Um Novo Método Iterativo Exato Para o Problema das Sequências Justas Ponderadas

Pablo Suria Pereira Mousinho

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2025

Pablo Suria Pereira Mousinho

Um Novo Método Iterativo Exato Para o Problema das Sequências Justas Ponderadas

Monografia apresentada ao Programa de Pós-Graduação em Informática do Centro de Informática, da Universidade Federal da Paraíba, como requisito para a obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Bruno Pessoa

João Pessoa, 2025

Catalogação na publicação
Seção de Catalogação e Classificação

M932n Mousinho, Pablo Suria Pereira.
Um novo método iterativo exato para o problema das sequências justas ponderadas / Pablo Suria Pereira Mousinho. - João Pessoa, 2025.
55 f. : il.

Orientação: Bruno Pessoa.
Dissertação (Mestrado) - UFPB/CI.

1. Escalonamento de processos. 2. Sequências justas. 3. Programação Inteira Mista. I. Pessoa, Bruno. II. Título.

UFPB/BC CDU 004.451.26(043)

Elaborado por RUSTON SAMMEVILLE ALEXANDRE MARQUES DA SILVA - CRB-15/0386

UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Ata da Sessão Pública de Defesa de Dissertação de Mestrado de Pablo Suria Pereira Mousinho, candidato ao título de Mestre em Informática na área de Sistemas de Computação, realizada em 30 de agosto de 2024.

Aos trinta dias do mês de agosto do ano de dois mil e vinte e quatro, às quinze horas, no Centro de Informática da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para julgar o Trabalho Final do discente Pablo Suria Pereira Mousinho, vinculado a esta Universidade sob a matrícula nº 20221003244, candidato ao grau de Mestre em Informática, na área de *"Sistemas de Computação"*, na linha de pesquisa *"Computação Distribuída"*, do Programa de Pós-Graduação em Informática. A comissão examinadora foi composta pelos professores: Bruno Jefferson de Sousa Pessoa, Orientador e Presidente da banca; Gilberto Farias de Sousa Filho, Examinador Interno; Daniel Aloise, Examinador Externo à Instituição. Dando início aos trabalhos, o Presidente da Banca cumprimentou os presentes, comunicou a finalidade da reunião e passou a palavra ao candidato para que ele fizesse a exposição oral do trabalho de dissertação intitulado "**Um Novo Método Iterativo Exato Para o Problema das Sequências Justas Ponderadas**". Concluída a exposição, o candidato foi arguido pela Banca Examinadora que emitiu o seguinte parecer: "***aprovado***". Do ocorrido, eu, Clauirton de Albuquerque Siebra, Vice-Coordenador do Programa de Pós-Graduação em Informática, lavrei a presente ata que vai assinada por mim e pelos membros da Banca Examinadora. João Pessoa, 30 de agosto de 2024.

gov.br Documento assinado digitalmente
CLAUIRTON DE ALBUQUERQUE SIEBRA
Data: 10/09/2024 11:07:16-0300
Verifique em https://validar.iti.gov.br

Clauirton de Albuquerque Siebra
Vice-Coordenador do Programa de Pós-Graduação em Informática

gov.br Documento assinado digitalmente
BRUNO JEFFERSON DE SOUSA PESSOA
Data: 04/09/2024 15:53:27-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Bruno Jefferson de Sousa Pessoa
Orientador (PPGI-UFPB)

gov.br Documento assinado digitalmente
GILBERTO FARIAS DE SOUSA FILHO
Data: 31/08/2024 10:10:58-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Gilberto Farias de Sousa Filho
Examinador Interno (PPGI-UFPB)

Prof. Dr. Daniel Aloise
Examinador Externo à Instituição (EPM)

DEDICATÓRIA

Dedico esse trabalho a toda minha família, sobretudo meus pais, por todo amor que nunca faltou em minha vida.

AGRADECIMENTOS

A Deus, pela minha vida, e por me permitir ultrapassar todos os obstáculos encontrados ao longo do caminho.

Aos meus pais e irmãos, que me apoiaram nos momentos difíceis e compreenderam a minha ausência enquanto eu seguia meu caminho de aprendizado.

Aos professores do Centro de Informática pelos seus ensinamentos e conselhos. Em particular, agradeço ao meu orientador Dr. Bruno Pessoa, sem ele este trabalho não seria possível.

Aos meus colegas de turma, por compartilharem comigo tantos momentos que sempre lembrarei com carinho e por todo o apoio ao longo deste percurso que passou pela graduação e, agora, pelo mestrado.

RESUMO

Este trabalho aborda o Problema das Sequências Justas Ponderadas (PSJP), um problema de otimização introduzido recentemente na literatura que faz parte da classe de problemas de sequências justas. Ele abrange grande número de aplicações, em diferentes áreas, as quais variam desde a minimização de custos em uma linha de montagem de automóveis ao sequenciamento de serviços de manutenção das máquinas de um fábrica. O PSJP é um problema de escalonamento periódico, com horizonte de tempo finito, que, dado um conjunto de atividades com diferentes prioridades, tem como objetivo produzir uma sequência de execuções tal que o máximo produto, definido como o produto entre a maior distância temporal entre duas execuções consecutivas de uma mesma tarefa e sua prioridade, seja minimizado. O presente trabalho propõe aprimoramentos que fortalecem a formulação matemática clássica do PSJP, além de um algoritmo iterativo baseado em recentes avanços da literatura. Os experimentos computacionais realizados mostram que o método iterativo proposto encontra mais soluções ótimas e em menor tempo computacional quando comparado à abordagem exata do estado da arte.

Palavras-chave: <Escalonamento>, <Sequências Justas>, <Programação Inteira Mista>.

ABSTRACT

This work addresses the Weighted Fair Sequences Problem (WFSP), a recently introduced optimization problem that belongs to the class of fair sequence problems. It encompasses a wide range of applications across different fields, varying from cost minimization in an automobile assembly line to the sequencing of maintenance services for factory machines. The WFSP is a periodic scheduling problem with a finite time horizon, where, given a set of activities with different priorities, the goal is to produce a sequence of executions such that the maximum product, defined as the product of the largest temporal distance between two consecutive executions of the same task and its priority, is minimized. This work proposes enhancements that strengthen the classical mathematical formulation of the WFSP, as well as an iterative algorithm based on recent advances in the literature. The computational experiments conducted demonstrate that the proposed iterative method finds more optimal solutions in less computational time compared to the state-of-the-art exact approach.

Key-words: <Scheduling>, <Fair Sequences>, <Integer-Mixed Programming>.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

EDF - *Earliest Deadline First*

NMC - Modelo matemático que trata do número máximo de cópias dos símbolos

MI - Método Iterativo

PLIM - Programação Linear Inteira Mista

PSJP - Problema das Sequências Justas Ponderadas

RM - *Rate Monotonic*

RTVP - *Response Time Variability Problem*

Sumário

1 INTRODUÇÃO

1.1 Definição do Tema

Muito embora não haja uma definição consensual de justiça em problemas de escalonamento, ela é frequentemente associada ao compartilhamento proporcional de recursos entre tarefas ao longo do tempo. Tarefas com maior importância são executadas com mais frequência, e suas execuções são distribuídas de maneira uniforme no decurso do tempo. Problemas de escalonamento que impõem restrições sobre as distâncias temporais entre execuções consecutivas de uma tarefa ou atividade são conhecidos na literatura como problemas de escalonamento com distâncias restritas [10] ou de sequências justas [12]. Recentemente, Pessoal et al. [16] apresentaram o Problema das Sequências Justas Ponderadas (PSJP), que pertence a essa classe de problemas. No PSJP, quanto maior a importância de uma tarefa, menores e mais uniformemente distribuídas são as distâncias entre suas execuções consecutivas.

Pessoa et al. [16] descrevem o PJSP da seguinte forma. Seja $X = \{x_1, x_2, ..., x_n\}$ um conjunto de símbolos, cada um deles com tamanho unitário, e $c : X \rightarrow \mathbb{Z}^+$ uma função que determina suas prioridades. Para um símbolo x_i, sua prioridade é denotada por $c_i = c(x_i)$. Considere uma sequência $S = s_1, s_2, ..., s_T$ de T símbolos, com $T \leq TMAX$, onde $TMAX$ é um parâmetro do problema que corresponde ao tamanho máximo das sequências viáveis. A frequência ou o número de ocorrências dos símbolos em S é uma variável do problema e no mínimo igual a um. A distância $d_{i,k,k+1}$ entre as ocorrências ou cópias k e $k+1$ do símbolo $x_i \in X$ é o número de símbolos entre eles mais um. Assumindo que l é o índice da última cópia do símbolo x_i em uma sequência qualquer, $d_{i,l,1}$ é a distância entre a última cópia de x_i em um ciclo e sua primeira cópia no ciclo subsequente. Sendo $D_i = \max\{\max_{k=1,...,l-1}\{d_{i,k,k+1}\}, d_{i,l,1}\}$ a maior distância entre todas as ocorrências sucessivas do símbolo x_i, incluindo $d_{i,l,1}$, o objetivo é encontrar uma sequência S que minimize o maior produto $D_i c_i$, para $i = 1, ..., n$.

A Figura 1 extraída de [16] ilustra diferentes soluções viáveis para um conjunto de símbolos X = {A,B,C,D,E,F} cujas prioridades são descritas como $c_A > c_B = c_C = c_D > c_E = c_F$. Na primeira solução, as cópias de cada símbolo foram posicionados uma após a outra. Na segunda, as cópias foram distribuídas de modo mais uniforme com o intuito de minimizar a diferença ente as distâncias das cópias consecutivas de um mesmo símbolo. Comparando a efetividade de cada solução com base no símbolo A de maior prioridade, podemos observar que houve uma redução na maior distancia $D_A = 9$ na primeira solução para $D_A = 3$ na segunda solução. Dessa forma, houve também uma redução no produto $D_A c_A$. Como todos os símbolos tiveram sua maior distância reduzida na solução 2, podemos afirmar que a solução 2 é melhor que a solução 1.

Este trabalho apresenta uma abordagem exata para resolver o PSJP, estendendo o trabalho inicial de Pessoal et al. [16].

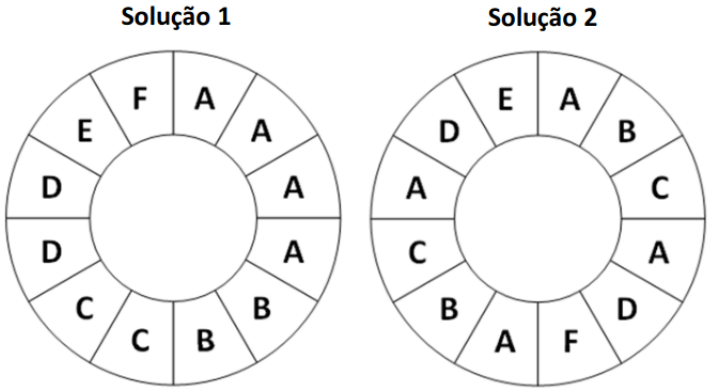

Figura 1: Exemplo de soluções para PSJP. Reprodução com permissão de Pessoa et al. [16].

1.2 Justificativa

Analisando o PSJP em um contexto mais amplo, observa-se que ele surge sempre que clientes, eventos, produtos ou tarefas precisam ser sequenciados, a fim de minimizar o tempo que esperam para a próxima oportunidade de obterem os recursos necessários para avançarem. Desse modo, uma quantidade enorme de aplicações são abrangidas pelo PSJP, tais como minimização de custos em uma linha de montagem de automóveis, manutenção de máquinas, coleta de lixo hospitalar, escala de plantões em UTIs, transmissão de conteúdo multimídia, escalonamento de processos em sistemas de tempo real, dentre outros. Algumas dessas aplicações são detalhadas abaixo.

No problema de manutenção periódica de máquinas [2], um escalonamento de manutenções deve ser elaborado com o objetivo de minimizar o tempo decorrido entre duas manutenções consecutivas, tendo em vista que os custos de operação das máquinas crescem com o tempo decorrido desde a última manutenção realizada. Como cada máquina possui um custo de operação próprio, algumas delas devem ser priorizadas no escalonamento dos serviços de manutenção. Portanto, a frequência ideal dos serviços de manutenção para cada máquina deve ser vista como uma variável do problema.

O planejamento hospitalar é outra área em que o PSJP pode ser aplicado, especialmente no escalonamento de plantões médicos em Unidades de Terapia Intensiva (UTI) [9]. Em alguns hospitais, médicos são contratados por meio de contratos individuais, o que pode gerar uma grande variação no número de plantões mensais dentro de uma mesma equipe, tornando a criação de escalas médicas um desafio. Por exemplo, se alguns médicos concentram seus plantões em uma única semana, os plantões consecutivos podem ficar muito próximos, afetando a qualidade do trabalho devido ao estresse acumulado, algo comum em UTIs. Por outro lado, se o intervalo entre plantões consecutivos é

muito longo, há uma quebra na continuidade dos protocolos adotados pelos plantonistas, o que também pode ser prejudicial. O objetivo seria elaborar uma escala médica onde os intervalos entre plantões consecutivos fossem o mais uniformes possível.

Devido à vasta gama de aplicações que o PSJP possui e à dificuldade em resolvê-lo, a busca por aprimoramentos nos métodos de solução disponíveis na literatura abre espaço para contribuições significativas. Estudos recentes, por exemplo, têm proposto a ideia de dividir o PSPJ em uma série de subproblemas de menor complexidade. Além disso, uma lacuna importante se destaca na pesquisa sobre o PSJP, já que, conforme apontado em [16], os autores usam um corte baseado em uma conjectura que, por não ter sido formalmente validada, compromete a garantia de otimalidade dos resultados obtidos. Esses fatores não apenas evidenciam a necessidade de um aprofundamento na pesquisa, mas também serviram como motivadores essenciais para a realização do presente trabalho.

1.3 Objetivos

1.3.1 Objetivo Geral

O presente trabalho tem a finalidade de estender a solução proposta para o PSJP em [16], aprimorando sua formulação matemática e introduzindo um novo método exato e iterativo que seja capaz de encontrar soluções inéditas para instâncias ainda não resolvidas, com garantia de otimalidade e em menor tempo computacional.

