-heurísticas.

Como já citado, o JSSP é um problema NP-Difícil para pelo menos três máquinas e dois trabalhos. Dessa maneira, a resolução de modelos inteiros para instâncias desse problema podem ser bastante custosas do ponto de vista computacional. Uma maneira de tentar melhorar essa resolução é obter limites superiores para a função objetivo, pois dessa maneira limitamos o espaço de busca que os algoritmos de programação inteira exploram. Logo, a utilização de heurísticas pode ser bastante útil para encontrar essas soluções viáveis que podemos utilizar como *upper bounds* de partida num resolvedor comercial, como por exemplo o IBM LOG CPLEX.

Em 3.2.3 utilizamos a combinação de duas meta-heurísticas que iremos

descrever na seção. A primeira é o GRASP para o JSSP e mostramos a abordagem de Bissoli (2018). Já a segunda é o ELS e mostramos uma abordagem baseada em Merz e Wolf (2006). Contudo, essa abordagem não é para o JSSP e por isso mostraremos de maneira genérica o procedimento.

3.2.1 Meta-heurística GRASP para o JSSP

Em Bissoli (2018) podemos encontrar uma aplicação da meta-heurística GRASP para o JSSP. Inicialmente a GRASP foi desenvolvida para o problema da cobertura de conjuntos (do inglês *set covering problem*). Basicamente, em Feo e Resende (1989) os autores adaptaram uma heurística *greedy*, adicionando um fator probabilístico. Dessa maneira, o procedimento tornou-se não determinístico. O GRASP genérico, ou seja para qualquer tipo de problema, informado em Bissoli (2018) pode ser visto em no Procedimento 1 com os parâmetros f representando a função que se deseja minimizar, $num\_it \in \mathbb{N}$ o número de iterações, $tam\_lista \in \mathbb{N}$ o tamanho da lista e $r \in \mathbb{N}$ uma semente para geração de números aleatórios.

Procedimento 1: GRASP genérico para problema de minimização

Entrada : f, num_it, tam_lista, r;
Saída : s^*: melhor solução encontrada;
Início
  $MIN = \infty$;
  Para *i de 1 até num_it* **faça**
    $s \leftarrow Construir\_solucao(tam\_lista, r)$;
    $s \leftarrow Buscar\_local(s)$;
    Se *s é viável e* $f(s) < MIN$ **então**
      $s^* \leftarrow s$;
      $MIN \leftarrow f(s)$;
    Fim
  Fim
  Retorne : s^*;
Fim

O procedimento funciona basicamente em duas etapas: construímos uma solução s, com algum fator aleatório, e aplicamos uma busca local passando a solução

construída s como ponto de partida. Caso essa solução s seja menor do que a última solução conhecida s^*, então atualizamos o valor de s^*. Essas etapas são repetidas de acordo com o número de iterações desejado. Contudo, devemos ter atenção para não utilizarmos um número de iterações demasiadamente alto, visto que o tempo de execução da heurística também aumentará. Vale ressaltar também que caso o problema seja de maximização, então devemos trocar a condição para atualizar a solução, além de alterar as funções de construção da solução e busca local.

A fase de construção da solução deve ser específica para cada problema, logo há características do JSSP que são incorporadas na meta-heurística. Para o JSSP, em Bissoli (2018), utiliza-se o conceito de operação, que basicamente é o processamento de um trabalho em uma máquina. Ou seja, existem mn operações para um problema com n trabalhos e m máquinas. Dessa forma, para cada iteração da fase de construção adiciona-se uma operação à solução de modo que ao final todas as operações sejam sequenciadas. Obviamente, a ordem em que essas operações são sequenciadas respeitam a ordem de precedência das máquinas de um trabalho. Com isso, em cada iteração existe uma lista de operações que podem ser sequenciadas, chamada lista de candidatos (LC) e representada por O_c. Após isso, é necessário selecionar um novo trabalho para ser adicionado no sequenciamento. Para isso, Bissoli (2018) define uma função $h^s(o)$ que retorna o *makespan* ao adicionar a operação o na solução s. Ademais, define-se:

$$\underline{o}^s = \min(h(o^s)|o \in O_c) \tag{3.23}$$

e

$$\overline{o}^s = \max(h(o^s)|o \in O_c) \tag{3.24}$$

Ou seja, $\underline{o}^s$ é a operação cujo o sequenciamento em s terá o menor *makespan* e $\overline{o}^s$ o maior *makespan*. Com isso, pode-se definir um parâmetro $\alpha \in (0,1)$ de modo a restringir LC a um conjunto menor da seguinte forma: $R^s = \{o^s \in O_c | \underline{o}^s \leq h(o^s) \leq \underline{o}^s + \alpha(\overline{o}^s - \underline{o}^s)\}$. Dessa maneira, tem-se um conjunto menor de operações que se pode ajustar de acordo com o parâmetro α. Dessa lista retrista de candidatos (LRC) R^s será selecionada uma operação aleatoriamente. Em Bissoli (2018), todas as operações terão a mesma probabilidade. Dessa forma, quando $\alpha = 0$, tem-se uma seleção totalmente gulosa, pois a operação selecionada será a que terá menor acréscimo na solução,

e quando $\alpha = 1$ tem-se uma solução totalmente aleatória, pois todas as operações que ainda não foram sequenciadas em s podem ser selecionadas. Geralmente α é escolhido empiricamente, mas pode variar de acordo com o interesse do usuário.

Dessa forma, basta atualizar a lista O_c a cada iteração e repetir o processo até que todos as operações sejam adicionadas à solução. Então, seja M o conjunto de máquinas, J o conjunto de trabalhos, O conjunto de operações, $tam\_lista \in \mathbb{N}$ o tamanho da lista e $r \in \mathbb{N}$ uma semente para geração de números aleatórios, o procedimento segue abaixo:

Procedimento 2: Construção da solução do GRASP para o JSSP

Entrada : M, J, O, tam_lista, r;

Saída : s: solução construída;

Início

Inicializa $\overline{O}$ com todas as operações;

Enquanto $\overline{O} \neq \emptyset$ **faça**

$O_C \leftarrow$ lista de operações que podem ser sequenciadas;

$R^s \leftarrow \{o^s \in O_c | \underline{o}^s \leq h(o^s) \leq \underline{o}^s + \alpha(\overline{o}^s - \underline{o}^s)\}$;

Selecionar uma operação o de R^s com a r;

Adicionar a operação o à solução s;

Excluir operação o de $\overline{O}$;

Fim

Retorne : s

Fim

Com uma solução construída dessa maneira não há garantias de que a solução seja ótima, nem mesmo localmente, ou seja, pode ser que uma simples troca de ordem de duas operações resulte em uma solução melhor. Logo, para tentar melhorar soluções sem muito esforço computacional, pois vale ressaltar que o objetivo da meta-heurística é encontrar soluções em tempo computacional aceitável, basta realizar uma busca local para verificar se há soluções melhores. Bissoli (2018) realiza a busca em cima do caminho crítico da solução, ou seja, do maior caminho entre o início do processamento dos trabalhos até o término do processamento de todos os trabalhos em todas as máquinas. Pensando na perspectiva de gerenciamento de projetos, em que os trabalhos do projeto são alocados para os responsáveis (no nosso contexto

seriam as máquinas) pelo desenvolvimento, o caminho crítico é entendido como o trabalho que é um gargalo, pois mesmo que todos os outros trabalhos anteriores a ele sejam adiantados, o projeto não poderá ser adiantado se aquele trabalho específico não for. Dessa maneira, Bissoli (2018) tenta realizar trocas em todos os pares consecutivos de operações no caminho crítico. Caso essa troca gere um *makespan* menor, então atualiza-se o caminho crítico e continua-se a busca por um novo caminho crítico. Caso contrário, ou seja, não há melhora na troca das operações do caminho crítico, então pula-se para a próxima operação. Esse processo é repetido até se percorrer todo o caminho crítico. Então, seja M o conjunto de máquinas, J o conjunto de trabalhos, O o conjunto de operações, s a solução construída e f: função que se deseja minimizar, segue abaixo o procedimento 3.

Procedimento 3: Busca local da solução do GRASP para o JSSP

Entrada : M, N, O, s, f;

Saída : s^*: solução após busca local;

Início

$s^* \leftarrow s' \leftarrow s$;

$c \leftarrow$ caminho crítico da solução s';

$o_c \leftarrow$ operações do caminho crítico c;

Para *i de 1 até (número de operações - 1) de* o_c **faça**

Trocar operação $o_c[i]$ com $o_c[i+1]$ em s';

Reconstruir s';

Se $f(s') < f(s^*)$ **então**

$s^* \leftarrow s'$

Fim

$s' \leftarrow s$;

Fim

Retorne : s^*;

Fim

3.2.2 Meta-heurística ELS

A ELS combina uma técnica de meta-heurísticas evolutivas e busca local em torno de uma solução atual. Essas meta-heurísticas evolutivas se inspiram no processo

de evolução biológica. Basicamente, há uma mutação ao acaso em um indivíduo, se essa mutação for benéfica, então as próximas gerações provenientes desse indivíduo terão esse benefício também. Caso contrário, então eventualmente esse indivíduo e as gerações provenientes dele serão extintas. É o princípio da seleção natural, indivíduos com características mais fortes dentre os demais tendem a perpetuar essa característica para as próximas gerações. Pensando do ponto de vista de otimização, podemos pensar em uma solução s como um indivíduo. A mutação ao acaso seria gerar uma perturbação aleatória nessa solução, ou seja, realizar alterações aleatórias nessa solução s. Se a solução gerada for melhor do que a existente, então deve-se atualizar essa solução com essa perturbação encontrada. Essas perturbações podem ser realizadas diversas vezes e, geralmente, define-se um parâmetro para a quantidade de mutações que devem ser realizadas. Essa perturbação realizada em cima de uma solução inicial, para o JSSP, pode ser feita trocando a ordem dos trabalhos de uma máquina e recalculando o tempo de início de cada trabalho em cada máquina, obviamente a ordem de precedência das máquinas de um trabalho deve ser respeitado.

Já o procedimento de busca local, como o próprio nome já diz, é uma busca em soluções vizinhas da solução atual e retorna a melhor delas. Esse conceito de solução vizinha é específico para cada problema. Para o JSSP podemos utilizar o mesmo conceito descrito em Bissoli (2018).

Então, basicamente dada uma solução atual s, para um determinado número de iterações e de mutações, é realizada uma duplicação s' da solução s. Dessa maneira, para cada cópia s', uma perturbação aleatória é realizada, seguida por uma busca local para tentar melhorar s'. Então, se a função objetivo aplicada em s' for melhor do que em s, então a solução atual é atualizada, ou seja, $s \leftarrow s'$. Então, seja f: função que se deseja minimizar, $num\_it \in \mathbb{N}$ o número de iterações, $num\_mt \in \mathbb{N}$ o número de mutações, $r \in \mathbb{N}$ e s uma solução viável, uma semente para geração de números aleatórios, podemos ver o Procedimento 4 a seguir.

Procedimento 4: ELS

Entrada : f, num_it, num_mt, r, s;

Saída : s^*: melhor solução encontrada;

Início

$MIN = f(s)$;

Para *i de 1 até* num_it **faça**

Para *j de 1 até* num_mt **faça**

$s' \leftarrow s$

$s' \leftarrow Perturbar(s', r)$;

$s' \leftarrow Busca\_local(s')$;

Se s' *é viável e* $f(s') < MIN$ **então**

$s \leftarrow s'$;

$MIN \leftarrow f(s)$;

Fim

Fim

Fim

Retorne : s^*;

Fim

Note que para a utilização dessa meta-heurística é necessário uma solução inicial, visto que ela trabalha com uma perturbação da solução corrente. Dessa maneira, a seguir mostramos uma combinação do GRASP e ELS, de modo que a solução encontrada do GRASP será utilizada como ponto de partida do ELS.

3.2.3 Meta-heurística GRASP-ELS para o JSSP

Na resolução de modelos de programação inteira, podemos limitar superiormente o espaço de busca com *upper bounds* para a função objetivo. Dessa forma, usamos uma combinação de GRASP com ELS, doravante chamada de GRASP-ELS, para encontrar uma solução viável e utilizar esse valor como *upper bound* do *makespan*.

A estrutura do GRASP será mantida como em 3.2.1 e no procedimento 1. Inicialmente construiremos uma solução com um fator aleatório e depois aplicaremos uma busca local para melhorar essa solução. Essa solução possivelmente melhorada com a busca local será a solução inicial da meta-heurística ELS, descrita em 3.2.2 e no

procedimento 4. Basicamente a solução atual é perturbada e depois é realizada uma nova busca local. Definindo os parâmetros sendo f a função que se deseja minimizar, $num\_it\_grasp \in \mathbb{N}$ o número de iterações do GRASP, $num\_it\_els \in \mathbb{N}$ o número de iterações do ELS, $num\_mt \in \mathbb{N}$ o número de mutações do ELS, $\alpha \in \mathbb{N}$ o parâmetro de aleatoridade e $r \in \mathbb{N}$ uma semente para geração de números aleatórios, então podemos ver o pseudocódigo dessa combinação do GRASP-ELS no Procedimento 5.

Procedimento 5: GRASP ELS genérico para problema de minimização

Entrada : f, num_it_grasp, num_it_els, num_mt, α, r;

Saída : s^*: melhor solução encontrada;

Início

$MIN = \infty$;

Para *i de 1 até* $numero\_it\_grasp$ **faça**

$s \leftarrow Construir\_solucao(\alpha, r)$;

$s \leftarrow Busca\_local(s)$;

$s \leftarrow ELS(f, s, numero\_it\_els, r)$;

Se *s é viável e* $f(s) < MIN$ **então**

$s^* \leftarrow s$;

$MIN \leftarrow f(s)$;

Fim

Fim

Retorne : s^*;

Fim

A construção de uma solução é uma adaptação do procedimento de Aiex *et al.* (2003) para o JSSP. É uma heurística construtiva que começa com uma solução vazia e na qual cada operação, ou seja, processamento de um trabalho em uma máquina, é adicionado aleatoriamente a cada iteração usando a LRC, até que uma solução completa seja obtida. Sejam O_c o conjunto de operações que podem ser adicionadas à solução parcial atual s, respeitando obrigatoriamente a precedência das operações nas máquinas; o^s o acréscimo da operação o à solução s; $h(o^s)$ a função que retorna o *makespan* ao acrescentar a operação o na solução s , $\underline{o}^s$ a operação que retorna o menor *makespan* quando adicionada à s, ou seja $\underline{o}^s = \min(h(o^s)|o \in O_c)$; e $\overline{o}^s$ a operação que retorna o maior *makespan* quando adicionada à s, ou seja $\overline{o}^s =$

$\max(h(o^s)|o \in O_c)$. Então, a LRC é $R^s \leftarrow \{o^s \in O_c | \underline{o}^s \leq h(o^s) \leq \underline{o}^s + \alpha(\overline{o}^s - \underline{o}^s)\}$. Ou seja, todas as operações quando adicionadas à s estão dentro da combinação linear de $\underline{o}^s$ e $\overline{o}^s$. Vale ressaltar que quanto menor o α o procedimento tende a ser guloso e quanto maior o α o procedimento tende a ser aleatório.

A busca local é baseada em Nowicki e Smutnicki (2005). O procedimento, dada uma solução viável s, consiste em escanear o caminho crítico de s, o maior caminho entre o início do processamento dos trabalhos até o término do processamento de todos os trabalhos em todas as máquinas, desde a última operação até a primeira. Cada vez que um arco disjuntivo é encontrado, ou seja, dois processamentos consecutivos de trabalhos diferentes na mesma máquina, troca-se as operações, uma nova solução s' é obtida e o *makespan* correspondente é avaliado. Se o *makespan* de s' for melhor do que o atual, ele será atualizado e a busca local começará novamente a partir do final do novo caminho crítico. Caso contrário, a busca local continua até percorrer todas as operações.

Após a construção da solução e a busca local, então será aplicado o procedimento ELS, baseado em Merz e Wolf (2006).

No próximo Capítulo apresentamos um novo modelo para o JSSP baseado no problema do arranjo linear.

4 MODELO MINLA PARA O JSSP

Neste capítulo apresentamos um novo modelo para o JSSP com base em modelagem de arranjo linear, denominado como (MLJ). Antes de introduzir esse novo modelo, apresentamos a definição do MinLA e o modelo de Andrade *et al.* (2017).

O MinLA consiste em atribuir rótulos distintos $\pi_{v_1}, ..., \pi_{v_n}$ a todos os n vértices de um grafo $G = (V, E)$, com $V = \{v_1, ..., v_n\}$ e π_v, para todo $v \in V$, pertencente a $\{1, ..., n\}$, de forma a minimizar a soma dos pesos das arestas do grafo, sendo que o peso de uma aresta $\{i, j\}$ em E é a diferença absoluta $|\pi_i - \pi_j|$ dos valores dos rótulos atribuídos aos vértices i e j dessa aresta. É um problema essencialmente teórico, porém na tese Seitz (2010) tem-se aplicações como desenho de grafos, representação de modelo de entidade relacional e diagramas de fluxo de dados.

Definição 4.0.1 *Seja $G = (V, E)$ um grafo simples não orientado, com V representando o conjunto de vértices e $E \subseteq V \times V$ o conjunto de arestas e $\pi : V \to \{1, 2, \cdots, |V|\}$ uma atribuição de rótulos aos vértices. O problema do arranjo linear mínimo MinLA é a atribuição de rótulos distintos π aos nós de modo que a soma dos pesos das arestas dada por $\sum_{\{ij\} \in E} |\pi_i - \pi_j|$ seja a menor possível.*

Por exemplo, na Figura 2 podemos verificar em (a) um grafo simples, em (b) uma atribuição de rótulos aos nós e, consequentemente, os pesos para suas arestas e, por fim, em (c) podemos ver a atribuição ótima.

(a) Um grafo simples G. (b) Uma rotulação não ótima. (c) Uma rotulação ótima.

