Universidade Federal de Itajubá

Instituto de Matemática e Computação

Ciência da Computação

Algoritmos de Programação por Restrições para o Problema do Dimensionamento de Lotes

Jonas de Freitas Ramos

Itajubá, Julho de 2021

Jonas de Freitas Ramos

Algoritmos de Programação por Restrições para o Problema do Dimensionamento de Lotes

Monografia apresentada como parte dos requisitos necessários para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Pedro Henrique Del Bianco Hokama
Co-Orientador: Prof. Dr. Mario César San Felice

Esta versão corresponde à versão da Monografia entregue à banca antes da defesa.

Itajubá
Julho de 2021

Na versão final esta página será substituída pela Ficha Catalográfica

Jonas de Freitas Ramos

Algoritmos de Programação por Restrições para o Problema do Dimensionamento de Lotes

---

Prof. Dr. Pedro Henrique Del Bianco Hokama

Orientador

Prof. Dr. Mario César San Felice

Co-Orientador

Prof. Dr. Rafael de Magalhaes Dias Frinhani

Convidado

Itajubá

Julho de 2021

Agradecimentos

Na versão final esta página será substituída pelos agradecimentos.

Resumo

Neste trabalho estamos interessados em uma formulação em Programação por Restrições para o problema do Dimensionamento de Lotes Capacitado de Item Único, que consiste em planejar a produção de um único tipo de item ao longo de um horizonte de planejamento finito e discreto de maneira a atender integralmente a demanda a cada período e minimizar o custo total envolvido no processo. A cada período é possível produzir itens para o período atual e/ou guardar itens para os próximos períodos, sabendo que haverá um custo decorrente dessa escolha.

Há algoritmos consolidados para a resolução do problema que utilizam paradigmas clássicos como a Programação Dinâmica e a Programação Linear Inteira. Nesse trabalho apresentamos uma resolução utilizando o paradigma da Programação por Restrições, um paradigma relativamente novo se comparado com os demais. Como não existem na literatura muitas abordagens para esse problema usando Programação por Restrições, nosso objetivo foi desenvolver um algoritmo competitivo, em particular para instâncias onde as abordagens mais tradicionais têm dificuldade. Além disso a Programação por Restrições permite a inclusão de restrições práticas, normalmente de maneira mais simples que as outras abordagens. Uma ferramenta chave nesse processo de elaboração do algoritmo são as restrições redundantes. Essas restrições não são obrigatórias na formulação do problema, mas são restrições que visam aumentar o poder de poda das restrições primárias, diminuindo o tempo de processamento necessário para a resolução do problema.

Propomos três restrições redundantes que foram incorporadas na formulação original. O algoritmo foi implementado utilizando o IBM CP Optimizer e obteve resultados competitivos às abordagens tradicionais em instâncias onde as capacidades de produção e estoque são grandes.

Palavras-chave: Programação por Restrições. Programação Linear Inteira. Programação Dinâmica. Pesquisa Operacional.

Esse projeto está vinculado ao projeto *Algoritmos para o problema de dimensionamento de lote, estoque e roteirização com restrições de empacotamento* registrado na DIP com o número 732 e financiado pela Chamada Universal de 2018 do CNPq.

Abstract

In this work we are interested in a Constraint Programming formulation for the Capacitated Single Item Lot Sizing Problem, which consists of planning the production of a single type of item over a finite and discrete planning horizon in order to fully meet the demand at each period and minimize the total cost involved in the process. At each period it is possible to produce items for the current period and/or save items for the next periods, knowing that there will be a cost from this choice.

There are consolidated algorithms to solve the problem that use classical paradigms such as Dynamic Programming and Integer Linear Programming. In this work we present a solution using the Constraint Programming paradigm, a relatively new paradigm compared to the others. As there are not many approaches in the literature for this problem using Constraint Programming, our goal is to develop a competitive algorithm, in particular for instances where more traditional approaches find difficulties. In addition, Constraint Programming allows the inclusion of practical constraints, usually in a simpler way than other approaches. A key tool in this algorithm elaboration process is the redundant constraints. These restrictions are not mandatory in the formulation of the problem, but they are restrictions that aim to increase the pruning power of the primary restrictions, reducing the processing time needed to solve the problem.

We propose three redundant constraints that were incorporated in the original formulation. The algorithm was implemented using IBM CP Optimizer and achieved competitive results to traditional approaches in instances where production and inventory capacities are large.

Key-Words: Constraint Programming. Integer Linear Programming. Dinamic Programming. Operational Research.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas

1. LSP - Lot Sizing Problem
2. CSILSP - Capacitated Single Item Lot Sizing Problem
3. USILSP - Uncapacitated Single Item Lot Sizing Problem
4. PL - Linear Programming
5. PLI - Integer Linear Programming
6. MILP - Mixed Integer Linear Programming
7. DP - Dinamic Programming
8. CP - Contraint Programming
9. LB - Lower Bound
10. RFF - Relaxed Flow Filter
11. WMF - Window Max Flow

1 Introdução

Os Problemas de Dimensionamento de Lotes (Lot-sizing Problems - LSP) têm fundamental importância em áreas que dependem do planejamento de produções ao longo de um determinado período de tempo. Saber planejar a produção de maneira a aumentar a produtividade e reduzir os custos é um fator crucial no processo de tomada de decisão, presente na maioria das grandes empresas na atualidade.

Existem diversas variantes dos LSP que buscam maior similaridade com os problemas enfrentados em casos reais. Alguns dos exemplos são com demandas variando ou constantes, produções limitadas ou ilimitadas, custos de inventário etc. Cada variante tende atender um segmento específico.

Os algoritmos clássicos mais conhecidos para os LSP se utilizam de paradigmas mais conhecidos e consolidados no meio acadêmico, como é o caso da Programação Dinâmica e da Programação Linear Inteira. Esses paradigmas conseguem obter bons resultados com formulações genéricas, sendo eficientes em boa parte das variantes do problema. Por outro lado, a Programação por Restrições é um paradigma mais recente e não tão explorado, que permite a elaboração de filtros que tem por objetivo o fortalecimento do algoritmo principal.

A presente monografia tem por objetivo investigar o Problema de Dimensionamento de Lotes Capacitado de Item Único (Capacitated Single Item Lot-sizing Problem - CSILSP), que é uma das variantes do LSP. O CSILSP consiste em planejar a produção de um único tipo de item ao longo de um período finito de tempo (T períodos discretos), de maneira a atender integralmente a demanda a cada período e minimizar os custos envolvidos nesse processo (custos de produção, inventário e setup). Iremos apresentar os algoritmos clássicos para a resolução do problema e uma formulação utilizando da Programação por Restrições, que é o foco da investigação.

Este trabalho está divido em - (1) Introdução, apresentando uma visão geral do trabalho - (2) Fundamentação Teórica, que apresenta uma revisão bibliográfica do problema e a descrição das técnicas utilizadas - (3) Definição do Problema, onde apresentamos a definição formal do problema e os algoritmos em Programação por Restrições, Programação Linear Inteira e Programação Dinâmica - (4) Restrições Redundantes, onde apresentamos as 3 restrições redundantes elaboradas para fortalecimento da formulação principal, (5) Resultados, onde apresentamos os resultados obtidos através da execução dos algoritmos - (6) Conclusões, onde apresentamos uma visão geral do que foi desenvolvido.

2 Fundamentação Teórica

O problema de dimensionamento de lotes não-capacitado (Uncapacitated Single Item Lot Sizing Problem - USILSP) pode ser resolvido em $O(T \log T)$ na maioria dos casos. Há casos que podem ser resolvidos em $O(T)$ quando assumimos algumas premissas. Em particular, quando não existem motivos especulativos para segurar o inventário, ou seja, onde a produção e estoque satisfazem a condição $p_{t-1} + h_t \geq p_t$ (sendo p_t o custo de produção de uma unidade do item no período t e h_t o custo de armazenamento de uma unidade do item do período $t-1$ para o t) em todos os períodos. Neste caso é melhor produzir sempre o mais tarde possível [25], estratégia conhecida como *just in time*. O primeiro algoritmo exato desenvolvido para o USILSP foi o algoritmo de programação dinâmica proposto por Wagner e Within em 1958, que pode ser resolvido em $O(T^2)$ [24]. No final dos anos 80 esse algoritmo foi aprimorado e resolve o problema em $O(T \log T)$ [3].

Já, ao tratarmos do problema com capacidade de produção (CSILSP), se a produção for constante (i. e. com um limite constante na produção) e considerarmos custos de setup, o problema pode ser resolvido em $O(T^4)$ com custos côncavos [7] e em $O(T^3)$ com custos lineares [12]. Quando a capacidade de produção varia de acordo com o tempo, o problema se torna NP-Difícil [2]. Todas as variantes do CSILSP já resolvidos em tempo polinomial são apresentados em [3, 2] e alguns resolvidos em $O(T \log T)$ são apresentados em [15, 6]. Quando consideramos algoritmos de programação por restrições (CP), em [14] encontramos uma variante que resolve o problema em tempo polinomial e visa minimizar os custos de inventário (custos de setup e produção são zero e custo de inventário é constante) utilizando uma restrição global.

Em Houndji et. al. [13] é apresentado um CSILSP com capacidade de inventário variando de acordo com o tempo, custo de inventário constante e capacidade da máquina constante. A resolução proposta se dá através de uma restrição global chamada StockingCost que é decomposta e aprimorada, com a utilização de um algoritmo em $O(n)$ que filtra os valores das variáveis e aumenta o poder de poda da StockingCost. Como auxiliares no processo, são apresentadas as restrições globais gcc [18] e cost_gcc [21] que auxiliam no processo de filtragem das variáveis.

O problema que será discutido no presente trabalho é o caso geral do CSILSP onde todos os valores variam de acordo com o tempo e não seguem um padrão quanto à valores crescentes e/ou decrescentes. Esse problema foi resolvido em tempo pseudo-polinomial utilizando programação dinâmica por Cheng et. al. [4] e por Grigori German usando uma combinação de programação por restrições e um algoritmo de filtragem que usa programação dinâmica [9]. Há

também uma resolução utilizando MILP (Mixed Integer Linear Programming) por Pochet et. al. em [17].

2.1 Programação por Restrições

A Programação por Restrições ou Constraint Programming (CP) é um paradigma de programação que tem se destacado nas últimas décadas, se mostrando bastante eficiente na resolução de grandes problemas combinatórios, em particular em áreas de planejamento e escalonamento. Outro ponto de destaque da CP é para problemas cuja formulação em Programação Linear Inteira é complicada ou muito complexa.

Uma formulação em CP se dá através da criação de restrições. Essas podem ser vistas como formalizações entre o mundo real e uma modelagem matemática. Dentro da CP, uma restrição nada mais é que a relação lógica entre as variáveis do problema. Dessa maneira, ela pode restringir os valores que as variáveis podem assumir, individualmente ou coletivamente. Uma forte característica de uma restrição é sua natureza declarativa, que especifica uma relação entre variáveis e não o procedimento que garante essa restrição.

Os primeiros traços da Programação por Restrições podem ser vistos nos estudos de Inteligência Artificial na década de 60, tendo o paradigma de programação lógica como seu precursor. Essas e outras formas de programação declarativa se preocupam em estabelecer o que deve ser resolvido, e não como resolver.

Restrições Uma restrição é representada por uma expressão envolvendo um conjunto ou subconjunto de variáveis. Em linhas gerais, uma restrição $R_{1,\ldots,n}$ entre as variáveis $x_1, \ldots, x_n$ é um subconjunto das combinações de valores de $x_1, \ldots, x_n$, isto é, $R_{1,\ldots,n} \subseteq Dom(x_1) \times \cdots \times Dom(x_n)$. O subconjunto define as combinações de valores que a restrição admite. Essa definição enfatiza que as restrições não precisam ser expressões simples, nem lineares. Alguns exemplos de restrições seriam, $x_1 \neq 2x_2, x_3 = 3x_2 + x_1$.

Satisfação de Restrições Formalmente, um Problema de Satisfação de Restrições (Constrtaint Satisfaction Problem - CSP) é definido da seguinte forma:

- um conjunto finito de variáveis $X = x_1, ..., x_n$,
- um conjunto finito $Dom(x_i)$ para toda variável x_i, com os possíveis valores de cada variável chamado domínio de x_i,

- um conjunto de restrições $C(X)$ para os valores que as variáveis podem assumir simultaneamente.

Em um CSP nós podemos encontrar:

- uma e somente uma solução, que não tenha preferência alguma sobre as demais;
- todas as possíveis soluções;
- uma solução boa ou ótima, para uma dada função objetivo.

Soluções para um CSP podem ser encontradas explorando, de maneira sistemática, todas as atribuições de valores às variáveis.

Técnica de Busca No paradigma de Programação por Restrições a técnica de busca utilizada consiste em uma busca exaustiva que utiliza-se da técnica do *backtracking* e é auxiliado por uma propagação de restrições que visa diminuir consistentemente o domínio das variáveis. A propagação ocorre em dois estágios:

1. **Propagação Inicial:** Ocorre no início da execução e reflete em toda a árvore do *backtraking*;
2. **Propagação Durante a Busca:** Ocorre durante a execução e de maneira temporária, sendo revertida caso a escolha se mostre inviável. Novas escolhas podem ser realizadas a todo momento.

Durante a propagação cada restrição é analisada individualmente, sempre podando os domínios das variáveis, de maneira a não permitir quaisquer violações nas restrições. Note que cada poda no domínio de uma variável pode resultar na poda dos domínios de outras variáveis. Esse processo ocorre em cadeia e exaustivamente, até que nenhuma outra poda seja possível.

Restrições Globais A Restrição Global (Global Constraint)[20] [22] é uma das grandes forças da Programação por Restrições. Uma Restrição Global encapsula um conjunto de restrições que são propagadas junto com a principal com o objetivo de haver uma filtragem mais eficiente.

Quando pensamos em Restrições Globais, pensamos em não deixar as restrições agirem por conta própria, mas consideramos um grupo de restrições como um todo que, usualmente, agem em cima de um padrão definido. Além disso, é muito mais prático designar uma restrição global ao invés de designar cada restrição isoladamente.