1.3.2 Objetivos Específicos

a) Análise da literatura referente a problemas de escalonamento e, em especial, do Problema de Sequências Justas Ponderadas e de suas soluções propostas.

b) Aprimoramento do modelo de Programação Linear Inteira Mista proposto em [16].

c) Elaboração de um método exato e iterativo para resolver o PSJP.

d) Análise da eficiência dos métodos propostos em comparação à literatura.

1.4 Estrutura da Monografia

O restante do trabalho está organizado como segue:

- A Seção 2 apresenta uma revisão da literatura disponível sobre problemas de sequências justas e, em especial, artigos que estudam o PSJP;
- A Seção 3 detalha as melhorias propostas e a solução desenvolvida neste trabalho;

- A Seção 4 discute os resultados computacionais;
- A Seção 5 traz as conclusões e estudos futuros deste trabalho.

2 REVISÃO DA LITERATURA

Problemas de sequências justas têm sido abordados historicamente nas áreas de Ciência da Computação e Pesquisa Operacional [16]. Na primeira área, os maiores interesses têm sido o desenvolvimento de algoritmos e teorias para o escalonamento de processos em sistemas de tempo real e problemas que surgem devido à transmissão de dados via *broadcast*.

Ao abordar o *problema de escalonamento de tarefas periódicas* pela primeira vez, o trabalho [15] abriu espaço para todo esse universo de problemas. Neste trabalho, ainda considerado um dos mais influentes da área de escalonamento de processos em sistemas de tempo real, o problema estudado consiste em escalonar um conjunto de tarefas de modo que cada uma delas seja executada exatamente uma vez em intervalos de tempo pré-definidos.

Em 1989, Holte et al. [11] formalizaram um problema de escalonamento de processos esporádicos tendo como base a comunicação de satélites com estações terrestres denominado *problema de escalonamento pinwheel*. Os satélites transmitem informações periodicamente para a estação, que tem capacidade limitada de processamento podendo se comunicar com um único satélite por vez. A frequência de transmissão é uma característica específica de cada satélite e, sendo assim, deve ser tratada como uma entrada do problema. Portanto, o problema consiste na construção de uma sequência de transmissões que sacie as necessidades de cada satélite e evite, simultaneamente, a perda de qualquer informação, levando em consideração a limitação de processamento da estação terrestre.

Alguns anos depois, um novo conceito denominado de progresso proporcional ou *P-Fairness* foi introduzido no escalonamento periódico de tarefas. Desenvolvida por [5], esse conceito pretendia trazer uma noção de justiça para o escalonamento ao agendar cada tarefa progressivamente de acordo com um peso calculado a partir do seu período e tempo de execução. Desta forma, as tarefas são agendadas com uma certa uniformidade ao longo do tempo. No mesmo ano, [10] introduziram o *problema de escalonamento com distâncias restritas*, que consiste no escalonamento de um conjunto de tarefas com execução periódica, com a adição de que essas tarefas possuem um limite máximo, definido previamente, para o valor da distância temporal entre suas execuções. O problema de escalonamento *pinwheel* pode ser considerado como a versão discreta deste [16].

No estudo de transmissão de dados em ambientes de *broadcast*, destaca-se o *problema de broadcast de dados*, introduzido por [1]. Ele pode ser observado em sistemas de transmissão onde os canais de comunicação de servidores para clientes possuem uma largura de banda muito superior àquela reservada para transmissão na via inversa. Nesse cenário, a transmissão de arquivos é feita pelo servidor de forma cíclica, e o cliente deve escutar o meio de comunicação até a chegada dos arquivos desejados. Desta forma, torna-se

necessário encontrar um escalonamento de transmissão que reduza o tempo de download dos arquivos.

O *problema de escalonamento de serviços de manutenção* estudado por [2] é um problema de escalonamento de atividades de vários tipos em um horizonte de tempo infinito. Neste trabalho, os autores abordam um conjunto de máquinas cujos os custos de operação aumentam à medida que o tempo vai passando desde sua última manutenção. Portanto, o objetivo do problema é minimizar o custo médio, por unidade de tempo discreto, de realizar manutenções em diferentes máquinas ao longo do tempo ao encontrar um escalonamento adequado para essas manutenções. [4] estudaram= uma versão generalizada deste problema, ao adicionar a possibilidade de várias máquinas serem escalonadas para manutenção em um mesmo *slot* de tempo.

[8] introduziram um problema de sequenciamento nomeado de *problema de variabilidade do tempo de resposta* (RTVP, do inglês *response time variability problem*), que pode ser identificado sempre que tarefas precisam ser escalonadas de forma que reduza a variabilidade das distâncias temporais entre suas execuções. Neste problema, a sequência de tarefas é dita finita e periódica já que as frequências ou demandas de cada tarefa são fornecidas como entradas para o problema.

De todos os problemas mencionados, apenas o RTVP pode ser inserido em um contexto onde atividades precisam ser executadas em um intervalo de tempo finito ou preestabelecido. Todos os outros problemas consideram o horizonte de tempo como infinito. Considerar um horizonte de tempo finito exige o tratamento das distâncias temporais entre ciclos ou períodos, o que tem um impacto direto nas definições, formulações e soluções dos problemas [16]. Assim como o RTVP, o PSJP lida com o escalonamento de ciclos com horizonte de tempo finito, mas os tamanhos de suas sequências não são definidos a priori já que encontrar a melhor frequência para cada tarefa é um dos objetivos do problema. Portanto, o PSJP se encaixa em uma série de situações reais onde as frequências de cada tarefa não são conhecidas previamente. Devido a essas similaridades e diferenças entre o RTVP e o PSJP, o último pode ser considerado como uma generalização do primeiro.

Todos os problemas mencionados acima são detalhados nas seções que seguem.

2.1 Problema de Escalonamento de Tarefas Periódicas

Ter uma sequência viável de tarefas que devem ser executadas periodicamente enquanto respeitam restrições de tempo de execução permite que possíveis problemas sejam encontrados e eliminados. Com esse objetivo, deve-se encontrar algoritmos que, dado um conjunto de tarefas, sejam capazes de montar sequências viáveis que atendam às restrições temporais.

O problema estudado neste tópico veio à tona pela primeira vez em [15] e é descrito

da seguinte forma:

Seja $X = \{x_1, x_2, ..., x_n\}$ um conjunto contendo n tarefas periódicas, caracterizadas por um tempo de execução t_i, um período p_i e um deadline d_i. Considerando que $p_i = d_i$, o problema consiste em encontrar um escalonamento periódico de modo que a tarefa $x_i, \forall x_i \in X$, seja executada exatamente uma vez, em um sistema monoprocessado, no intervalo $[p_{ik}, p_{i(k+1)}], k \in \mathbb{N}$.

Um algoritmo de escalonamento deve definir um conjunto de regras que determina qual tarefa deve ser executada em cada momento. Uma das classes de algoritmos mais presentes na literatura é denominada como escalonamento dirigido à prioridade. Nesta classe de algoritmos, essas prioridades são atribuídas baseadas nas restrições temporais de cada tarefa. Um algoritmo dessa classe pode ser classificado baseado na variação de prioridades entre cada execução (*job*) de uma tarefa. Se a prioridade permanece fixa para cada *job* de uma tarefa, o algoritmo é dito dirigido à prioridades estáticas, caso contrário, ele é classificado como dirigido à prioridades dinâmicas.

[15] propuseram dois algoritmos clássicos na área de escalonamento periódico dirigido à prioridades: o *Rate Monotonic* (RM) e o *Earliest Deadline First* (EDF). O primeiro toma as prioridades como fixas e é considerado como ótimo para sua classe de problemas, logo, nenhum outro algoritmo desta classe pode escalonar um conjunto de tarefas que não seja escalonável por ele [3]. A política de prioridades utilizada pelo RM faz com que as tarefas de menor período sejam executadas com uma frequência maior. O escalonamento RM baseia-se nas seguintes premissas:

1. Todas as tarefas são periódicas;
2. O *deadline*(d_i) de cada tarefa é igual ao seu período(p_i);
3. Todas as tarefas são independentes;
4. As tarefas possuem um tempo de execução (t_i) fixo;
5. Assume-se que o tempo de transição entre tarefas é nulo;

Dentro dessas premissas, [15] demonstraram que um conjunto de tarefas periódicas é escalonável pelo RM se:

$$\sum_{i=1}^{n} \frac{t_i}{p_i} \leq n(2^{\frac{1}{n}} - 1)$$

Essa condição é dita suficiente mas não necessária já que conjuntos de tarefas com utilização de processador maior que o lado direito da inequação podem ainda ser escalonados pelo algoritmo.

O algoritmo EDF é dirigido a prioridades dinâmicas e também é considerado ótimo para sua classe de problemas. Sua política consiste em atribuir prioridades dinâmicas baseadas nos deadlines dos *jobs* correntes. A tarefa cujo *job* atual possui um tempo de deadline mais cedo ganha prioridade sobre as outras. Assim, toda vez que uma nova tarefa é liberada para execução, a fila de tarefas correntes é reordenada baseada na política de prioridades.

Utilizando as mesmas premissas do algoritmo de escalonamento RM, um conjunto de tarefas periódicas é escalável através do EDF se, e somente se,

$$\sum_{i=1}^{n} \frac{t_i}{p_i} \leq 1$$

Diferente do *Rate Monotonic*, esta condição é suficiente e necessária para avaliar a viabilidade de um certo conjunto de tarefas a ser escalonado através do EDF.

Testes de viabilidade tratam do problema de decidir se um conjunto de tarefas é de alguma forma viável ou não, ou seja, se existe algum escalonamento tal qual todos os *jobs* das tarefas são executados antes de seu deadline. Enquanto testes e condições de escalonabilidade estão sempre vinculados a certo algoritmo, testes de viabilidade são independentes do algoritmo a ser utilizado. Em ([14], [13]) o problema de viabilidade é definido com NP-completo para $p_i \neq d_i$, tornando este um dos grandes desafios do escalonamento de tarefas periódicas.

2.2 Escalonamento P-Fair

[5] defende a utilização do conceito de progresso proporcional como forma de propagar um senso de justiça aos problemas de escalonamento periódico. O termo justiça pode ser interpretado, neste contexto, como uma métrica em função da qual um conjunto de tarefas deve ser escalonado. Para justificar tal conceito, os autores utilizam o seguinte exemplo:

Considere uma companhia aérea que possue m aeronaves e n tripulações, tal que $n > m$. Assuma que m tripulações são escaladas para trabalhar em um determinado dia. Devido a questões como antiguidade, performance ou outros fatores, algumas tripulações são escaladas para trabalhar com uma frequência maior que outras. Para cada tripulação $x_i \in X$, seja w_i a fração do total de dias que x_i deve trabalhar em um dado período, de modo que $\sum_{i=1}^{n} w_i = m$, onde X corresponde ao conjunto de tripulações da companhia. A ideia é gerar uma escala, de acordo com a qual cada tripulação trabalhe em uma taxa uniforme, isto é, após u dias de trabalho, a tripulação x_i terá que trabalhar entre $\lfloor w_i u \rfloor$ ou $\lceil w_i u \rceil$ dias.

Tal conceito de justiça, denominado *P-Fairness*, consiste em um tipo de escalonamento com restrições mais fortes do que aquelas utilizadas num escalonamento periódico

de [15]. Desse modo, pode-se considerar um escalonamento P-Fair como periódico, mas o inverso não é verdade. Segue abaixo a definição formal deste escalonamento:

Seja $X = \{x_1, x_2, ..., x_n\}$ um conjunto de tarefas definidas pelo par ordenado (t_i, p_i), respectivamente, o tempo e o período de execução de x_i, para i $= 1, \ldots, n$. O escalonamento P-Fair impõe que, a cada $u \in N$ unidades ou *slots* de tempo, a tarefa x_i deve ser executada por $\lfloor w_i u \rfloor$ ou $\lceil w_i u \rceil$ *slots*, onde $w_i = \frac{t_i}{p_i}$ representa o peso da tarefa x.

Podemos observar a diferença entre um escalonamento periódico e um escalonamento P-Fair na Figura 2 que apresenta os dois tipos de escalonamento para a seguinte situação: considere a tarefa x_i com $p_i = 7$ e $t_i = 4$. De acordo com a definição do escalonamento periódico, a tarefa x_i deve ser executada por quatro ($t_i = 4$) unidades de tempo a cada sete ($p_i = 7$). Portanto, o escalonamento da parte superior da figura pode ser considerado como periódico. No caso do escalonamento P-Fair, a cada instante de tempo u, a tarefa x_i deve ser executada por $\lfloor w_i u \rfloor$ ou $\lceil w_i u \rceil$ *slots*. De acordo com a Tabela 1, verifica-se que no instante $u = 2$, a tarefa x_i só poderia ter sido executada por no máximo dois *slot* de tempo e, da mesma forma, no instante $u = 4$, a mesma tarefa só poderia ter sido executada por no máximo três *slots*. Portanto, o escalonamento superior não atende às restrições de quantidade de *slots* definidas para o escalonamento P-Fair, ao contrário do que pode ser visto no escalonamento inferior que apresenta um certo grau de uniformidade na distribuição dos escalonamentos ao longo do tempo. Outra consequência desta uniformidade é a diminuição das distância temporais entre execuções de um *job*.

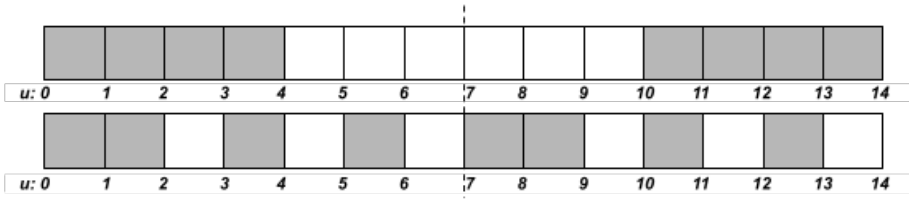

Figura 2: Dois escalonamentos para uma tarefa x_i com $p_i = 7$ e $x_i = 4$. O escalonamento de cima é periódico, enquanto que o outro é P-Fair.

Além de definir o P-Fair, [5] provaram que qualquer instância do problema de escalonamento periódico possui um escalonamento P-Fair correspondente, caso $\sum_{i=1}^{n} w_i \leq m$, onde m representa o número de recursos compartilhados. Os autores também descreveram e provaram a corretude de um algoritmo de tempo polinomial capaz de gerar um escalonamento P-Fair para qualquer conjunto de tarefas viáveis e, por consequência de que todo escalonamento P-Fair é periódico, tal algoritmo também é capaz de resolver o problema de escalonamento de tarefas periódicas.