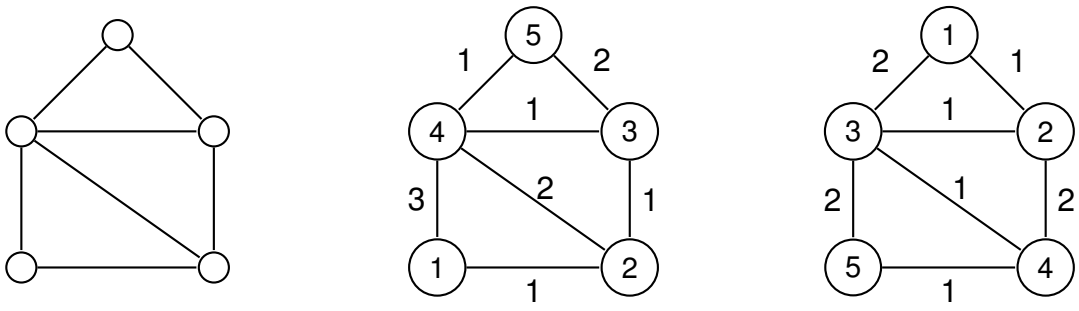

Figura 2 – Exemplo de um grafo G, uma atribuição não ótima e ótima para o MinLA.
Fonte: Ferreira (2016)

Cada atribuição pode ser vista como uma permutação dos vértices, de modo que a posição corresponde a seu rótulo. Por exemplo, para facilitar a verificação da

atribuição, podemos nomear os vértices do grafo (a) da Figura 2 como segue na Figura 3.

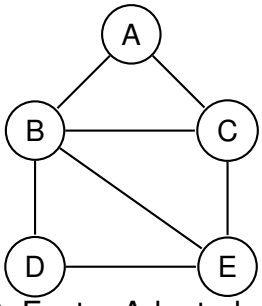

Figura 3 – Grafo G. Fonte: Adaptado de Ferreira (2016)

Dessa forma, seguindo a ordem da atribuição ótima vista em (c) da Figura 2, podemos mostrar o grafo da seguinte forma:

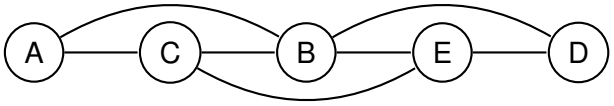

Figura 4 – Grafo G em outra perspectiva. Fonte: Adaptado de Ferreira (2016)

Logo, podemos ver a permutação $\pi_A = 1$, $\pi_B = 3$, $\pi_C = 2$, $\pi_E = 4$ e $\pi_D = 5$. Notamos que qualquer outra permutação gera uma atribuição válida, mesmo que não seja a ótima, e a permutação simétrica $\pi_D = 1$, $\pi_E = 2$, $\pi_B = 3$, $\pi_C = 4$ e $\pi_A = 5$ resulta no mesmo valor da ponderação das arestas.

O problema do arranjo linear mínimo foi recentemente tratado com uma formulação eficiente do ponto de vista computacional em Andrade *et al.* (2017). Nesse artigo são apresentadas uma formulação não-linear e sua versão linearizada. Esta última mostrou melhor performance computacional que outras modelagens existentes para o problema. Essa nova formulação tem $\mathcal{O}(|V|^2)$ restrições e variáveis.

4.1 Modelo do arranjo linear mínimo de Andrade *et al.* (2017)

Para introduzir essa modelagem do MinLA, considere um grafo como o da definição 4.0.1. Construímos um grafo completo e orientado $D = (V,A)$ obtido de G com $A = \{uv \mid u,v \in V, u \neq v\}$. Considere o conjunto $A(E) \subseteq A$ de arcos de D em que $uv, vu \in A(E) \Leftrightarrow \{uv\} \in E$.

Definimos: para cada $v \in V$, uma variável π_v representando o rótulo do

vértice v; e variáveis binárias $x \in \{0,1\}^{|A|}$, em que x_{uv} é 1 se o arco uv pertencer à solução ou, equivalentemente, se $\pi_u < \pi_v$. Por fim, o peso do arco uv será dado por uma variável $w_{uv} \in \mathbb{R}_+$, para todo $uv \in A$.

Então, o modelo para o MinLA de Andrade *et al.* (2017) é:

$$(Q_1) \quad \min \sum_{uv \in A(E)} w_{uv} x_{uv} \tag{4.1}$$

$$s.a \quad x_{uv} + x_{vu} = 1, \forall uv \in A \tag{4.2}$$

$$\pi_v - \pi_u \leq w_{uv} + n(1 - x_{uv}), \forall uv \in A \tag{4.3}$$

$$\pi_v - \pi_u \geq w_{uv} - n(1 - x_{uv}), \forall uv \in A \tag{4.4}$$

$$1 \leq \pi_v \leq n, \forall v \in V \tag{4.5}$$

$$1 \leq w_{uv} \leq n - 1, \forall uv \in A \tag{4.6}$$

$$x_{uv} \in \{0,1\}, \forall uv \in A \tag{4.7}$$

Em que (4.1) é a função objetivo, minimizando a soma dos pesos dos arcos associados às arestas do grafo original G, dado pelo conjunto $A(E)$. As restrições (4.2) garantem que para quaisquer dois vértices de D, então o rótulo de um será obrigatoriamente maior que o rótulo do outro. As restrições (4.3) e (4.4) garantem que se x_{uv} pertencer à solução, então $w_{uv} = \pi_v - \pi_u$, caso contrário essas restrições tornam-se redundantes. Por fim, as restrições (4.5), (4.6) e (4.7) representam o domínio de cada variável.

O modelo acima é quadrático. Andrade *et al.* (2017) mostraram como linearizá-lo, bem como propõem várias desigualdades válidas para o mesmo, obtendo assim o seguinte modelo linear inteiro misto. A interpretação de suas restrições será feita à medida que o adaptarmos para o JSSP mais adiante. Para mais detalhes sobre a linearização, consultar Andrade *et al.* (2017).

$$(Q_2) \quad \min z = \sum_{uv \in A(E)} w_{uv}$$

$$s.a \quad (4.2), (4.5), (4.7) \text{ e}$$

$$\pi_v - \pi_u \leq w_{uv}, \quad \forall uv \in A \tag{4.8}$$

$$\pi_v - \pi_u \geq w_{uv} - n(1 - x_{uv}), \quad \forall uv \in A, \tag{4.9}$$

$$z \geq |E| + \sum_{i=1}^{p} (|V| - i)i + \overline{p}(p+1) \tag{4.10}$$

$$x_{uv} + x_{vk} + x_{ku} \leq 2, \quad \forall uv \in A, k \in V \setminus \{u, v\} \tag{4.11}$$

$$\sum_{uv \in A} w_{uv} = \frac{|V|(|V|-1)(|V|+4)}{6} \tag{4.12}$$

$$\pi_u + \sum_{uv \in A} x_{uv} = |V|, \quad \forall u \in V \tag{4.13}$$

$$\pi_u - \sum_{vu \in A} x_{vu} = 1, \quad \forall u \in V \tag{4.14}$$

$$w_{uv} + \sum_{tu \in A} x_{tu} \leq |V|, \quad \forall uv \in A \tag{4.15}$$

$$w_{uv} + w_{vu} \leq w_{vk} + w_{kv} + w_{ku} + w_{uk} - 1, \quad \forall uv \in A, k \in V \setminus \{u, v\} \tag{4.16}$$

$$1 \leq w_{uv} \leq 1 + (|V| - 2)x_{uv}, \quad \forall uv \in A \tag{4.17}$$

Sendo p, em (4.10), o maior número natural tal que $\sum_{i=1}^{p} i \leq |E|$ e $\overline{p} = |E| - \sum_{i=1}^{p} (|V| - i)$. Vale ressaltar que essa restrição representa um *lower bound* (LB) para a função objetivo.

Com base nas ideias anteriores, introduzimos a seguir o novo modelo para o JSSP, denominado (MLJ).

4.2 Modelo MLJ

Como no JSSP existem as permutações dos trabalhos nas máquinas e o modelo do MinLA {(4.2),(4.5)-(4.17)} contém desigualdades fortes para o problema de permutação, então é razoável adaptar essas desigualdades para o modelo disjuntivo de Manne (1960). Dessa forma, o esperado é obter uma relaxação linear mais forte do modelo {(3.1) - (3.8)} e consequentemente melhorar o tempo de execução ou diminuir o *gap* de integralidade na resolução de instâncias do JSSP. Além disso, pode-se desenvolver novas restrições para melhorar os aspectos citados anteriormente.

Em particular, fazendo a conexão do JSSP com o MinLA, podemos imaginar cada máquina como sendo um grafo completo, onde cada trabalho é um vértice.

Nota-se que uma enumeração dos vértices do grafo associado a cada máquina pode ser interpretada como a ordem (permutação) dos trabalhos nessa máquina. Para exemplificar, dado o problema mostrado na Figura 1, podemos representar o grafo para a máquina 1 na Figura 5.

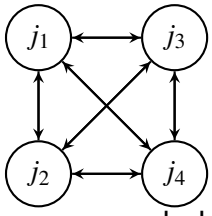

Figura 5 – Interpretação do MinLA para uma dada máquina no JSSP em relação ao problema da Figura 1. Fonte: autor.

A permutação de trabalhos para cada máquina, a variável z_{ijk} do modelo disjuntivo para uma dada máquina i é equivalente à variável x_{uv} do MinLA, desconsiderando-se o índice da máquina i. Ou seja, z_{ijk} é 1 se o trabalho j for processado antes de k na máquina i. Assim, todas as desigualdades válidas do MinLA que utilizam a variável x podem ser adaptadas para o JSSP.

Além disso, a variável π_u, que representa o rótulo ou a posição do vértice u, também pode ser estendida para cada máquina. Ou seja, π_{ij} é a posição do trabalho j na máquina i. Porém, utilizando a igualdade (4.14), temos que $\pi_u = 1 + \sum_{vu \in A} x_{vu}$. Então, a variável π_{ij} pode ser substituída por:

$$\pi_{ij} = 1 + \sum_{u \in J | u < j} z_{iuj} + \sum_{u \in J | u > j} (1 - z_{iju}) \tag{4.18}$$

Dessa forma, também podemos utilizar as desigualdades válidas que contém a variável π para fortalecer o modelo. Obviamente na implementação das restrições utilizaremos a expressão acima para não aumentar o número de variáveis do modelo disjuntivo de Manne (1960). Contudo, para facilitar a notação e entendimento das restrições apresentadas, mostraremos as desigualdades com a variável π_{ij} ao invés da expressão. Vale ressaltar que a substituição acima pode ser feita pelo fato de as restrições (3.3) e (3.4) capturarem o valor das variáveis z. No caso do MinLA, a própria variável π_u é utilizada para isso. Então não é necessário fazer essa substituição.

A variável w_{uv} também pode ser estendida para cada máquina, tornando-se w_{ijk} e representaria o número de trabalhos executados entre j e k na máquina i. Vale ressaltar que no JSSP a variável z_{ijk} está definida para todo $i \in M$ e $j,k \in J$, ou seja,

trabalhos distintos. Logo, é necessário que nas restrições seja utilizada essa variável para modelar os casos disjuntivos. Dessa maneira, temos w_{ijk} definida para todo $i \in M$ e $j,k \in J$. Diferentemente do MinLA que utiliza nas restrições a variável x_{uv} para todo $uv \in A$, no JSSP a variável z_{ijk} não está definida para todas as combinações de trabalhos, está definida apenas para as combinações distintas. Logo, é necessário garantir ambos os casos nas restrições. Quando j precede k na máquina i, temos que:

$$w_{ijk} \geq \pi_{ik} - \pi_{ij} - Pz_{ijk}, \quad \forall i \in M, \forall j,k \in J, j < k \tag{4.19}$$

$$w_{ijk} \leq \pi_{ik} - \pi_{ij} + Pz_{ijk}, \quad \forall i \in M, \forall j,k \in J, j < k \tag{4.20}$$

De maneira análoga, temos para quando k precede j na máquina i:

$$w_{ijk} \geq \pi_{ij} - \pi_{ik} - P(1 - z_{ijk}), \quad \forall i \in M, \forall j,k \in J, j < k \tag{4.21}$$

$$w_{ijk} \leq \pi_{ij} - \pi_{ik} + P(1 - z_{ijk}), \quad \forall i \in M, \forall j,k \in J, j < k \tag{4.22}$$

Mesmo sendo possível fazer a adaptação da variável w, isso só é razoável com o auxílio de restrições para abranger os casos em que j precede k na máquina i e k precede j na máquina i. Ou seja, em um somatório não seria possível fazer essa distinção sem uma variável auxiliar, o que elevaria a quantidade de variáveis do modelo e poderia ser bastante pesado para sua resolução. Logo, optou-se por não adaptar essas restrições. Dessa forma, com todas as variáveis devidamente transformadas, podemos analisar as restrições do MinLA e avaliar quais podem ser transformadas para o JSSP.

Inicialmente, temos o conjunto de restrições (4.2). Contudo, como no JSSP a variável z_{ijk} é definida para trabalhos distintos, então a adaptação dessa restrição não faz sentido. Note que a transformação seria $z_{ijk} + (1 - z_{ijk}) = 1$ e isso, claramente, após a manipulação não restará variável, logo não poderá ser uma restrição.

Já as restrições (4.8) e (4.9), que são utilizadas no MinLA para garantir o valor de w_{uv}, não serão adaptadas ao modelo do JSSP, visto que a variável w_{ijk} não será, de fato, utilizada.

A restrição (4.10) também não será adaptada, visto que é um *lower bound* da função objetivo do MinLA. Dessa maneira, como a função objetivo do JSSP é diferente, não faz sentido o mapeamento da desigualdade.

A desigualdade (4.11) também pode ser modelada, contudo vale ressaltar que como a variável z_{ijk} está definida para todo $i \in M$ e $j,k \in J, j < k$, ou seja, trabalhos j e k distintos, então é necessário modelar os casos em que a variável u é $u < j < k$, $j < u < k$ e $j < k < u$ para quando j preceder k na máquina i e quando k preceder j na máquina i. Então, essa desigualdade modela o fato de que dados três trabalhos distintos, deve haver no máximo duas precedências, ou seja, se j precede k e k precede u, então obrigatoriamente u não pode preceder j ou k. Isso pode ser modelado como:

$$z_{iuj} + z_{ijk} - 1 \leq z_{iuk}, \quad \forall i \in M, \forall u, j, k \in J, u < j < k \tag{4.23}$$

$$z_{iuk} + (1 - z_{ijk}) - 1 \leq z_{iuj}, \quad \forall i \in M, \forall u, j, k \in J, u < j < k \tag{4.24}$$

$$(1 - z_{iju}) + z_{ijk} - 1 \leq z_{iuk}, \quad \forall i \in M, \forall u, j, k \in J, j < u < k \tag{4.25}$$

$$z_{iuk} + (1 - z_{ijk}) - 1 \leq (1 - z_{iju}), \quad \forall i \in M, \forall u, j, k \in J, j < u < k \tag{4.26}$$

$$(1 - z_{iju}) + z_{ijk} - 1 \leq (1 - z_{iku}), \quad \forall i \in M, \forall u, j, k \in J, j < k < u \tag{4.27}$$

$$(1 - z_{iku}) + (1 - z_{ijk}) - 1 \leq (1 - z_{iju}), \quad \forall i \in M, \forall u, j, k \in J, j < k < u \tag{4.28}$$

As restrições (4.12), (4.15), (4.16) e (4.17) não serão modeladas para o JSSP, visto que utilizam a variável w_{uv} do MinLA e essa variável não será definida para o JSSP.

O conjunto de restrições (4.13) não será modelado para o JSSP, visto que utilizamos uma expressão para representar as variáveis π e, dessa maneira, após manipulações matemática podemos concluir que são expressões que resultam em uma identidade. Note que a intepretação da restrição (4.13) do MinLA é que a posição do vértice $u \in V$ mais a quantidade de vértices que tem uma posição maior que u é igual ao número de vértices. Dessa maneira, a adaptação ao JSSP significa que para cada máquina $i \in M$ e $j \in J$, então a posição de execução do trabalho j na máquina i mais a quantidade de trabalhos que j precede em i é igual ao número de máquinas. Portanto, temos que:

$$\pi_{ij} + \sum_{u \in J | u < j} (1 - z_{iuj}) + \sum_{u \in J | u > j} z_{iju} = n, \quad \forall i \in M, \forall j \in J \tag{4.29}$$

Contudo, substituindo π_{ij} pela expressão de (4.18) em (4.29) temos:

$$1+\sum_{u\in J|u<j} z_{iuj}+\sum_{u\in J|u>j}(1-z_{iju})+\sum_{u\in J|u<j}(1-z_{iuj})+\sum_{u>j} z_{iju}=n,\quad \forall i\in M,\forall j\in J \quad (4.30)$$

Claramente temos que a expressão (4.30) resulta em uma identidade, por isso, (4.13) não será adaptada para o JSSP.

Podemos pensar de maneira análoga para a restrição (4.14). A interpretação dessa restrição para o MinLA é que a posição do vértice $u\in V$ subtraído da quantidade de vértices com posições menores que u será um. Note que no JSSP essa restrição significará que para todo trabalho $j\in J$ e toda máquina $i\in M$, a posição do trabalho j em i subtraída de todos os vértices que antecedem j em i será um, ou seja:

$$\pi_{ij}-\sum_{u\in J|u<j} z_{iuj}-\sum_{u\in J|u>j}(1-z_{iju})=1,\quad \forall i\in M,\forall j\in J \quad (4.31)$$

Substituindo π_{ij} definida em (4.18) em (4.31), vemos que:

$$1+\sum_{u\in J|u<j} z_{iuj}+\sum_{u\in J|u>j}(1-z_{iju})-\sum_{u\in J|u<j} z_{iuj}-\sum_{u\in J|u>j}(1-z_{iju})=1,\quad \forall i\in M,\forall j\in J \quad (4.32)$$

Temos claramente que a restrição resultará em uma identidade. Portanto, também não será utilizada no JSSP.

As expressões (4.33) para quando o trabalho j precede k na máquina i e (4.34) para quando o trabalho k precede j na máquina i são suficientes para definir uma permutação na nossa adaptação do MinLA para o JSSP, visto que as expressões garantem que a diferença de rótulos seja no mínimo igual a 1 pelo fato de o grafo ser completo.

$$-n\cdot(1-z_{ijk})+1\leq \pi_{ik}-\pi_{ij},\quad \forall i\in M,\forall j,k\in J, j<k \quad (4.33)$$

$$-nz_{ijk}+1\leq \pi_{ij}-\pi_{ik},\quad \forall i\in M,\forall j,k\in J, j<k \quad (4.34)$$

Ainda adaptando conceitos do MinLA para o JSSP, podemos modelar $\sum_{j\in J}\pi_{ij}$ para toda máquina $i\in M$. Note que π_{ij} é a posição de execução do trabalho j na máquina i, então será sempre, independente da ordem, o somatório de uma progressão

aritmética com o primeiro elemento sendo 1 e a razão sendo 1. Dessa maneira, temos que:

$$\sum_{j \in J} \pi_{ij} = \frac{n(n+1)}{2}, \quad \forall i \in M \tag{4.35}$$

Pode-se observar que a restrição acima é válida para $n \geq 3$. De qualquer modo, isso não é um problema, pois para uma quantidade inferior de trabalhos o JSSP é um problema trivial.