Como exemplo, vejamos a restrição global $\texttt{AllDifferent}(X_1, ..., X_n)$, que afirma que o valor de todas as variáveis de X_1 a X_n devem assumir valores diferentes e usa a Teoria de Combinação (Macthing Teory) [20] com o objetivo de filtrar mais que a decomposição das restrições de diferença isoladamente. $\texttt{AllDifferent}(X, Y, Z)$ define a mesma região factível, uma vez que $\{X \neq Y, Y \neq Z, Z \neq X\}$, porém a restrição global, em comparação com as restrições decompostas, se mostra mais eficiente. Considere o seguinte exemplo:

$$X \in \{1, 2, 3\}, Y \in \{1, 2\}, \text{ e } Z \in \{1, 2\}.$$

Se propagarmos uma restrição por vez, podemos perceber que nenhum valor será removido, visto que são localmente consistentes. Porém, quando propagamos um $\texttt{AllDifferent}(X, Y, Z)$ podemos observar as 3 restrições agindo juntas e removendo os valores 1 e 2 do domínio do X, visto que X só pode assumir o valor 3.

Restrições Redundantes Uma restrição r é redundante em relação a um conjunto de restrições $\mathcal{R}$ se $\bigcap_{R \in \mathcal{R}} R \subseteq r$ (i.e. se r é satisfeita quando $\mathcal{R}$ é satisfeito). As Restrições Redundantes, são quaisquer restrições que não fazem parte do conjunto $\mathcal{R}$ de restrições essenciais da formulação do problema, ou seja, a ausência dessas restrições não impactam na corretude da solução encontrada pela formulação principal do problema [5, 11]. Uma Restrição Redundante é especialmente útil quando há a intenção de fortalecer a formulação, acrescentando no modelo informações que não estão presentes em $\mathcal{R}$ e aumentando seu poder de poda. Normalmente uma restrição redundante aplica um algoritmo especifico para manter alguma propriedade do domínio das variáveis válida.

Para mais informações sobre Programação por Restrições, consulte os trabalhos de Gaschnig [8], Haralick et. al. [10] e Sabin et. al [23].

2.2 Programação Linear e Programação Linear Inteira

A Programação Linear é um paradigma de programação que consiste na resolução de um problema de minimização ou maximização de uma função linear. Esta função deve, obrigatoriamente, estar sujeita a restrições de igualdades ou desigualdades, que sejam também lineares [1]. Em linhas gerais, um Programa Linear (PL) se encarrega de conseguir encontrar a melhor solução possível dada a função em questão.

Componentes de um PL Um PL é divido em dois componentes fundamentais:

1. uma função objetivo, de minimização ou maximização;
2. um conjunto de restrições lineares de igualdade ou desigualdade.

Um exemplo e um problema de Programação Linear é mostrado nas equações abaixo (adaptado de [16]):

$$\begin{aligned} \text{min} \quad & 3x_1 + 5x_2, && (1)\\ \text{sujeito a} \quad & x_1 \leq 4, && (2)\\ & 2x_2 \leq 12, && (3)\\ & 3x_1 + 2x_2 \leq 18, && (4)\\ & x_1, x_2 \geq 0. && (5) \end{aligned}$$

Resolução de um PL Para resolvermos um PL existem diversas abordagens. A mais conhecida e mais utilizada é através do algoritmo criado por G. B. Datzing, chamado de Simplex. O Simplex é um algoritmo que melhora o resultado da função objetivo do problema através da movimentação da solução pelos vértices do problema [16]. Utilizaremos o exemplo apresentado acima para mostrar como, através de uma representação gráfica, podemos otimizar a função objetivo.

A Figura 1 mostra, de forma gráfica, as regiões geradas pelo problema anterior. A região escura representa a região factível do problema e cada um dos vértices são os candidatos a soluções ótimas [16].

Através do gráfico podemos então substituir os valores dos vértices na função objetivo. Ao substituirmos todos os vértices, encontramos o melhor resultado com o vértice (2,6), para x_1

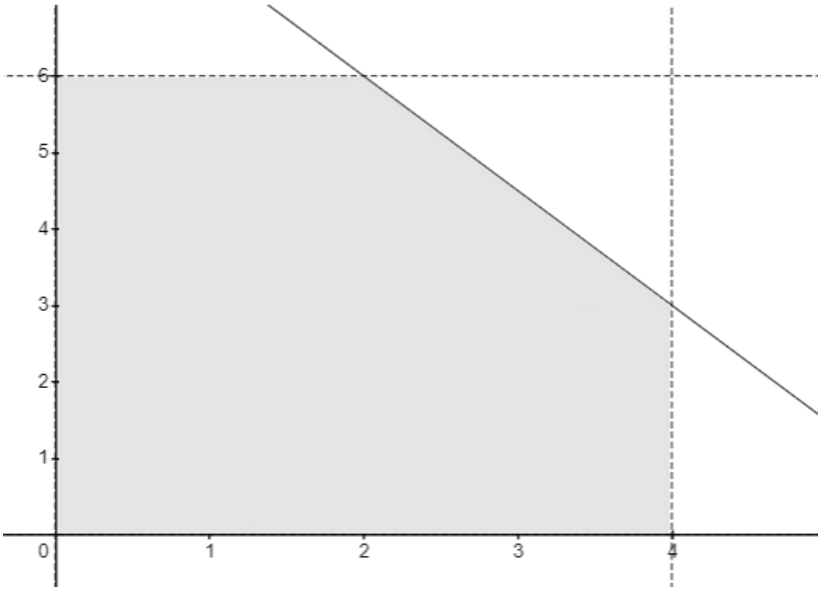

Figura 1: Gráfico da região factível do problema, com x_1 representado no eixo horizontal e x_2 representado no eixo vertical.

e x_2 respectivamente. Para esse vértice temos o valor ótimo para a função, que é de 36. O Simplex não percorre de maneira exaustiva todos os vértices do problema. Ao invés disso, ele sempre percorre vértices adjacentes visando melhorar a solução objetivo até que não seja mais possível [26].

Programação Linear Inteira Diferentemente da Programação Linear (PL), a Programação Linear Inteira (PLI) se caracteriza pelo fato de todas as sua variáveis possuem, obrigatoriamente, domínios $\in \mathbb{Z}$, ou seja, domínios inteiros.

Note que essa mudança no domínio das variáveis torna o problema um problema da classe NP-Completo. Assim sendo, não basta resolver um PLI como um PL e em seguida arredondar os resultados. Isso pode gerar uma solução infactível ou sub-ótima para o problema [26].

Quando temos um PL que possui restrições de domínio inteiro em apenas algumas variáveis, ao passo que as demais continuam com domínio contínuo, temos um Programa Linear Inteiro Misto (Mixed Integer Linear Programming - MILP).

Relaxação Linear Uma técnica que será utilizada no trabalho que vale a pena ser mencionada é a Relaxação Linear. Essa técnica consistem em remover a característica do PLI que obriga todas as variáveis a terem um domínio inteiro e as transformam em variáveis contínuas.

Essa técnica não visa encontrar uma solução para o problema. Como mencionado acima,

essa relaxação no domínio gera valores de solução infactíveis. No entanto, ao resolvermos um problema relaxado conseguimos obter um limitante válido para o problema, que pode, em seguida, ser utilizado no problema de PLI como auxiliar na obtenção do valor ótimo da função.

3 Definição do Problema

O problema do dimensionamento de lotes capacitado de item único (Capacitated Single Item Lot Sizing Problem - CSILSP) consiste em planejar a produção de um único tipo de item ao longo de um horizonte finito e discreto de tempo de maneira a atender integralmente a demanda a cada período e minimizar o custo total envolvido no processo. A cada período é possível produzir itens para período atual e/ou guardar itens para o próximo período sabendo que haverá um custo decorrente dessa escolha. Abaixo é apresentada a descrição de cada um dos parâmetros de entrada do problema e a Figura 2 apresenta uma representação do problema em forma de grafo.

- $T \in \mathbb{N}$: Número de períodos, sendo os períodos indexados em $\{0, \ldots, T-1\}$,
- $d_t \in \mathbb{N}$: Demanda no período t,
- $p_t \in \mathbb{Q} > 0$: Custo de produção de uma unidade do item no período t,
- $h_t \in \mathbb{Q} > 0$: Custo de inventário uma unidade do item do período $t-1$ para o período t,
- $s_t \in \mathbb{Q} > 0$: Custo de setup pago se pelo menos uma unidade foi produzida no período t,
- $\underline{\alpha_t} \in \mathbb{N}$: Produção mínima do item no período t,
- $\overline{\alpha_t} \in \mathbb{N}$: Produção máxima do item no período t,
- $\underline{\beta_t} \in \mathbb{N}$: Inventário mínimo do item do período $t-1$ para o período t,
- $\overline{\beta_t} \in \mathbb{N}$: Inventário máximo do item do período $t-1$ para o período t.

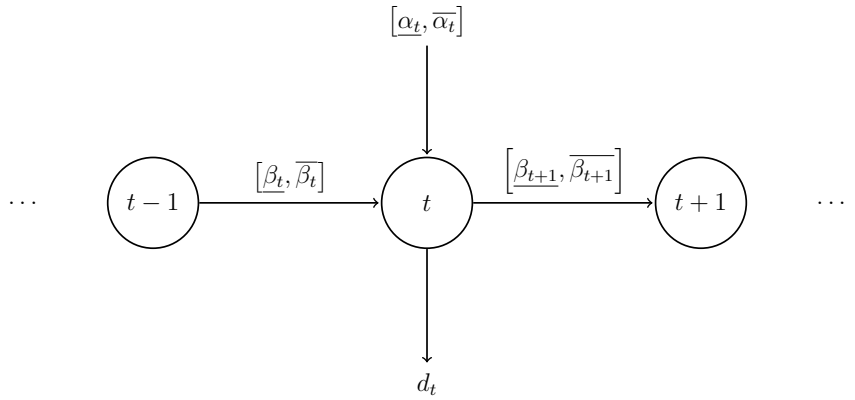

Figura 2: Representação do CSILSP em Grafo

O custo p_t é pago por unidade do item produzido no período t, e caso haja produção deve ser pago o custo s_t. Caso opte por guardar itens para o próximo período, deve ser pago o custo h_{t+1} por unidade de inventário. A produção no período t deve respeitar os limitantes α_t e o inventário no período t deve respeitar os limitantes β_t.

O objetivo do problema é decidir, a cada período t, quanto deve ser produzido. Escolher quanto será produzido a cada período implica também em escolher o quanto será guardado e levado ao período seguinte. Como produzir e guardar itens geram custos, o desafio é atender a cada período a respectiva demanda e no final dos T períodos obter o menor custo possível.

Dados os parâmetros apresentados acima, adicionaremos variáveis que auxiliarão na resolução do problema. A Figura 3 ilustra um exemplo do problema com os parâmetros e variáveis em grafo, semelhante ao encontrado em [9].

- $X_t \in \mathbb{N}$: Produção do item no período t,
- $I_t \in \mathbb{N}$: Inventário do item no período t,
- $Y_t \in \{0, 1\}$: Variável de setup no período t, sendo 1 caso tenha produzido e 0 caso não tenha produzido,
- $Ch \in \mathbb{N}$: Custo total de inventário, somados os T períodos,
- $Cp \in \mathbb{N}$: Custo total de produção, somados os T períodos,
- $Cs \in \mathbb{N}$: Custo total de setup, somados os T períodos,
- $C \in \mathbb{N}$: Custo total, somados todos os custos dos T períodos.

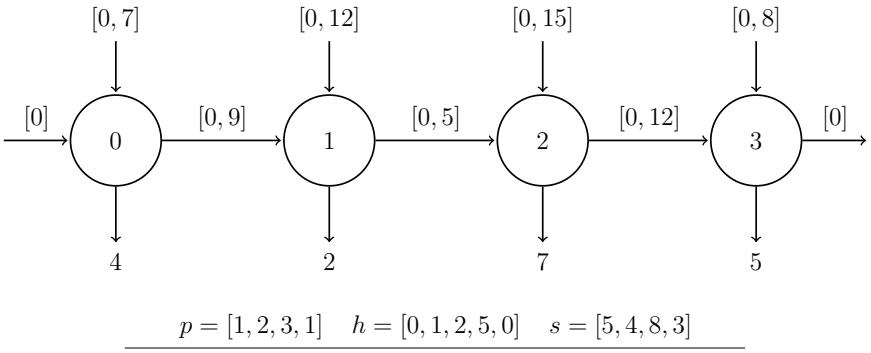

Figura 3: Representação do CSILSP com Parâmetros e Variáveis.

Podemos representar os valores obtidos pelas variáveis X, Y, I ao final do processo como vetores $< X_0, \ldots, X_{T-1} >$,$< Y_0, \ldots, Y_{T-1} >$ e $< I_0, \ldots, I_{T-1} >$, que representam as respectivas quantidades a cada período. Na Figura 4, temos X como $< 6, 0, 7, 5 >$, Y como $< 1, 0, 1, 1 >$ e I como $< 0, 2, 0, 0, 0 >$. Os custos $Ch = 2$, $Cp = 32$, $Cs = 16$, $C = 50$ são respectivamente a soma do custo de inventário, a soma do custo de produção, a soma do custo de setup e a soma total de todos os custos.

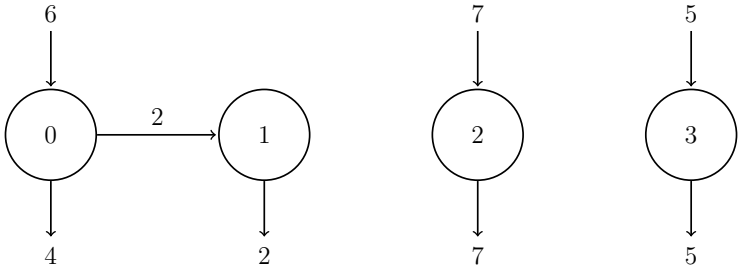

Figura 4: Exemplo de Solução Ótima para a Instância da Figura 2 do CSILSP.