Tabela 1: Intervalo de tempo (em slots) permitido para a execução tarefa x_i da Figura 2 a cada instante de tempo u, segundo o escalonamento P-Fair.

u	$\lfloor w_i u \rfloor$	$\lceil w_i u \rceil$
1	0	1
2	0	2
3	1	2
4	2	3
5	2	3
6	3	4
7	3	4
8	4	5
9	5	6
10	5	6
11	6	7
12	6	7
13	7	8
14	7	8

2.3 Problema de Escalonamento Pinwheel

Dado um conjunto de n inteiros positivos $D = \{d_1, d_2, ..., d_n\}$, o problema *pinwheel* tem como objetivo encontrar uma sequência infinita de símbolos de $X = \{x_1, x_2, ..., x_n\}$ de modo que exista no mínimo um símbolo x_i dentro de qualquer subsequência de d_i símbolos consecutivos para $i = 1, ..., n$ [11].

O problema de escalonamento *pinwheel* foi inspirado pelas restrições na comunicação entre satélites e estações terrestres. As estações só conseguiam ler uma mensagem por *slot* de tempo, portanto, cada satélite s_i enviava uma mesma mensagem por d_i *slots* em seu canal de comunicação reservado. A estação precisava escolher um canal a ser lido a cada *slots* de tempo de forma que nenhuma mensagem fosse perdida. As sucessivas escolhas de canais criava uma sequência infinita de mensagens, em que qualquer subsequência de tamanho d_i teria que apresentar no mínimo uma mensagem do satélite s_i para que nenhuma mensagem fosse perdida.

A Figura 3 ilustra o estilo de comunicação descrito acima para um conjunto de satélites $S = \{1, 2, 3\}$. O conjunto $D = \{2, 4, 6\}$ determina a quantidade de *slots* consecutivos onde um satélite transmite suas mensagens. x_{ij} representa a j-ésima mensagem do satélite s_i. A escolha de canal realizada pela estação é representada pelos *slots* em cinza. Através da figura, podemos ver que no *slot* 1, a estação lê a primeira mensagem do satélite 1, no *slot* 2 a primeira mensagem do satélite 2, no *slot* 3 a segunda mensagem do satélite 1 e assim por diante. A sequência resultante dessas leituras é 1,2,1,3,1,2,1,3,... onde os valores indicam o canal lido naquele *slot*. Podemos notar que a restrição que diz respeito a existência de um símbolo x_i dentro de qualquer subsequência de d_i *slots* consecutivos foi respeitada e que seguindo essa sequência nenhuma mensagem será perdida.

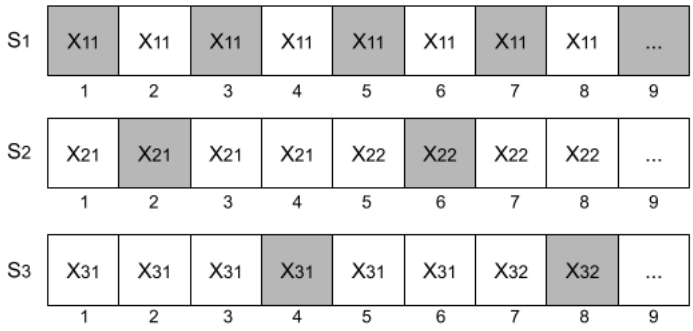

Figura 3: Mensagens transmitidas pelo conjunto de satélites $S = \{1, 2, 3\}$ conforme a estratégia de escalonamento *pinwheel*.

O problema *pinwheel* difere do problema de escalonamento de tarefas periódicas por tratar de processos esporádicos e, portanto, faz parte de uma classe diferente de problemas denominada de problemas de escalonamento de processos em tempo real.

Em [15] os processos são definidos por 2 parâmetros principais: o tempo de execução (t) e o período do processo (p). Esses processos solicitam serviço periodicamente apenas nos instantes de tempo np, onde $n \in \mathbb{N}$. A diferença entre os problemas surgem porque os satélites são modelados como processos esporádicos que podem solicitar serviços a qualquer momento, a única restrição é que duas mensagens distintas de um mesmo satélite estejam separadas por no mínimo p unidades de tempo consecutivas. Outra diferença é que, no *pinwheel*, uma mensagem não pode ser interrompida durante sua transmissão, um fato que inviabiliza a preempção entre processos.

Outra contribuição de [11] foi a definição do conceito de densidade de uma instância como $\sum_{i=1}^{n} \frac{1}{d_i}$ e a demonstração que instâncias com densidade maior que um não são escalonáveis.

2.4 Problema de Escalonamento com Distâncias Restritas

Nesta classe de problemas, um conceito bastante presente em vários sistemas de tempo real é introduzido. No modelo de escalonamento periódico, as diferentes execuções (*jobs*) de uma mesma tarefa eram tidos como independentes, mas podemos observar exatamente o contrário em vários sistemas. Imagine, por exemplo, um problema análogo à troca de óleo de um carro onde o período indicado para a troca seja de seis meses. Se a primeira troca ocorreu em janeiro, em um escalonamento periódico a segunda troca poderia ser agendada para dezembro já que a única preocupação deste tipo de escalonamento é que a tarefa seja realizada uma vez em cada fatia de 6 meses. Podemos observar que o mais indicado seria que a distância temporal entre cada troca fosse de no máximo 6 meses, com o propósito de evitar danos para o veículo. Essa limitação de distância temporal é denominada distâncias restritas e foi definida em [10]. Formalmente, esse problema pode ser definido da seguinte maneira:

Considere um conjunto $X = \{x_1, x_2, ..., x_n\}$ de tarefas que são executadas em um número infinito de vezes. Cada execução de uma tarefa é chamada de *job*, denotado por J_{ij} (job j da tarefa x_i). Cada tarefa x_i tem um tempo de execução t_i e uma restrição de distância r_i. A distância entre dois *jobs* consecutivos é igual a $f_{i(j+1)} - f_{ij} \leq r_i$, onde f_{ij} representa o instante de tempo no qual o *job* j da tarefa x_i é concluído. Assumindo que os *jobs* são preemptíveis e que o *job* $J_{i(j+1)}$ inicia sua execução após o job J_{ij} , o problema consiste encontrar um escalonamento de tarefas que atenda às restrições de distância.

Observe que o requisito de ocorrência de pelo menos um símbolo x_i em qualquer subsequência de d_i símbolos consecutivos presente no problema *Pinwheel*, equivale a dizer que, a distância temporal entre duas ocorrências consecutivas de um símbolo x_i deve ser menor ou igual a d_i. Desta forma, temos o problema *Pinwheel* como um caso especial da versão discreta do problema de escalonamento com substâncias restritas, se diferenciando somente por não permitir preempção e por limitar o tempo de execução das tarefas a unidades de tempo discreto.

[10] propuseram um algoritmo, denominado **Sr**, com capacidade de escalonar um conjunto de n tarefas com densidade menor ou igual a $n(2^{\frac{1}{n}} - 1)$.

2.5 Problema da Variabilidade do Tempo de Resposta

O *problema de variabilidade de tempo de resposta* (RTVP, do inglês *response time variability problem*) é um problema de otimização NP-difícil que pode ser observado sempre que eventos, tarefas, clientes ou produtos precisam ser sequênciados objetivando minimizar a variabilidade do tempo esperado entre aplicações consecutivas de recursos [8].

O conceito de dependência entre *jobs* de uma tarefa introduzido no problema de escalonamento com distâncias restritas é revisado no RTVP. Enquanto que as escolhas de distâncias restritas poderiam tornar um conjunto de tarefas inviáveis no primeiro, temos no segundo, apenas a obrigação de minimizar a variabilidade das distâncias temporais entre as ocorrências de execução de uma tarefa. Ao mesmo tempo em que contorna o problema mencionado, o RTVP conserva a ideia principal de uma certa uniformidade nas distâncias temporais de sucessivas alocações de recursos para um mesmo cliente, produto ou tarefa [16].

O RTVP foi definido formalmente por [8] da seguinte forma:

Dados n inteiros positivos $m_1 \leq ... \leq m_n$, defina $T = \sum_{i=1}^{n} m_i$ e a taxa $r_i = \frac{m_i}{T}$ para $i = 1, ..., n$. Considere uma sequência $S = s_1 s_2 ... s_T$ de tamanho T, onde x_i (tarefa, produto ou cliente) ocorre exatamente m_i vezes. A distância d entre duas ocorrências consecutivas de x_i é definida como o número de posições entre elas mais um, já que todos os elementos da sequência possuem o mesmo tamanho. Desse modo, se x_i ocorre m_i vezes, existem exatamente m_i distâncias $d_{i1}, ..., d_{im_i}$ para x_i, onde $d_i m_i$ é a distância da última

ocorrência de x_i de um ciclo para a primeira ocorrência de x_i do ciclo subsequente. O objetivo é minimizar a variabilidade do tempo de resposta total, definida como:

$$RTV = \sum_{i=1}^{n} \sum_{i \leq j \leq m_i} (d_{ij} - \overline{d_i})^2,$$

onde $\overline{d_i} = \frac{T}{m_i} = \frac{1}{r_i}$.

2.6 Problema do Broadcast de Dados

Ambientes de Comunicação Assimétrica foram definidos por [1] como ambientes onde a capacidade para comunicação na direção servidor-cliente é muito maior que a capacidade no sentido contrário. São listados dois motivos primários para a ocorrência de cenários que se enquadram como comunicações assimétricas: As limitações naturais de banda impostas pelos meios físicos de comunicação ou pela natureza dos padrões de fluxo das informações que transmitidas entre clientes e servidores. Redes telefônicas podem ser encaixadas no primeiro já que temos as estações base equipadas com poderosos transmissores enquanto que clientes possuem apenas pequenas antenas instaladas em seus aparelhos.

É natural a escolha de uma comunicação unidirecional denominada *broadcast* quando num contexto onde o número de clientes ultrapassa consideravelmente o número de servidores para tentar atenuar a incapacidade de atender as solicitações excessivas do grande número de clientes.

Em seu trabalho, [1] propuseram uma arquitetura de sistema, denominada *Broadcast Disk*, objetivando explorar as vantagens da comunicação *broadcast* em ambientes onde a unica forma viavel de comunicação é a assimétrica. Nesta arquitetura proposta, o servidor transmitiria dados, repetidamente, de forma circular como é mostrado na Figura 4 extraída de [1].

Desta forma o cliente poderia obter a informação desejada apenas ouvindo o canal de *broadcast* e esperando a chegada da informação para iniciar seu download. O nome da arquitetura vem diretamente do fato que as informações são transmitidas de forma circular lembrando a ideia de um disco. O número ideal de discos, seus tamanhos e a velocidade com que "giram" são todos parâmetros que podem ser ajustados afim de minimizar o tempo de espera do cliente pelos itens requisitados.

[4] definiram o problema formalmente da seguinte maneira. Seja $X = \{x_1, x_2, ..., x_n\}$ um conjunto de páginas acessadas pelos clientes, tal que a página x_i possui uma probabilidade de acesso c_i, para $i = 1, ..., n$. Um cliente que deseja acessar uma página x_i a partir do *slot* de tempo t tem que esperar $w + it = t' - t + 1$ *slots* de tempo, onde $t' \leq t$ é o próximo *slot* de tempo em que x_i será transmitido. O objetivo do problema é encontrar uma sequência infinita de transmissões de páginas que minimize

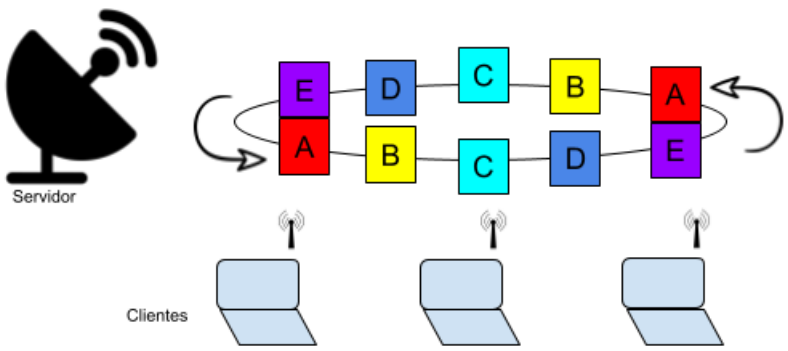

Figura 4: Transmissão de mensagens na arquitetura Broadcast Disks. Adaptada de Acharya (1995).

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{i=1}^{n} c_i w_{it}.$$

O problema do escalonamento de serviços de manutenção [2] pode ser modelado de forma similar ao problema do *broadcast* de dados. Neste problema, o custo de operação de um conjunto de máquinas aumenta à medida que o tempo vai decorrendo, portanto deve ser encontrado uma sequência de manutenções realizadas num horizonte infinito de tempos discretos que minimize o custo médio de operação. [4] definiram formalmente uma generalização para este problema onde mais de um serviço de manutenção possa ocorrer em um mesmo *slot* de tempo da seguinte forma:

Considere um escalonamento de serviços de manutenção $S = S_1, S_2, ...$ com n máquinas, onde $S_t \subseteq \{x_1, x_2, ..., x_n\}$ e $|S_t| \leq M$, número máximo de máquinas que podem ser escalonadas para manutenção em um mesmo *slot* de tempo. O escalonamento da máquina x_i para manutenção no *slot* t é denotado por $x_i \in S_t$, e o custo de manutenção no *slot* t é $\sum_{x_i \in S_t} a_i$. O custo de operação o_{it} da máquina x_i é $c_i(t - t')$, onde c_i é uma constante associada a máquina x_i e $t' \leq t$ é o último *slot* em que x_i foi escalonada para manutenção. Pretende-se, portanto, encontrar um escalonamento S que minimize

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} (\sum_{x_i \in S_t} a_i + \sum_{i=1}^{n} o_{it}).$$

A diferença principal entre os dois problemas definidos por [4] está na forma como os custos referente às distâncias temporais aumentam ou diminuem. No problema do *broadcast* de dados, o custo de espera por uma página x_i diminui ao longo do tempo até a transmissão consecutiva de x_i. Por outro lado, no problema do escalonamento de serviços de manutenção, o custo de operação da máquina x_i aumenta com o decorrer do tempo até a ocorrência do próximo serviço de manutenção de x_i [16].