Logo, após testes preliminares selecionamos as restrições que de fato ajudam na melhora da relaxação linear. Então, com base nas ideias anteriores, introduzimos a seguir o novo modelo para o JSSP, denominado (MLJ).

$$\begin{aligned} (MLJ) \quad & \min C_{max} \\ s.a \quad & (3.2) - (3.8), (4.23) - (4.28), (4.33) - (4.35) \end{aligned} \tag{4.36}$$

Podemos adicionar ao modelo acima desigualdades válidas, assim como nos demais modelos da literatura, para melhorar a sua relaxação linear. No próximo capítulo descrevemos uma abordagem da estrutura do JSSP para a construção de desigualdades válidas.

5 NOVAS DESIGUALDADES VÁLIDAS PARA O JSSP

Como uma das contribuições deste trabalho, nesta seção introduzimos novas desigualdades válidas para os modelos disjuntivos do JSSP.

Inicialmente apresentamos definições de novos parâmetros usados nas desigualdades. Em seguida, introduzimos novas desigualdades válidas para o problema com base na estrutura ótima de sua solução. Mais precisamente, no acúmulo dos tempos de processamento a *priori* e a *posteriori* da execução de um trabalho em uma dada máquina.

Antes de desenvolvermos as novas restrições é necessário introduzir novos parâmetros para ajudar no seu entendimento. Definimos $\sigma^{j^{-1}}(i)$ uma função que retorna a posição da máquina i em σ_j, ou seja, a posição de execução da máquina i para o trabalho j. Por exemplo, seja $\sigma^j = (m_3, m_1, m_2)$. O trabalho j executará primeiro na máquina m_3, depois na m_1 e, por fim, na m_2. Então, $\sigma^{j^{-1}}(m_1) = 2$, pois a máquina m_1 é a segunda máquina em que o trabalho j deve ser executado, ou seja $\sigma_2^j = m_1$. Nesse caso,

$$\sigma^{k^{-1}}(i) = h \leftrightarrow \sigma_h^k = i$$

Então, podemos definir novos parâmetros.

- P_{ij}^{-} representa a soma dos tempos de processamento do trabalho j na máquina i e em todas as máquinas que antecedem i:

$$P_{ij}^{-} = \sum_{r=1}^{\sigma^{j^{-1}}(i)} p_{\sigma_r^j j}$$

- P_{ij}^{+} representa a soma dos tempos de processamento do trabalho j na máquina i e em todas as máquinas que sucedem i:

$$P_{ij}^{+} = \sum_{r=\sigma^{j^{-1}}(i)}^{m} p_{\sigma_r^j j}$$

Para um melhor entendimento, ilustramos a ideia desses parâmetros na Figura 6.

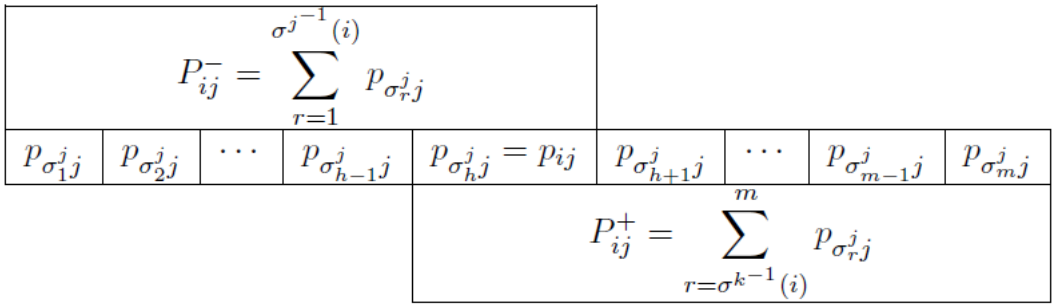

Figura 6 – Ilustração de P_{ij}^{+} e P_{ij}^{-}. Note que $P_{ij}^{-} + P_{ij}^{+} - p_{ij} = P_j$.

Para exemplificar, dado o problema da Tabela 1, podemos ver que:

- P_{14}^{-} é a soma do tempo de processamento do trabalho 4 na máquina 1 e em todas as máquinas anteriores. Pela Tabela 1 vemos que o trabalho 4 deve ser executado anteriormente somente na máquina 3, logo $P_{14}^{-} = p_{34} + p_{14} = 14$;
- P_{14}^{+} é a soma do tempo de processamento do trabalho 4 na máquina 1 em todas as máquinas posteriores. Pela Tabela 1 vemos que o trabalho 4 deve ser executado posteriormente somente na máquina 2 $P_{14}^{+} = p_{14} + p_{24} = 16$;
- P_{23}^{-} de maneira análoga aos exemplos acima, é a soma do tempo de processamento do trabalho 3 na máquina 2 e em todas as anteriores. Vemos na Tabela 1 que esse trabalho deve ser processamento anteriormente em 1 e 3, então $P_{23}^{-} = p_{13} + p_{33} + p_{23} = 17$;
- P_{23}^{+} é a soma do tempo de processamento do trabalho 3 na máquina 2 somente, visto que a máquina 2 é a última máquina, então $P_{23}^{+} = p_{23} = 8$

No desenvolvimento das desigualdades válidas para o JSSP a seguir, ressaltamos que a variável z_{ijk} do modelo Manne (1960) e Liao e You (1992) está definida para pares de trabalhos distintos e sempre com $j < k$.

Note inicialmente que a expressão $P((n+1)(1-z_{ijk}) + (\pi_{ik} - \pi_{ij} - 1))$ é 0 se o trabalho j preceder k na máquina i e se k for consecutivo a j em i. Caso contrário, é um número suficientemente grande. Notamos que caso k preceda j em i, então $(1-z_{ijk}) = 1$ e a expressão se resume em $P((n+1) + (\pi_{ik} - \pi_{ij} - 1))$. Como π_{ik} e π_{ij} representam as posições dos processamentos dos trabalhos nas máquinas, então, obrigatoriamente, no pior cenário possível, considerando que k precede j em i, a diferença entre eles será de $1-n$. Esse cenário ocorre quando k é o primeiro trabalho executado em i e j é o último (n-ésimo). De qualquer modo, no mínimo essa expressão resultará em $P(n+1+1-n-1) = P$. Caso j preceda k em i, então $(1-z_{ijk}) = 0$ e

existem dois cenários. O primeiro cenário possível é quando j e k não são consecutivos em i e dessa maneira a diferença de posições dos trabalhos será maior que um. Dessa forma, a subtração de π_{ik} e π_{ij}, será no mínimo um número maior ou igual a 1. Logo, no mínimo a expressão resultará em $P((n+1)0+1) = P$. O segundo cenário possível é quando j e k são consecutivos em i, ou seja, a diferença das posições deles é exatamente um. Dessa forma, temos que $(\pi_{ik} - \pi_{ij} - 1) = 0$ e, consequentemente, a expressão resultaria em $P((n+1)0+0) = 0$.

Com essa expressão, podemos definir *upper bounds* para os tempos de início de cada trabalho, representados pelas variáveis x. Considerando que o trabalho k esteja na sua primeira máquina de execução, ou seja σ_1^k, e seja executado logo após j nessa mesma máquina, logo o tempo de início de k será igual ao tempo de término de j. Isso é verdade, pois k não está sendo processado em nenhuma outra máquina, então não haverá tempo de espera. Podemos formular matematicamente esse fato para quando j precede k na máquina i como:

$$x_{\sigma_1^k k} \leq P((n+1)(1 - z_{\sigma_1^k jk}) + (\pi_{\sigma_1^k k} - \pi_{\sigma_1^k j} - 1)) + x_{\sigma_1^k j} + p_{\sigma_1^k, j}, \quad \forall j,k \in J, j < k \tag{5.1}$$

Modelando o caso disjuntivo de maneira análoga, ou seja, quando k precede j na máquina i, temos:

$$x_{\sigma_1^j j} \leq P((n+1) z_{\sigma_1^j jk} + (\pi_{\sigma_1^j j} - \pi_{\sigma_1^j k} - 1)) + x_{\sigma_1^j k} + p_{\sigma_1^j, k}, \quad \forall j,k \in J, j < k \tag{5.2}$$

Um novo conjunto de restrições pode ser construído, baseado em (5.1) e (5.2), de modo que seja válido para qualquer máquina σ_h^k, com $2 \leq h \leq m$ e $k \in J$. Seja um trabalho k na sua h-ésima máquina σ_h^k, então há duas situações: ou k começa logo após o trabalho anterior terminar na mesma máquina; ou k começa logo após k ser processado na sua máquina anterior σ_{h-1}^k. Assim, se o trabalho k for consecutivo a j em σ_h^k, então o tempo de início de k em σ_h^k dado por $x_{\sigma_h^k, k}$ deverá ser o máximo entre o tempo de término de j na máquina σ_h^k, representado pela variável $x_{\sigma_h^k j}$, e o tempo de término de k na máquina σ_{h-1}^k, representado pela variável $x_{\sigma_{h-1}^k k}$. Caso j preceda k em i, então a expressão é modelada de maneira análoga. Matematicamente, podemos formular essa ideia como:

$$\begin{aligned} x_{\sigma_h^k k} \leq \quad & P((n+1)(1 - z_{\sigma_h^k jk}) + (\pi_{\sigma_h^k k} - \pi_{\sigma_h^k j} - 1)) \\ & + \max\{x_{\sigma_h^k j} + p_{\sigma_h^k j}, x_{\sigma_{h-1}^k k} + p_{\sigma_{h-1}^k k}\}, \qquad h = 2, \cdots, m, \forall j,k \in J, j < k \end{aligned} \tag{5.3}$$

$$x_{\sigma_h^j j} \leq \quad P((n+1)z_{\sigma_h^j jk} + (\pi_{\sigma_h^j j} - \pi_{\sigma_h^j k} - 1)) \\ + \max\{x_{\sigma_h^k j} + p_{\sigma_h^k j}, x_{\sigma_{h-1}^k k} + p_{\sigma_{h-1}^k k}\}, \quad h = 2, \cdots, m, \forall j,k \in J, j < k \tag{5.4}$$

Podemos linearizar as restrições acima com o uso de variáveis auxiliares $y_{\sigma_h^k k}$ e $y_{\sigma_h^j j}$, que indica o máximo entre $\{x_{\sigma_h^k, j} + p_{\sigma_h^k, j}, x_{\sigma_{h-1}^k, k} + p_{\sigma_{h-1}^k, k}\}$ e $\{x_{\sigma_h^k j} + p_{\sigma_h^k j}, x_{\sigma_{h-1}^k k} + p_{\sigma_{h-1}^k k}\}$, respectivamente. Logo, os conjunto de restrições (5.3) e (5.4) podem ser reescritos como:

$$x_{\sigma_h^k k} \leq P((n+1)(1 - z_{\sigma_h^k jk}) + (\pi_{\sigma_h^k k} - \pi_{\sigma_h^k j} - 1)) + y_{\sigma_h^k k}, \quad h = 2, \cdots, m, \forall j,k \in J, j < k \tag{5.5}$$

$$x_{\sigma_h^j j} \leq P((n+1)z_{\sigma_h^j jk} + (\pi_{\sigma_h^j j} - \pi_{\sigma_h^j k} - 1)) + y_{\sigma_h^j j}, \quad h = 2, \cdots, m, \forall j,k \in J, j < k \tag{5.6}$$

Para o vetor de variáveis y capturar o valor de $\max\{x_{\sigma_h^k, j} + p_{\sigma_h^k, j}, x_{\sigma_{h-1}^k, k} + p_{\sigma_{h-1}^k, k}\}$, devemos garantir que $y_{\sigma_h^k k}$ seja maior que ambos os termos, quando k for consecutivo a j na máquina σ_h^k, para todo $j \in J$ tal que $j < k$. Por conseguinte, para garantir que $y_{\sigma_h^k k}$ seja maior que $x_{\sigma_h^k, j} + p_{\sigma_h^k, j}$ apenas quando k for consecutivo a j, utilizamos um número suficientemente grande positivo P, definido anteriormente, de modo que quando não for consecutivo, $y_{\sigma_h^k jk}$ estará ilimitado inferiormente, tornando a restrição redundante. De maneira análoga para $y_{\sigma_h^j j}$. Matematicamente, podemos formular como:

$$y_{\sigma_h^k k} \geq x_{\sigma_h^k j} + p_{\sigma_h^k j} - P(1 - z_{\sigma_h^k jk}), \quad h = 2, \cdots, m, \quad \forall j,k \in J, j < k \tag{5.7}$$

$$y_{\sigma_h^j j} \geq x_{\sigma_h^k j} + p_{\sigma_h^k j} - P z_{\sigma_h^j jk}, \quad h = 2, \cdots, m, \quad \forall j,k \in J, j < k \tag{5.8}$$

Para garantir que $y_{\sigma_h^k k}$ seja maior que o término do trabalho k na máquina $h-1$ não precisamos modelar o caso disjuntivo ($y_{\sigma_h^j j}$), visto que ao fazer a restrição para todo $h = 2, \cdots, m$ e todo $k \in J$ já contemplamos esse caso. Então, temos que:

$$y_{\sigma_h^k k} \geq x_{\sigma_{h-1}^k k} + p_{\sigma_{h-1}^k k}, \quad h = 2, \cdots, m, \forall j, k \in J, j < k \tag{5.9}$$

Por fim, para garantir que $y_{\sigma_h^k k}$ e $y_{\sigma_h^j j}$ sejam, de fato, o máximo, devemos forçar que na solução ótima sejam o menor valor viável. Para isso, podemos associar as variáveis ao c_{max}:

$$y_{\sigma_h^k k} \leq c_{max}, \quad h = 2, \cdots, m, \forall k \in J \tag{5.10}$$

Observe que o início do processamento de cada trabalho $k \in J$ em uma máquina i, representado por x_{ik}, deverá ser pelo menos o tempo de processamento de todo trabalho $j \in J, j \neq k$ executado antes de k na máquina i. Note que para identificar se j foi processado antes de k na máquina i, temos que verificar dois casos de indexação de variável: $j < k$ ou $k < j$. Assim, para $j < k$ temos que a variável $z_{ijk} = 1$ caso j preceda k na máquina i. Por outro lado, caso $k < j$, então a variável $z_{ikj} = 0$ caso j preceda k na máquina i. Dessa forma, basta unir os dois casos da indexação dessas variáveis em uma única expressão:

$$x_{ik} \geq \sum_{j \in J | j < k} z_{ijk} p_{ij} + \sum_{j \in J | j > k} (1 - z_{ikj}) p_{ij}, \quad \forall i \in M, \forall k \in J \tag{5.11}$$

Agora, suponha que o trabalho k seja executado após j na máquina i. Assim, o início de k em i, representado por x_{ik}, deverá ser pelo menos o tempo de processo de j em todas as máquinas onde j executou antes de i, incluindo i. Vale notar que P_{ij}^{-} (definido anteriormente) representa exatamente a soma desse tempo de processamento. Por fim, o caso em que o trabalho k não precede j em i pode ser modelado de maneira análoga. Logo, temos:

$$x_{ik} \geq z_{ijk} P_{ij}^{-}, \quad \forall i \in M, \forall j, k \in J, j < k \tag{5.12}$$

$$x_{ij} \geq (1 - z_{ijk})P_{ik}^{-}, \quad \forall i \in M, \forall j,k \in J, j < k \tag{5.13}$$

Podemos também criar *lower bounds* para o *makespan*. Suponha que o trabalho k seja executado após j na máquina i. Então, no melhor cenário possível, o *makespan* será o término de j em i, ou seja $x_{ij} + p_{ij}$, mais o tempo de processamento de k em todas as máquinas posteriores a i, adicionando seu processamento em i. Note que P_{ik}^{+} (também definido anteriormente) representa essa soma dos tempos de processamento. O caso em que j não precede k em i pode ser formulado de maneira análoga.

$$C_{max} \geq x_{ij} + p_{ij} + z_{ijk}P_{ik}^{+}, \quad \forall i \in M, \forall j \in J, j < k \tag{5.14}$$

$$C_{max} \geq x_{ik} + p_{ik} + (1 - z_{ijk})P_{ij}^{+}, \quad \forall i \in M, \forall j \in J, j < k \tag{5.15}$$

Por fim, o *makespan* será pelo menos o tempo de início de um trabalho j em uma máquina i mais o tempo de processamento de j em todas as suas máquinas posteriores a ela, adicionando seu processamento em i. Nota que quando $i = \sigma_m^j$, teremos o conjunto de restrições (3.5). Logo, podemos substituir essas restrições por:

$$C_{max} \geq x_{ij} + P_{ij}^{+}, \quad \forall i \in M, j \in J \tag{5.16}$$

Com isso, finalizamos a introdução de diversas desigualdades válidas para o problema. Contudo, após testes preliminares observamos que apenas as restrições (5.11) a (5.16) foram úteis para alguma melhora dos modelos. O acréscimo das demais restrições não trouxe melhoras significativas na relaxação linear ou tornaram a busca por soluções viáveis bastante demorada. Dessa maneira, não compensa adicioná-las aos modelos.

Assim, incorporamos de forma satisfatória apenas as restrições citadas acima, (5.11) a (5.16), nos modelos anteriores de Manne (1960), Liao e You (1992) e (MLJ) com o intuito de melhorar suas relaxações lineares.

O modelo abaixo ($M2$) é o modelo de Manne (1960) com o acréscimo das desigualdades válidas (5.11) a (5.16).

$$(M2) \quad \min C_{max}$$
$$s.a \quad (3.2)-(3.8),(5.11)-(5.16)$$

O modelo abaixo ($L2$) é o modelo de Liao e You (1992) com o acréscimo dessas desigualdades.

$$(L2) \quad \min C_{max}$$
$$s.a \quad (3.2),(3.5)-(3.10),(5.11)-(5.16)$$

Por fim, o modelo abaixo ($MLJ2$) é o modelo (MLJ) com o acréscimo das desigualdades válidas (5.11) a (5.16).

$$(MLJ2) \quad \min C_{max}$$
$$s.a \quad (3.2)-(3.8),(4.23)-(4.28),(4.33)-(4.35),(5.11)-(5.16)$$

Para incorporar as novas desigualdades válidas não foi necessário adicionar nenhuma nova variável. Logo, todos os modelos têm o mesmo número de variáveis. Com isso, podemos comparar as dimensões dos modelos com e sem essas desigualdades válidas.