Note que caso haja a necessidade de se definir o problema sem limitantes inferiores de produção e estoque (i.e. $\underline{\alpha_t}$ e $\underline{\beta_t}$) basta definir os limitantes como 0. Nenhuma outra mudança é necessária para adaptação do problema, mais que isso, essas versões são equivalentes [9].

3.1 Algoritmo em Programação por Restrições e Programação Linear Inteira

Usaremos as variáveis de decisão X_t que indicarão a quantidade de unidades do produto que serão produzidas no período t. Além disso variáveis auxiliares I_t e Y_t que indicarão a quantidade de unidade do produto em inventário no período t e se o custo de setup será pago no período t, respectivamente. Definimos $[K]$ como o conjunto de índices $\{0, 1, \ldots K-1\}$, dessa forma $[T]$ é o conjunto de índices de todos os períodos. Teremos então a seguinte formulação:

$$\text{min} \quad C = Ch + Cp + Cs \tag{6}$$

$$\text{sujeito a} \quad X_t + I_t = d_t + I_{t+1} \qquad t \in [T-1], \tag{7}$$

$$X_t + I_t = d_t \qquad t = T-1, \tag{8}$$

$$X_t \leq \overline{\alpha_t}\, Y_t \qquad t \in [T], \tag{9}$$

$$Cp = \sum_{t \in [T]} p_t\, X_t \tag{10}$$

$$Cs = \sum_{t \in [T]} s_t\, Y_t \tag{11}$$

$$Ch = \sum_{t \in [T]} h_t\, I_t \tag{12}$$

$$X_t \in \{\underline{\alpha_t}, ..., \overline{\alpha_t}\} \qquad t \in [T], \tag{13}$$

$$I_t \in \{\underline{\beta_t}, ..., \overline{\beta_t}\} \qquad t \in [T], \tag{14}$$

$$Y_t \in \{0, 1\} \qquad t \in [T]. \tag{15}$$

A função objetivo (6) visa minimizar o custo total, composto pelo custo de produzir os produtos, o custo de estoque e o custo de setup. A igualdade (8) garante a correta correlação entre as variáveis I_t e garante que a demanda de cada produto seja atendida. Já a desigualdade (9) garante que $Y_t = 1$ se o item for produzido no período t. As restrições (10), (11) e (12) asseguram que as variáveis Cp, Cs, Ch sejam igual o somatório do custos de produção, setup e inventário. As restrições (13), (14), (15) garantem o domínio das variáveis.

Note que a formulação apresentada para o paradigma da Programação por Restrições também é uma formulação em PLI, já que todas as restrições são expressões lineares.

3.2 Algoritmo em Programação Dinâmica

A Programação Dinâmica também se destaca no resolução do problema e sua formulação pode ser encontrada em [7]. Apresentamos aqui o algoritmo sem limitantes inferiores em produção e inventário, que itera sobre o número de períodos e níveis de inventário. Denotamos por $f(t, I_{t+1})$ o custo ótimo de produção das demandas de d_0 a d_t, sabendo que o estoque final

período t é I_{t+1}:

$$\forall t \in [0, T-1] \text{ e } \forall I_t \in [0, I_{max}], \tag{16}$$

$$f(t, I_{t+1}) = \min_{I_t = a \ldots b} \left\{ f(t-1, I_t) + \left\lceil \frac{X_t}{\overline{\alpha_t}} \right\rceil s_t + p_t X_t + h_{t+1} I_{t+1} \right\}, \tag{17}$$

onde $a = \max\{0, d_t + I_{t+1} - \overline{\alpha_t}\}$, $b = \min\{\overline{\beta_t}, d_t + I_{t+1}\}$ e $X_t = I_{t+1} + d_t - I_t$. Também temos que $I_{max} = \max\{\overline{\beta_t}, t \in [0, T-1]\}$ e os valores iniciais como $f(-1, 0) = 0$ e $f(-1, I_0) = +\infty$, $\forall I_0 \in [1, I_{max}]$. O valor $f(T-1, 0)$ corresponde ao valor ótimo do CSILSP.

4 Restrições Redundantes

Nesta seção apresentaremos três restrições redundantes que têm por objetivo aumentar o poder de poda do algoritmo principal de Programação por Restrições. São elas as restrições `LowerBound`, `RelaxedFlowFilter` e `WindowMaxFlow`.

4.1 Lower Bound

A restrição `LowerBound` foi criada com o intuito de, durante a execução do algoritmo principal, realizer podas no valor do custo global da solução, i.e., no valor de C. O valor de C, como já dito anteriormente, é o somatório de $Cs + Ch + Cp$ que são, custos de setup, custos de inventário e custos de produção, respectivamente.

Como C é composto de $Cs + Ch + Cp$, uma poda no seu domínio afeta esses 3 valores, podendo gerar podas também nos domínios de $Cs + Ch + Cp$. Por sua vez, uma poda em Ch, por exemplo, afeta os valores de $h_t,\ \forall\ t \in [T]$, podendo gerar podas também os domínios de h_t. Dessa forma vemos como uma poda em C pode ter grande impacto em todas as variáveis do problema.

Antes de falarmos da restrição em si, mostraremos um exemplo que mostra como uma poda no valor de C pode gerar podas em todas as demais variáveis do problema. Considere o exemplo apresentado na Figura 5. Suponhamos que em determinado *branch*, durante a execução do algoritmo principal, temos como *lower bound* de C o valor de 25. Tendo como referência este valor, podemos perceber algumas oportunidades de podas nos domínios das variáveis do problema. Como exemplo, vamos citar:

1. No período 1, há a possibilidade de produzir até 9 itens. Com a produção desses 9 itens, há o suprimento das demandas dos períodos 1, 2 e 3, desde que os itens sejam devidamente guardados pagando um custo de inventário. Porém, quando olhamos para o custo de produzir esses itens, temos que:

$$9 * p_1 + s_1 = 29,$$

   este custo já excede o melhor *lower bound* encontrado que é 25. Logo, podemos realizar a poda do valor 9 de X_2.

2. No período 0, há a possibilidade de produzir até 15. Se somarmos a demanda total, temos

12 itens. Logo, o domínio de X_0 pode ser podado para $X[0, 12]$. Supondo que vamos produzir os 12 itens possíveis e carregá-los para os demais períodos, temos:

$$12 * p_0 + s_0 + 9 * h_1 + 6 * h_2 + 3 * h_3 = 31,$$

este custo também excede o melhor *lower bound* encontrado que é 25. Logo, podemos realizar a poda de X_0.

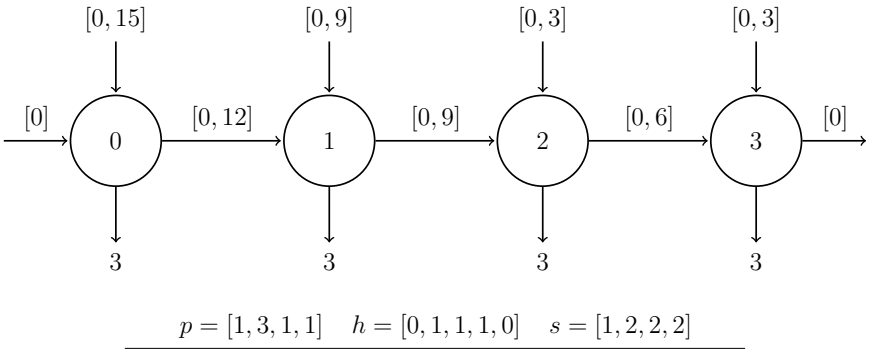

Figura 5: Exemplo Lower Bound

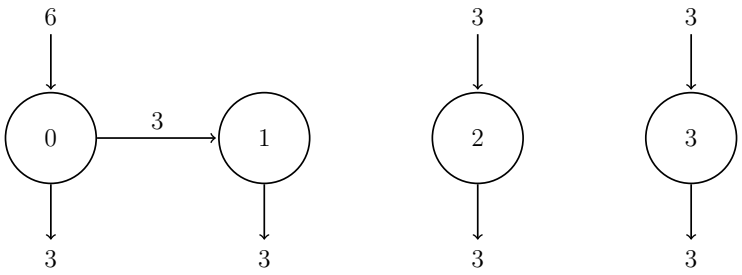

Figura 6: Exemplo Lower Bound Resolvido

Esses cenários exemplificam o possível ganho que a restrição pode trazer em poder de poda. Há muitos outros exemplos de poda, alguns não tão triviais de percebermos. A junção dessas podas podem prover uma otimização muito significativa em tempo de processamento.

Exemplificado o valor dessas podas, vamos descrever a restrição `LowerBound`. Para descrevê-la usaremos as notações:

1. $\underline{X_t}$: Menor valor do domínio de X no período t no ramo atual
2. $\overline{X_t}$: Maior valor do domínio de X no período t no ramo atual
3. $\underline{I_t}$: Menor valor do domínio de I no período t no ramo atual

4. $\overline{I_t}$: Maior valor do domínio de I no período t no ramo atual

5. $\underline{Y_t}$: Menor valor do domínio de Y no período t no ramo atual

6. $\overline{Y_t}$: Maior valor do domínio de Y no período t no ramo atual

$$lowerbound = \sum_{t=0}^{T-1} \underline{X_t} * p_t + \underline{I_t} * h_t + \underline{Y_t} * s_t \tag{18}$$

$$lowerbound += \sum_{t=0}^{T-1} min_{k=0...t}\{p_k * (\sum_{j=0}^{t} d_j - \sum_{j=0}^{t} \underline{X_j}) + setup + stock\} \tag{19}$$

$$setup = 0 \text{ se } k \neq t \text{ e } s_t \text{ se } k = t \tag{20}$$

$$stock = 0 \text{ se } k = t \text{ e } \sum_{j=k}^{t} h_j * I_j \text{ se } k \neq t \tag{21}$$

$$\text{se } lowerbound < \underline{C},\ \underline{C} = lowerbound \tag{22}$$

Durante a execução do algoritmo principal, as restrições redundantes serão chamadas a cada nova ramificação. A expressão (18) representa um limitante inferior válido para o valor de $\underline{C}$ no ramo atual, uma vez que somando o mínimo a ser produzido em cada período, o mínimo a ser guardado em cada período e os respectivos custos de setup, temos o menor valor possível pra C.

Tendo essa informação, temos que fortalecer $\underline{C}$. Para fazer isso, iremos considerar a expressão (18). Vamos entender esse algoritmo dividindo-o em partes.

O valor de $\sum_{j=0}^{t} d_j - \sum_{j=0}^{t} \underline{X_j}$ representa o que ainda tem que ser produzido no problema até o período t. Somando as demandas, menos o mínimo que se deve produzir, temos o restante que deverá ser produzido, gerando aí, um problema de decisão.

Sabendo que temos uma quantidade de itens a ser produzida, devemos considerar quais as possíveis maneiras de produzir esses itens, que são:

1. Produzir todos os itens no período atual;

2. Produzir os itens em algum período anterior e pagar os custos de inventário para trazer essa produção até período atual.

Quando produzimos os itens no período atual, sabemos que há o custo de setup a ser pago. Quando consideramos que o item pode ser produzido em um período anterior ao atual, se consi-

derarmos custos de setup, podemos estar desconsiderando um custo de setup já pago e podemos invalidar o *lower bound* ao realizar esse cálculo. Por isso, a expressão (19) diz respeito à soma do custo de setup do período em questão, que incorre somente quando a produção é realizada no período atual.

Já, quando há a produção em períodos anteriores devemos considerar o custo de inventário para levar o item do período de produção até o período atual. A expressão (20) mostra que, quando a produção é feita em períodos anteriores, devemos pagar os devidos custos de inventário e não pagamos nada do contrário.

Iterando a expressão (18) por todos os períodos, teremos a cada período um custo mínimo válido a ser pago. Logo, somamos todos esses valores e comparamos com o valor de $\underline{C}$ atual. Caso esse valor seja maior, atualizamos o valor de $\underline{C}$.

4.2 Relaxed Flow Filter

A restrição redundante `RelaxedFlowFilter` se encarrega de resolver uma relaxação linear do problema original. Relaxar o problema permite atingir uma complexidade muito menor, possibilitando fornecer podas consistentes no domínio das variáveis do problema. Essa restrição é executada sobre os valores do ramo atual e tem como pré-requisito que todos os valores de Y estejam fixados. Sempre que chegamos em um ramo com essa condição, uma chamada é feita à restrição.

A poda é executada sobre a variável C, que indica o custo total do problema. Podando o domínio de C, interferimos, indiretamente, nos valores de Cp, Cs e Ch, que, por sua vez, interfere no domínio de X_t e I_t. Essas relações existentes entre os parâmetros e variáveis asseguram as devidas podas.

A relaxação, quando comparada com o algoritmo principal, possui as seguintes diferenças:

1. $X_t \in \mathbb{Q}$ tal que $X_t > 0$
2. $I_t \in \mathbb{Q}$ tal que $I_t > 0$
3. $\underline{\beta_t}$ e $\overline{\beta_t}$ contém os valores relativos ao domínio no ramo atual, ou seja $\underline{I_t}$ e $\overline{I_t}$
4. $\underline{\alpha_t}$ e $\overline{\alpha_t}$ contém os valores relativos ao domínio no ramo atual, ou seja $\underline{X_t}$ e $\overline{X_t}$
5. A variável Y passa a ser um parâmetro, uma vez que seus valores já estão fixados

6. O domínio da variável X_t será sempre $[0, 0]$ se $Y_t = 0$.

Resolver a relaxação linear é equivalente a resolver um problema de fluxo de rede de custo mínimo [19]. O gráfico deste fluxo é apresentado na Figura 7. Em cada arco, (u, c, m) representa a capacidade (u), o custo unitário (c) e o fluxo mínimo (m) do arco. As unidades fluem do nó de source S para o nó sink W.