2.7 Trabalho de Pessoa et al. [16]

Como mencionado na Seção 1, este trabalho discute o PSJP introduzido no trabalho de [16]. Para podermos apresentar as melhorias introduzidas ao modelo, é necessário

que se entenda as três principais contribuições propostas no trabalho em questão: a formulação matemática proposta para o PSJP, os cortes utilizados para reduzir o o espaço de solução do problema e, finalmente, as soluções propostas.

2.7.1 Formulação Matemática

A formulação proposta para o problema descrito na Seção **??** foi dividida em duas partes. A primeira apresentando os dados de uma instância do problema junto com as restrições que calculam as distâncias entre cópias de um mesmo símbolo contidas no mesmo ciclo. A segunda apresenta as restrições responsáveis pelo cálculo das distâncias entre cópias consecutivas de diferente ciclos.

Dados:

n : Número de símbolos.

X : Conjunto de símbolos $x_1, ..., x_n$.

M_i : Número máximo de cópias do símbolo $x_i \in X$.

K_i : Conjunto de índices das cópias $x_i \in X, i.e., K_i = \{1, ..., M_i\}$.

$TMAX$: Tamanho máximo das sequência viáveis.

c_i : Prioridade do símbolo $x_i \in X$.

H_{ik} : Conjunto de posições que podem ser ocupadas pela k-ésima cópia do símbolo $x_i \in X, i.e., H_{ik} = \{k, ..., TMAX\}$.

Variáveis:

y_{ikh} : 1 se a k-ésima cópia do símbolo $x_i \in X$ está na posição $h \in H_{ik}$; 0 caso contrário.

$d_{i,k,k+1}$: Distância entre a k-ésima e a (k + 1)-ésima cópias do símbolo $x_i \in X, k \in K_i \setminus \{M_i\}$.

D_i : Maior distância entre duas cópias consecutivas do símbolo $x_i \in X$.

P : Maior produto $D_i c_i, \ \forall \ x_i \in X$.

p_{ik} : Posição da k-ésima cópia do símbolo $x_i \in X$.

Modelo:

$$\text{Minimize } P \tag{1}$$

$$\text{s.t.} \quad P \geq D_i c_i, \qquad \forall i = 1, ..., n; \tag{2}$$

$$D_i \geq d_{i,k,k+1}, \qquad \forall i = 1, ..., n, \forall k \in K_i \setminus \{M_i\}; \tag{3}$$

$$\sum_{i:x_i \in X} \sum_{\substack{k \in K_i \\ h \in H_{ik}}} y_{ikh} \leq 1, \qquad \forall h \in \{1, ..., TMAX\}; \tag{4}$$

$$\sum_{h \in H_{ik}} y_{ikh} \leq 1, \qquad \forall i = 1, ..., n, \forall k \in K_i; \tag{5}$$

$$\sum_{k \in K_i} \sum_{h \in H_{ik}} y_{ikh} \geq 1, \qquad \forall i = 1, ..., n; \tag{6}$$

$$p_{ik} = \sum_{h \in H_{ik}} h y_{ikh}, \qquad \forall i = 1, ..., n, \forall k \in K_i; \tag{7}$$

$$\sum_{h \in H_{ik}} y_{ikh} \geq \sum_{h \in H_{i,k+1}} y_{i,k+1,h}, \qquad \forall i = 1, ..., n, \forall k \in K_i \setminus \{M_i\}; \tag{8}$$

$$\sum_{i:x_i \in X} \sum_{\substack{k \in K_i \\ h \in H_{ik}}} y_{ikh} \geq \sum_{i:x_i \in X} \sum_{\substack{k \in K_i \\ h \in H_{ik}}} y_{i,k,h+1}, \qquad \forall h \in \{1, ..., TMAX - 1\}; \tag{9}$$

$$p_{i,k+1} \geq p_{ik} - \big(1 - \sum_{h \in H_{i,k+1}} y_{i,k+1,h}\big) TMAX, \qquad \forall i = 1, ..., n, \forall k \in K_i \setminus \{M_i\}; \tag{10}$$

$$d_{i,k,k+1} \geq p_{i,k+1} - p_{ik}, \qquad \forall i = 1, ..., n, \forall k \in K_i \setminus \{M_i\}; \tag{11}$$

$$\sum_{i:x_i \in X} \sum_{k \in K_i} \sum_{h \in H_{ik}} y_{ikh} \leq TMAX; \tag{12}$$

$$P, D_i, p_{ik} \geq 0, \qquad \forall i = 1, ..., n, \forall k \in K_i; \tag{13}$$

$$d_{i,k,k+1} \geq 0, \qquad \forall i = 1, ..., n, \forall k \in K_i \setminus \{M_i\}; \tag{14}$$

$$y_{ikh} \in \{0, 1\}, \qquad \forall i = 1, ..., n, \forall k \in K_i, \forall h \in H_{ik}; \tag{15}$$

A função objetivo (1) tem a finalidade de minimizar o máximo produto entre a maior distância de duas cópias consecutivas (D_i) e sua prioridade (c_i), considerando os produtos para cada símbolo x_i, os quais são definidos nas restrições (2). As restrições (3) garantem que D_i assuma o valor da maior distância entre duas cópias consecutivas de um mesmo símbolo, levando em consideração apenas as cópias pertencentes a um mesmo ciclo. As restrições (4), (5) e (6) asseguram que cada posição seja ocupada por apenas uma cópia de um símbolo, que cada cópia seja alocada uma única vez, e que cada símbolo seja alocado pelo menos uma vez na sequência. As restrições (7), por sua vez determinam as posições das cópias de um símbolo. As restrições (8) e (9) estabelecem que a alocação

da $(k+1)$-ésima cópia de um símbolo depende da alocação da k-ésima cópia do mesmo símbolo e que a posição $(h+1)$ na sequência só pode ser ocupada após a alocação de uma cópia na posição h. As restrições (10) garantem a alocação ordenada das cópias de um símbolo em relação a k, ou seja, $p_{i,k+1} \geq p_{ik}$. As demais restrições asseguram a validade da inequação mesmo quando a cópia $k+1$ não estiver alocada, pois, nesse caso, $p_{i,k+1} = 0$ e $p_{ik} - TMAX \leq 0$. As restrições (11) vinculam as variáveis que representam as distâncias entre cópias consecutivas com as variáveis que definem suas posições. Já a restrição (12) determina o tamanho máximo de uma sequência válida. Por fim, as restrições (13) - (15) especificam os domínios das variáveis utilizadas no modelo.

Devido à natureza cíclica do PSJP, ilustrada na Figura 5, o modelo descrito acima precisa ser estendido com restrições que lidam especificamente com o cálculo das distâncias entre a última cópia de um símbolo em um ciclo e a primeira cópia deste símbolo no próximo ciclo. Seguem as variáveis e restrições utilizadas por Pessoa et al. [16] para representar esse aspecto do problema:

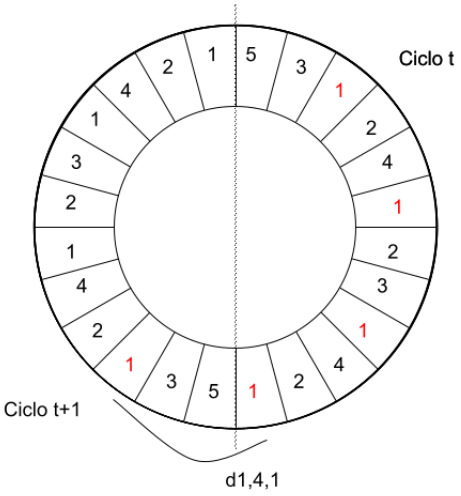

Figura 5: Sequência ótima representada de forma cíclica para instância do problema com $X = \{1,2,3,4,5\}$, e prioridades $c_1 = 9, c_2 = 6, c_3 = 5, c_4 = 4, c_5 = 1$, número máximo de cópias $M_i = 5, \forall x_i \in X$ e $TMAX = 15$. Adaptada de Pessoa (2018).

W_i : Distância entre a última cópia alocada de um ciclo e a primeira do próximo ciclo do símbolo $x_i \in X$.

U_i : Tamanho da sequência mais a distância do início da sequência para a primeira cópia do símbolo $x_i \in X$.

R_i : Distância entre o início da sequência e a última cópia alocada do símbolo $x_i \in X$.

α_{ik}, w_{ik} : Variáveis auxiliares para encontrar a última cópia alocada do símbolo $x_i \in X$.

$$D_i \geq W_i, \qquad \forall i = 1, ..., n; \tag{16}$$

$$W_i = U_i - R_i, \qquad \forall i = 1, ..., n; \tag{17}$$

$$U_i = \sum_{j:x_j \in X} \sum_{k \in K_j} \sum_{h \in H_{jk}} y_{jkh} + p_{i1}, \qquad \forall i = 1, ..., n; \tag{18}$$

$$R_i = \sum_{k \in k_i} \alpha_{ik}, \qquad \forall i = 1, ..., n; \tag{19}$$

$$\sum_{k \in k_i} w_{ik} = 1, \qquad \forall i = 1, ..., n; \tag{20}$$

$$\alpha_{ik} \leq 1 - w_{ik} + p_{ik}, \qquad \forall i = 1, ..., n, \forall k \in K_i; \tag{21}$$

$$\alpha_{ik} \leq w_{ik} TMAX, \qquad \forall i = 1, ..., n, \forall k \in K_i; \tag{22}$$

$$W_i, U_i, R_i, \alpha_{ik} \geq 0, \qquad \forall i = 1, ..., n, \forall k \in K_i; \tag{23}$$

$$w_{ik} \in \{0, 1\}, \qquad \forall i = 1, ..., n, \forall k \in K_i; \tag{24}$$

As restrições (16) garantem que as distâncias W_i, que representam as distâncias entre a última cópia de um símbolo num ciclo e a primeira cópia desse símbolo no ciclo seguinte, sejam incluídas na minimização da função objetivo (1). As restrições (17) mostram como W_i é calculado pela diferença entre U_i e R_i, que são definidos respectivamente pelas restrições (18) e (19) - (22). Em (18), U_i é definida como a soma do tamanho da sequência e a posição da primeira cópia do símbolo x_i. R_i representa a distância entre o início da sequência e a última cópia de x_i e, já que o número de ocorrências de um símbolo é variável, precisamos das restrições (19) - (22) para determiná-lo. As restrições (20) garantem que a variável w_{ik} será igual a 1 para uma única cópia k de um símbolo i. As restrições (21) e (22), por sua vez, fazem $\alpha_{ik} \leq 0$, caso $w_{ik} = 0$, ou $\alpha_{ik} \leq p_{ik}$, caso $w_{ik} = 1$. Uma vez que a função objetivo (1) minimiza P e, por consequência, W_i, temos que o modelo maximizará R_i, que é calculado em (19). R_i assume o maior valor possível quando o w_{ik} escolhido pelo modelo representa a última cópia k alocada de um símbolo x_i, portanto, com o maior valor p_{ik}. A Figura 6 exibe os valores de W_i, U_i e R_i para uma sequência qualquer.

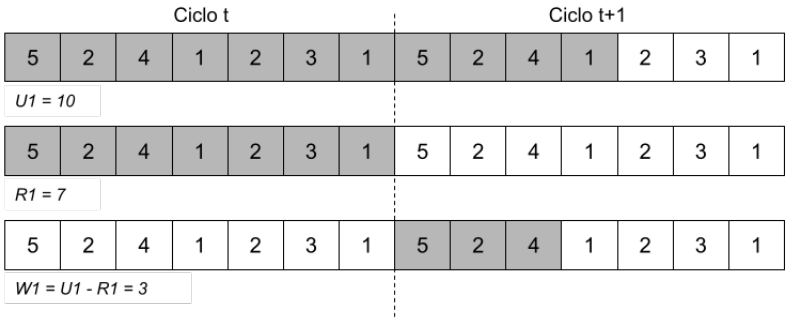

Figura 6: Demonstração dos valores W_i, U_i e R_i para uma sequência qualquer. Reprodução com permissão de Pessoa et al. [16].

Ao analisar o modelo matemático, podemos ver que diversas variáveis e restrições

estão atreladas à quantidade máxima de cópias para cada símbolo(M_i). Pessoal et al. [16] desenvolveram um modelo de programação matemática que calcula limites superiores para o número máximo de cópias para cada símbolo, já que esse é um parâmetro que tem um grande impacto na performance do modelo. Na próxima seção, esse modelo e várias outras otimizações desenvolvidas pelos autores são apresentados.

2.7.2 Cortes

Além da formulação matemática, Pessoa et al. [16] propuseram vários cortes válidos que reduzem o espaço de soluções do problema, um modelo de programação matemática capaz de encontrar limites superiores para o número máximo de cópias de cada símbolo e limites inferiores para o problema, e uma conjectura que diminui consideravelmente o tempo de resolução que, embora intuitiva, não foi provada matematicamente. Seguem as preposições extraídas de [16]:

Proposição 1. *Seja x_s um dos símbolos de uma instância do PSJP. Existe uma solução ótima, onde o símbolo x_s é alocado na primeira posição da sequência solução, i.e., $y_{s11} = 1$.*

Proposição 2. *Considere uma instância do PSJP com n símbolos tal que $c_i \geq c_{i+1}$, para $i = 1, ..., n-1$. Existe uma solução ótima para tal instância onde $D_i \leq D_{i+1}$, para $i = 1, ..., n-1$.*

Proposição 3. *Considere uma instância do PSJP com $n \geq 2$ símbolos. Existe uma sequência ótima na qual não há duas cópias de um mesmo símbolo localizadas em posições consecutivas. Logo, as desigualdades*

$$y_{ikh} + y_{i,k+1,h+1} \leq 1, \qquad \forall i = 1, ..., n, \forall k \in K \setminus \{M_i\}, \forall h \in (H_{ik} \cap H_{i,k+1}), \tag{25}$$

não eliminam integralmente o conjunto de solução ótimas do PSJP.