Modelo	**# Restrições**	**# Variáveis Binárias**	**# Variáveis Contínuas**
(MANNE)	mn^2	$\frac{mn(n-1)}{2}$	mn+1
(M2)	$m(2n^2+2n-1)$	$\frac{mn(n-1)}{2}$	mn+1
(LIAO)	$\frac{m(n^2+2n-1)}{2}$	$\frac{mn(n-1)}{2}$	$\frac{mn(n+1)+2}{2}$
(L2)	$\frac{m(3n^2+4n-3)}{2}$	$\frac{mn(n-1)}{2}$	$\frac{mn(n+1)+2}{2}$
(MLJ)	$\frac{6n(n-1)+mn(n^2+3n+2)+6m}{6}$	$\frac{mn(n-1)}{2}$	mn+1
(MLJ2)	$\frac{6n(n-1)+mn(n^2+9n+14)}{6}$	$\frac{mn(n-1)}{2}$	mn+1

Tabela 3 – Dimensões dos modelos.

6 RESULTADOS COMPUTACIONAIS

O ambiente para a realização dos testes foi um computador Intel Computer Core i7-7700, CPU 3.60 GHz com 32 Giga bytes de RAM executando em Linux 18.04 LTS/64 bits. O *solver* usado para resolver os modelos é o CPLEX versão V12.10, por ter sido o mais eficiente em Ku e Beck (2016) e ser um dos principais do mercado. Para a construção dos modelos foi utilizada a linguagem *Python*, versão 3.6.7, com a biblioteca *docplex*. O tempo máximo de execução para cada instância foi configurado para 7200 segundos e uma tolerância de *gap* do CPLEX de 10^{-6}.

Para as métricas de avaliação dos modelos iremos definir alguns parâmetros. Inicialmente, seja o conjunto de instâncias I. Então, seja $UB_{inst,mod}$ o *upper bound* do modelo mod na instância $inst$ e $RL_{inst,mod}$ a relaxação linear do modelo mod na instância $inst$. Então, $\overline{UB}_{mod}$ representa a média dos *upper bounds* do modelo mod, matematicamente:

$$\overline{UB}_{mod} = \frac{\sum_{inst \in I} UB_{inst,mod}}{|I|} \tag{6.1}$$

Além disso, $\overline{RL}_{mod}$ representa a média das relaxações lineares para o modelo mod, matematicamente:

$$\overline{RL}_{mod} = \frac{\sum_{inst \in I} RL_{inst,mod}}{|I|} \tag{6.2}$$

Então, a primeira métrica para comparar dois modelos é a variação da média das relaxações lineares. Para essa métrica, quanto maior a variação então consideramos que $mod1$ obteve um desempenho melhor em relação às relaxações lineares do que $mod2$. Matematicamente podemos escrever como:

$$VRL_{mod1,mod2} = \frac{\overline{RL}_{mod1} - \overline{RL}_{mod2}}{\overline{RL}_{mod2}} \tag{6.3}$$

A segunda métrica é a variação da média dos *upper bounds* dos modelos. Podemos descrever matematicamente:

$$VUB_{mod1,mod2} = \frac{\overline{UB}_{mod1} - \overline{UB}_{mod2}}{\overline{UB}_{mod2}} \tag{6.4}$$

Para ‘cpu(s)’, ‘RL’ e ‘UB’ vamos construir novas medidas denominadas ‘% cpu(s)’, ‘% RL’ e ‘% UB’, respectivamente. O intuito é obter métricas que estarão na

mesma escala para todos os modelos e, dessa forma, evitar distorções nos gráficos. Então, sendo $cpu(s)_{inst,mod}$ o tempo de execução para resolução do problema de programação linear inteira do modelo $mod \in \{(MANNE),(M2),(Liao),(L2)\}$ com a instância $inst$ e $modelos = \{(MANNE),(M2),(Liao),(L2)\}$, temos que:

$$\%cpu(s)_{inst,mod} = 100\% \frac{cpu(s)_{inst,mod}}{\max\limits_{mod' \in modelos} \{cpu(s)_{inst,mod'}\}}$$

$$\%RL_{inst,mod} = 100\% \frac{RL_{inst,mod}}{\max\limits_{mod' \in modelos} \{RL_{inst,mod'}\}}$$

$$\%UB_{inst,mod} = 100\% \frac{UB_{inst,mod}}{\max\limits_{mod' \in modelos} \{UB_{inst,mod'}\}}$$

Ou seja, para cada instância e modelos dividimos cada métrica pela maior métrica entre os modelos. Dessa maneira, garantimos que os valores estarão entre 0 e 100%.

Foram definidos dois conjuntos de testes. O primeiro, denominando como I_1 e com os resultados descritos na seção 6.1, foi preliminarmente usado para comparar a performance dos modelos de $(MANNE)$ com $(M2)$ e $(LIAO)$ com $(L2)$. As instâncias utilizadas foram adaptações das instâncias de Taillard (1993). A adaptação foi realizada da seguinte forma. Seja um arquivo da instância de Taillard (1993):

```
Times
 8 9 7
 6 7 5
 7 6 1
 4 2 1
Machines
 3 1 2
 3 2 1
 2 1 3
 2 3 1
```

Então, temos que $n = 4$, $m = 3$, a matriz *Times* contém uma linha para cada trabalho com os tempos de processamento e a matriz *Machines* com cada

linha representando a ordem das máquinas de cada trabalho. Além disso, o tempo de processamento de *Times* é referente à ordem em *Machines*, ou seja, o primeiro elemento de *Times* indica o tempo de processamento do trabalho 1 na máquina do primeiro elemento de *Machines*. Dessa maneira, no exemplo acima, temos p_{11} = 9, $p_{12} = 7$, $p_{13} = 8$, $\cdots$, $p_{14} = 1$, $p_{24} = 4$ e $p_{34} = 2$. A adaptação que realizamos é considerar o tempo de processamento tal qual é mostrada em *Times*, então o primeiro tempo de cada linha representa a máquina 1, o segundo tempo a máquina 2 e assim até o m-ésimo tempo representando a máquina m. A ordem seria a mesma descrita em *Machines*. Dessa forma, no exemplo acima, a ordem é p_{11} = 8, $p_{12} = 9$, $p_{13} = 7$, $\cdots$, $p_{14} = 4$, $p_{24} = 2$ e $p_{34} = 1$.

Dessa forma, foram 10 instâncias com 15 trabalhos e 15 máquinas, 10 instâncias com 15 trabalhos e 20 máquinas e 10 instâncias com 20 trabalhos e 20 máquinas, adaptadas de Taillard (1993).

O segundo conjunto de teste são instâncias *benchmark* da literatura, denominado como I_2 e com os resultados descritos na seção 6.2, avalia o impacto das desigualdades nos modelos de $(MANNE)$ e (MLJ) empregando como valor de corte (solução viável) o *upper bound* encontrado pela meta-heurística GRASP-ELS. Devido ao fator aleatório da meta-heurística GRASP-ELS, cada execução dela pode encontrar um valor de *makespan* distinto. Logo, para evitar distorções nas comparações dos modelos, executamos a meta-heurística uma vez e utilizamos os valores encontrados de *makespan* para cada instância em todos os modelos. Dessa forma, garantimos que todos os modelos terão os mesmos *upper bounds* para o *makespan*. Foram utilizadas 43 instâncias, sendo 3 de Fisher (1963), com prefixo ft, 20 de Lawrence (1984), com prefixo la, e 20 de Taillard (1993), com prefixo ta. As dimensões de todas as instâncias podem ser vistas na Tabela 4 juntamente com a quantidade de instâncias para cada dimensão.

Nas próximas seções 6.1 e 6.2 apresentamos os resultados para os dois conjuntos de instâncias I_1 e I_2, respectivamente. Os resultados da seção 6.1 permitem comparar os modelos de Manne (1960) e Liao e You (1992) com base nas novas instâncias do conjunto I_1. A finalidade é mostrar que o primeiro modelo é de fato melhor que o segundo como é de conhecimento da literatura. Assim, justificamos o motivo pelo qual usamos apenas os modelos $(MANNE)$, $(M2)$ e $(MLJ2)$ com as instâncias *bench-*

$\|J\|$	$\|M\|$	# Instâncias
6	6	1
10	5	5
10	10	6
15	5	5
15	15	10
20	5	6
20	15	10

Tabela 4 – Quantidade de instâncias para cada combinação de dimensão.

mark da literatura na seção 6.2. Isso, por conta do expressivo tempo computacional necessário para realizar esses testes computacionais para as 43 instâncias do conjunto I_2 com esses 3 modelos e usando um limite de tempo de 2 horas para cada instância.

6.1 Resultados computacionais para novas instâncias de teste com modelos Manne (1960) e Liao e You (1992)

Os resultados para o conjunto de instâncias I_1 estão descritos nas Tabelas 5 e 6 para os modelos originais e fortalecidos de Manne (1960) e Liao e You (1992), respectivamente. Esses resultados obtidos preliminarmente foram apresentados em Cavalcante *et al.* (2020).

A legenda dessas tabelas é como segue. A quantidade de trabalhos e de máquinas são dadas por $|J|$ e $|M|$, respectivamente. Para distinguir instâncias com mesmo número de trabalhos e máquinas, usamos um identificador 'ID' para cada instância. Para cada modelo e sua variante, o valor da melhor solução viável encontrada pelo CPLEX é denotado por 'UB'. O tempo de execução de cada modelo é dado na coluna 'cpu(s)', em segundos, com o limite máximo de 7200s. O *gap* retornado pelo CPLEX, que é a diferença relativa em porcentagem entre um limite inferior para a solução da instância e o valor UB, é dado por '*gap*'. O valor da solução da relaxação linear de cada modelo é dado por 'RL' e serve para observar o impacto das novas desigualdades propostas para o problema quando são acrescidas ao modelo original. O tempo de execução para obter a solução da relaxação linear de cada modelo é dado na coluna 'cpu-rl(s)', em segundos. Utilizamos o símbolo asterisco ('*') nas colunas de UB e cpu(s) para identificar resultados obtidos a partir do log do CPLEX quando houve estouro de memória, ou seja, os últimos resultados encontrados pelo modelo nos respectivos *logs*. Por fim, destacamos em negrito o valor do 'UB' quando for

estritamente menor do que em comparação ao outro modelo da tabela e o valor da 'RL' quando for estritamente maior.

Instância			(MANNE)					(M2)				
ID	$\|J\|$	$\|M\|$	UB	cpu (s)	gap%	RL	cpu-rl (s)	UB	cpu (s)	gap%	RL	cpu-rl (s)
1	15	15	1210	238.2	0.00	963.000	0.019	1210	457.3	0.00	**982.47**	0.083
2	15	15	1177	978.0	0.00	942.000	0.024	1177	4502.1	0.00	**945.625**	0.123
3	15	15	1197	2538.9	0.00	921.000	0.026	1197	4593.7	0.00	**947.846**	0.094
4	15	15	1172	5906.5	0.00	911.000	0.020	1172	2190.3	0.00	911.000	0.095
5	15	15	**1232**	7200.0	6.10	940.000	0.024	1243	7200.0	8.10	**975.694**	0.097
7	15	15	1162	1578.6	0.00	849.000	0.027	1162	1828.1	0.00	**915.057**	0.108
7	15	15	1243	295.8	0.00	935.000	0.021	1243	688.56	0.00	**979.433**	0.086
8	15	15	1181	51.8	0.00	963.000	0.016	1181	60.8	0.00	963.000	0.079
9	15	15	1293	7200.0	5.60	982.000	0.024	1293	7200.0	5.80	**1012.376**	0.101
10	15	15	1206	7200.0	7.90	896.000	0.022	1206	629.7	0.00	**988.883**	0.097
11	15	20	**1382**	7202.0	16.90	949.000	0.043	1447	7200.0	24.80	**992.180**	0.290
12	15	20	**1440**	7200.0	22.60	1012.000	0.050	1479*	4791.8*	EM	**1031.464**	0.285
13	15	20	**1367**	7200.0	15.30	919.000	0.046	1379	7200.0	19.20	**1057.76**	0.239
14	15	20	**1341**	7201.0	19.90	990.000	0.054	1398	7200.0	23.40	**1004.136**	0.282
15	15	20	**1406**	7200.0	22.90	880.000	0.042	1442	7200.0	21.30	**1053.151**	0.248
16	15	20	**1464**	7200.0	25.30	932.000	0.052	1471	7200.0	16.50	**1141.346**	0.238
17	15	20	**1457**	7200.0	20.40	979.000	0.056	1487	7200.0	6.40	**1106.498**	0.288
18	15	20	**1470**	7200.0	26.00	900.000	0.098	1529	7200.0	26.80	**1027.487**	0.284
19	15	20	**1371**	7200.0	22.90	920.000	0.054	1392*	4270.3*	EM	**1026.778**	0.279
20	15	20	**1397**	7200.0	21.80	928.000	0.084	1436	7200.0	20.90	**1026.908**	0.305
21	20	20	**1604**	7200.0	15.10	1217.000	0.068	1627	7200.0	20.20	**1218.974**	0.337
22	20	20	**1616**	7200.0	15.40	1223.000	0.085	1624	7200.0	18.40	1223.000	0.287
23	20	20	**1643**	7200.0	17.60	1164.000	0.079	1665	7200.0	20.70	**1212.621**	0.291
24	20	20	**1652**	7200.0	19.10	1151.000	0.097	1665	7200.0	20.70	**1216.473**	0.323
25	20	20	**1687**	7200.0	18.10	1170.000	0.110	1745	7200.0	20.80	**1298.272**	0.283
26	20	20	**1647**	7200.0	17.90	1207.000	0.085	1669	7200.0	15.70	**1276.443**	0.269
27	20	20	**1699**	7200.0	15.00	1291.000	0.079	1724	7200.0	16.30	1291.000	0.282
28	20	20	1614	7200.0	11.30	1221.000	0.071	**1581**	7200.0	11.50	**1246.331**	0.331
29	20	20	1692	7200.0	16.70	1227.000	0.077	**1654**	7200.0	13.00	**1253.470**	0.351
30	20	20	1654	7200.0	16.10	1212.000	0.087	**1640**	7200.0	15.80	**1275.337**	0.246
Médias			1 422.53	5 906.4	13.20	1 026.47	0.050	1 438.11	5 676.9	12.37	1 086.70	0.220

Tabela 5 – Resultados para os modelos (MANNE) e (M2).

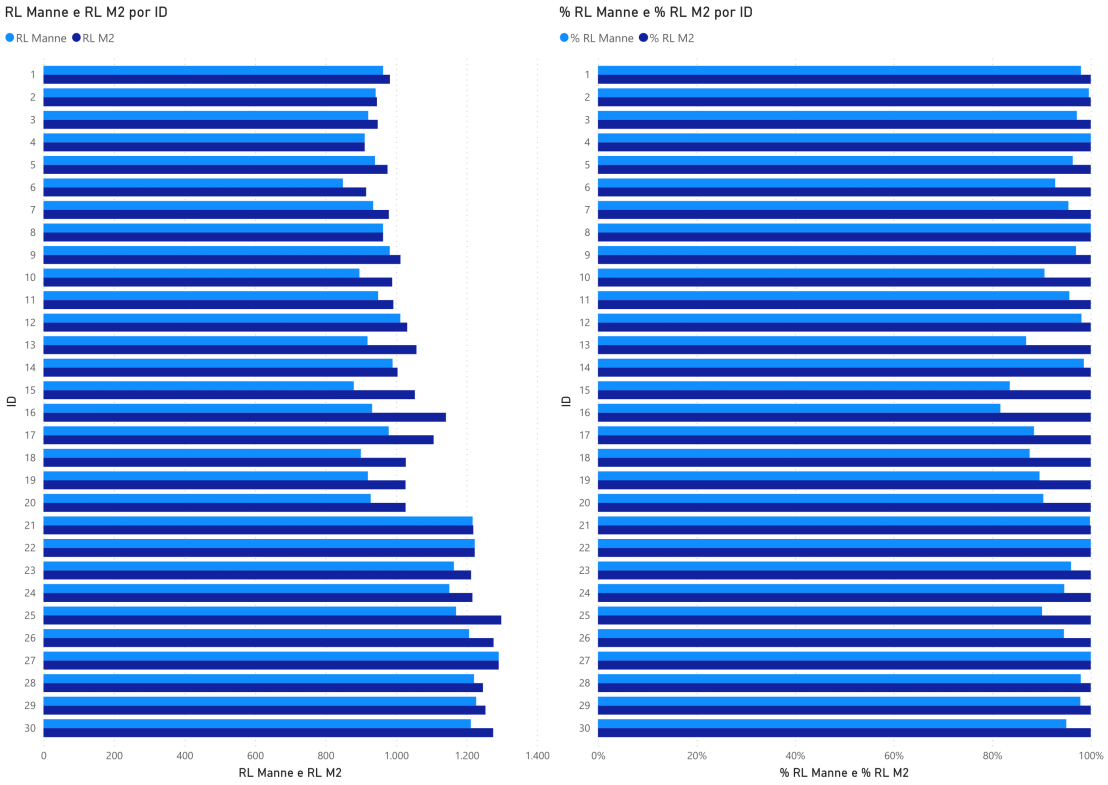

Figura 7 – Resultados da RL para os modelos de $(MANNE)$ e $(M2)$ para o conjunto de instâncias I_1.

Analisando a relaxação linear na Tabela 5 verificamos $VRL_{M2,Manne} = 5,86\%$, o que não significa uma melhora muito considerável, contudo $(M2)$ obteve valores de relaxação linear estritamente maiores do que de $(MANNE)$ para 26 das 30 instâncias. A Figura 7 apresenta os gráficos da RL e $\%RL$ para cada instância.Dessa maneira, podemos verificar que o modelo $(M2)$ obteve, como esperado, um comportamento dominante em relação ao modelo de $(MANNE)$, principalmente nas instâncias 12 a 20, onde vemos uma maior distância entre as proporções.