Em cada arco de produção (S, t), a capacidade de produção do período é $\overline{X_t}$, o custo é p_t e o fluxo mínimo é $\underline{X_t}$. Com relação aos arcos de estoque $(t-1, t)$, a capacidade é $\overline{I_t}$, o custo é h_t e o fluxo mínimo é $\underline{I_t}$. Em cada arco de demanda (t, W), deve haver o número da demanda do período t, d_t e o custo é 0. O problema de fluxo pode ser resolvido em $O(T^2)$ com o algoritmo successive shortest path [19] ou através de uma formulação em PL.

Basta então resolver essa relaxação linear com o objetivo de prover um limitante inferior válido para a variável C. Em seguida esse utilizamos esse valor na formulação principal, aumentando seu poder de poda.

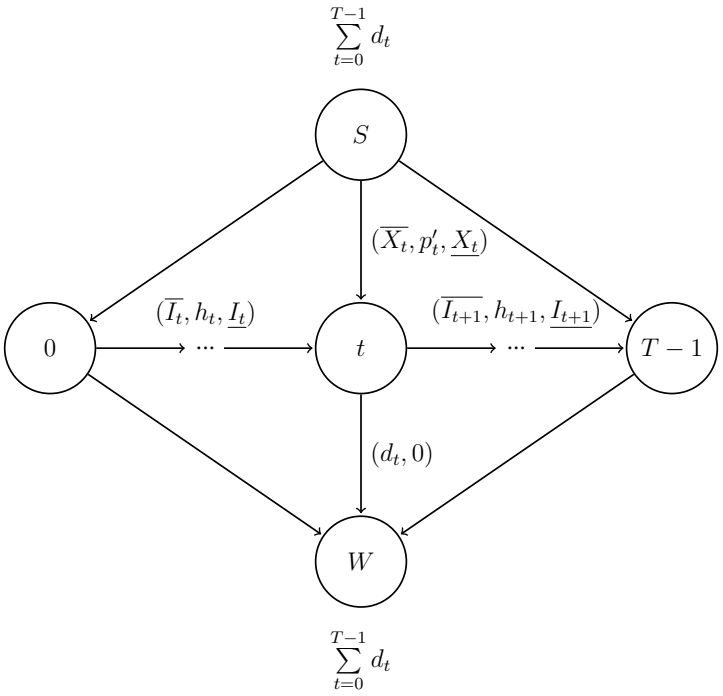

Figura 7: Relaxação Linear

4.3 Window Max Flow

Como mencionado na Seção 3, a variante do problema a ser resolvido envolve capacidades de produção e estoque, i.e., sempre há uma quantidade mínima e máxima permitida de produção e de estoque a cada período. Essas capacidades são gerais, ou seja, não precisam seguir um padrão de crescimento ou decrescimento, constância ou proporção. Sabendo disso, a restrição `WindowMaxFlow` foi desenvolvida com o intuito de realizar possíveis podas no domínio de X e I. Mais especificamente estamos interessados nos valores $\underline{X}, \overline{X}, \underline{I}$ e $\overline{I}$.

Na restrição `WindowMaxFlow`, a cada mudança no domínio de uma das 3 variáveis de decisão X, Y ou I, o domínio das variáveis X e I é analisado e tentamos identificar valores inviáveis.

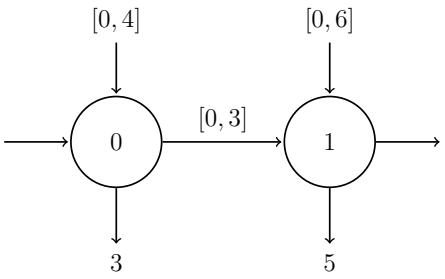

Figura 8: Exemplo de Instância da CSILSP

Para melhor compreensão, vejamos o exemplo proposto na Figura 8 que possui apenas 2 períodos. Podemos notar que, no período 1, temos uma capacidade de produção de 0 a 6 itens e uma demanda de 5 itens. Para o período 1, temos os seguinte possíveis cenários:

1. Receber 1 item do período 0 através do inventário e produzir mais 4 itens no período 1;
2. Não receber nenhum item do período 0 e produzir 5 itens no período 1.

Observando os possíveis cenários, podemos perceber que alguns dos valores presentes nos domínios das variáveis nunca poderão ser utilizados. No período 1, por exemplo, temos um cenário onde produzimos 4 e outro onde produzimos 5. Logo, os valores $\{0, 1, 2, 3\}$ podem ser removidos do domínio de X_1. Note que, no último período o somatório $I_t + X_t - d_t = 0$, visto que não é permitido que sobrem itens ao final do horizonte de tempo definido. Assim, o valor $\{6\}$ também pode ser removido do domínio de X_1.

Além dessa poda, podemos perceber também que na capacidade de inventário, isto é, I_1, só há a possibilidade passarmos 0 ou 1 item. Logo, seu domínio também pode ser podado. Temos como novo domínio de I_1 os valores $\{0, 1\}$.

Dado esse exemplo, podemos perceber que há possibilidade de poda nos domínios das variáveis X e I, e a análise e poda desses domínios pode ser uma contribuição interessante, não somente na programação por restrições como também em outros paradigmas.

A `WindowMaxFlow` possui duas partes: uma para podar os *upper bounds* e outra para podar os *lower bounds* dos domínios das variáveis.

4.3.1 Criação de Janelas

Antes de prosseguirmos com a apresentação da restrição, será apresentado o conceito de janela. Em linhas gerais uma janela é um subproblema do CSILSP, cujos períodos são uma subsequência contígua dos períodos do problema principal. Para exemplificarmos o conceito da janela, utilizaremos o exemplo apresentado na Figura 9.

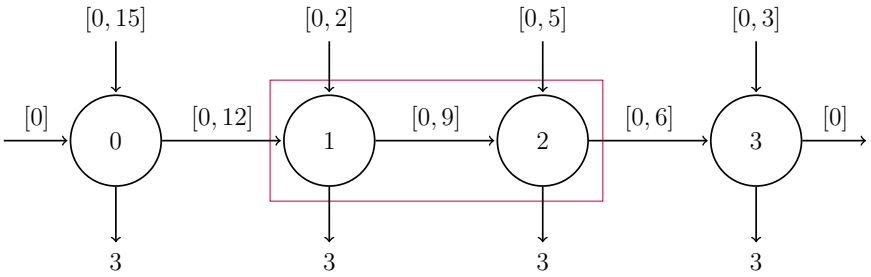

Figura 9: Exemplo da Criação de Janelas

Podemos definir uma janela que engloba os períodos 1 e 2. Quando olhamos para essa subproblema, temos dois pontos que diferem do problema original:

1. O inventário entrando na janela (no exemplo, o inventário que é passado do período 0 para o período 1) pode ser maior que 0. No exemplo $I_1 \in [0, 12]$.

2. O inventário que sai da janela (no exemplo, o inventário que é passado do período 2 para o período 3) pode ser maior que 0. No exemplo, $I_3 \in [0, 6]$.

Como uma janela é definida pelos seus períodos inicial e final, dado um problema com n períodos podemos definir $\binom{n}{2} = \frac{n(n-1)}{2}$ janelas distintas. A relação entre janelas e podas vem da observação de que tudo que entra em uma janela deve igualar tudo que sai. Assim, dada uma janela com período inicial t_i e final t_f,

$$I_{t_i} + \sum_{j=t_i}^{t_f} X_j = \sum_{j=t_i}^{t_f} d_j + I_{t_{f+1}}. \tag{23}$$

4.3.2 Poda de I_t pelas janelas a direita

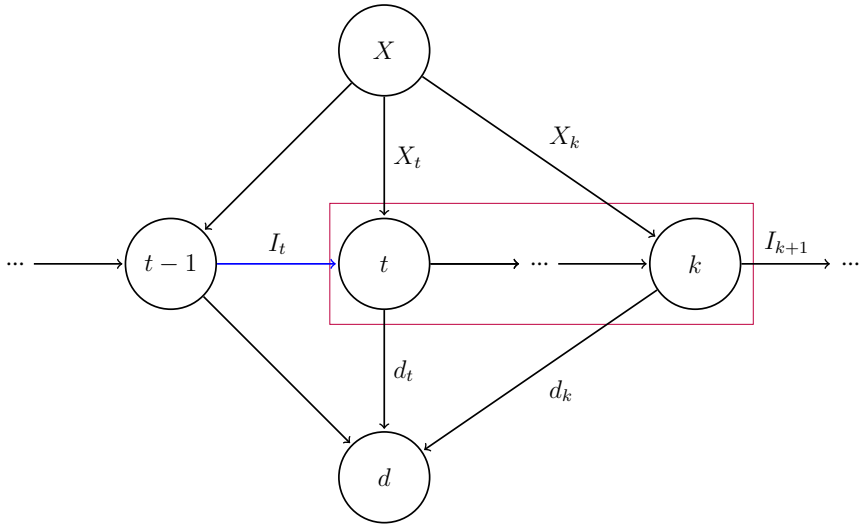

Figura 10: Criação das janelas da expressão (27) e (29).

Nesta Subseção, veremos como é feita a poda dos limitantes das variáveis I_t. Para tanto, utilizaremos o conceito de janelas e a equação (23). Fazendo $t_i = t$ em (23) e isolando I_t temos

$$I_t = \sum_{j=t}^{t_f} (d_j - X_j) + I_{t_{f+1}}, \tag{24}$$

vamos primeiramente procurar um limitante inferior de I_t. Note que o menor valor que o lado direito da equação pode assumir, é quando X_j tem os maiores valores possíveis e $I_{t_{f+1}}$ tem o menor valor possível. Dessa forma obtemos a seguinte expressão

$$\underline{I_t} \geq \sum_{j=t}^{t_f} \left(d_j - \overline{X_j}\right) + \underline{I_{t_{f+1}}}. \tag{25}$$

Essa desigualdade deve valer para toda janela com $t_f \leq t$, substituindo t_f por k, conforme representado pela Figura 10, chegamos à expressão

$$\underline{I_t} \geq \max_{k=t,\ldots,T-1} \{\underline{I_{k+1}} + \sum_{j=t}^{k} (d_j - \overline{X_j})\}, \tag{26}$$

dessa forma podemos atualizar $\underline{I_t}$ caso seja encontrado um limitante maior que o atual, dessa

forma concluimos que

$$\underline{I_t} = \max\{\underline{I_t}, \max_{k=t,\dots,T-1}\{\underline{I_{k+1}} + \sum_{j=t}^{k}(d_j - \overline{X_j})\}\} \tag{27}$$

Podemos partir da também da expressão (24), para encontrar um limitante superior para I_t. Note que o maior valor que o lado direito da expressão pode assumir é quando X_j assume o menor valor e $I_{t_{f+1}}$ assume o maior valor possível.

$$\overline{I_t} \leq \sum_{j=t}^{t_f}\left(d_j - \underline{X_j}\right) + \overline{I_{t_{f+1}}}. \tag{28}$$

Seguindo o mesmo raciocínio que usamos para obter o limitante (27), obtemos a seguinte expressão

$$\overline{I_t} = \min\{\overline{I_t}, \min_{k=t,\dots,T-1}\{\overline{I_{k+1}} + \sum_{j=k}^{t}(d_j - \underline{X_j})\}\}. \tag{29}$$

4.3.3 Poda de I_t pelas janelas a esquerda

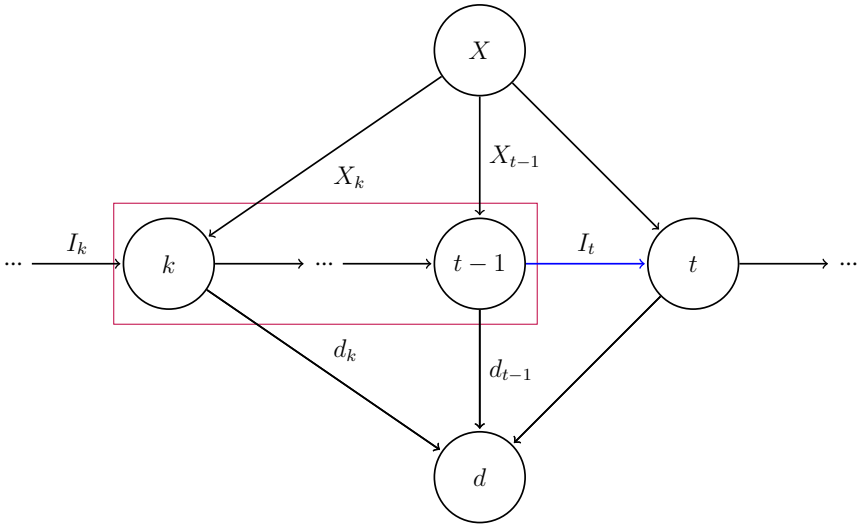

Figura 11: Criação das janelas da expressão (31) e (32).

Para obter os próximos limitantes também partimos de (23), mas fazemos $t_{f+1} = t$, e isolamos I_t no lado esquerdo, obtendo

$$I_t = I_{t_i} + \sum_{j=t_i}^{t-1}(X_j - d_j). \tag{30}$$

Utilizando essa expressão podemos encontrar limitantes inferiores e superiores para as

variáveis I_t. Para o limitante inferior trocamos I_t por $\underline{I_t}$ e, no lado direito, substituindo todo termo positivo por seu *lower bound*. Por fim trocamos t_i por k e variando k de 0 até $t-1$, conforme ilustrado na Figura 11, e chegamos a expressão

$$\underline{I_t} = max\{\underline{I_t}, \max_{k=0,\ldots,t-1}\{\underline{I_k} + \sum_{j=k}^{t-1}(\underline{X_j} - d_j)\}\}. \tag{31}$$

Para o limitante superior podemos usar o mesmo raciocínio, e obtemos a expressão

$$\overline{I_t} = \min\{\overline{I_t}, \min_{k=0,\ldots,t-1}\{\overline{I_k} + \sum_{j=k}^{t-1}(\overline{X_j} - d_j)\}\}. \tag{32}$$

4.3.4 Poda de X_t pelas janelas a direta

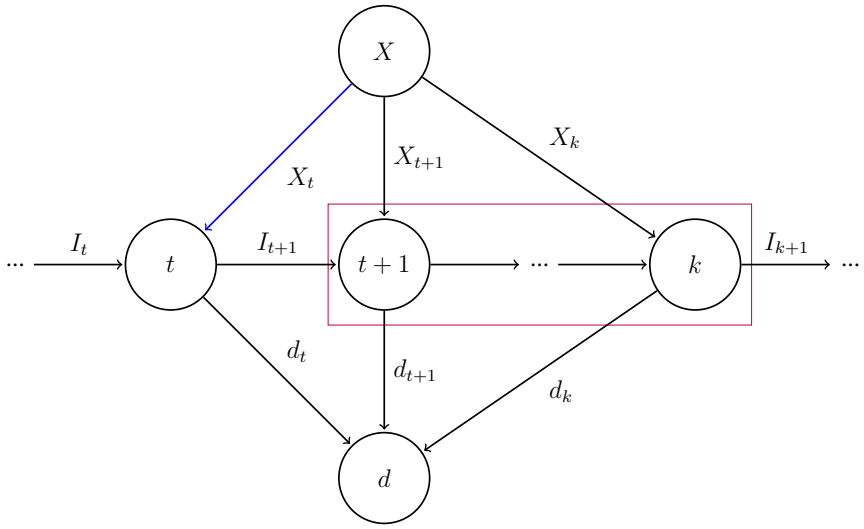

Figura 12: Criação das janelas da expressão (34) e (35).