Para evitar uma explosão no número de variáveis e restrições, os autores desenvolveram um método para encontrar limitantes superiores para o número máximo de cópias M_i para cada símbolo $x_i \in X$. Um valor trivial para M_i seria o tamanho máximo das sequências viáveis ($TMAX$). Uma alternativa ligeiramente melhor, baseada na Proposição 1 e no fato de que cada símbolo deve estar presente pelo menos uma vez na solução, seria $M_i = \left\lceil \frac{TMAX}{2} \right\rceil$ para $i = 1, ..., n$. Contudo, ambas não suficientes para fortalecer o modelo de forma adequada. A solução encontrada pelos autores foi calcular limites superiores através de um modelo de Otimização Não-Linear descrito na proposição abaixo:

Proposição 4. *Seja $X = \{x_1, ..., x_n\}$ um conjunto de símbolos tal que $c_1 \geq ... \geq c_n$. O número máximo de cópias M_i^*, para cada $x_i^* \in X$, pode ser feito igual a ρ_i^*, onde*

$$\rho_i^* = \max m_i^* \tag{26}$$

$$\text{s.t.} \quad P \geq D_i c_i, \qquad \forall i = 1, ..., n; \tag{27}$$

$$T \geq m_i D_i, \qquad \forall i = 1, ..., n; \tag{28}$$

$$T = \sum_{x_i \in X} m_i; \tag{29}$$

$$m_i \geq 1, \qquad \forall i = 1, ..., n; \tag{30}$$

$$T \leq TMAX; \tag{31}$$

$$P \leq \overline{f}; \tag{32}$$

$$P, D_i, T, m_i \geq 0, \qquad \forall i = 1, ..., n; \tag{33}$$

A função objetivo (26) deste modelo tenta maximizar o número de cópias do símbolo $x_i^* \in X$. As restrições (27) veem diretamente do modelo PSPJ e fazem com que P seja maior do que todos os produtos entre a maior distâncias e a prioridade de um símbolo. As restrições (28) tem origem de um outro fato encontrado por Pessoa que afirma que o tamanho da sequência (T) encontrada como solução para o PSPJ é maior ou igual ao número de cópias de um símbolo $x_i \in X$ e a maior distância entre cópias consecutivas deste símbolo(D_i). As restrições (29) definem T como a soma de todas as cópias de símbolos utilizadas nesta sequência. As restrições (30) estabelecem que pelo menos uma cópia de cada símbolo esteja presente na solução. As restrições (31) e (32) impõem, respectivamente, limites superiores para o tamanho máximo das sequências possíveis T e o valor do maior produto P. Pessoa et al. [16] nomeiam esse modelo de MNC (do inglês, *Maximum Number of Copies*) e demonstram que, ao substituir (26) por (1), o modelo torna-se uma relaxação para o PSJP.

Por fim, os autores introduzem um conjunto de cortes que fortalecem a formulação de forma relevante, baseado em uma conjectura não provada matematicamente:

Conjectura 1. *Considere uma instância do PSJP com n símbolos, tal que* $c_i \geq c_{i+1}$*, para* $i = 1, ..., n-1$*. Existe uma solução ótima para essa instância em que* $m_i \geq m_{i+1}$*, para* $i = 1, ..., n1$*, onde* m_i *é o número de cópias utilizadas do símbolo* x_i*, de modo que*

$$\sum_{h \in H_{i+1,k}} y_{i+1,k,h} \leq \sum_{h \in H_{ik}} y_{ikh}, \qquad \forall i = 1, ..., n-1, \forall k \in K_{i+1}. \tag{34}$$

2.7.3 Método Solução

O Algoritmo 1 desenvolvido por Pessoa et al. [16] consiste em um método iterativo que integra os modelos descritos anteriormente. Os parâmetros de entrada são os mesmos

da formulação do PSJP, com a adição de alguns valores que administram as repetições dos passos do algoritmo. São eles: X, o conjunto de símbolos; $c : X \to \mathbb{Z}^+$, a função prioridade; $TMAX$, representando o tamanho máximo de uma sequência viável; Δ , um número real no intervalo [0,1] representando o passo no valor do limite inferior a cada execução; it_{max}, representando o número máximo de iterações realizadas pelo algoritmo.

O conceito principal deste algoritmo é utilizar de um limitante inferior (LI) obtido através do modelo NMC (linha 2) e, a partir dele, estimar um limitante superior (linha 3). Assim, o número de variáveis e restrições indexadas por M_i, para $i = 1, ..., n$ é reduzido. Com esses valores estimados, o laço das linhas 4-12 é executado até it_{max} vezes. No laço interno das linhas 5-7, o modelo NMC é executado para cada símbolo $x_i \in X$, utilizando o valor $\overline{f}$ na desigualdade (32). Após encontrar os valores M_i para o atual limitante superior, o modelo PSJP é resolvido na linha 8 tomando tais valores como o número máximo de cópias K_i, para $i_1, ..., n$ e utilizando o corte $P \leq \overline{f}$. Se este corte não inviabilizar o espaço de solução, o modelo PSJP é resolvido e o algoritmo se encerra. Caso contrário, o limitante superior é incrementado por um fator Δ na linha 12 e a próxima iteração é executada.

Note que cada incremento de $\overline{f}$ traz consigo uma gama de variáveis e restrições. Portanto, o valor Δ deve ser escolhido cuidadosamente. Um valor pequeno de Δ acarreta num número maior de iterações e um valor excessivo pode acabar incrementando muito os valores M_i, deixando o modelo mais caro a cada iteração. Os resultados desse algoritmo serão discutidos em comparação com os resultados obtidos pelas melhorias propostas pelo presente trabalho na Seção 4.

Algoritmo 1: Método iterativo baseado em PLIM para o PSJP.

```
1  procedimento MétodoIterativo(X, c, TMAX, Δ, it_max)
2  LI ← valor de (1) sujeito a (27) - (33)
3  f̄ ← LI + LI × Δ
4  para it = 1 até it_max faça
5  |  para i = 1 até n faça
6  |  |  M_i ← resultado de NMC_i*
7  |  fim
8  |  resolva o PSJP com o corte P ≤ f̄
9  |  se a solução ótima foi encontrada então
10 |  |  retorna
11 |  fim
12 |  f̄ ← f̄ + LI * Δ
13 |  it ← it + 1
14 fim
```

2.8 Trabalho de Sinnl [19]

Em [19], os autor apresenta um algoritmo exato iterativo como solução para o PSJP. O algoritmo proposto baseia-se na execução iterativa de um modelo de programação linear inteira mista que fixa o tamanho da sequência em cada iteração. A seguir serão apresentados o algoritmo, o modelo em questão, bem como algumas proposições interessantes apresentadas pelo autor.

Algoritmo 2: Algoritmo Iterativo de [19]

```
1 z* ← ∞
2 para Σ_{i∈N} M_i ≤ T ≤ TMAX faça
3 |  z‘ ← Valor ótimo para o PSJP com o tamanho da sequência fixado em T
4 |  se z‘ ≤ z* então
5 |  |  z* ← z‘
6 |  fim
7 fim
```

O Algoritmo 2 itera sobre todas os possíveis tamanhos T que uma sequência que resolve o PSJP possa tomar e utiliza o modelo matemático 2.8.1 que será apresentando a seguir para encontrar a sequência ótima para esse dado tamanho. Ao considerar um tamanho fixo, a modelagem do problema pode ser simplificada permitindo que o modelo contenha um número menor de variáveis e restrições [19]. O autor também menciona que existem outros problemas, cuja reformulação da modelagem com o intuito de fixar uma certa dimensão do problema, somada à exploração dessa dimensão através de um método iterativo, pode levar as soluções mais efetivas que as abordagens que tentam resolver o problema de forma direta, citando como exemplos [6], [7] e [18].

2.8.1 Modelo PLIM para um T fixo

Seja x_i^t, $\forall i \in N, \forall t \in 1, \ldots, T$ uma variável binária denotando que uma cópia do símbolo i foi posicionada em t na sequência. Considere as variáveis $s_i^{t,t‘}$ e $p_i^{t,t‘}$, $\forall i \in N, 1 \leq t, t‘ \leq T$. A primeira indica que se um símbolo i for colocado na posição t da sequência, a próxima ocorrência desse símbolo será na posição $t‘$ levando em consideração que a sequência é circular. A segunda, por sua vez, indica que, se um símbolo i for colocado na posição t da sequência, a última ocorrência desse símbolo antes de t foi na posição $t‘$ tendo em vista que a sequência é circular. A função auxiliar que calcula a distância entre duas posições t e $t‘$, assim como a formulação matemática seguem abaixo:

$$dist(t, t‘) = \begin{cases} t \text{ - } t‘, & \text{se} \quad t > t‘ \\ T + t \text{ - } t‘, & \text{caso contrario.} \end{cases}$$

Modelo:

$$\text{Minimize } \theta \tag{35}$$

$$\text{s.t.} \quad \sum_{1 \le t \le T} x_i^t \ge f_i, \qquad \forall i \in N; \tag{36}$$

$$\sum_{i \in N} x_i^t = 1, \qquad 1 \le t \le T; \tag{37}$$

$$x_i^t - \sum_{1 \le t' \le T} p_i^{t,t`} = 0, \qquad \forall i \in N, 1 \le t \le T; \tag{38}$$

$$x_i^t - \sum_{1 \le t' \le T} s_i^{t,t`} = 0, \qquad \forall i \in N, 1 \le t \le T; \tag{39}$$

$$s_i^{t,t`} - p_i^{t,t`} = 0, \qquad \forall i \in N, 1 \le t, t' \le T; \tag{40}$$

$$\theta - \sum_{i \in N} \sum_{1 \le t' \le T} C_i \text{ dist(t,t`) } p_i^{t,t`} \ge 0, \qquad 1 \le t \le T; \tag{41}$$

$$\theta - \sum_{i \in N} \sum_{1 \le t' \le T} C_i \text{dist(t,t`) } s_i^{t,t`} \ge 0, \qquad 1 \le t \le T; \tag{42}$$

$$x_i^t \in \{0,1\}, \qquad \forall i \in N, 1 \le t \le T; \tag{43}$$

$$p_i^{t,t`} \in \{0,1\}, \qquad \forall i \in N, 1 \le t, t' \le T; \tag{44}$$

$$s_i^{t,t`} \in \{0,1\}, \qquad \forall i \in N, 1 \le t, t' \le T; \tag{45}$$

As restrições (36) garantem que cada símbolo i, $\forall i \in N$, apareça pelo menos f_i vezes na sequência, onde f_i representa a frequência miníma do símbolo i. As restrições (37) garantem que cada posição da sequência só receba um símbolo. As restrições (38) e (39) asseguram que para cada símbolo posicionado na sequência em uma dada posição, variáveis que indicam as posições das ocorrência anterior e futura desse símbolo são selecionadas. As restrições (40) relacionam as variáveis s e p. De acordo com o autor, as restrições (38), (39) e (40) estabelecem que para cada símbolo i colocado numa posição t, alguma outra posição $t`$, onde o símbolo i também está posicionado, seja definida como uma posição "predecessora"e "sucessora". Porém, por si só, essas restrições não garantem que as verdadeiras posições sucessoras e predecessoras para uma dada posição sejam selecionadas. O modelo não precisa que essa condição seja verdade para todas as variáveis s e p, mas somente para as combinações de posições que criam o maior produto entre distâncias e o peso (prioridade) do símbolo para a sequência representada pelas variáveis x. Como o objetivo do PSJP é a minimização do produto entre distâncias e a prioridade do símbolo, essas restrições são o suficiente para resolver o problema. As restrições (41) e (42), em combinação com a função objetivo, garantem que o valor do produto entre as

distâncias e o peso do símbolo seja calculado correntemente.

2.8.2 Resultados

O método iterativo de Sinnl [19] foi capaz de resolver 404 das 440 instâncias da literatura até a otimalidade, a maioria delas sendo resolvidas dentro de 5 minutos. Os resultados apresentados superaram significativamente os obtidos por Pessoa et al. [16], tanto em relação ao número de instâncias resolvidas até a otimalidade, quanto ao tempo de execução.

2.9 Trabalho de Rocha et al. [17]

Rocha et al. [17] propuseram o primeiro algoritmo baseado em metaheurística para resolver o PSJP. Ao utilizar uma implementação eficiente da heurística *Variable Neighborhood Search*, baseada em uma estrutura de dados especializada, o trabalho obteve resultados semelhantes ao método de Sinnl [19] no quesito qualidade das soluções, porém em um tempo computacional bem menor.

O Algoritmo 3 descreve a metaheurística proposta. Assim como [19], o algoritmo divide o PSJP numa série de subproblemas de tamanhos fixos $T = n, \ldots, TMAX$. O algoritmo inicia a partir de uma solução trivial do PSJP contendo uma única cópia de cada símbolo $x_i \in X$, que é denominada como a melhor solução atual S^*. Daí, seguimos para o procedimento na linha 4 que aumenta o tamanho da sequência ao adicionar uma nova cópia de algum símbolo de X na melhor posição possível. O próximo passo (executado na linha 5) é o cerne do algoritmo e consiste na execução de um VNS que resolve o subproblema atual do PSJP para uma sequência de tamanho $|S|$. Caso a solução encontrada na linha 6 seja melhor do que a melhor solução encontrada até o momento, S^* é atualizada na linha 7. O loop contendo as linhas 3-9 se repete até que o subproblema com tamanho TMAX for otimizado. A melhor solução S^* encontrada é retornada após o fim do loop na linha 10.

A heurística VNS utilizada neste algoritmo foi desenvolvida seguindo um preceito denominado LIMA (do inglês, *Less Is More Aproach*), que defende que o uso mínimo de ingredientes algoritmos (numero de buscas locais, redução de dados, decomposições, etc.) como uma dimensão para a avaliação de uma meta-heurística [17]. Para se adequar a essas restrições de design, apenas duas vizinhanças foram utilizadas para explorar o espaço de solução do PSJP. Essas duas vizinhanças são descritas como:

- Flip: Uma cópia de um símbolo é substituída por uma cópia de outro símbolo;
- Shift-one: Uma cópia de um símbolo é deslocada para o próxima posição da sequência.