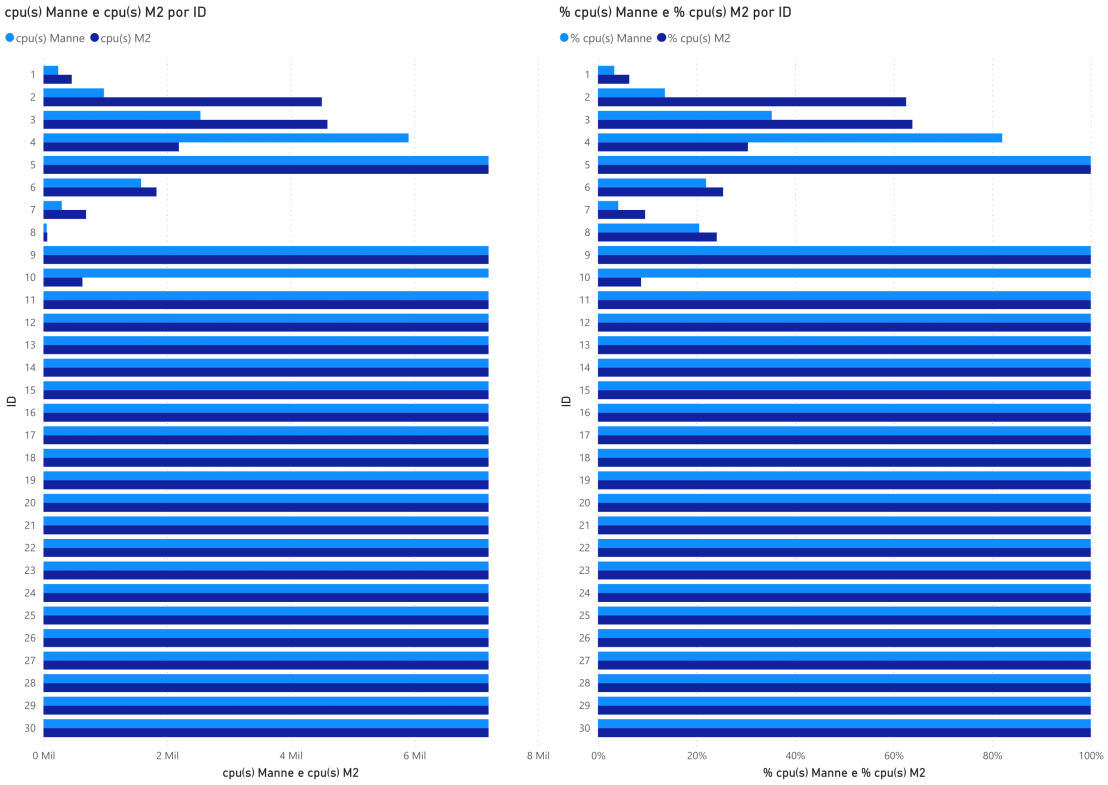

Figura 8 – Resultados do cpu(s) para os modelos de $(MANNE)$ e $(M2)$ para o conjunto de instâncias I_1.

Analisando a Tabela 5, podemos verificar que para a maioria das instâncias o tempo limite foi atingido, evidenciando a dificuldade para resolvê-las. Em apenas sete das 30 instâncias o modelo de $(MANNE)$ não o atingiu, enquanto para o modelo $(M2)$ foram oito instâncias. Para duas instâncias houve um estouro de memória (EM) para $(M2)$. A Figura 8 apresenta os resultados em gráficos de barras. O gráfico da esquerda com os valores absolutos para cada instância e o gráfico da direita com os valores proporcionais. Dessa forma, notamos que o comportamento do tempo de execução foi semelhante, principalmente pelo fato de na maioria das instâncias o limite de tempo ser atingido. Porém, em algumas instância é o tempo de execução do modelo $(M2)$ é evidentemente maior do que comparado ao modelo $(MANNE)$, como na instância 2 e 3. Contudo, nas instâncias 4 e 10 o comportamento foi o inverso, o modelo $(M2)$ que teve 'cpu(s)' bem menores em relação ao modelo de $(MANNE)$.

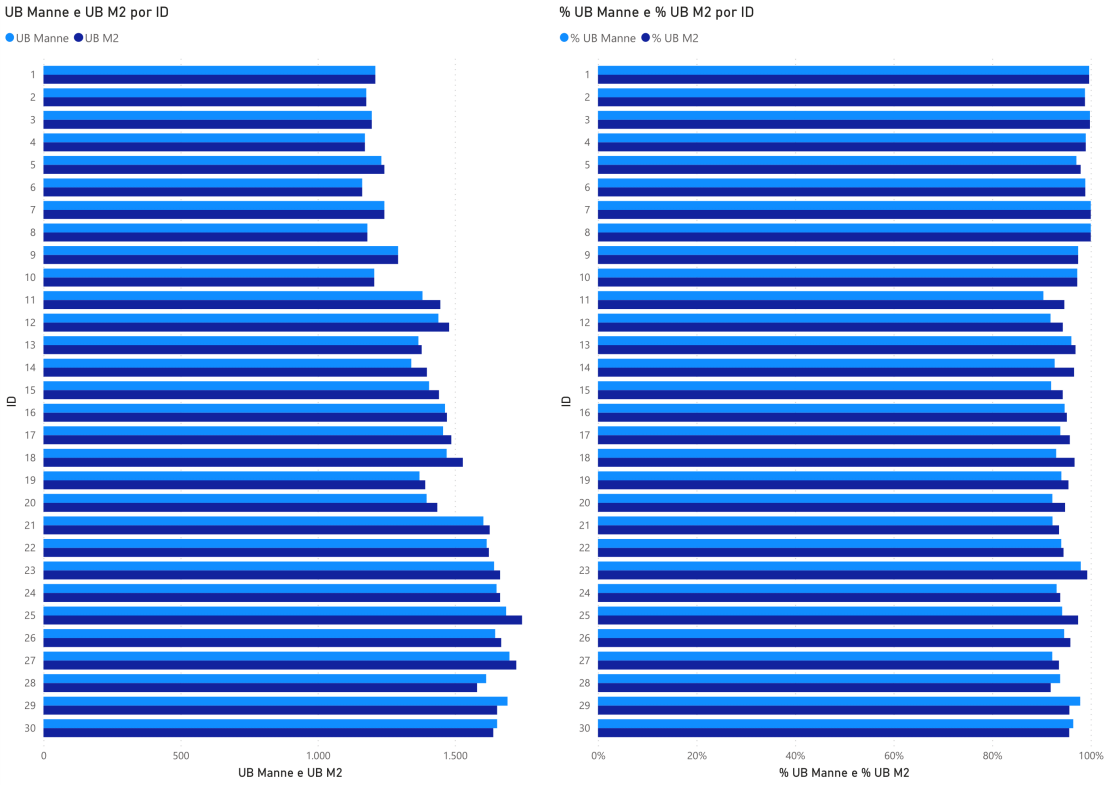

Figura 9 – Resultados do UB para os modelos de $(MANNE)$ e $(M2)$ para o conjunto de instâncias I_1.

Observamos na Tabela 5 que para metade das instâncias o modelo $(M2)$ obteve um *gap* tão bom quanto o modelo de $(MANNE)$. Além disso, o modelo $(M2)$ obteve em 12 das 30 instâncias um *makespan* tão bom quanto do modelo de $(MANNE)$, sendo três estritamente menores. Contudo, $VUB_{M2,Manne} = 10.95\%$, ou seja, o modelo de $(MANNE)$ puro foi melhor do que o $(M2)$ em relação ao *upper bound*. Na Figura 9 podemos identificar os resultados do UB e $\%UB$. Constatamos visualmente que, de fato, o comportamento foi bem semelhante.

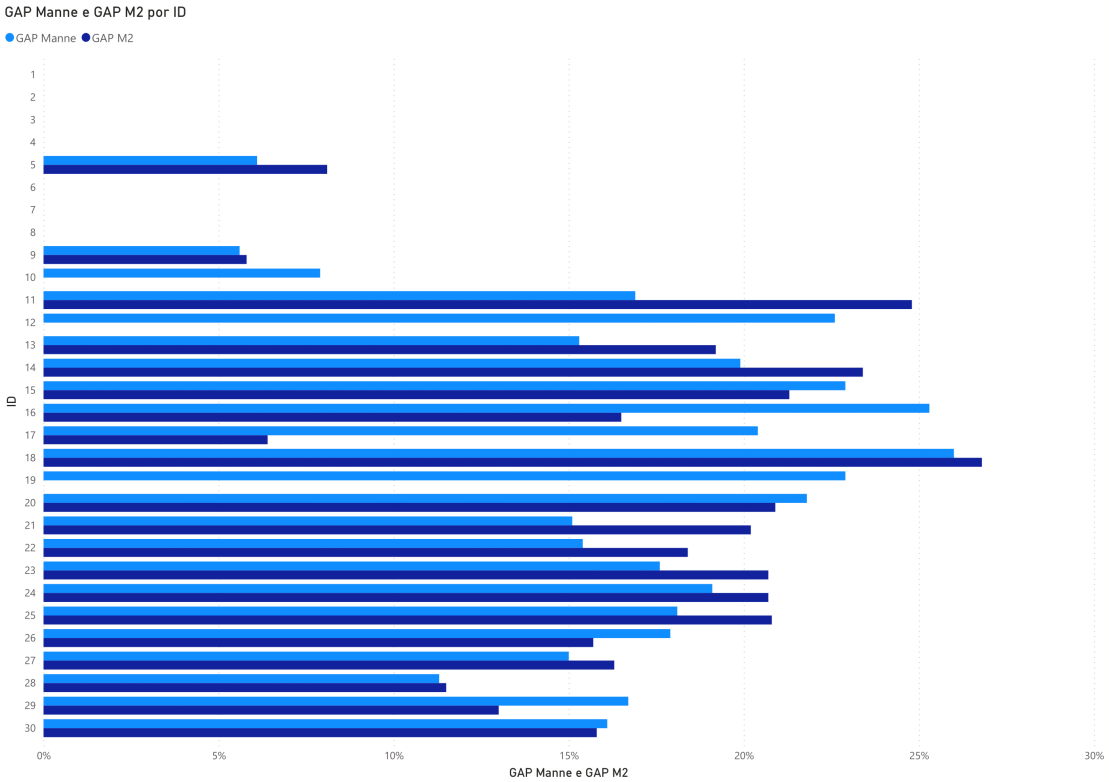

Figura 10 – Resultados do *gap* para os modelos de $(MANNE)$ e $(M2)$ para o conjunto de instâncias I_1.

Vemos na Tabela 5 que a média do *gap* do modelo $(M2)$ foi de 12.37%, ligeiramente menor que a média do *gap* do modelo de $(MANNE)$ que foi de 13.20%, sendo que para 8 instâncias o *gap* foi estritamente menor. Na Figura 10 podemos verificar o comportamento do *gap* para cada instância. Notamos que em algumas instâncias, por exemplo a 11, o *gap* do modelo de $(MANNE)$ foi bem menor do que o do modelo de $(M2)$. Contudo, vemos na instância 17 que o inverso ocorreu.

Instância			(LIAO)					(L2)				
ID	$\|J\|$	$\|M\|$	UB	cpu (s)	gap%	RL	cpu-rl (s)	UB	cpu (s)	gap%	RL	cpu-rl (s)
1	15	15	1214	7200.0	11.10	963.000	0.008	**1210**	2034.8	0.00	**982.470**	0.083
2	15	15	**1184**	7200.0	9.80	942.000	0.008	1191	7200.0	10.70	**945.625**	0.095
3	15	15	**1198**	7200.0	11.40	921.000	0.009	1199	7200.0	11.30	**947.846**	0.130
4	15	15	**1181**	7200.0	13.40	911.000	0.008	1184	7200.0	12.30	911.000	0.073
5	15	15	**1247**	7200.0	17.50	940.000	0.008	1269	7200.0	17.60	**975.694**	0.070
6	15	15	1175	7200.0	9.20	849.000	0.008	**1168**	7200.0	6.70	**915.057**	0.126
7	15	15	1243	7200.0	9.90	935.000	0.008	1243	2236.0	0.00	**979.433**	0.082
8	15	15	1181	252.1	0.00	963.000	0.008	1181	184.6	0.00	963.000	0.078
9	15	15	**1317**	7200.0	16.50	982.000	0.008	1327	7200.0	16.20	**1012.376**	0.092
10	15	15	1240	7200.0	16.10	896.000	0.008	**1206**	4182.8	0.00	**988.883**	0.080
11	15	20	**1504**	7200.0	28.60	949.000	0.017	1529	7200.0	29.80	**992.180**	0.588
12	15	20	**1419**	7200.0	25.00	1012.000	0.016	1568	7200.0	27.70	**1031.464**	0.525
13	15	20	**1409**	7200.0	25.70	919.000	0.016	1423	7200.0	22.10	**1057.760**	0.669
14	15	20	**1376**	7200.0	26.20	990.000	0.016	1447	7200.0	26.30	**1004.136**	0.559
15	15	20	**1464**	7200.0	29.70	880.000	0.021	1529	7200.0	25.30	**1053.151**	0.588
16	15	20	1546	7200.0	31.30	932.000	0.016	**1489**	7200.0	19.70	**1141.346**	0.563
17	15	20	**1475**	7200.0	26.80	979.000	0.016	1553	7200.0	19.70	**1106.498**	0.600
18	15	20	**1560**	7200.0	38.20	900.000	0.018	1581	7200.0	32.00	**1027.487**	0.635
19	15	20	1458	7200.0	30.40	920.000	0.017	**1414**	7200.0	24.20	**1026.778**	0.600
20	15	20	**1441**	7200.0	30.90	928.000	0.017	1515	7200.0	28.10	**1026.908**	0.548
21	20	20	**1642**	7200.0	21.40	1217.000	0.026	1739	7200.0	26.20	**1218.974**	0.857
22	20	20	**1637**	7200.0	22.30	1223.000	0.033	1719	7200.0	24.80	1223.000	1.179
23	20	20	**1659**	7200.0	23.50	1164.000	0.026	1677	7200.0	21.10	**1212.621**	0.792
24	20	20	**1687**	7200.0	24.60	1151.000	0.026	1775	7200.0	27.60	**1216.473**	1.192
25	20	20	1791	7200.0	27.50	1170.000	0.026	**1747**	7200.0	23.20	**1298.272**	0.769
26	20	20	**1681**	7200.0	22.70	1207.000	0.028	1741	7200.0	22.30	**1276.443**	0.754
27	20	20	**1724**	7200.0	20.40	1291.000	0.024	1843	7200.0	25.10	1291.000	0.725
28	20	20	**1649**	7200.0	17.80	1221.000	0.028	1721	7200.0	20.20	**1246.331**	0.813
29	20	20	**1711**	7200.0	20.60	1227.000	0.026	1729	7200.0	18.70	**1253.470**	0.817
30	20	20	1715	7200.0	23.80	1212.000	0.024	**1663**	7200.0	19.20	**1275.337**	1.611
Médias			1 457.60	6 968.5	21.08	1 026.47	0.020	1 486.00	6 528.0	18.60	1 086.70	0.540

Tabela 6 – Resultados para os modelos (LIAO) e (L2).

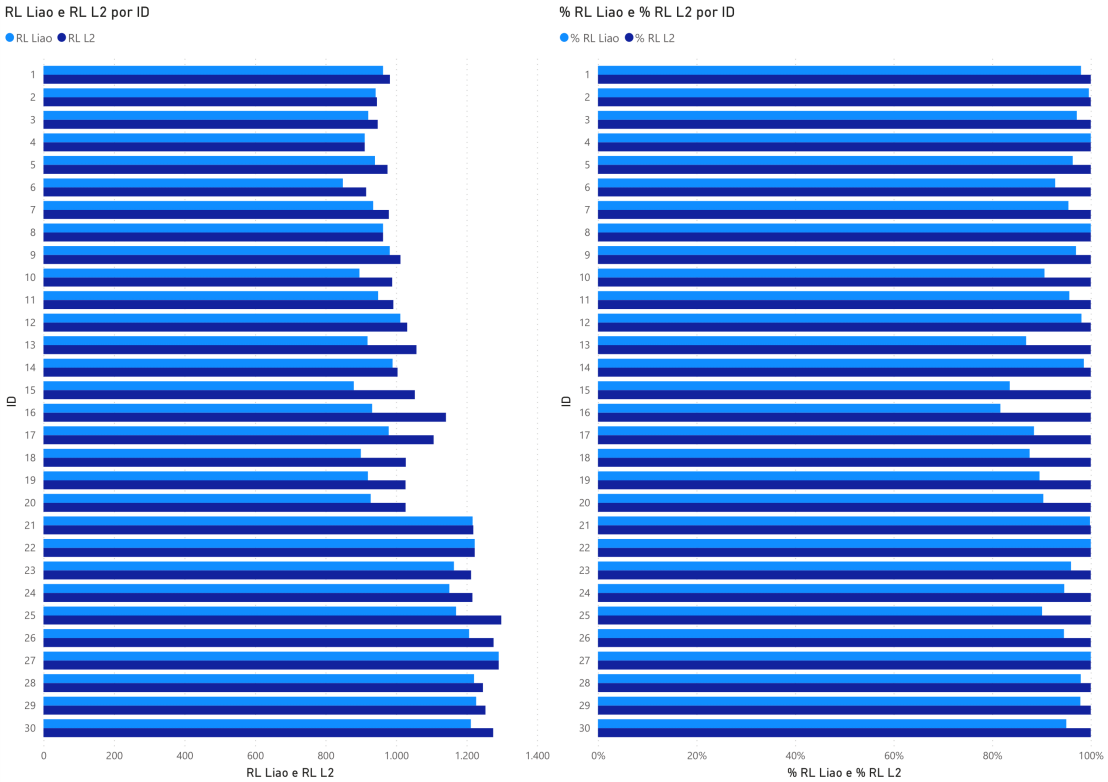

Figura 11 – Resultados da RL para os modelos de $(LIAO)$ e $(L2)$ para o conjunto de instâncias I_1.

Analisando a relaxação linear na Tabela 6, notamos que $VLR_{L2,Liao} = 5,8\%$, contudo o modelo $(L2)$ também obteve valores de relaxação linear estritamente maiores às dos modelos originais de $(LIAO)$ para 26 das 30 instâncias foram. Verificando a Figura 11 notamos a dominância da RL de $(L2)$ em relação à RL de $(LIAO)$, principalmente nas instâncias 15 a 20.