Nos próximos limitantes partimos de (23), mas focamos no que é produzido no período t_i. Assim, substituimos t_i por t, isolamos X_t,

$$X_t = d_t + \sum_{j=t+1}^{t_f}(d_j - X_j) + I_{t_{f+1}} - I_t. \tag{33}$$

Agora podemos encontrar um limitante inferior para X_t, observando que o menor valor que o lado direito da expressão pode assumir, é quando os termos positivos assumem o menor valor possível e os termos negativos assumem o maior valor possível. Além disso a expressão deve ser verdadeira para qualquer $t_f > t$, conforme ilustrado na Figura 12, dessa forma, trocamos t_f pot k e, variando k de t até $T-1$, obtemos a expressão

$$\underline{X_t} = \max\{\underline{X_t}, \max_{k=t,\ldots,T-1}\{d_t + \sum_{j=t+1}^{k}(d_j - \overline{X_j}) + \underline{I_{k+1}} - \overline{I_t}\}\} \tag{34}$$

Por último podemos encontrar um limitante superior obtido analogamente ao anterior,

$$\overline{X_t} = \min\{\overline{X_t}, \min_{k=t,\ldots,T-1}\{d_t + \sum_{j=t+1}^{k}(d_j - \underline{X_j}) + \overline{I_{k+1}} - \underline{I_t}\}\}. \tag{35}$$

4.3.5 Compondo a restrição

Finalmente a restrição `WindowMaxFlow` calcula e mantêm válidos os limitantes (27), (31), (34), (29), (32) e (35). Como o ajuste de um dos limitantes pode afetar outro, eles são calculados em um laço até que nenhuma alteração seja feita no domínio das variáveis.

5 Resultados

Os algoritmos foram implementados em C++ e executados em uma máquina com processador AMD Ryzen 5 3400G, 16GB de RAM e sistema operacional Ubuntu 18.04 LTS. Foi utilizado também o solver CPLEX v12.10 de PL e PLI e o solver CP Optimizer V12.10 de CP, disponibilizado pela IBM, que atuou como framework nas modelagens dos problemas. Os testes utilizaram apenas 1 thread e tempo máximo de execução de 3600 segundos.

5.1 Instâncias

Os testes foram desenvolvidos utilizando um conjunto de instâncias próprias, visto que para o nosso conhecimento não existem instâncias presentes na literatura. O conjunto é composto de 63 instâncias, e os parâmetros estão apresentados na Tabela 1.

Sigla	**Descrição**	**Valores**
T	Número de períodos	3, 6, 9, 12, 15, 18, 21
davg	Demanda média	0.5
alpha-min	Valores mínimos de capacidade de inventário e capacidade de produção	5000
alpha-max	Valores máximos de capacidade de inventário e capacidade de produção	500000
ratio-h	Porcentagem da capacidade de produção que pode ser levada ao próximo período	0.1, 0.3, 0.6
ratio-d	Porcentagem da capacidade de produção que será transformada em demanda	0.5
custos-s-p	Balanco entre custos de produção e custos de setup (somados são sempre 100%)	[80,20] - [20,80] - [50,50]

Tabela 1: Parâmetros de geração das instâncias.

Para cada uma das instâncias serão executados testes utilizando os algoritmos apresentados na Tabela 2.

5.2 Análise dos Resultados

Nesta seção serão apresentados os testes computacionais, bem como a análise dos resultados obtidos. As métricas utilizadas para a avaliação dos resultados são:

1. Tempo de Execução: O tempo que o algoritmo levou para encontrar a solução ótima do problema;

Algoritmos	
Sigla	**Descrição**
PLI	Formulação em Programação Linear Inteira
DP	Formulação em Programação Dinâmica
CP	Formulação em Programação por Restrições
CP + RFF	Formulação em Programação por Restrições com auxílio da restrição `RelaxedFlowFilter`
CP + RFF + LB	Formulação em Programação por Restrições com auxílio das restrições `Lowerbound` e `RelaxedFlowFilter`
CP + RFF + WMF	Formulação em Programação por Restrições com auxílio das restrições `RelaxedFlowFilter` e `WindowMaxFlow`

Tabela 2: Algoritmos utilizados nos testes.

2. Branches: Os algoritmos que utilizam a programação por restrições nos indicam quantas ramificações foram criadas para a resolução do problema;

3. Fails: Os algoritmos que utilizam a programação por restrições também nos indicam quantas das ramificações criadas sofreram podas;

4. Gap: Porcentagem que indica a distância entre melhor solução encontrada até o momento e o melhor limitante dual.

A Tabela 3 apresenta os resultados obtidos pelo algoritmo de PLI para a resolução do problema. O algoritmo de PLI se mostrou o mais eficiente para este conjunto de instâncias, sendo capaz de resolver todas as instâncias à otimalidade com um tempo de execução curto se comparado com os demais algoritmos.

A Tabela 4 apresenta os resultados obtidos pelo algoritmo de CP para a resolução do problema. O algoritmo de CP sem a ajuda das restrições redundantes apresenta resultados ruins para este conjunto de instâncias, não sendo capaz de resolver todas as instâncias à otimalidade. As instâncias que o algoritmo não foi capaz de resolver, após o tempo limite de execução, ainda apresentam valores de Gap altos.

A Tabela 5 apresenta os resultados obtidos pelo algoritmo de CP utilizando como restrição redundante a restrição `RelaxedFlowFilter`. Este algoritmo não apresenta os melhores resultados, se comparado com o algoritmo de PLI, porém ao compararmos com formulação de CP sem as restrições, temos grandes melhorias na maioria das instâncias. O primeiro ponto a se mencionar é o fato de, agora, a formulação passar a resolver todas as instâncias do conjunto proposto, além de reduzir drasticamente o tempo de execução em maioria das instâncias. Vejamos, por exemplo, a instância de número 0031, que passou de 3196.23 segundos para apenas 9.3

segundos em tempo de execução, e a maioria das instâncias após a instância 0050 que passaram a resolver o problema.

Em seguida, utilizamos a restrição `WindowMaxFlow` somada à `RelaxedFlowFilter` para obtermos mais melhorias, como apresentado na Tabela 6. A melhoria não foi tão expressiva quanto a observada anteriormente, porém conseguimos aprimorar ainda mais a formulação de CP. Tomemos como exemplo a instância de número 0061, que passou de 762.14 para 692.11 segundos em tempo de execução. Mais uma vez, todas as instâncias foram resolvidas e em grande maioria foram obtidas melhoras.

Por fim, como apresentado na Tabela 7 utilizamos a restrição `RelaxedFlowFilter` somada à `LowerBound` para obtermos os melhores resultados a ser apresentados no presente trabalho. Obtivemos, mais uma vez, melhorias consistentes, quando comparadas com o melhor encontrado até o momento. Tomemos como exemplo as instâncias 0062, 0059, 0058, onde obtivemos um tempo de execução de 10 a 20 vezes menor. Se compararmos com a formulação básica de CP, notamos uma diferença consistente em todas as instâncias.

A Tabela 8 apresenta os resultados obtidos pelo algoritmo de DP para a resolução do problema. Os algoritmos de PD são, quando falamos do Problema do Dimensionamento de Lotes, particularmente ineficientes em instâncias com altos valores de capacidade de inventário. Como as instâncias propostas possuem essa característica, podemos ver que o algoritmo só conseguiu resolver a primeiras instâncias, e ainda assim, de maneira ineficiente.

A Tabela 9 apresenta a comparação dos resultados obtidos entre os diferentes algoritmos de CP. Esta comparação está sendo realizada em tempo de execução e gap, que são as métricas mais relevantes na pesquisa. Vemos a evolução da eficiência do algoritmo devido às restrições redundantes acrescentadas. Já a Tabela 10 compara o melhor resultado obtido com um algoritmo de CP (i.e. CP + RFF + LB) com os demais paradigmas.

Por fim, a Figura 13 apresenta um gráfico comparativo entre os resultados obtidos através dos métodos apresentados. O eixo vertical apresenta o número de instâncias resolvidas até o ótimo pelo método, ao passo que o eixo horizontal apresenta o tempo de execução do algoritmo. Note que, os algoritmos mais eficientes apresentam linhas mais à esquerda e mais ao topo, o que significa que em um tempo menor, mais instâncias foram resolvidas. Podemos observar que houve uma melhora significativa entre o CP puro e o CP com a adição das restrições desenvolvidas, obtendo tempos competitivos com o PL.

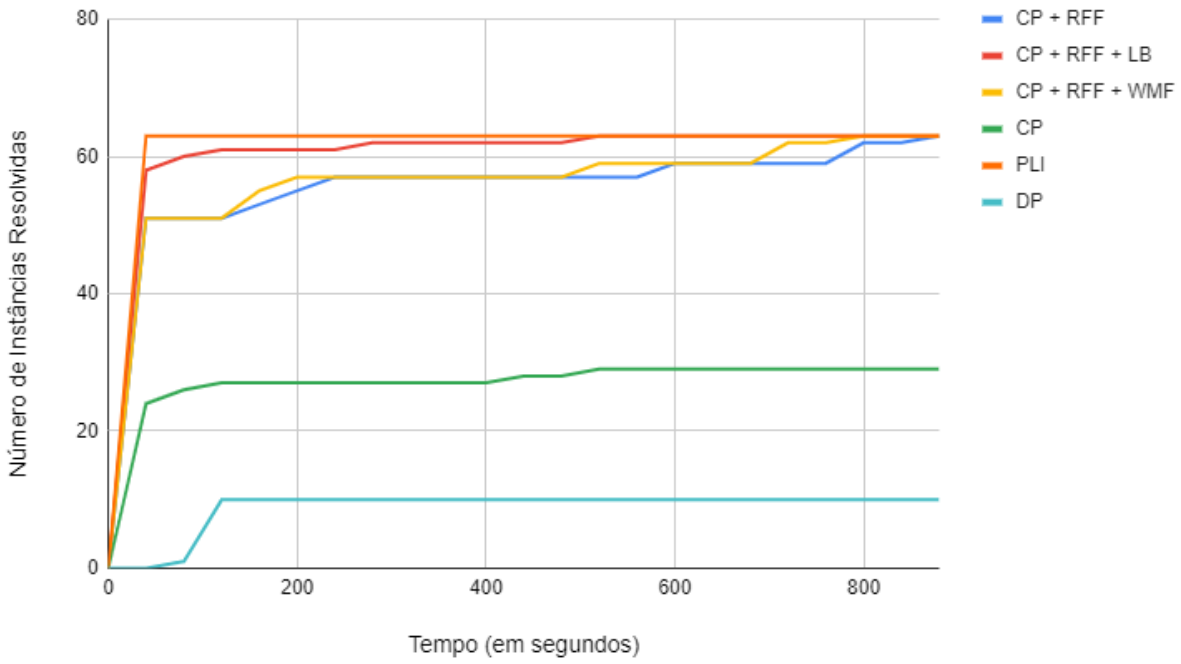

Figura 13: Comparação entre os métodos apresentados.