Algoritmo 3: Algoritmo Iterativo heurístico de [17]

```
1  procedimento Iterativo
2     S* ← solução trivial com |S| = n
3     enquanto |S| ≤ TMAX faça
4     |  S ← IncreaseSequence(S)
5     |  S ← VNS(S)
6     |  se S ≤ S* então
7     |  |  S* ← S
8     |  fim
9  fim
10    retorna S*
11 fim do procedimento
```

Em adição às duas vizinhanças bastante simples e eficientes, os autores apresentaram um conjunto de estruturas de dados que reduzem sobremaneira o custo computacional de avaliação das soluções. Considerando que uma solução para o PSJP é representada por um arranjo unidimensional $S = (s_1, s_2, \ldots, s_T)$, de tamanho T, contendo cópias de símbolos pertencentes ao conjunto de símbolos X, as estruturas de dados propostas podem ser descritas como segue:

- $\Gamma^+[x_i, j]$: Posição da primeira cópia do símbolo x_i que aparece após a posição j;
- $\Gamma^-[x_i, j]$: Posição da primeira cópia do símbolo x_i que aparece antes da posição j;
- d(j) : Distância entre a cópia do símbolo s_j presente na posição j e a próxima cópia de s_j em S, ou seja, d(j) = $\Gamma^+[s_j, j] - j$, se $\Gamma^+[s_j, j] \geq j$; caso contrario, d(j) = $\Gamma^+[s_j, j] - j + T$;
- $D(x_i)$: Distancia máxima encontrada entre dois pares consecutivos do símbolo $x_i \in X$ presentes em S;
- $p(j)$: Produto $d(j) * c(s_j)$ associado com a cópia do símbolo s_j presente na posição j;
- $m(x_i)$: Número de cópias do símbolo $x_i \in X$ presentes em S;
- cost(s) : Custo da sequência S, ou seja, $cost(S) = \max\{D(x_i)|x_i \in X\}$

Outro conceito relevante apresentado pelo trabalho é denominado de "intervalo dominante", que corresponde ao intervalo de posições na sequência que determinam o máximo produto. Vários intervalos dominantes podem existir em uma sequência, podendo ser associados a diferente símbolos [17]. Um dos recursos mais eficientes do algoritmo apresentado é que ele concentra as buscas locais para melhorar a solução apenas nos

intervalos dominantes de sequência S. Uma vez que a função objetivo tenta minimizar o maior produto $D_i * c_i$, $i \in X$, apenas os intervalos dominantes precisam ser avaliados e atualizados durante o algoritmo [17].

Após cada iteração do Algoritmo 3, o tamanho da sequência S deve ser aumentado para que o VNS possa explorar o espaço de soluções de um novo subproblema do PSJP [17]. O procedimento *IncreaseSequence* procura pela melhor posição do intervalo dominante de S onde pode ser inserida uma cópia do símbolo $x_i \in X$, em que $D(x_i) = cost(S)$, assim aumentando a frequência do símbolo que está determinando o valor objetivo da sequência S.

Para demostrar a efetividade da heurística, o artigo compara seus resultados para as 440 instâncias disponibilizadas por [16] com os resultados obtidos pelo método do estado da arte definido por [19]. A heurística iterativa é tão efetiva quanto o método de [19] em relação ao número de instâncias resolvidas até a otimalidade. Ela foi capaz de resolver 413 instâncias, enquanto que o método de [19] resolveu 408. A otimalidade das soluções retornada pelo método heurístico foi comprovada para 359 das 413 Instâncias resolvidas. É possível que outras soluções retornadas pelo algoritmo heurístico iterativo sejam ótimas, contudo não há prova de sua otimalidade. Por fim, a razão do tempo computacional médio gasto pelo método exato de [19] e o tempo gasto pelo método heurístico é por volta de 40.

3 METODOLOGIA

As principais contribuições deste trabalho foram desenvolvidas com o objetivo de suprir algumas deficiências da formulação matemática proposta em [16] e aproveitar avanços recentes da literatura para o PSJP. Desse modo, nas próximas seções são apresentados um conjunto de cortes que visam fortalecer a formulação mencionada e um método iterativo que resolve o PSJP para sequências sequências de tamanho fixo.

3.1 Distância miníma entre cópias

Um dos pontos mais frágeis da formulação apresentada na Seção 2.7.1 reside na ausência de uma restrição de distância mínima entre as cópias de um determinado símbolo. Como resultado, as variáveis do modelo que correspondem às posições das cópias dos símbolos recebem valores fracionários muitos próximos uns dos outros nas soluções das relaxações lineares do *branch-and-bound*, fazendo com que as distâncias entre cópias e, por consequência, o valor objetivo, se aproximem de zero e evoluam lentamente durante a exploração da árvore. O corte a seguir introduz uma distância mínima entre as cópias intercaladas de um símbolo

Proposição 3.1. *Considere uma solução ótima S^* para uma instância do PSJP com n símbolos. Sejam p_{ik} e $p_{i,k+2}$ as posições de duas cópias de um símbolo x_i, com pelo menos três cópias em S^*, e D_i a maior distância entre suas cópias consecutivas. Existe uma solução ótima para tal instância em que*

$$p_{i,k+2} \geq p_{ik} + D_i + 2, \qquad i = 1, ..., n, k = 1, ..., m_i - 2, \tag{46}$$

onde m_i é o número de cópias de x_i.

Demonstração. Sem perda de generalidade, assuma que o símbolo x_i possui quatro cópias em S^*, distribuídas conforme a Figura 7(a). Uma vez que $D_i = 4$, pode-se eliminar a segunda cópia do símbolo x_i (vide 7(b)) sem que haja impacto em D_i e, portanto, no valor da função objetivo. Note que, para não afetar D_i, a distância entre as cópias k e $k + 2$ deve ser no máximo $D_i + 1$. Assim, ao proceder com a eliminação de todas as cópias dos símbolos em S^* que satisfazem essa restrição, obtém-se uma solução S', também ótima, em que $p_{i,k+2} \geq p_{ik} + D_i + 2, i = 1, ..., n, k = 1, ..., m_i - 2$. □

Como D_i é uma variável do problema, seu valor tende a aproximar-se de zero à medida que as cópias consecutivas se aproximam nas relaxações lineares. Para que o corte produza o efeito desejado, é fundamental substituir D_i por uma constante que funcione

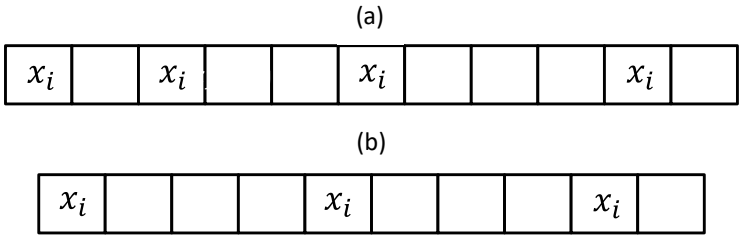

Figura 7: Distância mínima entre as cópias k e $k+2$.

como um limitante inferior para essa variável. Para determinar esse limitante, utiliza-se o modelo MNC, substituindo a função objetivo (26) por

$$D_{min(i)} = \min D_i. \tag{47}$$

Além disso, para fortalecer o modelo MNC, foi adicionada a desigualdade

$$D_i \leq D_{i+1}, \tag{48}$$

que é um corte proposto por [16] para o PSJP, e proposto um novo corte descrito na proposição abaixo.

Proposição 3.2. *Considere o modelo MNC modificado por meio da substituição de (26) por (3.3) e da adição de (48), e uma instância com n símbolos tal que $c_i \geq c_{i+1}$, para $i = 1, \ldots, n-1$. Existe uma solução ótima para essa instância tal que $m_i \geq m_{i+1}$, onde m_i é o número de cópias do símbolo x_i.*

Demonstração. Seja S^* uma solução ótima para a instância dada, com valor objetivo f^*. Suponha que S^* possua dois símbolos, x_i e x_j, onde $i < j$ e $m_j > m_i$. Ao trocar os valores de m_i e m_j, e vice-versa, não há alteração no número total de cópias dos símbolos T nem no produto $c_i * D_i$, $i = 1, \ldots, n-1$. Assim, as restrições (27), (29), (30), (31) e (32) continuam sendo satisfeitas. Além disso, como $D_i \leq D_j$ (3.3), as restrições (28) também permanecem válidas. Portanto, permutando os valores de m_i e m_j, e vice-versa, obtemos uma solução viável S', cujo valor objetivo também é f^*. Isso é válido para todos os cenários, inclusive nos casos em que $x_k \neq x_i$ e x_j, $x_k = x_i$, e $x_k = x_j$, onde $x_k \in X$ e $c_k D_k$ corresponde ao maior produto considerando todos os símbolos. □

3.2 Distância máxima entre cópias

As distâncias máximas entre cópias podem ser calculadas a partir do modelo MNC, bastando modificar a função objetivo pra maximizar D_i. Contudo, existe uma maneira mais eficiente de se obter um limitante para as distâncias máximas entre cópias. Seja $\overline{f}$ um limitante superior para uma instância do PSJP,

$$D_{max(i)} = \begin{cases} \lfloor \frac{\overline{f}}{c_i} \rfloor, & \text{if} \quad \lfloor \frac{\overline{f}}{c_i} \rfloor \leq TMAX, \\ TMAX, & \text{caso contrário.} \end{cases} \qquad \forall x_i \in X \tag{49}$$

Caso as sequências tenham um tamanho fixo, como é o caso das sequências abordadas pelo método iterativo apresentado mais adiante, substitui-se $TMAX$ pelo tamanho da sequência.

3.3 Número mínimo de cópias

Determinar o número mínimo de cópias de cada símbolo é essencial para o modelo, pois, além de ajudar a definir o intervalo de valores que uma variável-chave do modelo pode assumir, permite que o modelo fixe antecipadamente uma quantidade mínima de cópias a serem alocadas nas sequências. Da mesma forma que a distância mínima entre as cópias, esse valor pode ser obtido utilizando o modelo MNC, substituindo a função objetivo (26) por

$$\rho_i = \min m_i.$$

Nesse caso, o corte descrito na Proposição 3.2 não pode ser utilizado, já que não é um corte válido para o modelo.

Caso as sequências possuam tamanho fixo, pode-se utilizar corte proposto em [19] como alternativa para o MNC. A abordagem proposta é descrita a seguir. Seja

$$k_i^*(S,t) = \min\{k_i : c(x_i)\lceil t/k_i \rceil < f(S); k_i \in \mathbb{N}\} \tag{50}$$

o número mínimo de cópias $x_i \in X$ necessárias para produzir uma solução de custo menor que $f(S)$ para $|S| = t$. O valor de $k_i^*(S,t)$ é obtido ao se distribuir de modo uniforme as cópias de x_i em intervalos de $\lceil t/k_i \rceil$ ao longo da sequência S. Assim, $k_i^*(S,t)$ oferece uma alternativa computacionalmente mais barata para encontrar um limitante inferior para o número de cópias de cada símbolo $x_i \in X$.

3.4 Número máximo de cópias

Ao combinar a distância mínima entre cópias intercaladas, discutida na Seção 3.1, com a distância mínima entre cópias consecutivas ($d_{i,k,k+1} \geq 2$), proposta por [16], é possível definir um limite máximo para o número de cópias de cada símbolo, evitando assim a necessidade de resolver o modelo MNC para esse propósito. Considerando $D_{min(i)} = 4$ e $TMAX = 12$, a Figura 8 ilustra o número máximo de cópias M_i de um

símbolo x_i, conforme as restrições mencionadas. Por exemplo, a cada $D_{min(i)+2}$ posições, o símbolo x_i pode ter no máximo duas cópias. Formalmente,

$$M_i = 2\lambda + \mu,$$

onde $\lambda = \lfloor \frac{TMAX}{D_{min(i)}+2} \rfloor$ e

$$\mu = \begin{cases} 2, & \text{if } \nu \geq 2, \\ 1, & \text{if } 1 \leq \nu \leq 2, \\ 0, & \text{if } \nu = 0. \end{cases} \qquad \nu = TMAX \bmod (D_{min(i)} + 2)$$

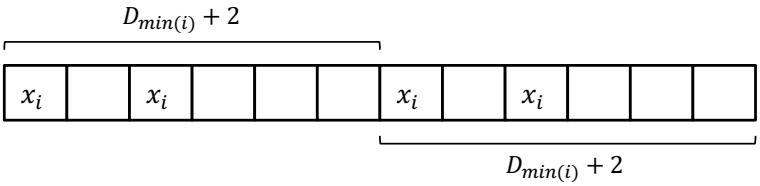

Figura 8: Número máximo de cópias do símbolo x_i.

3.5 Alterações no modelo matemático proposto por Pessoa et al. [16]

As melhorias discutidas nas seções anteriores foram convertidas em restrições e incorporadas à formulação matemática de [16]. Como mencionado anteriormente, essas melhorias visam restringir o intervalo de valores que principais variáveis do modelo podem assumir, resultando em um impacto mais robusto em comparação com os cortes baseados na conjectura não provada, os quais são os cortes mais eficazes propostos por [16]. A seguir, apresentamos todas as restrições que foram adicionadas ou alteradas a partir dos melhoramentos propostos:

$$p_{i,k+2} \geq p_{ik} + D_{min(i)} + 2, \forall i = 1, \ldots, n, \forall k = 1, \ldots, M_{min(i)}, \tag{51}$$

$$p_{i,k+2} \geq p_{ik} + D_{min(i)} + 2 - \left(1 - \sum_{h \in H_{i,k+2}} y_{i,k+2,h}\right) TMAX,$$
$$\forall i = 1, \ldots, n, \forall k = M_{min(i)} + 1, \ldots, M_i - 2 \tag{52}$$

$$W_i + d_{i,1,2} \geq D_{min(i)} + 2, \qquad \forall i = 1, \ldots, n, \tag{53}$$

$$\sum_{h \in H_{ik}} y_{ikh} = 1, \qquad \forall i = 1, \ldots, n, \forall k = 1, \ldots, M_{min(i)}, \tag{54}$$

$$D_i \geq D_{min(i)}, \qquad \forall i = 1, \ldots, n, \tag{55}$$

$$D_i \leq D_{max(i)}, \qquad \forall i = 1, \ldots, n. \tag{56}$$

Os conjuntos de restrições (51)-(52) correspondem à adição do corte que estabelece a distância mínima entre cópias intercaladas dentro do mesmo ciclo. O primeiro conjunto se aplica a cópias com alocação garantida na sequência, conforme o número mínimo de cópias, enquanto o segundo se refere a cópias cuja alocação pode não ocorrer. Por outro lado, as restrições (53) aplicam o mesmo corte à última cópia de um ciclo e à segunda cópia do ciclo seguinte. As restrições (54) asseguram a alocação do número mínimo de cópias, e, por fim, as restrições (55) e (56) definem o intervalo de valores para a variável $D_i, i = 1, ..., n$. De agora em diante, o modelo resultante das melhorias propostas será denominado PSJP-1.