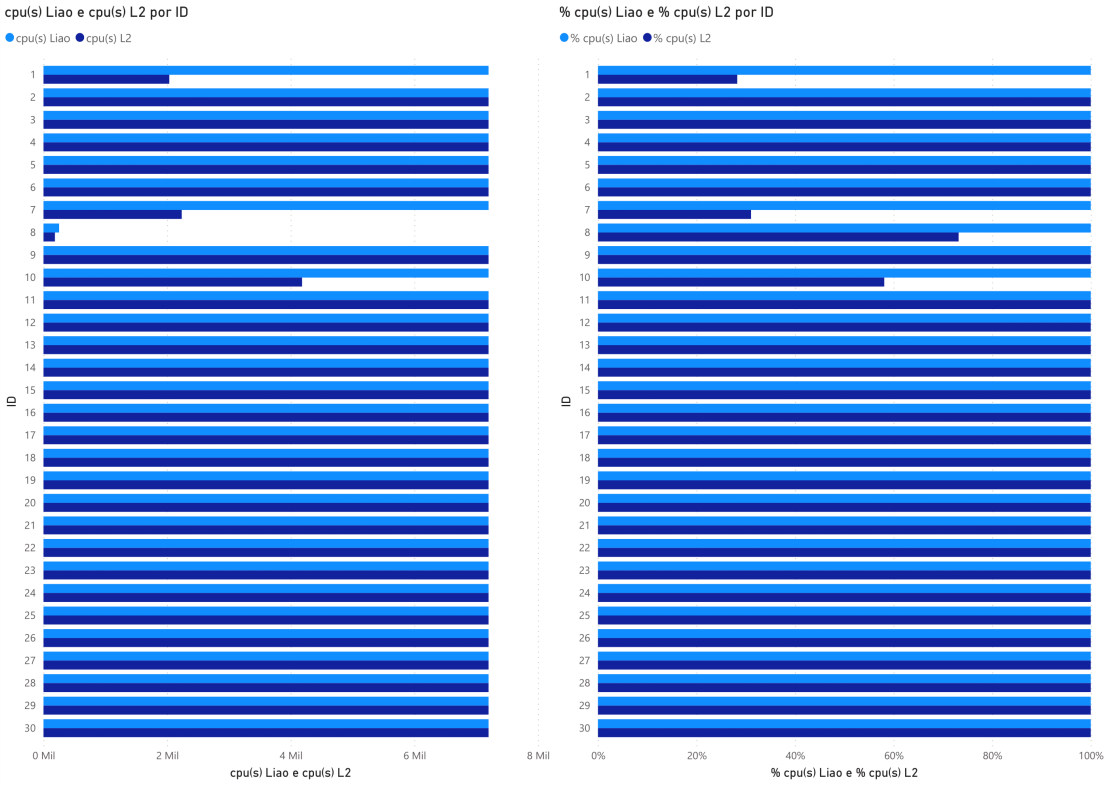

Figura 12 – Resultados do cpu(s) para os modelos de $(LIAO)$ e $(L2)$ para o conjunto de instâncias I_1.

Na Tabela 6 podemos constatar que o modelo de $(LIAO)$ não atingiu o tempo máximo de execução para apenas uma instância, enquanto $(L2)$ não o atingiu em quatro instâncias. Dessa maneira, podemos ver na Figura 12 que nas instâncias que $(L2)$ não atingiu o máximo, obteve um tempo consideravelmente menor do que $(LIAO)$. Inclusive, se verificarmos a instância 8, que ambos os modelos não precisaram do tempo máximo de execução, podemos ver que houve uma diferença de cerca de 25% entre $(L2)$ e $(LIAO)$, ou seja, podemos dizer que em geral os modelos tiveram tempos de execução semelhantes, mas o modelo de $(L2)$ obteve resultados melhores em algumas instâncias.

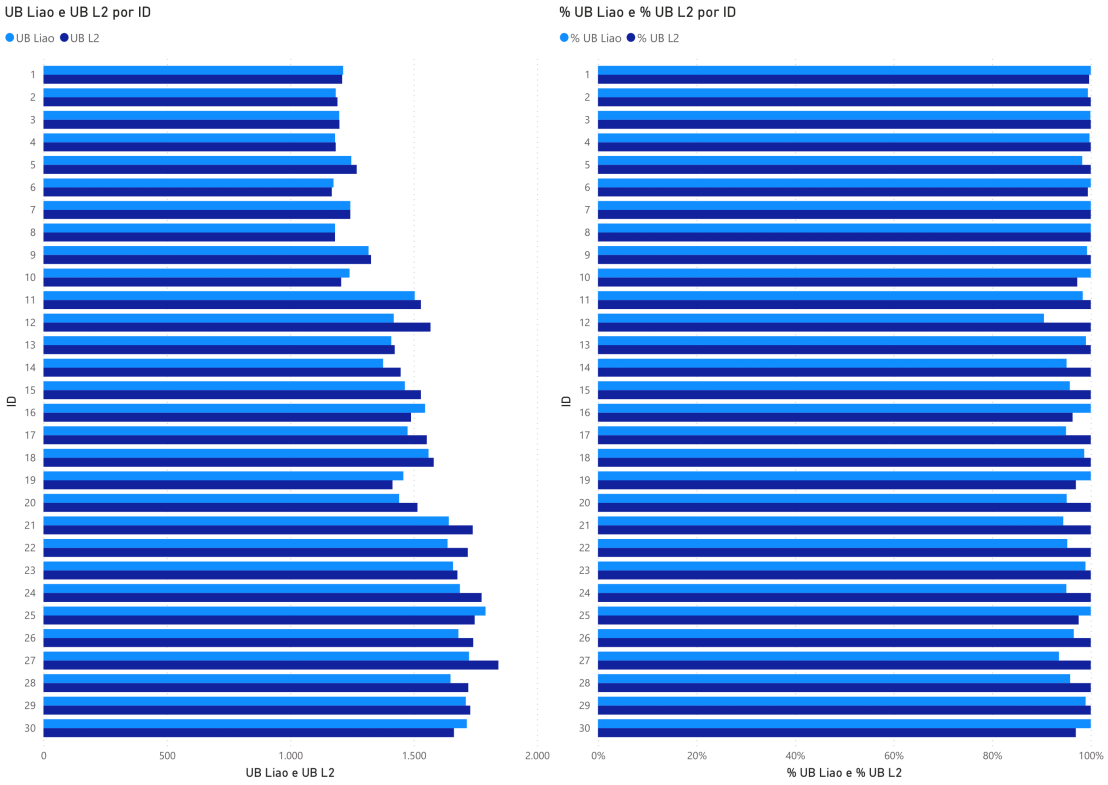

Figura 13 – Resultados do UB para os modelos de $(LIAO)$ e $(L2)$ para o conjunto de instâncias I_1.

Analisando a Tabela 6 podemos ver que em sete instâncias o *makespan* de $(L2)$ foi menor que o modelo de $(LIAO)$, tendo $VUB_{L2,Liao} = 19,48\%$. Contudo, pela Figura 13 notamos que o comportamento foi bem semelhante.

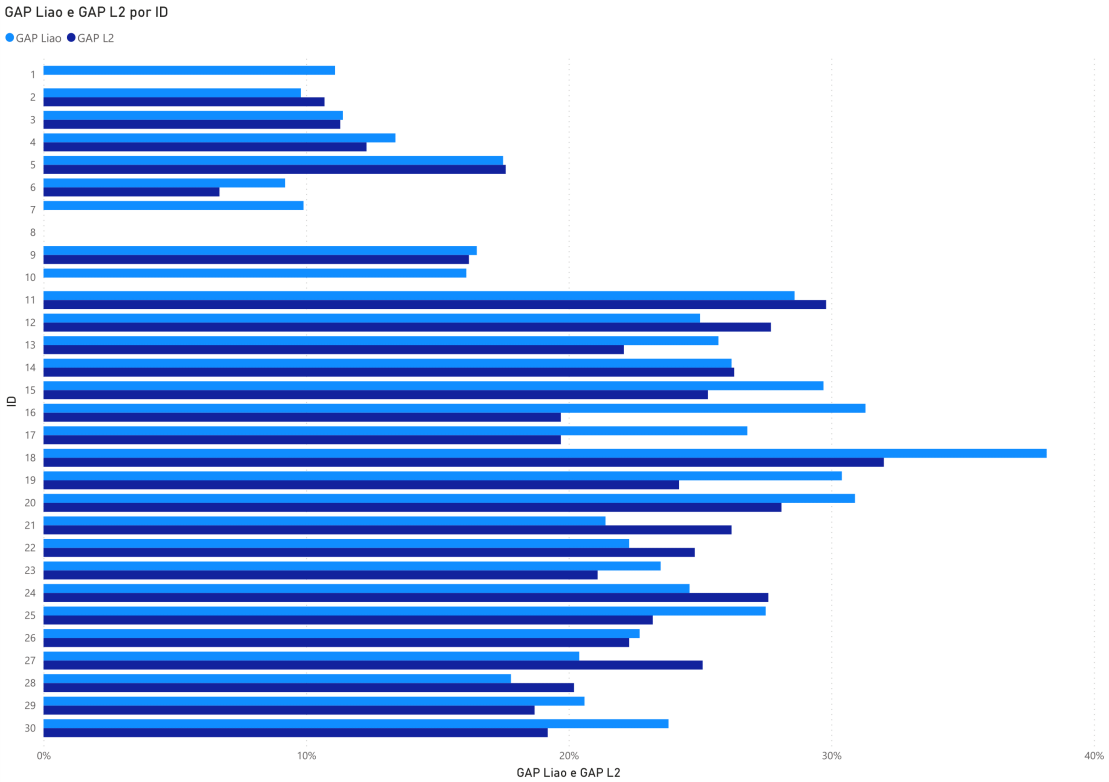

Figura 14 – Resultados do *gap* para os modelos de $(LIAO)$ e $(L2)$ para o conjunto de instâncias I_1.

Pela a Tabela 6 podemos ver que o modelo $(L2)$ obteve uma média de *gap* de 18.60%, menor que o do modelo de $(LIAO)$ de 21.08%, sendo que em 19 das 30 instâncias foi estritamente menor. Pela Figura 14 vemos que o comportamento em algumas instâncias foi bem diferente, por exemplo na instância 18 e 30, o modelo de $(L2)$ obteve *gaps* consideravelmente menores do que o modelo de $(LIAO)$.

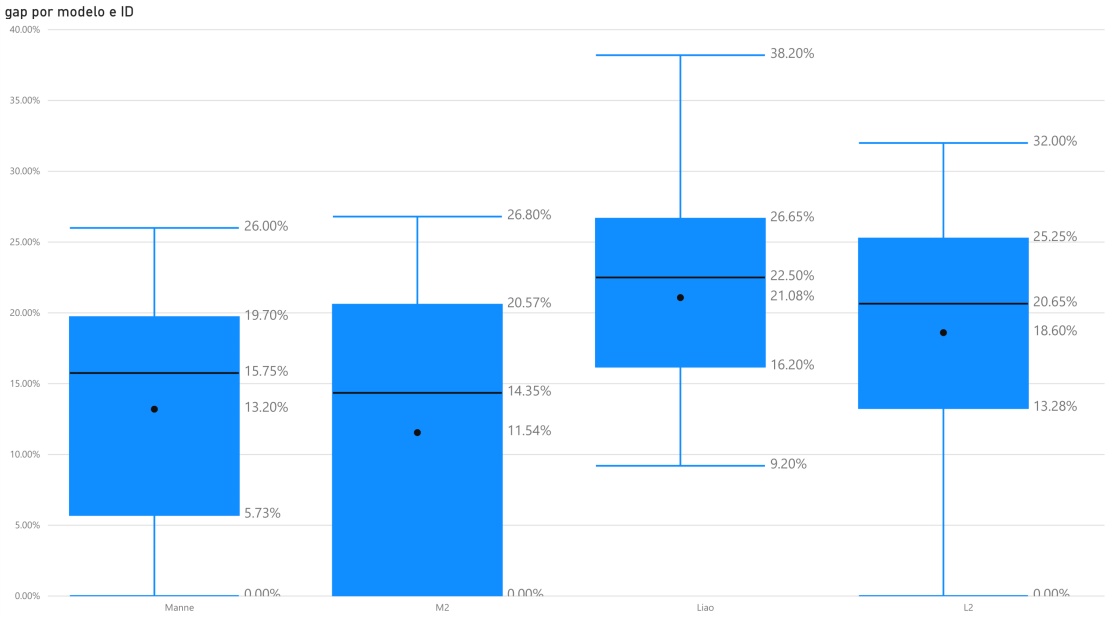

Figura 15 – Resultados do *gap* em gráfico de caixa para os modelos de $(MANNE), (M2), (LIAO)$ e $(L2)$ para o conjunto de instâncias I_1.

Por fim, podemos comparar os *gaps* de todos os modelos entre si. A Figura 15 representa um gráfico de caixa dos *gaps*. Podemos notar que os modelos de $(MANNE)$ e $(M2)$ tiveram o comportamento em geral menor do que $(LIAO)$ e $(L2)$. Inclusive, o modelo de $(LIAO)$ teve um comportamento com os maiores *gaps*, tendo a maior média e mediana, e o modelo $(M2)$ obteve a menor média e mediana.

6.2 Resultados computacionais para instâncias da literatura para os modelos Manne (1960), (M2) e (MLJ2)

Para os testes realizados para as instâncias da literatura, escolhemos o modelo de $(MANNE)$, por ter tido um comportamento melhor do que o de $(LIAO)$ em relação aos *upper bounds*. Dessa maneira, testamos o conjunto de instâncias I_2 com $(MANNE)$, $(M2)$ e $(MLJ2)$.

Para obtenção dos *upper bounds* a meta-heurística GRASP-ELS foi executada com 200 iterações do GRASP, $\alpha = 0.6$, 100 iterações do ELS e 20 permutações. Esses parâmetros foram encontrados de maneira empírica e, vale ressaltar, que diversas avaliações poderiam ser feitas para estimá-los de maneira mais eficiente, assim como obter mais análises acerca dos valores encontrados pela meta-heurística. Contudo, o objetivo do trabalho não é analisar a qualidade da meta-heurística. Então,

focamos no comportamento dos modelos de programação linear inteira para resolução do problema do sequenciamento e não abordaremos essas questões referentes à meta-heurística com mais profundidade. Dessa forma, o c_{max} obtido em cada instância foi utilizado como *upper bound* nos 3 modelos. A Figura 16 abaixo representa o tempo de execução para cada instância.

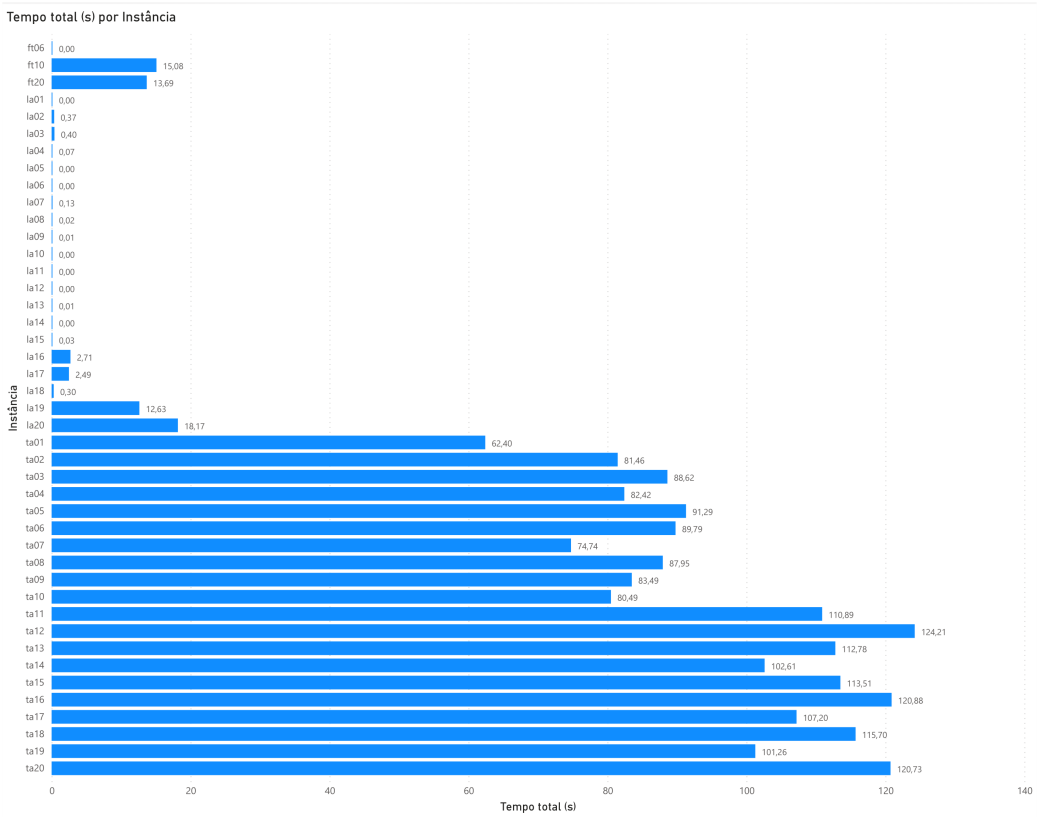

Figura 16 – Tempo (em segundos) para obtenção das soluções de partida usando a meta-heurística GRASP-ELS para cada instância do conjunto I_2.

Notamos que, em geral, o tempo de execução foi baixo, sendo o maior de aproximadamente 125 segundos e a média de cerca de 47 segundos. Também observamos claramente que as instâncias de Taillard (1993) são mais demoradas do que as demais, visto que tiveram os maiores tempos de execução.

Os resultados da execução dos modelos estão descritos na Tabela 7. A legenda dessa tabela é a mesma das tabelas anteriores, sendo que HEUR indica o valor de UB obtido pela GRASP-ELS. Esse valor pode não ser o melhor valor de solução viável conhecido da literatura. Quando coincidir, o valor estará em negrito na coluna HEUR. Além disso, a RL do modelo $(MLJ2)$ estará em negrito quando for maior que a RL de $(MANNE)$ e $(M2)$.