Instância	T	ratio-h	ratio-d	custos-s-p	Tempo (s)	Gap (%)
0000	3	0.1	0.5	[80, 20]	0.00	0
0001	3	0.3	0.5	[80, 20]	0.00	0
0002	3	0.6	0.5	[80, 20]	0.00	0
0003	3	0.1	0.5	[50, 50]	0.00	0
0004	3	0.3	0.5	[50, 50]	0.00	0
0005	3	0.6	0.5	[50, 50]	0.00	0
0006	3	0.1	0.5	[20, 80]	0.00	0
0007	3	0.3	0.5	[20, 80]	0.00	0
0008	3	0.6	0.5	[20, 80]	0.00	0
0009	6	0.1	0.5	[80, 20]	0.18	0
0010	6	0.3	0.5	[80, 20]	0.19	0
0011	6	0.6	0.5	[80, 20]	0.03	0
0012	6	0.1	0.5	[50, 50]	0.06	0
0013	6	0.3	0.5	[50, 50]	0.04	0
0014	6	0.6	0.5	[50, 50]	0.03	0
0015	6	0.1	0.5	[20, 80]	0.04	0
0016	6	0.3	0.5	[20, 80]	0.06	0
0017	6	0.6	0.5	[20, 80]	0.02	0
0018	6	0.1	0.5	[80, 20]	0.00	0
0019	9	0.3	0.5	[80, 20]	0.00	0
0020	9	0.6	0.5	[80, 20]	0.00	0
0021	9	0.1	0.5	[50, 50]	0.00	0
0022	9	0.3	0.5	[50, 50]	0.00	0
0023	9	0.6	0.5	[50, 50]	0.00	0
0024	9	0.1	0.5	[20, 80]	0.00	0
0025	9	0.3	0.5	[20, 80]	0.00	0
0026	9	0.6	0.5	[20, 80]	0.00	0
0027	9	0.1	0.5	[80, 20]	0.02	0
0028	12	0.3	0.5	[80, 20]	0.06	0
0029	12	0.6	0.5	[80, 20]	0.05	0
0030	12	0.1	0.5	[50, 50]	0.03	0
0031	12	0.3	0.5	[50, 50]	0.05	0
0032	12	0.6	0.5	[50, 50]	0.05	0
0033	12	0.1	0.5	[20, 80]	0.06	0
0034	12	0.3	0.5	[20, 80]	0.02	0
0035	12	0.6	0.5	[20, 80]	0.01	0
0036	15	0.1	0.5	[80, 20]	0.22	0
0037	15	0.3	0.5	[80, 20]	0.18	0
0038	15	0.6	0.5	[80, 20]	0.14	0
0039	15	0.1	0.5	[50, 50]	0.11	0
0040	15	0.3	0.5	[50, 50]	0.09	0
0041	15	0.6	0.5	[50, 50]	0.25	0
0042	15	0.1	0.5	[20, 80]	0.06	0
0043	15	0.3	0.5	[20, 80]	0.35	0
0044	15	0.6	0.5	[20, 80]	0.21	0
0045	18	0.1	0.5	[80, 20]	0.05	0
0046	18	0.3	0.5	[80, 20]	0.06	0
0047	18	0.6	0.5	[80, 20]	0.07	0
0048	18	0.1	0.5	[50, 50]	0.13	0
0049	18	0.3	0.5	[50, 50]	0.15	0
0050	18	0.6	0.5	[50, 50]	0.02	0
0051	18	0.1	0.5	[20, 80]	0.16	0
0052	18	0.3	0.5	[20, 80]	0.15	0
0053	18	0.6	0.5	[20, 80]	0.35	0
0054	18	0.1	0.5	[80, 20]	0.15	0
0055	21	0.3	0.5	[80, 20]	0.25	0
0056	21	0.6	0.5	[80, 20]	0.04	0
0057	21	0.1	0.5	[50, 50]	0.05	0
0058	21	0.3	0.5	[50, 50]	0.11	0
0059	21	0.6	0.5	[50, 50]	0.22	0
0060	21	0.1	0.5	[20, 80]	0.46	0
0061	21	0.3	0.5	[20, 80]	0.28	0
0062	21	0.6	0.5	[20, 80]	0.16	0

Tabela 3: Testes PLI.

Instância	T	ratio-h	ratio-d	custos-s-p	Tempo(s)	Branches	Fails	Gap(%)
0000	3	0.1	0.5	[80, 20]	4.45	14408	7100	0
0001	3	0.3	0.5	[80, 20]	0.23	6340	3183	0
0002	3	0.6	0.5	[80, 20]	0.17	6398	3214	0
0003	3	0.1	0.5	[50, 50]	0.21	6236	3120	0
0004	3	0.3	0.5	[50, 50]	0.16	6226	3136	0
0005	3	0.6	0.5	[50, 50]	0.15	7161	3594	0
0006	3	0.1	0.5	[20, 80]	0.54	6236	3121	0
0007	3	0.3	0.5	[20, 80]	0.19	6340	3183	0
0008	3	0.6	0.5	[20, 80]	0.14	6245	3183	0
0009	6	0.1	0.5	[80, 20]	0.23	7356	3662	0
0010	6	0.3	0.5	[80, 20]	0.12	5468	2700	0
0011	6	0.6	0.5	[80, 20]	0.17	8204	4053	0
0012	6	0.1	0.5	[50, 50]	0.12	7935	3925	0
0013	6	0.3	0.5	[50, 50]	0.26	7183	3540	0
0014	6	0.6	0.5	[50, 50]	0.17	9745	4833	0
0015	6	0.1	0.5	[20, 80]	3.17	85420	41839	0
0016	6	0.3	0.5	[20, 80]	482.87	2882956	1409161	0
0017	6	0.6	0.5	[20, 80]	–	5119K	–	31.47
0018	6	0.1	0.5	[80, 20]	0.55	11993	5976	0
0019	9	0.3	0.5	[80, 20]	0.63	11796	5849	0
0020	9	0.6	0.5	[80, 20]	0.45	9418	4638	0
0021	9	0.1	0.5	[50, 50]	1.40	17398	8566	0
0022	9	0.3	0.5	[50, 50]	0.69	17788	8818	0
0023	9	0.6	0.5	[50, 50]	50.97	667009	324172	0
0024	9	0.1	0.5	[20, 80]	–	13017K	–	40.51
0025	9	0.3	0.5	[20, 80]	–	12943K	–	42.56
0026	9	0.6	0.5	[20, 80]	–	12008K	–	46.03
0027	9	0.1	0.5	[80, 20]	1855.28	4091235	1982617	0
0028	12	0.3	0.5	[80, 20]	2.00	53725	26935	0
0029	12	0.6	0.5	[80, 20]	0.52	18817	9241	0
0030	12	0.1	0.5	[50, 50]	–	9528K	–	28.71
0031	12	0.3	0.5	[50, 50]	3196.23	14271554	7017843	0
0032	12	0.6	0.5	[50, 50]	–	21622K	–	26.12
0033	12	0.1	0.5	[20, 80]	–	19297K	–	51.74
0034	12	0.3	0.5	[20, 80]	–	15898K	–	48.57
0035	12	0.6	0.5	[20, 80]	–	16062K	–	46.63
0036	15	0.1	0.5	[80, 20]	–	3808736	1849276	8.09
0037	15	0.3	0.5	[80, 20]	60.26	826422	403548	0
0038	15	0.6	0.5	[80, 20]	5.92	152523	75142	0
0039	15	0.1	0.5	[50, 50]	–	23390768	11438710	23.77
0040	15	0.3	0.5	[50, 50]	–	19730845	9619924	24.79
0041	15	0.6	0.5	[50, 50]	–	19393339	9469639	27.19
0042	15	0.1	0.5	[20, 80]	–	24372892	11863501	49.82
0043	15	0.3	0.5	[20, 80]	–	28034497	13636975	47.91
0044	15	0.6	0.5	[20, 80]	–	23792178	11581737	50.61
0045	18	0.1	0.5	[80, 20]	–	13682599	6648594	5.84
0046	18	0.3	0.5	[80, 20]	104.93	1712286	824277	0
0047	18	0.6	0.5	[80, 20]	419.38	5176910	2537857	0
0048	18	0.1	0.5	[50, 50]	–	28405860	13833672	41.38
0049	18	0.3	0.5	[50, 50]	–	26917470	13093727	46.35
0050	18	0.6	0.5	[50, 50]	–	29777685	14465240	42.81
0051	18	0.1	0.5	[20, 80]	–	25324052	12244481	64.30
0052	18	0.3	0.5	[20, 80]	–	30660752	14870449	62.17
0053	18	0.6	0.5	[20, 80]	–	30062647	14626212	52.72
0054	18	0.1	0.5	[80, 20]	–	20608199	10144800	9.06
0055	21	0.3	0.5	[80, 20]	–	25721750	12671437	8.48
0056	21	0.6	0.5	[80, 20]	–	27466720	13552503	4.92
0057	21	0.1	0.5	[50, 50]	–	19220075	9314818	46.01
0058	21	0.3	0.5	[50, 50]	–	26796638	12966322	45.68
0059	21	0.6	0.5	[50, 50]	–	28569855	13873143	43.29
0060	21	0.1	0.5	[20, 80]	–	27901851	13450872	64.81
0061	21	0.3	0.5	[20, 80]	–	24356194	11788018	56.59
0062	21	0.6	0.5	[20, 80]	–	27395328	13296613	50.84

Tabela 4: Testes CP.

Instância	T	ratio-h	ratio-d	custos-s-p	Tempo(s)	Branches	Fails	Gap(%)
0000	3	0.1	0.5	[80, 20]	0.01	706	356	0
0001	3	0.3	0.5	[80, 20]	0.03	942	471	0
0002	3	0.6	0.5	[80, 20]	0.09	912	456	0
0003	3	0.1	0.5	[50, 50]	0.00	0	4	0
0004	3	0.3	0.5	[50, 50]	0.01	0	4	0
0005	3	0.6	0.5	[50, 50]	0.00	0	4	0
0006	3	0.1	0.5	[20, 80]	0.08	802	404	0
0007	3	0.3	0.5	[20, 80]	0.05	808	406	0
0008	3	0.6	0.5	[20, 80]	0.03	809	405	0
0009	6	0.1	0.5	[80, 20]	0.08	236	77	0
0010	6	0.3	0.5	[80, 20]	0.07	247	78	0
0011	6	0.6	0.5	[80, 20]	0.08	269	88	0
0012	6	0.1	0.5	[50, 50]	0.19	492	212	0
0013	6	0.3	0.5	[50, 50]	0.19	484	223	0
0014	6	0.6	0.5	[50, 50]	0.17	505	233	0
0015	6	0.1	0.5	[20, 80]	0.25	646	288	0
0016	6	0.3	0.5	[20, 80]	1.7	10501	5189	0
0017	6	0.6	0.5	[20, 80]	0.67	7488	3713	0
0018	6	0.1	0.5	[80, 20]	0.4	146	40	0
0019	9	0.3	0.5	[80, 20]	0.07	237	81	0
0020	9	0.6	0.5	[80, 20]	0.08	238	105	0
0021	9	0.1	0.5	[50, 50]	0.3	833	375	0
0022	9	0.3	0.5	[50, 50]	0.83	5433	2600	0
0023	9	0.6	0.5	[50, 50]	0.42	1103	481	0
0024	9	0.1	0.5	[20, 80]	0.57	1500	618	0
0025	9	0.3	0.5	[20, 80]	1.19	8464	4084	0
0026	9	0.6	0.5	[20, 80]	1.16	10563	5187	0
0027	9	0.1	0.5	[80, 20]	1.92	7340	3622	0
0028	12	0.3	0.5	[80, 20]	3.67	17812	8808	0
0029	12	0.6	0.5	[80, 20]	1.85	7531	3717	0
0030	12	0.1	0.5	[50, 50]	3.8	13959	6861	0
0031	12	0.3	0.5	[50, 50]	9.3	36745	18041	0
0032	12	0.6	0.5	[50, 50]	4.83	20545	10078	0
0033	12	0.1	0.5	[20, 80]	4.51	21958	10778	0
0034	12	0.3	0.5	[20, 80]	3.94	14213	6939	0
0035	12	0.6	0.5	[20, 80]	4.67	25419	12514	0
0036	15	0.1	0.5	[80, 20]	2.78	9831	4764	0
0037	15	0.3	0.5	[80, 20]	2.83	11882	5627	0
0038	15	0.6	0.5	[80, 20]	5.37	25593	12716	0
0039	15	0.1	0.5	[50, 50]	14.25	47404	23286	0
0040	15	0.3	0.5	[50, 50]	12.87	39771	19467	0
0041	15	0.6	0.5	[50, 50]	11.71	42662	20922	0
0042	15	0.1	0.5	[20, 80]	24.27	76933	38017	0
0043	15	0.3	0.5	[20, 80]	24.53	80362	39302	0
0044	15	0.6	0.5	[20, 80]	22.32	76134	37109	0
0045	18	0.1	0.5	[80, 20]	5.06	19380	9540	0
0046	18	0.3	0.5	[80, 20]	14.54	54233	26692	0
0047	18	0.6	0.5	[80, 20]	7.27	23888	11696	0
0048	18	0.1	0.5	[50, 50]	186.32	297549	146225	0
0049	18	0.3	0.5	[50, 50]	195.44	335202	164421	0
0050	18	0.6	0.5	[50, 50]	150.87	260047	128160	0
0051	18	0.1	0.5	[20, 80]	211.71	338344	166286	0
0052	18	0.3	0.5	[20, 80]	202.66	347702	171057	0
0053	18	0.6	0.5	[20, 80]	146.48	267197	131193	0
0054	18	0.1	0.5	[80, 20]	22.73	63707	30979	0
0055	21	0.3	0.5	[80, 20]	22.01	11796	5849	0
0056	21	0.6	0.5	[80, 20]	25.51	56774	27830	0
0057	21	0.1	0.5	[50, 50]	790.91	1739085	852209	0
0058	21	0.3	0.5	[50, 50]	760.21	1557932	764824	0
0059	21	0.6	0.5	[50, 50]	573.09	1278490	625875	0
0060	21	0.1	0.5	[20, 80]	871.07	1736190	849636	0
0061	21	0.3	0.5	[20, 80]	762.14	1649755	806080	0
0062	21	0.6	0.5	[20, 80]	580.69	1376875	673779	0

Tabela 5: Testes CP + RFF.