3.6 Algoritmo iterativo para sequências de tamanho fixo

A ideia principal do algoritmo iterativo proposto neste trabalho consiste em atualizar os valores encontrados para reduzir cada vez mais o número de variáveis a partir de cada nova solução inteira encontrada. O processo, que é chamado para cada tamanho $t \in N, \ldots, TMAX$ possível consiste em descobrir os novos valores para os limitantes superiores e inferiores das distâncias entre cópias consecutivas e número de cópias para um dado tamanho t e atualizar as restrições que lidam diretamente com esses valores.

O Algoritmo 4 descreve o método desenvolvido após todas as alterações descritas nas seções anteriores. Os parâmetros de entradas são: X, o conjunto de símbolos; $c : X \rightarrow \mathbb{Z}^+$; $TMAX$, o tamanho máximo das sequências viáveis; TEMPO_LIMITE, número natural que descreve qual o tempo limite para a execução do algoritmo em milissegundos.

O algoritmo começa calculando um limite inferior (LI) para a instancia do PSJP em questão a partir de uma versão refinada do modelo NMC proposta por [16] e definindo um limite superior ($\overline{f}$) inicial a partir da execução da meta-heurística proposta por [17]. Partimos para o laço das linhas 4 - 24 que é executado enquanto o tempo de execução for menor do que o tempo limite. Dentro deste laço, iteramos por todos os tamanhos possíveis $t \in N, \ldots, TMAX$ no loop contido nas linhas 5 - 23. Para cada t, calculamos os valores para distância miníma ($Dmin_i$), distância máxima ($Dmax_i$), número mínimo de cópias

($Mmin_i$) e número máximo de cópias (M_i) para cada símbolo $i \in X$ no loop contido nas linhas 6 - 11. Na condição presente nas linhas 12 - 17, checamos se alguma sequência com tamanho t é valida, se não pulamos para a próximo t através da atribuição da linha 13. Em seguida, resolvemos o modelo PSJP-1 levando em consideração todos esses valores encontrados. Caso o valor encontrado pela resolução do PSJP-1 para o tamanho t for melhor do que o limitante superior $\overline{f}$, o limitante é atualizado na linha 21 e nos seguimos para a próxima iteração.

Algoritmo 4: Método *iterativo* para o PSJP.

```
1  procedimento PSPJ_Iterativo(X, c, TMAX, TEMPO LIMITE)
2  LI ← valor de (1) sujeito a (27) - (33) e (??) - (??)
3  f̄ ← VNS(X, c, TMAX)
4  enquanto TEMPO_EXECUÇÃO ≤ TEMPO_LIMITE faça
5      para t = n até TMAX faça
6          para i = 1 até n faça
7              Mmin_i ← resultado de Mmin_i^* para f̄
8              Dmin_i ← resultado de Dmin_{i*} para f̄
9              Dmax_i ← resultado de Dmax_{i*} para f̄
10             M_i ← resultado de Mmax_{i*} para f̄
11         fim
12         se Σ_i^n Mmin_i > t então
13             t ← Σ_i^n Mmin_i
14             continue
15         fim
16         \\Resolva o modelo revisado para o PSPJ-1 para t, Mmin, M,
            Dmin e Dmax
17         objValue ← PSPJ-1(X, c, t, Mmin, M, Dmin, Dmax)
18         se objValue ≤ f̄ então
19             f̄ ← objValue
20         fim
21     fim
22 fim
```

Os resultados desta solução serão mostrados em comparação com as solução proposta por [16] e por [19] na Seção 4.

4 RESULTADOS

Os modelos avaliados nesta seção foram todos resolvidos por meio do IBM ILOG CPLEX 12.9, e os algoritmos implementados na linguagem C++. Os experimentos foram executados em uma máquina com processador Intel Xeon E5-2680 com 2.5 GHz, rodando o sistema operacional Ubuntu 16.04 LTS. As instâncias usadas foram aquelas criadas em [16] e disponíveis em `https://sites.google.com/site/weightedfairsequencesproblem/instances`.

4.1 Impacto das melhorias propostas para a formulação de Pessoal et al. [16]

As tabelas a seguir comparam os resultados obtidos pela solução proposta por Pessoa et al.[16] com aqueles gerados pela formulação resultante das melhorias propostas neste trabalho, denominada PSJP-1. Os resultados foram obtidos a partir da execução de ambos os métodos, utilizando como entrada as 180 instâncias de tamanhos pequeno e médio, previamente definidas e analisadas em [16]. As instâncias são organizadas em classes conforme o número de símbolos do conjunto X e o valor estabelecido como tamanho máximo de uma sequência válida ($TMAX$). Por exemplo, a classe de instâncias com número de símbolos (n) igual a 5 e tamanho máximo 10 é referida nas tabelas como *class_5_10*. Os valores das soluções viáveis ($\overline{f}$ na restrição (32) do modelo MNC) necessários para a resolução do modelo MNC e suas variantes foram obtidos aumentando-se em 10% os limitantes inferiores reportados em [16]. Todos os resultados apresentados correspondem à média dos valores obtidos para cada instância dentro de uma determinada classe.

A Tabela 2 compara o desempenho das relaxações lineares das duas formulações na raiz do *Branch-and-Bound*, utilizando o *gap* entre o limitante superior $\overline{f}$ e os valores objetivos obtidos nas relaxações. Quanto menor o valor do *gap*, mais próximo o valor encontrado nas relaxações está do limitante superior. Como pode ser observado, a formulação PSJP-1 obtém melhores limitantes em todas as 18 classes de instâncias. O valor médio dos *gaps* para todas as classes é de 0,74 para o modelo proposto em [16] e de 0,2 para o modelo PSJP-1, o que evidencia o fortalecimento da formulação original após a inserção das melhorias propostas.

A Tabela 3 apresenta os resultados da execução do solver para ambas as formulações, considerando um tempo limite de 7200 segundos. A coluna **Tempo** indica o tempo total de execução, incluindo o tempo gasto nos modelos auxiliares. A coluna **Gap** mostra o *gap* (em %) entre a melhor solução obtida e o limite inferior encontrado pelo solver. Quando nenhuma solução inteira é encontrada dentro do tempo limite, o valor do

Tabela 2: Comparação da performance dos modelos no nó raiz.

	Pessoa et al.	PSJP-1
Classe	Gap(%)	Gap(%)
class_5_10	0.43	**0.21**
class_5_15	0.68	**0.18**
class_5_20	0.73	**0.18**
class_7_14	0.49	**0.14**
class_7_21	0.67	**0.20**
class_7_28	0.79	**0.14**
class_9_18	0.52	**0.22**
class_9_27	0.81	**0.18**
class_9_36	0.85	**0.17**
class_11_22	0.60	**0.23**
class_11_33	0.81	**0.22**
class_11_44	0.89	**0.19**
class_13_26	0.55	**0.17**
class_13_39	0.92	**0.23**
class_13_52	0.96	**0.23**
class_15_30	0.64	**0.23**
class_15_45	0.92	**0.23**
class_15_60	0.98	**0.21**

gap é registrado como 100%. Por fim, a coluna **Ótimos** informa o número de instâncias resolvidas de forma ótima em cada classe.

Tabela 3: Comparação de métricas entre algoritmos.

	Pessoa et al.			PSJP-1		
Classe	Tempo(s)	Gap(%)	Ótimos	Tempo	Gap(%)	Ótimos
class_5_10	0.05	0.0	10	**0.02**	0.0	10
class_5_15	0.90	0.0	10	**0.11**	0.0	10
class_5_20	33.11	0.0	10	**1.44**	0.0	10
class_7_14	0.18	0.0	10	**0.13**	0.0	10
class_7_21	50.67	0.0	10	**7.01**	0.0	10
class_7_28	1951.47	0.01	8	**28.27**	0.0	10
class_9_18	0.88	0.0	10	**0.33**	0.0	10
class_9_27	145.19	0.0	10	**10.55**	0.0	10
class_9_36	3722.02	10.99	7	**334.85**	0.0	10
class_11_22	**2.27**	0.0	10	4.45	0.0	10
class_11_33	1432.27	0.0	10	**10.22**	0.0	10
class_11_44	6746.82	51.30	2	**1247.64**	0.0	10
class_13_26	3.70	0.0	10	**1.22**	0.0	10
class_13_39	4198.72	10.82	7	**117.53**	0.0	10
class_13_52	7200	100.0	0	**3616.25**	0.003	8
class_15_30	212.60	0.0	10	**9.16**	0.0	10
class_15_45	7200	100.0	0	**1517.04**	0.0	8
class_15_60	7200	100.0	0	**6613.97**	0.6	2

Assim como nos testes realizados na raiz do *Branch-and-Bound*, a formulação aprimorada (PSJP-1) apresentou um desempenho muito superior à formulação original, tanto no tempo de execução quanto no número de soluções ótimas encontradas. Analisando apenas as classes em que ambos os métodos resolvem todas até a otimalidade, ou seja, das classes 05_10 a 07_28, a formulação PSJP-1 reduziu o tempo de execução por um fator de 9,75. Já considerando as classes em que somente o modelo PSJP-1 resolve todas as instâncias, a redução no tempo de execução foi de pelo menos 10,45 vezes. Quanto ao número de instâncias resolvidas, o método de Pessoa et al. [16] encontrou 134 soluções ótimas, enquanto o PSJP-1 alcançou 168.

4.2 Melhorias aplicadas à formulação de Sinnl et al. [19]

Com o intuito de avaliar se a formulação proposta em [19] se beneficia das mesmas estratégias elaboradas para fortalecer a formulação de Pessoal et al. [16], tentou-se adaptar as restrições introduzidas na Seção 3.5 para o modelo do autor em questão. Em particular, foram adicionadas restrições que impedem a criação de pares de cópias que violam as distâncias máximas para cada símbolo e também restrições que estabelecem uma distância mínima entre cópias consecutivas e intercaladas. As tabelas a seguir ilustram os resultados obtidos na resolução das instâncias pequenas (Tabela 4) e das instâncias maiores (Tabelas 4) de Pessoal et al. [16], exibindo o tempo de execução (**Tempo)** e o número de soluções ótimas (**Ótimos**) encontrados por cada método.

Tabela 4: Avaliação das adaptações realizadas na formulação proposta em [19] sobre as instâncias pequenas.

	Sinnl		Sinnl-Modificado	
Classe	Tempo	Ótimos	Tempo	Ótimos
class_05_10	**0.13**	**10**	0.14	9
class_05_15	0.33	10	0.33	10
class_05_20	**1.01**	10	1.03	10
class_07_14	0.30	10	**0.29**	10
class_07_21	1.93	10	**1.89**	10
class_07_28	9.96	10	**9.57**	10
class_09_18	**2.00**	10	2.03	10
class_09_27	**4.45**	10	4.60	10
class_09_36	**205.55**	10	208.76	10
class_11_22	**1.15**	10	1.25	10
class_11_33	**29.20**	10	29.72	10
class_11_44	662.29	10	**656.34**	10
class_13_26	**8.75**	10	8.82	10
class_13_39	**63.41**	10	67.60	10
class_13_52	1275.79	9	**1209.76**	9
class_15_30	**17.64**	10	18.21	10
class_15_45	**365.85**	10	380.59	10
class_15_60	4504.94	**7**	**3331.54**	5

Tabela 5: Comparação entre Sinnl (2022) e Sinnl-Modificado (Instâncias grandes).

	Sinnl		Sinnl-Modificado	
Classe	Tempo	Ótimos	Tempo	Ótimos
class_05_25	4.70	10	**4.59**	10
class_05_30	5.78	10	**5.74**	10
class_05_35	28.39	10	**27.46**	10
class_05_40	149.36	10	**145.24**	10
class_05_50	**875.51**	10	882.82	10
class_05_75	**902.66**	8	918.00	8
class_05_100	**880.95**	8	933.87	8
class_05_125	**2439.50**	4	2553.97	4
class_05_150	**328.04**	6	571.67	6
class_07_35	118.55	10	**113.37**	10
class_07_42	521.60	10	**520.41**	10
class_07_49	**1475.40**	10	1489.31	10
class_07_56	**1595.65**	8	1620.56	8
class_07_63	**2631.26**	7	2633.53	7
class_09_45	**1005.70**	9	1033.36	9
class_11_55	1080.25	7	**1074.75**	7
class_20_40	**80.67**	10	94.34	10
class_20_60	**680.99**	8	748.09	8
class_25_50	**82.22**	10	101.27	10
class_25_75	**3917.47**	**7**	4178.93	5
class_30_60	**107.62**	9	146.03	9
class_35_70	**1381.12**	7	1469.59	7
class_40_80	1283.71	7	**895.43**	7
class_45_90	**4777.97**	5	5133.68	**6**
class_50_100	**2465.59**	5	3631.11	5

Como pode-se notar, as restrições adicionadas não tiveram um impacto positivo no fortalecimento da formulação de Sinnl [19]. Em algumas classes de instâncias, há uma ligeira melhora no tempo despendido para a resolução das instâncias, enquanto que em outras há um piora. Isso pode ser explicado devido à natureza da referida formulação, que de forma implícita já elimina a possibilidade de que cópias de um mesmo símbolo fiquem muito próximas nas soluções das relaxações lineares durante a execução do *Branch-and-Bound*, evitando assim que os custos de tais soluções se aproximem de zero. Consequentemente, os conjuntos de restrições (51), (52) e (53), que são os mais efetivos no fortalecimento da formulação de Pessoal et al.[16], não surtem o mesmo efeito na formulação em análise.

4.3 Avaliando o método iterativo proposto

Nesta seção, são apresentados os resultados dos experimentos realizados para avaliar o desempenho do método iterativo proposto (MI), descrito na Seção 3.6. Além disso,

ele é comparado com a solução de Sinnl [19], que é o método exato da literatura que alcançou os melhores resultados até então. Os experimentos foram realizados em todas as 440 instâncias geradas em [16]. Assim como os resultados apresentados anteriormente, os dados agregados correspondem a valores médios calculados a partir das 10 instâncias de cada classe.

A Tabela 6 apresenta o tempo de execução (**Tempo**) e a quantidade de soluções resolvidas até a otimalidade (**Ótimos**) por cada um dos métodos avaliados, considerando apenas o grupo de instâncias menores.