Instância				$(MANNE)$					$(M2)$					$(MLJ2)$				
ID	$\|J\|$	$\|M\|$	HEUR	UB	gap%	cpu (s)	RL	cpu-rl(s)	UB	gap%	cpu (s)	RL	cpu-rl(s)	UB	gap%	cpu (s)	RL	cpu-rl(s)
ft06	6	6	**55**	55	0.00	0.01	47.00	0.00	55	0.00	0.01	47.00	0.00	55	0.00	0.01	47.00	0.00
ft10	10	10	**930**	930	0.00	30.11	655.00	0.00	930	0.00	19.43	741.58	0.01	930	0.00	37.25	741.58	0.03
ft20	20	5	**1165**			7200.00	387.00	0.01			7200	744.02	0.08			7200.00	**744.05**	0.33
la01	10	5	**666**	666	0.00	2.89	413.00	0.00	666	0.00	2.66	541.70	0.01	666	0.00	2.40	541.70	0.03
la02	10	5	**655**	655	0.00	2.56	394.00	0.00	655	0.00	2.39	484.12	0.01	655	0.00	2.85	484.12	0.02
la03	10	5	**597**	597	0.00	0.84	349.00	0.00	597	0.00	0.89	441.64	0.01	597	0.00	0.55	**442.26**	0.02
la04	10	5	**590**	590	0.00	0.37	369.00	0.00	590	0.00	0.32	463.44	0.01	590	0.00	0.35	463.44	0.03
la05	10	5	**593**	593	0.00	21.08	380.00	0.00	593	0.00	18.05	453.18	0.01	593	0.00	10.25	453.18	0.02
la06	15	5	**926**	926	29.00	7200.00	413.00	0.00	926	32.00	7200.00	667.85	0.03	926	30.00	7200.00	667.85	0.09
la07	15	5	**890**	890	25.00	7200.00	376.00	0.00	890	22.00	7200.00	630.94	0.03	890	27.00	7200.00	630.94	0.09
la08	15	5	**863**	863	23.00	7200.00	369.00	0.00	863	23.00	7200.00	602.52	0.05	863	23.00	7200.00	**603.34**	0.17
la09	15	5	**951**	951	22.00	7200.00	382.00	0.00	951	21.00	7200.00	666.24	0.04	951	23.00	7200.00	**666.30**	0.14
la10	15	5	**958**	958	31.00	7200.00	443.00	0.00	958	31.00	7200.00	684.63	0.09	958	32.00	7200.00	**684.64**	0.11
la11	20	5	**1222**	1222	49.00	7200.00	413.00	0.01	1222	48.00	7200.00	820.85	0.08	1222	46.00	7200.00	**820.89**	0.34
la12	20	5	**1039**	1039	31.00	7200.00	408.00	0.01	1039	35.00	7200.00	720.41	0.09	1039	34.00	7200.00	**720.79**	0.36
la13	20	5	**1150**	1150	41.00	7200.00	382.00	0.01	1150	42.00	7200.00	762.04	0.10	1150	40.00	7200.00	**762.19**	0.37
la14	20	5	**1292**	1292	52.00	7200.00	443.00	0.01	1292	53.00	7200.00	860.48	0.10	1292	51.00	7200.00	**860.59**	0.32
la15	20	5	**1207**	1207	51.00	7200.00	378.00	0.01	1207	48.00	7200.00	824.84	0.09	1207	46.00	7200.00	**824.92**	0.34
la16	10	10	**945**	945	0.00	2.70	717.00	0.00	945	0.00	2.49	758.18	0.01	945	0.00	2.71	758.18	0.03
la17	10	10	**784**	784	0.00	4.02	646.00	0.00	784	0.00	2.66	651.51	0.01	784	0.00	2.27	651.51	0.03
la18	10	10	**848**	848	0.00	0.73	663.00	0.00	848	0.00	0.38	667.38	0.01	848	0.00	0.30	667.38	0.03
la19	10	10	**842**	842	0.00	0.65	617.00	0.00	842	0.00	0.74	652.67	0.01	842	0.00	1.86	652.67	0.03
la20	10	10	**902**	902	0.00	0.45	756.00	0.00	902	0.00	0.32	756.00	0.02	902	0.00	0.44	756.00	0.03
ta01	15	15	1242	1231	0.00	1488.63	963.00	0.02	1231	0.00	1291.97	963.00	0.10	1231	0.00	768.90	963.00	0.21
ta02	15	15	1249	1244	0.00	3446.75	942.00	0.02	1244	0.00	3665.86	942.00	0.10	1244	6.00	7200.00	942.00	0.17
ta03	15	15	1233	1218	0.00	601.59	921.00	0.02	1218	0.00	819.94	921.00	0.14	1218	0.00	1084.27	921.00	0.20
ta04	15	15	1196	1175	0.00	1295.83	911.00	0.02	1175	0.00	659.79	972.23	0.08	1175	0.00	1244.89	972.23	0.17
ta05	15	15	1237	1235	7.00	7200.00	940.00	0.02	1235	8.00	7200.00	966.28	0.10	1231	5.00	7200.00	966.28	0.18
ta06	15	15	1252			7200.00	849.00	0.02			7200	969.85	0.11			7200.00	969.85	0.20
ta07	15	15	1228	1227	0.00	3494.16	935.00	0.02	1227	0.00	2801.10	950.47	0.09	1227	0.00	3581.36	950.47	0.16
ta08	15	15	1221	1217	0.00	1362.06	963.00	0.02	1217	0.00	1342.27	963.00	0.10	1217	0.00	535.37	963.00	0.20
ta09	15	15	1299	1274	0.00	1447.40	982.00	0.02	1274	0.00	2142.06	982.00	0.11	1274	0.00	1970.99	982.00	0.20
ta10	15	15	1265	1241	0.00	1625.30	896.00	0.02	1241	0.00	1685.59	975.35	0.10	1241	0.00	3282.43	975.35	0.20
ta11	20	15	1428			7200.00	949.00	0.04			7200.00	1061.12	0.23			7200.0	1061.12	0.81
ta12	20	15	1414			7200.00	1012.00	0.07			7200.00	1076.21	0.26			7200.0	**1076.26**	0.76
ta13	20	15	1395			7200.00	919.00	0.04			7200.00	1041.92	0.28			7200.0	**1041.93**	0.84
ta14	20	15	1356			7200.00	990.00	0.05			7200.00	1015.15	0.23			7200.0	1015.15	0.76
ta15	20	15	1410			7200.00	880.00	0.04			7200.00	961.93	0.26			7200.0	**962.06**	0.93
ta16	20	15	1408			7200.00	932.00	0.06			7200.00	1061.21	0.27			7200.0	1061.21	0.74
ta17	20	15	1496	1496	22.00	7200.00	979.00	0.04			7200.00	1053.58	0.30			7200.0	1053.58	0.93
ta18	20	15	1463			7200.00	900.00	0.06			7200.00	1067.60	0.22			7200.0	1067.60	0.83
ta19	20	15	1399			7200.00	920.00	0.05			7200.00	1019.38	0.29			7200.0	**1019.39**	0.86
ta20	20	15	1394			7200.00	928.00	0.04			7200.00	1052.59	0.29			7200.0	**1052.60**	0.84
Médias				983.22	11.97	4196	677	0.02	966.68	11.71	4187.42	806.03	0.11	966.55	11.71	4309.99	806.08	0.31

Tabela 7 – Resultados para os modelos $(MANNE)$, $(M2)$ e $(MLJ2)$ para as instâncias do grupo I_2

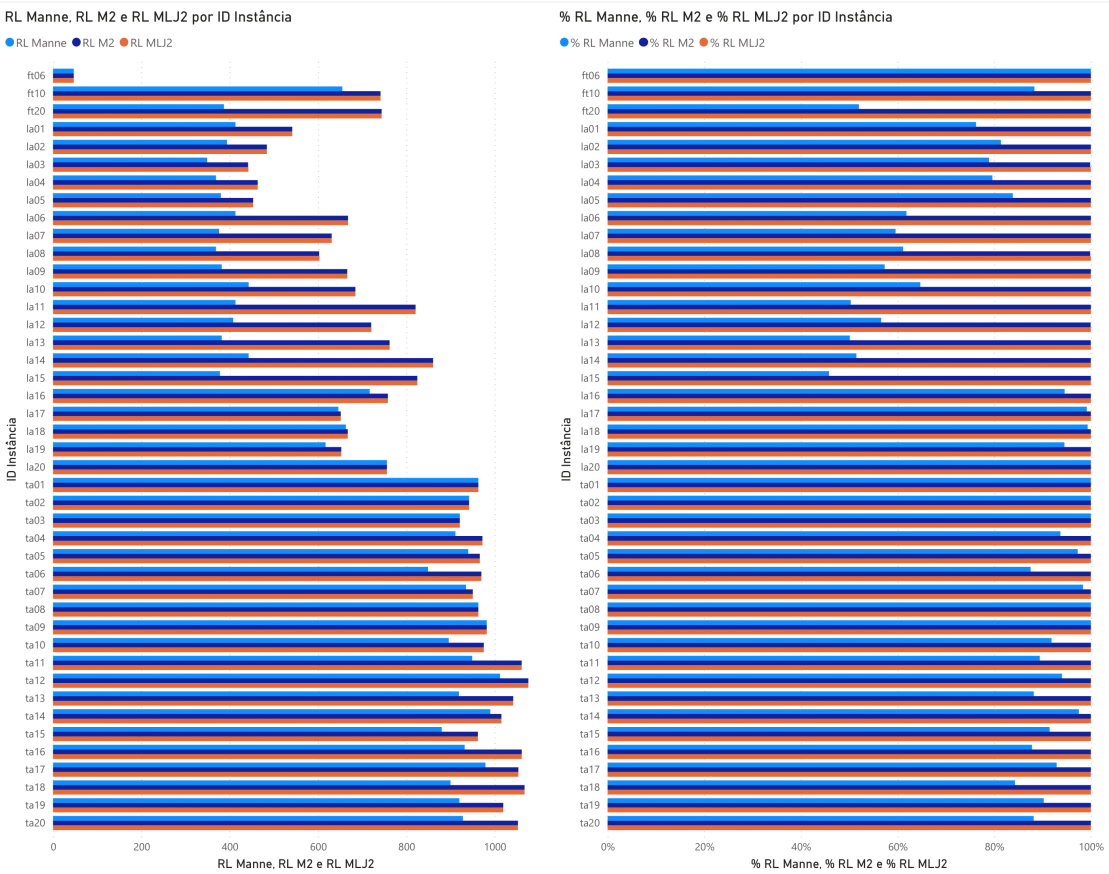

Figura 17 – Resultados da RL para os modelos de $(MANNE)$, $(M2)$ e $(MLJ2)$ para o conjunto de instâncias I_2.

Podemos ver na Tabela 7 que os modelos $(M2)$ e $(MLJ2)$ obtiveram um aumento considerável na relaxação linear e $VRL_{M2,Manne} = VRL_{MLJ2,Manne} = 19.06\%$, ou seja, um aumento da média do valor da relaxação linear em 19.6%. Pela Figura 17 notamos que em algumas instâncias pontuais, como em $la15$, o valor da relaxação linear dos modelos $(M2)$ e $(MLJ2)$ mais que dobrou em relação ao de $(MANNE)$. Em cerca de 84% das instâncias o modelo $(M2)$ e $(MLJ2)$ tiveram relaxações lineares estritamente maiores do que o modelo de $(MANNE)$. Além disso, na comparação de $(M2)$ com $(MLJ2)$ não notamos grande diferença nos valores avaliados. Contudo, a relaxação linear de $(MLJ2)$ é ligeiramente maior para 15 das 43 instâncias. Dessa forma, para esses experimentos computacionais, o modelo $(MLJ2)$ mostrou-se mais apertado que $(MANNE)$ e $(M2)$, sendo que a prova formal da validade ou não dessa afirmação para o caso geral não foi tratada neste trabalho.

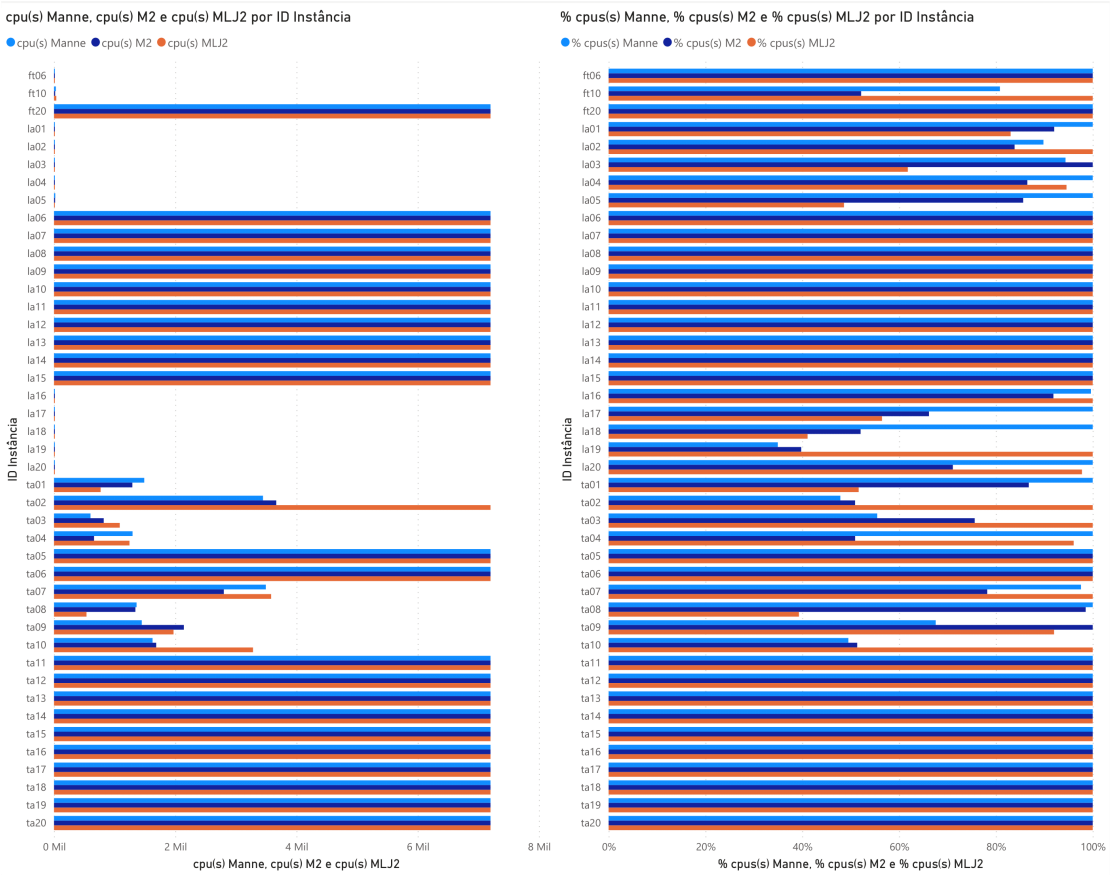

Figura 18 – Resultados do cpu(s) para os modelos de $(MANNE)$, $(M2)$ e $(MLJ2)$ para o conjunto de instâncias I_2.

Podemos verificar na Tabela 7 que os modelos de $(MANNE)$, $(M2)$ e $(MLJ2)$ atingiram o tempo limite de execução em 23 das 43 instâncias, ou seja cerca de metade das instâncias. Além disso, em 10 das 43 instâncias, ou seja 25%, o modelo de $(MANNE)$ não conseguiu achar pelo menos uma solução viável. Já os modelos $(M2)$ e $(MLJ2)$ não conseguiram encontrar solução viável em 11 instâncias. Pela Figura 18 notamos que em algumas instâncias, por exemplo $la90$, $ta02$, $ta03$ e $ta10$, o modelo $(MLJ2)$ obteve um tempo de execução bem superior, quando comparado aos modelos de $(MANNE)$ e $(M2)$. Isso evidencia a dificuldade de resolução das instâncias, principalmente para o conjunto de instâncias de Taillard (1993), pois em metade dessas instâncias o modelo de $(MANNE)$ não encontrou uma solução viável.

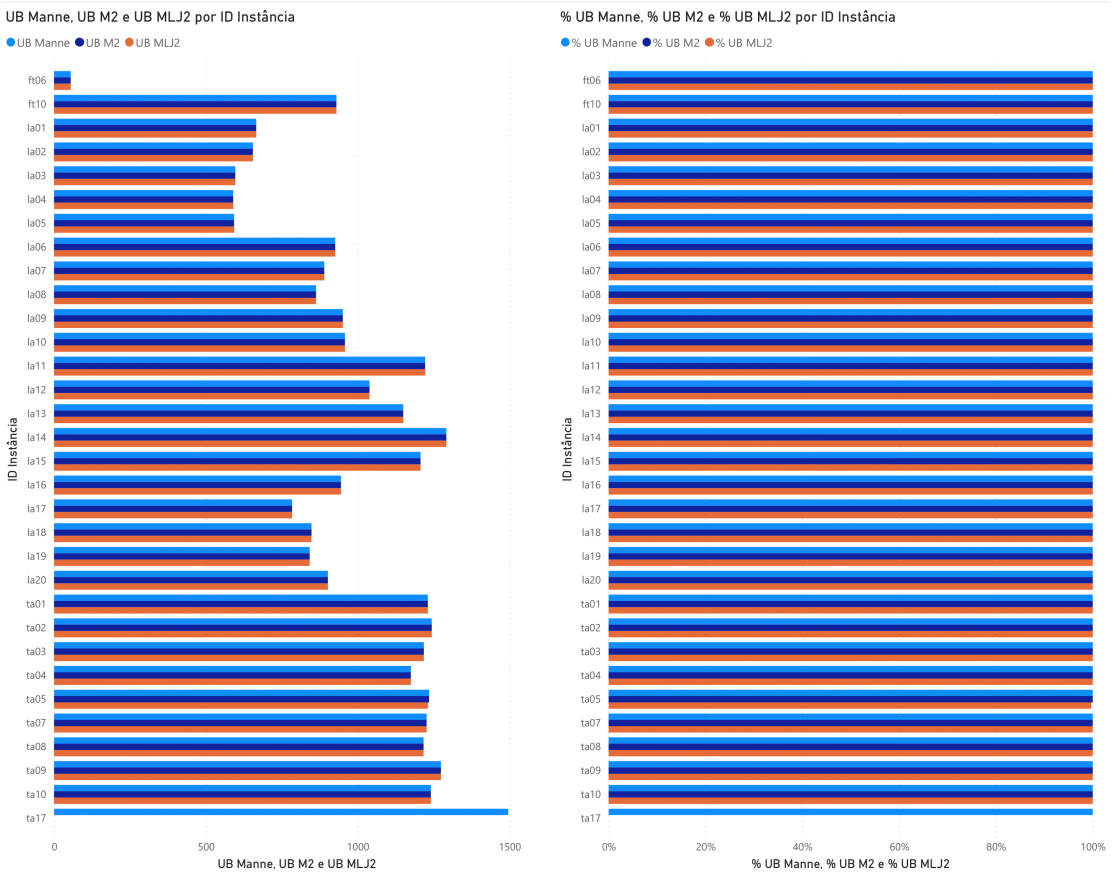

Figura 19 – Resultados do UB para os modelos de $(MANNE)$, $(M2)$ e $(MLJ2)$ para o conjunto de instâncias I_2.