Instância	T	ratio-h	ratio-d	custos-s-p	Tempo(s)	Branches	Fails	Gap(%)
0000	3	0.1	0.5	[80, 20]	0.22	706	356	0
0001	3	0.3	0.5	[80, 20]	0.03	942	471	0
0002	3	0.6	0.5	[80, 20]	0.08	912	456	0
0003	3	0.1	0.5	[50, 50]	0.00	0	4	0
0004	3	0.3	0.5	[50, 50]	0.00	0	4	0
0005	3	0.6	0.5	[50, 50]	0.00	0	4	0
0006	3	0.1	0.5	[20, 80]	0.09	802	404	0
0007	3	0.3	0.5	[20, 80]	0.03	808	406	0
0008	3	0.6	0.5	[20, 80]	0.03	809	405	0
0009	6	0.1	0.5	[80, 20]	0.08	236	77	0
0010	6	0.3	0.5	[80, 20]	0.06	247	78	0
0011	6	0.6	0.5	[80, 20]	0.12	269	88	0
0012	6	0.1	0.5	[50, 50]	0.21	492	212	0
0013	6	0.3	0.5	[50, 50]	0.19	484	223	0
0014	6	0.6	0.5	[50, 50]	0.28	505	233	0
0015	6	0.1	0.5	[20, 80]	0.35	646	288	0
0016	6	0.3	0.5	[20, 80]	1.44	10501	5189	0
0017	6	0.6	0.5	[20, 80]	0.84	7488	3713	0
0018	6	0.1	0.5	[80, 20]	0.5	146	40	0
0019	9	0.3	0.5	[80, 20]	0.08	237	81	0
0020	9	0.6	0.5	[80, 20]	0.09	238	105	0
0021	9	0.1	0.5	[50, 50]	0.42	833	375	0
0022	9	0.3	0.5	[50, 50]	1.13	5433	2600	0
0023	9	0.6	0.5	[50, 50]	0.66	1103	481	0
0024	9	0.1	0.5	[20, 80]	0.76	1500	618	0
0025	9	0.3	0.5	[20, 80]	1.9	9442	4616	0
0026	9	0.6	0.5	[20, 80]	1.45	10563	5187	0
0027	9	0.1	0.5	[80, 20]	1.96	7340	3622	0
0028	12	0.3	0.5	[80, 20]	3.22	17812	8808	0
0029	12	0.6	0.5	[80, 20]	1.92	7531	3717	0
0030	12	0.1	0.5	[50, 50]	3.99	13959	6861	0
0031	12	0.3	0.5	[50, 50]	8.3	36745	18041	0
0032	12	0.6	0.5	[50, 50]	4.82	20545	10078	0
0033	12	0.1	0.5	[20, 80]	5.67	21958	10778	0
0034	12	0.3	0.5	[20, 80]	3.54	14213	6939	0
0035	12	0.6	0.5	[20, 80]	5.12	25419	12514	0
0036	15	0.1	0.5	[80, 20]	4.2	9831	4764	0
0037	15	0.3	0.5	[80, 20]	2.98	11882	5627	0
0038	15	0.6	0.5	[80, 20]	5.27	25593	12716	0
0039	15	0.1	0.5	[50, 50]	12.15	47404	23286	0
0040	15	0.3	0.5	[50, 50]	10.1	39771	19467	0
0041	15	0.6	0.5	[50, 50]	12.9	42662	20922	0
0042	15	0.1	0.5	[20, 80]	19.11	76933	38017	0
0043	15	0.3	0.5	[20, 80]	20.5	80362	39302	0
0044	15	0.6	0.5	[20, 80]	20.8	76134	37109	0
0045	18	0.1	0.5	[80, 20]	7.06	19380	9540	0
0046	18	0.3	0.5	[80, 20]	14.54	54233	26692	0
0047	18	0.6	0.5	[80, 20]	7.27	23888	11696	0
0048	18	0.1	0.5	[50, 50]	151.2	297549	146225	0
0049	18	0.3	0.5	[50, 50]	166.17	335202	164421	0
0050	18	0.6	0.5	[50, 50]	142.33	260047	128160	0
0051	18	0.1	0.5	[20, 80]	187.12	338344	166286	0
0052	18	0.3	0.5	[20, 80]	140.16	347702	171057	0
0053	18	0.6	0.5	[20, 80]	135.05	267197	131193	0
0054	18	0.1	0.5	[80, 20]	21.2	63707	30979	0
0055	21	0.3	0.5	[80, 20]	31.13	11796	5849	0
0056	21	0.6	0.5	[80, 20]	28.37	56774	27830	0
0057	21	0.1	0.5	[50, 50]	710.1	1739085	852209	0
0058	21	0.3	0.5	[50, 50]	701.94	1557932	764824	0
0059	21	0.6	0.5	[50, 50]	517.16	1278490	625875	0
0060	21	0.1	0.5	[20, 80]	781.03	1736190	849636	0
0061	21	0.3	0.5	[20, 80]	692.11	1649755	806080	0
0062	21	0.6	0.5	[20, 80]	512.00	1376875	673779	0

Tabela 6: Testes CP + RFF + WMF.

Instância	T	ratio-h	ratio-d	custos-s-p	Tempo(s)	Branches	Fails	Gap(%)
0000	3	0.1	0.5	[80, 20]	0.01	706	356	0
0001	3	0.3	0.5	[80, 20]	0.01	942	471	0
0002	3	0.6	0.5	[80, 20]	0.07	912	456	0
0003	3	0.1	0.5	[50, 50]	0.00	0	4	0
0004	3	0.3	0.5	[50, 50]	0.00	0	4	0
0005	3	0.6	0.5	[50, 50]	0.00	0	4	0
0006	3	0.1	0.5	[20, 80]	0.07	802	404	0
0007	3	0.3	0.5	[20, 80]	0.04	808	406	0
0008	3	0.6	0.5	[20, 80]	0.03	809	405	0
0009	6	0.1	0.5	[80, 20]	0.05	236	73	0
0010	6	0.3	0.5	[80, 20]	0.05	248	80	0
0011	6	0.6	0.5	[80, 20]	0.06	268	90	0
0012	6	0.1	0.5	[50, 50]	0.04	245	96	0
0013	6	0.3	0.5	[50, 50]	0.04	272	109	0
0014	6	0.6	0.5	[50, 50]	0.04	340	143	0
0015	6	0.1	0.5	[20, 80]	0.15	3214	1515	0
0016	6	0.3	0.5	[20, 80]	0.66	8752	4337	0
0017	6	0.6	0.5	[20, 80]	0.55	13171	6488	0
0018	6	0.1	0.5	[80, 20]	0.04	199	46	0
0019	9	0.3	0.5	[80, 20]	0.05	255	65	0
0020	9	0.6	0.5	[80, 20]	0.07	420	92	0
0021	9	0.1	0.5	[50, 50]	0.03	321	104	0
0022	9	0.3	0.5	[50, 50]	0.05	545	192	0
0023	9	0.6	0.5	[50, 50]	0.09	711	307	0
0024	9	0.1	0.5	[20, 80]	0.08	581	234	0
0025	9	0.3	0.5	[20, 80]	0.56	7521	3678	0
0026	9	0.6	0.5	[20, 80]	0.55	10761	5306	0
0027	9	0.1	0.5	[80, 20]	1.03	8458	4174	0
0028	12	0.3	0.5	[80, 20]	0.73	9205	4531	0
0029	12	0.6	0.5	[80, 20]	0.82	11356	5589	0
0030	12	0.1	0.5	[50, 50]	0.53	9340	4577	0
0031	12	0.3	0.5	[50, 50]	0.58	11951	5873	0
0032	12	0.6	0.5	[50, 50]	0.49	10394	5021	0
0033	12	0.1	0.5	[20, 80]	4.03	14374	7069	0
0034	12	0.3	0.5	[20, 80]	2.82	16752	8149	0
0035	12	0.6	0.5	[20, 80]	4.42	43225	20987	0
0036	15	0.1	0.5	[80, 20]	0.82	9989	4823	0
0037	15	0.3	0.5	[80, 20]	1.59	16498	8069	0
0038	15	0.6	0.5	[80, 20]	1.75	22683	11123	0
0039	15	0.1	0.5	[50, 50]	0.86	13054	6385	0
0040	15	0.3	0.5	[50, 50]	0.78	13076	6320	0
0041	15	0.6	0.5	[50, 50]	0.77	12537	6010	0
0042	15	0.1	0.5	[20, 80]	1.91	15964	7634	0
0043	15	0.3	0.5	[20, 80]	2.17	25176	12218	0
0044	15	0.6	0.5	[20, 80]	7.29	77820	38005	0
0045	18	0.1	0.5	[80, 20]	1.78	16521	8051	0
0046	18	0.3	0.5	[80, 20]	1.35	13068	6367	0
0047	18	0.6	0.5	[80, 20]	1.75	16470	8033	0
0048	18	0.1	0.5	[50, 50]	4.01	38895	19101	0
0049	18	0.3	0.5	[50, 50]	4.69	48011	23576	0
0050	18	0.6	0.5	[50, 50]	4.08	44044	21655	0
0051	18	0.1	0.5	[20, 80]	94.96	307729	150834	0
0052	18	0.3	0.5	[20, 80]	58.7	191235	94068	0
0053	18	0.6	0.5	[20, 80]	33.89	155178	76175	0
0054	18	0.1	0.5	[80, 20]	5.96	44305	21720	0
0055	21	0.3	0.5	[80, 20]	3.16	28079	13650	0
0056	21	0.6	0.5	[80, 20]	3.57	35476	17427	0
0057	21	0.1	0.5	[50, 50]	19.87	127515	62451	0
0058	21	0.3	0.5	[50, 50]	23.69	194088	95399	0
0059	21	0.6	0.5	[50, 50]	34.46	227539	111230	0
0060	21	0.1	0.5	[20, 80]	504.76	1255950	612599	0
0061	21	0.3	0.5	[20, 80]	257.28	767598	375665	0
0062	21	0.6	0.5	[20, 80]	52.42	201553	98805	0

Tabela 7: Testes CP + RFF + LB.

Instância	T	ratio-h	ratio-d	custos-s-p	Tempo (s)
0000	3	0.1	0.5	[80, 20]	80.84
0001	3	0.3	0.5	[80, 20]	82.29
0002	3	0.6	0.5	[80, 20]	80.14
0003	3	0.1	0.5	[50, 50]	80.99
0004	3	0.3	0.5	[50, 50]	82.35
0005	3	0.6	0.5	[50, 50]	83.58
0006	3	0.1	0.5	[20, 80]	84.40
0007	3	0.3	0.5	[20, 80]	82.78
0008	3	0.6	0.5	[20, 80]	83.08
0009	6	0.1	0.5	[80, 20]	–
0010	6	0.3	0.5	[80, 20]	–
0011	6	0.6	0.5	[80, 20]	–
0012	6	0.1	0.5	[50, 50]	–
0013	6	0.3	0.5	[50, 50]	–
0014	6	0.6	0.5	[50, 50]	–
0015	6	0.1	0.5	[20, 80]	–
0016	6	0.3	0.5	[20, 80]	–
0017	6	0.6	0.5	[20, 80]	–
0018	6	0.1	0.5	[80, 20]	–
0019	9	0.3	0.5	[80, 20]	–
0020	9	0.6	0.5	[80, 20]	–
0021	9	0.1	0.5	[50, 50]	–
0022	9	0.3	0.5	[50, 50]	–
0023	9	0.6	0.5	[50, 50]	–
0024	9	0.1	0.5	[20, 80]	–
0025	9	0.3	0.5	[20, 80]	–
0026	9	0.6	0.5	[20, 80]	–
0027	9	0.1	0.5	[80, 20]	–
0028	12	0.3	0.5	[80, 20]	–
0029	12	0.6	0.5	[80, 20]	–
0030	12	0.1	0.5	[50, 50]	–
0031	12	0.3	0.5	[50, 50]	–
0032	12	0.6	0.5	[50, 50]	–
0033	12	0.1	0.5	[20, 80]	–
0034	12	0.3	0.5	[20, 80]	–
0035	12	0.6	0.5	[20, 80]	–
0036	15	0.1	0.5	[80, 20]	–
0037	15	0.3	0.5	[80, 20]	–
0038	15	0.6	0.5	[80, 20]	–
0039	15	0.1	0.5	[50, 50]	–
0040	15	0.3	0.5	[50, 50]	–
0041	15	0.6	0.5	[50, 50]	–
0042	15	0.1	0.5	[20, 80]	–
0043	15	0.3	0.5	[20, 80]	–
0044	15	0.6	0.5	[20, 80]	–
0045	18	0.1	0.5	[80, 20]	–
0046	18	0.3	0.5	[80, 20]	–
0047	18	0.6	0.5	[80, 20]	–
0048	18	0.1	0.5	[50, 50]	–
0049	18	0.3	0.5	[50, 50]	–
0050	18	0.6	0.5	[50, 50]	–
0051	18	0.1	0.5	[20, 80]	–
0052	18	0.3	0.5	[20, 80]	–
0053	18	0.6	0.5	[20, 80]	–
0054	18	0.1	0.5	[80, 20]	–
0055	21	0.3	0.5	[80, 20]	–
0056	21	0.6	0.5	[80, 20]	–
0057	21	0.1	0.5	[50, 50]	–
0058	21	0.3	0.5	[50, 50]	–
0059	21	0.6	0.5	[50, 50]	–
0060	21	0.1	0.5	[20, 80]	–
0061	21	0.3	0.5	[20, 80]	–
0062	21	0.6	0.5	[20, 80]	–

Tabela 8: Testes DP.