Tabela 6: Comparação entre algoritmos para as instâncias pequenas.

	PSJP-1		Sinnl		MI	
Classe	Tempo	Ótimos	Tempo	Ótimos	Tempo	Ótimos
class_5_10	0.02	10	**0.02**	10	0.05	10
class_5_15	0.11	10	0.07	10	**0.06**	10
class_5_20	1.44	10	0.28	10	**0.10**	10
class_7_14	0.13	10	0.07	10	**0.08**	10
class_7_21	7.01	10	0.52	10	**0.18**	10
class_7_28	28.27	10	5.93	10	**0.42**	**10**
class_9_18	0.33	10	0.20	10	0.20	10
class_9_27	10.55	10	0.82	10	**0.31**	10
class_9_36	334.85	10	13.43	10	**0.75**	**10**
class_11_22	4.45	10	0.50	10	**0.35**	10
class_11_33	10.22	10	2.78	10	**0.65**	10
class_11_44	1247.64	10	196.19	10	**2.05**	10
class_13_26	1.22	10	1.32	10	**0.56**	10
class_13_39	117.53	10	10.46	10	**1.29**	**10**
class_13_52	3616.25	8	391.71	8	**9.53**	**10**
class_15_30	9.16	10	3.06	10	**0.99**	10
class_15_45	1517.04	8	32.00	10	**2.67**	**10**
class_15_60	6613.97	2	378.72	9	**3.94**	**10**

Os resultados apontam uma melhora significativa do algoritmo MI com relação não apenas ao PJSP-1, mas também quando comparado à solução desenvolvida em [19]. O método iterativo proposto resolveu todas as 180 instâncias do experimento, quatro a mais que a solução de Sinnl [19]. Já com relação ao tempo de execução, MI foi em média 16,89 vezes mais rápido que a solução do estado da arte, considerando a razão média entre os tempos de execução agrupados por classe . Isso sugere que a formulação PSJP-1 é significativamente fortalecida quando adaptada para resolver sequências de tamanho fixo, e que o modelo MNC tem um papel importante em evitar que tamanhos de sequências não promissores sejam tratados pelo algoritmo.

Na Tabela 7, MI-VNS refere-se a uma versão do algoritmo MI, que utiliza os resultados da metaheurística VNS, introduzida por Rocha et al. [17], como limite superior

na sua primeira iteração. São reportados o tempo de execução e o número instâncias resolvidas até a otimalidade.

Tabela 7: MI utilizando como limitante superior resultados da meta-heurística VNS de Rocha et al. [17].

	MI		MI-VNS	
Classe	Tempo	Ótimo	Tempo	Ótimo
class_5_10	0.05	10	**0.04**	10
class_5_15	0.06	10	**0.03**	10
class_5_20	0.10	10	**0.08**	10
class_7_14	0.08	10	**0.04**	10
class_7_21	0.18	10	**0.10**	10
class_7_28	0.42	10	**0.27**	10
class_9_18	0.20	10	**0.09**	10
class_9_27	0.31	10	**0.17**	10
class_9_36	0.75	10	**0.37**	10
class_11_22	0.35	10	**0.14**	10
class_11_33	0.65	10	**0.19**	10
class_11_44	2.05	10	**1.10**	10
class_13_26	0.56	10	**0.19**	10
class_13_39	1.29	10	**0.23**	10
class_13_52	9.53	10	**3.89**	10
class_15_30	0.99	10	**0.19**	10
class_15_45	2.67	10	**0.48**	10
class_15_60	3.94	10	**0.93**	10

Os dados apresentados demonstram a capacidade do método MI de tirar proveito de bons limitantes superiores fornecidos em sua primeira iteração. Isso demonstra que modelo PSJP-1, adaptado para sequências de tamanho fixo, é fortalecido de forma significativa com cortes que são criados a partir de tais limitantes. Portanto, quanto melhor for o limitante, mais rápido será a execução de MI.

Por fim, a Tabela 8 compara os resultados da solução desenvolvida em [19] e o MI-VNS sobre as instâncias grandes.

Tabela 8: Comparação entre o método de Sinnl [19] e o MI-VNS sobre as instâncias grandes.

	Sinnl		MI-VNS	
Classe	Tempo	Ótimos	Tempo	Ótimos
class_5_25	2.35	10	**0.42**	10
class_5_30	3.05	10	**0.60**	10
class_5_35	10.98	10	**1.10**	10
class_5_40	173.92	10	**4.34**	10
class_5_50	407.71	10	**12.80**	10
class_5_75	837.11	8	**40.38**	**10**
class_5_100	450.99	9	**313.71**	**10**
class_5_125	1518.29	6	**278.09**	**8**
class_5_150	1618.53	6	**753.29**	**9**
class_5_200	2186.13	4	**1065.55**	**5**
class_7_35	79.36	10	**0.94**	10
class_7_42	91.21	10	**2.84**	10
class_7_49	566.62	9	**2.21**	**10**
class_7_56	798.94	8	**96.65**	**10**
class_7_63	1595.93	6	**450.38**	**10**
class_9_45	802.45	8	**5.56**	**10**
class_11_55	674.43	9	**3.02**	**10**
class_20_40	674.43	10	**0.47**	10
class_20_60	116.78	10	**1.01**	10
class_25_50	49.27	10	**1.04**	10
class_25_75	763.45	10	**1.86**	10
class_30_60	140.12	10	**1.94**	10
class_35_70	383.09	10	**4.27**	10
class_40_80	763.53	10	**39.03**	10
class_45_90	1976.25	10	**6.55**	10
class_50_100	3180.80	5	**7.64**	**10**

Pode-se perceber que o MI-VNS supera de forma significativa a solução de Sinnl [19], tanto em termos de tempo de execução quanto ao número de soluções ótimas encontradas. O MI-VNS resolveu até a otimalidade 24 instâncias a mais, do grupo de instâncias maiores, que a solução do estado da arte. Das 260 instâncias grandes, 252 instâncias foram resolvidas. Com base na razão entre o tempo de execução da solução apresentada em [19] e o MI-VNS, por classe de instâncias, conclui-se que o MI-VNS foi cerca de 134 vezes mais eficiente, em média. Considerando todas as 440 instâncias introduzidas em [16], o MI-VNS resolveu até a otimalidade 432 instâncias, 27 instâncias a mais que a solução de Sinnl [19] e 23 a mais que o método heurístico de Rocha et al. [17], até então o trabalho da literatura que havia encontrado o maior número de soluções ótimas.

5 CONCLUSÕES

Neste trabalho, abordamos o Problema das Sequências Justas Ponderadas, introduzido em [16], um problema de escalonamento justo com múltiplas aplicações potenciais. O objetivo do PSJP é criar uma sequência que determine a ordem de execução de determinadas atividades, de forma que o máximo produto, definido como a maior distância temporal entre duas execuções consecutivas de uma mesma tarefa multiplicada por sua prioridade, seja minimizado.

Devido ao uso de um corte cuja validade se baseia em uma conjectura ainda não comprovada, a solução proposta em [16], embora apresente características de uma abordagem exata, não consegue garantir a otimalidade das soluções obtidas. Observou-se também que, ao resolver o modelo proposto pelos referidos autores, a solução da relaxação linear no nó raiz do *branch-and-bound* tendia a valores próximos de zero, indicando a necessidade de ajustes na formulação. Neste trabalho, aprimoramos a solução de Pessoa et al. [16], fortalecendo a formulação matemática apresentada, ao mesmo tempo em que assegurou-se otimalidade das soluções. Além disso, foi proposto um novo método iterativo que, baseado na abordagem de Sinnl [19], divide o PSJP em subproblemas menores e os resolve em iterações distintas.

Como resultado, foi apresentado um método exato que se fortalece a cada iteração, encontrado soluções ótimas em um menor tempo computacional. Os experimentos demonstraram que a solução proposta superou de forma significativa o método exato do estado da arte, tanto em relação ao número de instâncias resolvidas até a otimalidade, quanto ao tempo de execução. Para o grupo de instâncias grandes, por exemplo, o método proposto chega a ser 134 vezes mais eficiente que a solução desenvolvida em [19]. Além do mais, em linha com os objetivos delineados na Seção 1.3.2, foram encontradas 23 soluções ótimas inéditas para instâncias até então não resolvidas.

Referências

[1] Swarup Acharya, Rafael Alonso, Michael Franklin e Stanley Zdonik. "Broadcast Disks: Data Management for Asymmetric Communication Environments". Em: *SIGMOD Rec.* 24.2 (1995), 199–210. ISSN: 0163-5808. DOI: 10.1145/568271.223816. URL: https://doi.org/10.1145/568271.223816.

[2] Shoshana Anily, Celia A. Glass e Refael Hassin. "The Scheduling of Maintenance Service". Em: *Discrete Appl. Math.* 82.1–3 (1998), 27–42. ISSN: 0166-218X. DOI: 10.1016/S0166-218X(97)00119-4. URL: https://doi.org/10.1016/S0166-218X(97)00119-4.

[3] Neil C. Audsley, Alan Burns, Robert I. Davis, Ken W. Tindell e Andy J. Wellings. "Fixed Priority Pre-Emptive Scheduling: An Historical Perspective". Em: *Real-Time Syst.* 8.2–3 (1995), 173–198. ISSN: 0922-6443. DOI: 10.1007/BF01094342. URL: https://doi.org/10.1007/BF01094342.

[4] Amotz Bar-Noy, Randeep Bhatia, Joseph Seffi, Naor e Baruch Schieber. "Minimizing Service and Operation Costs of Periodic Scheduling". Em: *Mathematics of Operations Research* 27 (set. de 2002), páginas 518–544. DOI: 10.1287/moor.27.3.518.314.

[5] Baruah, S.K., Cohen, N.K. e C.G. et al. Plaxton. "Proportionate progress: A notion of fairness in resource allocation". Em: *Algorithmica* (1996), 600–625. DOI: 10.1007/BF01940883. URL: https://doi.org/10.1007/BF01940883.

[6] Doron Chen e Reuven Chen. "New relaxation-based algorithms for the optimal solution of the continuous and discrete p-center problems". Em: *Computers Operations Research* 36.5 (2009). Selected papers presented at the Tenth International Symposium on Locational Decisions (ISOLDE X), páginas 1646–1655. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2008.03.009. URL: https://www.sciencedirect.com/science/article/pii/S0305054808000725.

[7] Claudio Contardo, Manuel Iori e Raphael Kramer. "A scalable exact algorithm for the vertex p-center problem". Em: *Computers Operations Research* 103 (2019), páginas 211–220. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2018.11.006. URL: https://www.sciencedirect.com/science/article/pii/S0305054818302910.

[8] Albert Corominas, Wieslaw Kubiak e Natalia Palli. "Response time variability". Em: *Reports de recerca de l'Institut d'Organització i Control de Sistemes Industrials, Nº. 8, 2004* 10 (fev. de 2007). DOI: 10.1007/s10951-006-0002-8.

[9] Melanie Erhard, Jan Schoenfelder, Andreas Fügener e Jens O. Brunner. "State of the art in physician scheduling". Em: *European Journal of Operational Research* 265.1 (2018), páginas 1 –18. ISSN: 0377-2217. DOI: https://doi.org/10.1016/j.ejor.2017.06.037. URL: http://www.sciencedirect.com/science/article/pii/S0377221717305787.

[10] Ching-Chih Han, Kwei-Jay Lin e Chao-Ju Hou. "Distance-constrained scheduling and its applications to real-time systems". Em: *IEEE Transactions on Computers* 45.7 (1996), páginas 814–826. DOI: 10.1109/12.508320.

[11] R. Holte, A. Mok, L. Rosier, I. Tulchinsky e D. Varvel. "The pinwheel: a real-time scheduling problem". Em: *[1989] Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume II: Software Track*. Volume 2. 1989, 693–702 vol.2. DOI: 10.1109/HICSS.1989.48075.

[12] Kubiak, Wieslaw, Leung e Joseph Y-T. *Fair sequencem, em Handbook of scheduling : algorithms, models, and performance analysis*. Boca Raton, Fla.: Chapman Hall/CRC, 2004. ISBN: 1584883979 9781584883975.

[13] Joseph Y.-T. Leung e Jennifer Whitehead. "On the complexity of fixed-priority scheduling of periodic, real-time tasks". Em: *Performance Evaluation* 2.4 (1982), páginas 237–250. ISSN: 0166-5316. DOI: https://doi.org/10.1016/0166-5316(82)90024-4. URL: https://www.sciencedirect.com/science/article/pii/0166531682900244.

[14] Joseph Y.T. Leung e M. L. Merrill. "A note on preemptive scheduling of periodic, real-time tasks". English (US). Em: *Information Processing Letters* 11.3 (jan. de 1980), páginas 115–118. ISSN: 0020-0190. DOI: 10.1016/0020-0190(80)90123-4.

[15] C. L. Liu e James W. Layland. "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment". Em: *J. ACM* 20.1 (1973), 46–61. ISSN: 0004-5411. DOI: 10.1145/321738.321743. URL: https://doi.org/10.1145/321738.321743.

[16] Bruno J. S. Pessoa, Daniel Aloise e Lucidio A.F. Cabral. "The Weighted Fair Sequences Problem". Em: *Computers Operations Research* 91 (2018), páginas 121–131. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2017.11.008. URL: https://www.sciencedirect.com/science/article/pii/S0305054817302848.

[17] Caroline Rocha, Bruno J.S. Pessoa, Daniel Aloise e Lucidio A. Cabral. "An efficient implementation of a VNS heuristic for the weighted fair sequences problem". Em: *International Transactions in Operational Research* n/a.n/a (). DOI: https://doi.org/10.1111/itor.13197. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/itor.13197. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.13197.

[18] Markus Sinnl. “A note on computational approaches for the antibandwidth problem”. Em: *Central European Journal of Operations Research* 29.3 (2021), páginas 1057–1077. DOI: 10.1007/s10100-020-00688-. URL: https://ideas.repec.org/a/spr/cejnor/v29y2021i3d10.1007_s10100-020-00688-4.html.

[19] Markus Sinnl. “An iterative exact algorithm for the weighted fair sequences problem”. Em: *Computers Operations Research* 148 (2022), página 106017. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2022.106017. URL: https://www.sciencedirect.com/science/article/pii/S0305054822002477.