Vemos na Tabela 7 que para o modelo $(MANNE)$ o CPLEX encontrou uma solução com mesmo *makespan* do *upper bound* da meta-heurística para 31 das 43 instâncias. Para o modelo $(M2)$ isso ocorreu em 27 instâncias e para o modelo $(MLJ2)$ para 28 instâncias. Vemos na Figura 19 que, quando os modelos encontraram soluções, o *upper bound* foi igual em praticamente todas as instâncias, com exceção da instância $ta05$ que o modelo $(MLJ2)$ encontrou um valor de *makespan* um pouco menor que os demais modelos.

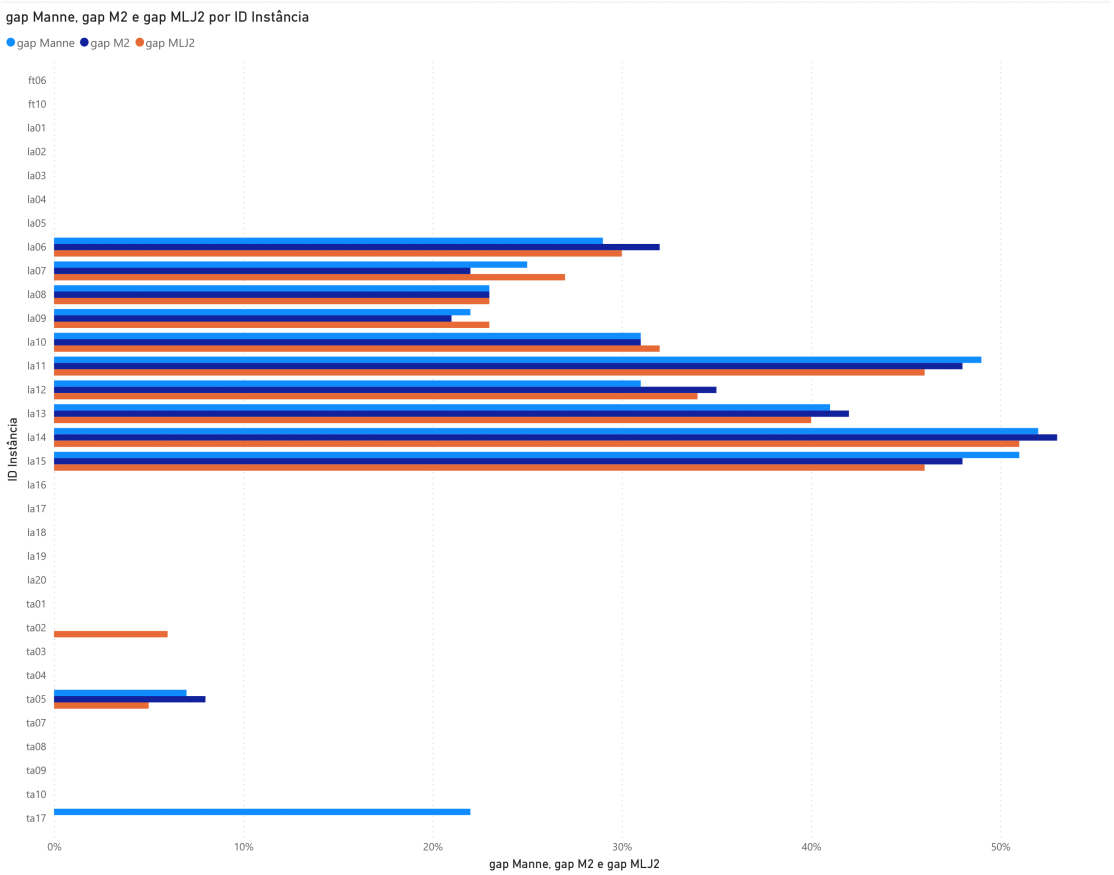

Figura 20 – Resultados do *gap* para os modelos de $(MANNE)$, $(M2)$ e $(MLJ2)$ para o conjunto de instâncias I_2.

Analisando a Tabela 7 temos que para o *gap*, em geral, todos os modelos tiveram valores muito próximos. Contudo, considerando as 33 instâncias em que o modelo de $(MANNE)$ encontrou pelo menos uma solução viável, em 27 delas o modelo $(M2)$ e $(MLJ2)$ encontraram um *gap* pelo menos tão bom quanto o de $(MANNE)$. De fato, em 4 dessas 27 instâncias os modelos $(M2)$ e $(MLJ2)$ tiveram *gaps* estritamente menores. Pela Figura 20 observamos que os valores de *gap* dos três modelos para as instâncias $la06$ a $la16$ foi bem semelhante.

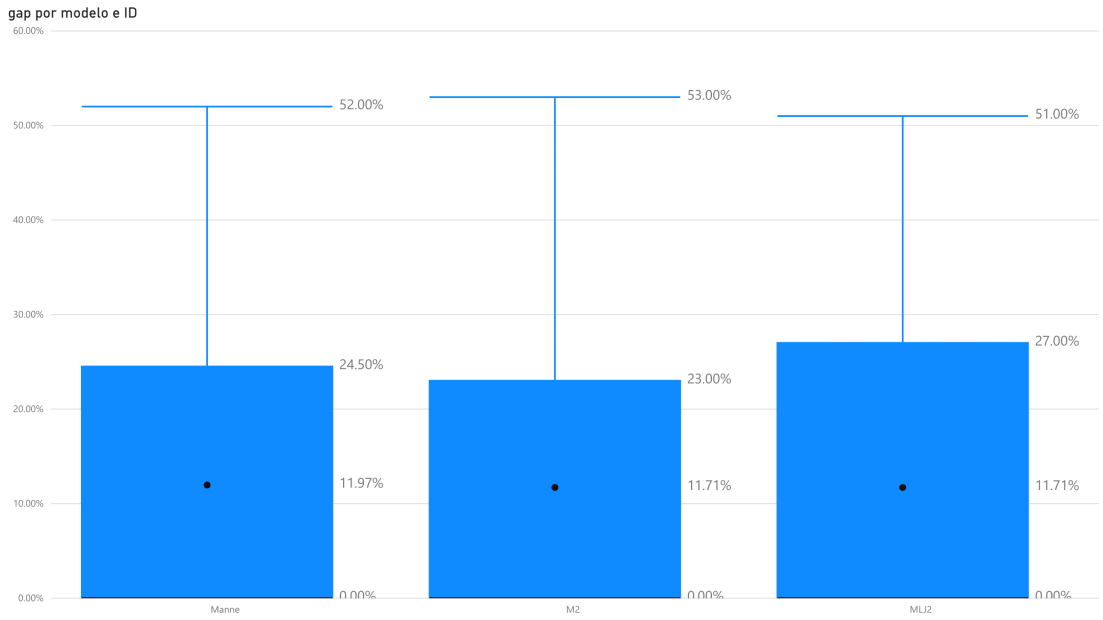

Figura 21 – Resultados do *gap* em gráfico de caixa para os modelos de $(MANNE)$, $(M2)$ e $(MLJ2)$ para o conjunto de instâncias I_2.

Por fim, ficou evidente que a melhora significativa na qualidade dos *lower bounds* não se refletiu na obtenção de melhores soluções viáveis ou para provar a otimalidade das soluções obtidas pelo CPLEX. Contudo, vemos no gráfico de caixa da Figura 21 que a média dos *gaps* dos modelos $(M2)$ e $(MLJ2)$ foram um pouco inferiores quando comparadas à media do modelo de $(MANNE)$. Além disso, em relação aos *upper bounds* da literatura, não tivemos êxito em encontrar soluções melhores. Esse ponto carece de mais estudo para descobrir como tirar proveito das novas desigualdades que fortaleceram as relaxações lineares dos modelos testados.

7 CONCLUSÃO

Abordamos neste trabalho o problema do sequenciamento de trabalhos com minimização do *makespan*. Desenvolvemos um modelo baseado em arranjo linear denominado (MLJ), bem como desigualdades válidas com base em propriedades estruturais do problema. O objetivo foi fortalecer a relaxação linear de alguns dos principais modelos matemáticos para o JSSP disponíveis na literatura. Precisamente, as novas desigualdades possibilitaram obter melhores *lower bounds* para as variáveis x de início de processamento das trabalhos nas máquinas e na variável de *makespan* C_{max}. Conforme observado em experimentos computacionais preliminares em Cavalcante *et al.* (2020), assim como nos novos experimentos, de fato as desigualdades propostas melhoram a qualidade do valor da solução da relaxação linear dos modelos testados.

De uma forma geral, os modelos $(M2)$ e $(MLJ2)$ foram eficazes na resolução de nossas instâncias quando comparados ao modelo $(MANNE)$. A melhora na relaxação linear foi bem superior com $(M2)$ e $(MLJ2)$, chegando a melhorar as médias dos valores da relaxação linear em cerca de 19% em relação ao modelo original de $(MANNE)$ sendo que em cerca de 84% das instâncias os modelos desenvolvidos testados tiveram relaxações lineares estritamente maiores. Dessa forma, evidenciando a eficácia das novas restrições desenvolvidas para o JSSP. Por outro lado, infelizmente não fomos capazes de fazer com que esses modelos tirassem proveito da melhora de suas relaxações lineares. Ainda assim, espera-se que tais resultados abram uma nova direção para o desenvolvimento teórico do problema quanto ao estudo do fortalecimento desses modelos como trabalhos futuros.

Existe a possibilidade de aproveitar famílias exponenciais de desigualdades válidas para o modelo de arranjo linear em um algoritmo de *branch-and-cut* (B&C) do MIP CPLEX que sabidamente fortalecem o modelo MinLA que não usamos neste trabalho. Igualmente, já que algumas das desigualdades propostas tornaram pesada, em termos de tempo de execução, a resolução dos modelos para o JSSP aqui testados, é possível explorá-las em um algoritmo de B&C como cortes pré-definidos em *user-cut-callbacks* do CPLEX à medida que forem sendo violadas ao longo da árvore de busca do CPLEX.

Além disso, como trabalhos futuros também podemos pensar em adaptar essas desigualdades válidas para a perspectiva de programação por restrições, testar

o modelo (MLJ) e $(MLJ2)$ com outras funções objetivos para o sequenciamento, estudando dessa maneira o impacto das desigualdades nas funções objetivos, e utilizar a ideia da adaptação do modelo MinLA como geração de colunas do JSSP.

REFERÊNCIAS

ADAMS, J.; BALAS, E.; ZAWACK, D. The shifting bottleneck procedure for job shop scheduling. **Management science**, United States, v. 34, n. 3, p. 391–401, 1988. Disponível em <https://www.jstor.org/stable/2632051>. Acesso em 17 mai. 2020.

AIEX, R. M.; BINATO, S.; RESENDE, M. G. C. Parallel GRASP with path-relinking for job shop scheduling. **Parallel Comput.**, v. 29, n. 4, p. 393–430, Apr. 2003. Disponível em <https://www.sciencedirect.com/science/article/pii/S0167819103000140>. Acesso em 17 jan. 2020.

ANDRADE, R.; BONATES, T.; CAMPÊLO, M.; FERREIRA, M. Minimum linear arrangements. **Electronic Notes in Discrete Mathematics**, Elsevier, Amsterdam, v. 62, p. 63–68, Nov. 2017. Disponível em <https://doi.org/10.1016/j.endm.2017.10.012>. Acesso em 18 set. 2018.

BISSOLI, D. de C. **Meta-heurísticas para resolução de alguns problemas de planejamento e controle da produção**. Tese (Doutorado) — Centro Tecnológico, Universidade Federal do Espírito Santo, Vitória, 2018. Disponível em <http://repositorio.ufes.br/handle/10/10728>. Acesso em 24 ago. 2020.

BOWMAN, E. H. The schedule-sequencing problem. **Operations Research**, United States, v. 7, n. 5, p. 621–624, 1959. Disponível em <https://www.jstor.org/stable/167010>. Acesso em 25 set. 2019.

CAVALCANTE, J.; ANDRADE, R.; DUHAMEL, C. Desigualdades válidas para o problema do job shop com minimização do makespan. **Anais do LII Simpósio Brasileiro de Pesquisa Operacional**, 2020. Disponível em <https://proceedings.science/sbpo-2020/papers/desigualdades-validas-para-o-problema-do-job-shop-com-minimizacao-do-makespan>. Acesso em 18 jan. 2021.

COLORNI, A.; DORIGO, M.; MANIEZZO, V.; TRUBIAN, M. Ant system for job-shop scheduling. **JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science**, v. 34, n. 1, p. 39–53, 1994. Disponível em <https://www.orbel.be/jorbel/index.php/jorbel/article/view/169>. Acesso em 18 mai. 2020.

DAVIS, L. Job shop scheduling with genetic algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS AND THEIR APPLICATIONS. **Proceedings of an international conference on genetic algorithms and their applications**. [S.l.], 1985. v. 140, p. 136–140.

FEO, T. A.; RESENDE, M. G. A probabilistic heuristic for a computationally difficult set covering problem. **Operations Research Letters**, Netherlands, v. 8, n. 2, p. 67 – 71, 1989. Disponível em <https://www.sciencedirect.com/science/article/abs/pii/0167637789900023>. Acesso em 09 jan. 2021.

FERREIRA, S. M. **Problema do arranjo linear mínimo**. 84 p. Dissertação (Mestrado) — Universidade Federal do Ceará, 2016. Disponível em <http://www.repositorio.ufc.br/handle/riufc/21501>. Acesso em 21 jan. 2019.

FISHER, H. Probabilistic learning combinations of local job-shop scheduling rules. **Industrial scheduling**, Prentice-Hall, p. 225–251, 1963.

JOHNSON, S. M. Optimal two-and three-stage production schedules with setup times included. **Naval research logistics quarterly**, United States, v. 1, n. 1, p. 61–68, Mar. 1954.

KONDILI, E.; PANTELIDES, C.; SARGENT, R. *et al.* A general algorithm for scheduling batch operations. In: INSTITUTION OF ENGINEERS. **PSE'88: Third International Symposium on Process Systems Engineering**. [S.l.]: In Affiliation with CHEMECA 88, a Bicentennial Event; Sydney, Australia, 28 August-2 September, 1998; Preprints of Papers, 1988. p. 62. Disponível em <https://www.researchgate.net/publication/272294074_A_General_Algorithm_for_Scheduling_Batch_Operations>. Acesso em 2 mai. 2019.

KU, W.-Y.; BECK, J. C. Mixed integer programming models for job shop scheduling: a computational analysis. **Computers & Operations Research**, v. 73, p. 165–173, Sept. 2016. Disponível em <https://www.sciencedirect.com/science/article/abs/pii/S0305054816300764>. Acesso em 25 mai. 2019.

LAARHOVEN, P. J. V.; AARTS, E. H.; LENSTRA, J. K. Job shop scheduling by simulated annealing. **Operations research**, United States, v. 40, n. 1, p. 113–125, 1992.

LAWRENCE, S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). **Graduate School of Industrial Administration, Carnegie-Mellon University**, Pittsburgh, PA, 1984.

LIAO, C.-J.; YOU, C.-T. An improved formulation for the job-shop scheduling problem. **Journal of the Operational Research Society**, United Kingdom, v. 43, n. 11, p. 1047–1054, 1992.

LITTLE, J. D.; MURTY, K. G.; SWEENEY, D. W.; KAREL, C. An algorithm for the traveling salesman problem. **Operations research**, United States, v. 11, n. 6, p. 972–989, 1963.

LOMNICKI, Z. A "branch-and-bound" algorithm for the exact solution of the three-machine scheduling problem. **Journal of the operational research society**, United Kingdom, v. 16, n. 1, p. 89–100, 1965.

MANNE, A. S. On the job-shop scheduling problem. **Operations Research**, United States, v. 8, n. 2, p. 219–223, 1960.

MERZ, P.; WOLF, S. Evolutionary local search for designing peer-to-peer overlay topologies based on minimum routing cost spanning trees. In: RUNARSSON, T. P.; BEYER, H.; BURKE, E. K.; GUERVÓS, J. J. M.; WHITLEY, L. D.; YAO, X. (Ed.). **Parallel Problem Solving from Nature - PPSN IX, 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Procedings**. [S.l.]: New York, Springer, 2006. (Lecture Notes in Computer Science, v. 4193), p. 272–281.

NOWICKI, E.; SMUTNICKI, C. An advanced tabu search algorithm for the job shop problem. **Journal of Scheduling**, United States, v. 8, n. 2, p. 145–159, 2005.

Disponível em <https://link-springer-com.ez11.periodicos.capes.gov.br/content/pdf/10.1007/s10951-005-6364-5.pdf>. Acesso em 17 jan. 2021.

OLIVEIRA, B. B.; CARRAVILLA, M. A. Understanding complexity in a practical combinatorial problem using mathematical programming and constraint programming. In: SPRINGER. **Congress of APDIO, the Portuguese Operational Research Society**. [S.l.], 2017. p. 269–295.

PINEDO, M.; HADAVI, K. Scheduling: theory, algorithms and systems development. In: **Operations Research Proceedings 1991**. [S.l.]: Springer, 1992. p. 35–42.

PIROOZFARD, H.; WONG, K. Y.; HASSAN, A. A hybrid genetic algorithm with a knowledge-based operator for solving the job shop scheduling problems. **Journal of Optimization**, United Kingdom, v. 2016, p. 1–13, 1, 2016. Disponível em <http://search-ebscohost-com.ez11.periodicos.capes.gov.br/login.aspx?direct=true&db=iih&AN=116637886&lang=pt-br&site=ehost-live>. Acesso em 04 mai. 2019.

SEITZ, H. **Contributions to the minimum linear arrangement problem**. Tese (Doutorado) — Heidelberg University, Faculty of Mathematics and Informatics, Heidelberg, 2010. Disponível em <http://archiv.ub.uni-heidelberg.de/volltextserver/10578/1/MinLA_Thesis_Seitz.pdf>. Acesso em 18 fev. 2019.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, , North-Holland, v. 64, n. 2, p. 278–285, 1993. Disponível em <https://www.sciencedirect.com/science/article/abs/pii/037722179390182M>. Acesso em 1 mar. 2019.

WAGNER, H. M. An integer linear-programming model for machine scheduling. **Naval Research Logistics Quarterly**, United States, v. 6, n. 2, p. 131–140, 1959.

ZHANG, J.; DING, G.; ZOU, Y.; QIN, S.; FU, J. Review of job shop scheduling research and its new perspectives under industry 4.0. **Journal of Intelligent Manufacturing**, United Kingdom, v. 30, n. 4, p. 1809–1830, 2019. Disponível em <http://search-ebscohost-com.ez11.periodicos.capes.gov.br/login.aspx?direct=true&db=iih&AN=135452240&lang=pt-br&site=ehost-live>. Acesso em 27 jan. 2021.