Instância	T	ratio-h	ratio-d	custos-s-p	CP		CP + RFF + WMF		CP + RFF + LS	
					Tempo(s)	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)
0000	3	0.1	0.5	[80, 20]	4.45	0	0.22	0	0.01	0
0001	3	0.3	0.5	[80, 20]	0.23	0	0.03	0	0.01	0
0002	3	0.6	0.5	[80, 20]	0.17	0	0.08	0	0.07	0
0003	3	0.1	0.5	[50, 50]	0.21	0	0.00	0	0.00	0
0004	3	0.3	0.5	[50, 50]	0.16	0	0.00	0	0.00	0
0005	3	0.6	0.5	[50, 50]	0.15	0	0.00	0	0.00	0
0006	3	0.1	0.5	[20, 80]	0.54	0	0.09	0	0.07	0
0007	3	0.3	0.5	[20, 80]	0.19	0	0.03	0	0.04	0
0008	3	0.6	0.5	[20, 80]	0.14	0	0.03	0	0.03	0
0009	6	0.1	0.5	[80, 20]	0.23	0	0.08	0	0.05	0
0010	6	0.3	0.5	[80, 20]	0.12	0	0.06	0	0.05	0
0011	6	0.6	0.5	[80, 20]	0.17	0	0.12	0	0.06	0
0012	6	0.1	0.5	[50, 50]	0.12	0	0.21	0	0.04	0
0013	6	0.3	0.5	[50, 50]	0.26	0	0.19	0	0.04	0
0014	6	0.6	0.5	[50, 50]	0.17	0	0.28	0	0.04	0
0015	6	0.1	0.5	[20, 80]	3.17	0	0.35	0	0.15	0
0016	6	0.3	0.5	[20, 80]	482.87	0	1.44	0	0.6	0
0017	6	0.6	0.5	[20, 80]	–	31.47	0.84	0	0.55	0
0018	9	0.1	0.5	[80, 20]	0.55	0	0.5	0	0.04	0
0019	9	0.3	0.5	[80, 20]	0.63	0	0.08	0	0.05	0
0020	9	0.6	0.5	[80, 20]	0.45	0	0.09	0	0.07	0
0021	9	0.1	0.5	[50, 50]	1.40	0	0.42	0	0.03	0
0022	9	0.3	0.5	[50, 50]	0.69	0	1.13	0	0.05	0
0023	9	0.6	0.5	[50, 50]	50.97	0	0.66	0	0.09	0
0024	9	0.1	0.5	[20, 80]	–	40.51	0.76	0	0.08	0
0025	9	0.3	0.5	[20, 80]	–	42.56	1.9	0	0.56	0
0026	9	0.6	0.5	[20, 80]	–	46.03	1.45	0	0.55	0
0027	12	0.1	0.5	[80, 20]	1855.28	0	1.96	0	1.03	0
0028	12	0.3	0.5	[80, 20]	2.00	0	3.22	0	0.73	0
0029	12	0.6	0.5	[80, 20]	0.52	0	1.92	0	0.82	0
0030	12	0.1	0.5	[50, 50]	–	28.71	3.99	0	0.53	0
0031	12	0.3	0.5	[50, 50]	3196.23	0	8.3	0	0.58	0
0032	12	0.6	0.5	[50, 50]	–	26.12	4.82	0	0.49	0
0033	12	0.1	0.5	[20, 80]	–	51.74	5.67	0	4.03	0
0034	12	0.3	0.5	[20, 80]	–	48.57	3.54	0	2.82	0
0035	12	0.6	0.5	[20, 80]	–	46.63	5.12	0	4.42	0
0036	15	0.1	0.5	[80, 20]	–	8.09	4.2	0	0.82	0
0037	15	0.3	0.5	[80, 20]	60.62	0	2.98	0	1.59	0
0038	15	0.6	0.5	[80, 20]	5.92	0	5.27	0	1.75	0
0039	15	0.1	0.5	[50, 50]	–	23.77	12.15	0	0.86	0
0040	15	0.3	0.5	[50, 50]	–	24.79	10.1	0	0.78	0
0041	15	0.6	0.5	[50, 50]	–	27.19	12.9	0	0.77	0
0042	15	0.1	0.5	[20, 80]	–	49.82	19.11	0	1.91	0
0043	15	0.3	0.5	[20, 80]	–	47.91	20.5	0	2.17	0
0044	15	0.6	0.5	[20, 80]	–	50.61	20.8	0	7.29	0
0045	18	0.1	0.5	[80, 20]	–	5.84	7.06	0	1.78	0
0046	18	0.3	0.5	[80, 20]	104.93	0	14.54	0	1.35	0
0047	18	0.6	0.5	[80, 20]	419.38	0	7.27	0	1.75	0
0048	18	0.1	0.5	[50, 50]	–	41.38	151.2	0	4.01	0
0049	18	0.3	0.5	[50, 50]	–	46.35	166.17	0	4.69	0
0050	18	0.6	0.5	[50, 50]	–	42.81	142.33	0	4.08	0
0051	18	0.1	0.5	[20, 80]	–	64.30	187.12	0	94.96	0
0052	18	0.3	0.5	[20, 80]	–	62.17	140.16	0	58.7	0
0053	18	0.6	0.5	[20, 80]	–	52.72	135.05	0	33.89	0
0054	21	0.1	0.5	[80, 20]	–	9.06	21.2	0	5.96	0
0055	21	0.3	0.5	[80, 20]	–	8.48	31.13	0	3.16	0
0056	21	0.6	0.5	[80, 20]	–	4.92	28.37	0	3.57	0
0057	21	0.1	0.5	[50, 50]	–	46.01	710.1	0	19.87	0
0058	21	0.3	0.5	[50, 50]	–	45.68	701.94	0	23.69	0
0059	21	0.6	0.5	[50, 50]	–	43.29	517.16	0	34.46	0
0060	21	0.1	0.5	[20, 80]	–	64.81	781.03	0	504.76	0
0061	21	0.3	0.5	[20, 80]	–	56.59	692.11	0	257.28	0
0062	21	0.6	0.5	[20, 80]	–	50.84	512.00	0	52.42	0

Tabela 9: Table Comparativa CP.

Instância	PLI		CP		DP		CP + RFF + LB	
	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)	Tempo(s)	Gap(%)
0000	0.00	0	0.23	0	80.84	–	0.01	0
0001	0.00	0	0.17	0	82.29	–	0.01	0
0002	0.00	0	0.21	0	80.14	–	0.07	0
0003	0.00	0	0.16	0	80.99	–	0.00	0
0004	0.00	0	0.15	0	82.35	–	0.00	0
0005	0.00	0	0.54	0	83.58	–	0.00	0
0006	0.00	0	0.19	0	84.40	–	0.07	0
0007	0.00	0	0.14	0	82.78	–	0.04	0
0008	0.00	0	0.18	0	77.71	–	0.03	0
0009	0.18	0	0.23	0	83.08	–	0.05	0
0010	0.19	0	0.12	0	–	–	0.05	0
0011	0.03	0	0.17	0	–	–	0.06	0
0012	0.06	0	0.12	0	–	–	0.04	0
0013	0.04	0	0.26	0	–	–	0.04	0
0014	0.03	0	0.17	0	–	–	0.04	0
0015	0.04	0	3.17	0	–	–	0.15	0
0016	0.06	0	482.87	0	–	–	0.6	0
0017	0.02	0	–	0	–	–	0.55	0
0018	0.00	0	0.55	0	–	–	0.04	0
0019	0.00	0	0.63	0	–	–	0.05	0
0020	0.00	0	0.45	0	–	–	0.07	0
0021	0.00	0	1.40	0	–	–	0.03	0
0022	0.00	0	0.69	0	–	–	0.05	0
0023	0.00	0	50.97	0	–	–	0.09	0
0024	0.00	0	–	0	–	–	0.08	0
0025	0.00	0	–	0	–	–	0.56	0
0026	0.00	0	–	0	–	–	0.55	0
0027	0.02	0	1855.28	0	–	–	1.03	0
0028	0.06	0	2.00	0	–	–	0.73	0
0029	0.05	0	0.52	0	–	–	0.82	0
0030	0.03	0	–	0	–	–	0.53	0
0031	0.05	0	3196.23	0	–	–	0.58	0
0032	0.05	0	–	26.12	–	–	0.49	0
0033	0.06	0	–	51.74	–	–	4.03	0
0034	0.02	0	–	48.57	–	–	2.82	0
0035	0.01	0	–	46.63	–	–	4.42	0
0036	0.22	0	–	8.09	–	–	0.82	0
0037	0.18	0	60.62	0	–	–	1.59	0
0038	0.14	0	5.92	0	–	–	1.75	0
0039	0.11	0	–	23.77	–	–	0.86	0
0040	0.09	0	–	24.79	–	–	0.78	0
0041	0.25	0	–	27.19	–	–	0.77	0
0042	0.06	0	–	49.82	–	–	1.91	0
0043	0.35	0	–	47.91	–	–	2.17	0
0044	0.21	0	–	50.61	–	–	7.29	0
0045	0.05	0	–	5.84	–	–	1.78	0
0046	0.06	0	104.93	0	–	–	1.35	0
0047	0.07	0	419.38	0	–	–	1.75	0
0048	0.13	0	–	41.38	–	–	4.01	0
0049	0.15	0	–	46.35	–	–	4.69	0
0050	0.02	0	–	42.81	–	–	4.08	0
0051	0.16	0	–	64.30	–	–	94.96	0
0052	0.15	0	–	62.17	–	–	58.7	0
0053	0.35	0	–	52.72	–	–	33.89	0
0054	0.15	0	–	9.06	–	–	5.96	0
0055	0.25	0	–	8.48	–	–	3.16	0
0056	0.04	0	–	4.92	–	–	3.57	0
0057	0.05	0	–	46.01	–	–	19.87	0
0058	0.11	0	–	45.68	–	–	23.69	0
0059	0.22	0	–	43.29	–	–	34.46	0
0060	0.46	0	–	64.81	–	–	504.76	0
0061	0.28	0	–	56.59	–	–	257.28	0
0062	0.16	0	–	50.84	–	–	52.42	0

Tabela 10: Table Comparativa CP, PLI e DP.

6 Conclusões

Neste trabalho apresentamos três novas restrições redundantes para fortalecer a formulação em Programação por Restrições para o Problema do Dimensionamento de Lotes Capacitado de Item Único (CSILSP).

O objetivo da pesquisa foi estudar restrições redundantes que aumentassem o poder de poda do algoritmo principal de maneira que a formulação do CSILSP em CP tivesse uma grande redução em tempo de execução, redução nas ramificações e aumento no número de fails.

A primeira restrição apresentada foi a `LowerBound`. Esta restrição se mostrou de grande valia para a formulação principal, provendo resultados positivos, quando comparamos com a formulação sem a restrição redundante.

A segunda restrição apresentada foi a `RelaxedFlowFilter`. Essa restrição, apesar de simples em termos de formulação e compreensão, quando comparada com as demais, proporcionou as melhores podas nos valores das variáveis do problema. Quando combinada com a `LowerBound` nos proporcionou os melhores resultados obtidos nesse trabalho.

A terceira restrição apresentada foi a `WindowMaxFlow`. Esta restrição apresentou resultados positivos, porém de maneira menos impactante em relação às demais. Apesar da formulação matemática sólida, houve pequenos ganhos em tempo de processamento. Mas acreditamos que ela pode ser de grande valia como auxiliar para instâncias que contem com uma grande capacidade de produção e grande capacidade de inventário, podendo ser especialmente útil para formulações em Programação Dinâmica.

Em trabalhos futuros pode-se aplicar as restrições apresentadas em variantes do problema apresentado, como por exemplo: considerar a produção de múltiplos tipos de itens ao invés de apenas um; permitir a satisfação da demanda de um determinado período produzindo os itens após esse período, normalmente pagando uma penalidade (variante também conhecida como Problema do Dimensionamento de Lotes com *Backlogging* de Produção) e diversas outras já apresentadas na literatura [2, 6, 15]. Além disso, é possível procurar aprimoramentos nas restrições apresentadas e/ou combiná-las com outras restrições de maneira a apresentar resultados interessantes.

Referências

[1] T. J. Bertsimas D. Introduction to linear optimization. *Athena Scientific*, 1997.

[2] G. R. Bitran and H. H. Computational complexity of the capacitated lot size problem. *Management Science*, (10), 1982.

[3] N. Brahimi, S. Dauzère-Pérès, N. M. Najid, and A. Nordli. Single item lot sizing problems. *European Journal of Operational Research*, (1), 2006.

[4] H. Chen, D. Hearn, and C. Lee. A new dynamic programming algorithm for the single item capacitated dynamic lot size model. *Journal of Global Optimization*, (4), 1994.

[5] W. J. Cheng BMW, Lee JHM. Speeding up constraint propagation by redundant modeling. *Freuder EC*, Principles and Practice of Constraint Programming:13, 1996.

[6] Y. Feng, S. Chen, A. Kumar, and B. Lin. Solving single-product economic lot-sizing problem with non-increasing setup cost, constant capacity and convex inventory cost in o(n log n) time. *Computers & Operations Research*, (4), 2011.

[7] M. Florian and M. Klein. Deterministic production planning with concave costs and capacity constraints. *Management Science*, (1), 1971.

[8] J. Gaschnig. Performance measurement and analysis of certain search algorithms. *Technical Report - Carnegie-Mellon University, Pittsburgh, PA*, 1979.

[9] G. German. *Constraint programming for lot-sizing problems*. PhD thesis, Université Grenoble Alpes, 2018.

[10] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *Artificial intelligence*, 14(3):263–313, 1980.

[11] S. Heipcke. Comparing constraint programming and mathematical programmimg approaches to discrete optimisation - the change problem. *The journal of the operational research society*, 50(6):581 – 595, 1999.

[12] S. V. Hoesel and A. P. M. Wagelmans. An o(t 3) algorithm for the economic lotsizing problem with constant capacities. *Management Science*, (1), 1996.

[13] V. R. Houndji, P. Schaus, L. A. Wolsey, and Y. Deville. The stockingcost constraint. *Principles and Practice of Constraint Programming*, (10):7, 2014.

[14] V. R. Houndji, P. Schaus, L. A. Wolsey, and Y. Deville. The stocking-cost constraint. *Principles and practice of constraint programming*, (1), 2014.

[15] M. Kovalyov and E. Pesch. An o(n log n) algorithm for a single-item capacitated lot-sizing problem with linear costs and no backlogging. *International Journal of Production Research*, (3), 2014.

[16] S. K. Papadimitriou C. Combinatorial optimization: Algorithms and complexity. *Dover Publications*, 2013.

[17] Y. Pochet and L. A. Wolsey. Production planning by mixed integer programming. *Springer Science & Business Media*, (1), 2006.

[18] C.-G. Quimper, P. van Beek, A. L´opez-Ortiz, and A. S. Golynski. An efficient bounds consistency algorithm for the global cardinality constraint. (2833), 2003.

[19] J. B. O. Ravindra K Ahuja, Thomas L Magnanti. Network flows: theory, algorithms, and applications. 1993.

[20] J.-C. Régin. A filtering algorithm for constraints of difference in csps. pp. 362–367, 1994.

[21] J.-C. Régin. Cost-based arc consistency for global cardinality constraint. *Principles and Practice of Constraint Programming*, (8), 2002.

[22] J.-C. Régin and M. Rueher. A global constraint combining a sum constraint and difference constraints. pp. 384–395, 2000.

[23] D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. in a. borning, editor, principles and practice of constraint programming. *volume 874 of Lecture Notes in Computer Science, pages 10–20*, Springer Berlin / Heidelberg.

[24] H. M. Wagner and T. M. Whitin. Dynamic version ot the economic lot size model. *Management Science*, (1), 1958.

[25] L. A. Wosley. Progress with single-item lot-sizing. *European Journal of Operational Research*, (3), 1995.

[26] L. A. Wosley. Integer programming. *Wiley Series in Discrete Mathematics and Optimization*, 1998.