Meta-heurística híbrida aplicada ao Problema das Sequências Justas Ponderadas

Bruno J. S. Pessoa[1], **Lucídio dos Anjos Cabral Formiga**[1], **Daniel Aloise** [2]

[1]Departamento de Computação Científica – Universidade Federal da Paraíba
João Pessoa – PB – Brasil

[2]Departamento de Engenharia da Computação – École Polytechnique de Montréal
Montreal – Canadá

Autores para correspondência: bruno@ci.ufpb.br, lucidio@ci.ufpb.br

RESUMO

O Problema das Sequências Justas Ponderadas (PSJP) é um problema de otimização que foi recentemente definido na literatura. Ele abrange um grande número de aplicações, em diferentes áreas, as quais variam desde a produção de automóveis em uma linha de montagem ao sequenciamento de aplicações interativas a serem transmitidas em um ambiente de TV Digital. O PSJP, que faz parte da classe de problemas de escalonamento denominada de sequências justas, foi provado ser NP-difícil por Pessoa et al. (2018), e um método iterativo foi proposto para resolvê-lo. Este trabalho propõe uma meta-heurística híbrida para resolver o PSJP e faz uma análise das dificuldades intrínsecas do problema. Os experimentos realizados mostram que a meta-heurística proposta produz soluções com a mesma qualidade do método iterativo citado em tempo computacional bastante inferior.

PALAVRAS-CHAVE: Escalonamento. Sequências Justas. Meta-heurísticas.

Área Principal: MH - Meta-heurísticas

ABSTRACT

The Weighted Fair Sequences Problem (WFSP) is an optimization problem that has been recently defined in the literature. It covers a large number of applications in different areas, ranging from automobile production on a mixed-model assembly line to the sequencing of interactive applications to be aired in a Digital TV environment. The WFSP, which is part of the class of scheduling problems called fair sequences, was proved to be NP-hard by Pessoa et al. (2018), and an iterative solution method was proposed for it. This paper proposes a hybrid metaheuristic for the WFSP and makes an analysis on the intrinsic difficulties of the problem. Computational experiments show that the proposed metaheuristic is able to find solutions with the same quality of the iterative method in much less computational time.

KEYWORDS: Scheduling. Fair Sequences. Metaheuristics.

Main Area: MH - Metaheuristics

https://proceedings.science/p/85285?lang=pt-br

1. Introdução

Problemas de escalonamento aos quais são impostas restrições relativas às distâncias temporais entre sucessivas execuções de uma tarefa ou atividade são frequentemente tratados na literatura como problemas de escalonamento de distâncias restritas (Han et al., 1996) ou simplesmente de sequências justas (Kubiak, 2004). Pessoa et al. (2018) introduziram um novo problema de otimização, chamado de Problemas das Sequências Justas Ponderadas (PSJP), que faz parte dessa classe de problemas. O PSJP consiste em um problema de escalonamento periódico, com horizonte de tempo finito, que, dado um conjunto de tarefas com diferentes prioridades, objetiva escalonar uma sequência de execuções tal que o máximo produto, envolvendo a maior distância temporal entre duas execuções sucessivas de uma mesma tarefa e sua prioridade, seja minimizado.

Em um contexto mais amplo, o PSJP 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. Abaixo são detalhadas algumas dessas aplicações.

Em um sistema de TV Digital, parte da banda alocada para as emissoras é reservada para a transmissão de aplicações interativas, isto é, programas de computador capazes de proporcionar interatividade entre os telespectadores e o conteúdo televisivo e anunciar produtos de forma mais dinâmica (Morris, 2005). Eles são transmitidos de forma cíclica, via *broadcast*, de servidores para clientes em um canal de comunicação unidirecional. Para baixá-los, os terminais dos clientes precisam escutar o canal de comunicação até a chegada dos dados solicitados. Um ciclo ou uma sequência de transmissões é constituído de um conjunto ordenado de aplicações, que podem ser transmitidas várias vezes de acordo com sua prioridade. Ao fim da transmissão da última aplicação no ciclo, a primeira é transmitida novamente. Considerando que aplicações com prioridades maiores (ex. aquelas cujos anunciantes pagam mais por sua veiculação) devem ser transmitidas em uma frequência maior que as demais, e que a frequência ótima também constitui uma variável do problema, objetiva-se minimizar a máxima distância temporal entre transmissões sucessivas de uma mesma aplicação e, consequentemente, o tempo de espera dos clientes pelos recursos requeridos.

Outro contexto no qual o PSJP pode ser aplicado diz respeito ao problema de manutenção periódica de máquinas (Anily et al., 1998; Bar-Noy et al., 2002). Nesse cenário, um escalonamento de manutenções deve ser elaborado visando minimizar o tempo decorrido entre duas manutenções sucessivas, dado que os custos de operação das máquinas crescem com o tempo decorrido desde a última manutenção efetivada. Como cada máquina possui um custo de operação particular, algumas delas são priorizadas no escalonamento dos serviços de manutenção. Por conseguinte, a frequência ideal dos serviços de manutenção de cada máquina constitui uma variável do problema. Além do mais, na maioria das situações reais, há uma limitação com relação ao horizonte de tempo (ex. um dia, uma semana) no qual os serviços de manutenção devem estar compreendidos, o qual deve ser finito e obedecer a um cronograma estabelecido a priori.

https://proceedings.science/p/85285?lang=pt-br

O escalonamento de processos em sistemas de tempo real também está dentro do escopo coberto pelo PSJP (Waldspurger e Weihl. W., 1995). Em alguns sistemas de tempo real, deseja-se estabelecer um escalonamento de processos baseado no chamado controle de compartilhamento proporcional, a fim de minimizar o tempo de resposta de cada processo quando solicitado. Para isso, os processos devem ser executados em uma taxa uniforme ao longo do tempo, mediante uma estratégia de atribuição de prioridades que tem por objetivo refletir o grau de importância de cada tarefa. Como nos exemplos anteriores, os processos com maiores prioridades devem ser executados em uma frequência maior que os demais. Para evitar situações nas quais processos menos prioritários demoram muito para ser executados, limita-se o período máximo das sequências criadas e se estabelece uma frequência mínima para cada processo.

Na literatura, problemas correlatos ao PSJP surgiram em épocas distintas e em contextos diferentes. Mais recentemente, Corominas et al. (2007) apresentaram um problema de sequenciamento denominado de Problema da Variabilidade do Tempo de Resposta (RTVP, do inglês *Response Time Variability Problem*), que surge sempre que tarefas (clientes ou produtos) precisam ser escalonadas de modo a reduzir a variabilidade da distância temporal entre suas execuções (Corominas et al., 2010; García-Villoria e Pastor, 2010; García-Villoria et al., 2013). No RTVP, as frequências ou demandas de cada tarefa são especificadas a priori, ou seja, são entradas para o problema. García-Villoria e Salhi (2014) abordaram um novo problema que ocorre na indústria de TV, envolvendo um esquema de transmissão de anúncios que satisfaça os anunciantes e, por conseguinte, as emissoras. O objetivo do problema é encontrar um escalonamento viável que minimize a irregularidade dos intervalos de tempo entre duas transmissões consecutivas de um mesmo anúncio. O PSJP pode ser considerado como uma generalização de tais problemas.

Além de proporem uma formulação matemática baseada em Programação Linear Inteira Mista para o PSJP, Pessoa et al. (2018) realizaram um estudo sobre a complexidade computacional do problema e propuseram um método iterativo para resolvê-lo. O algoritmo proposto baseia-se na resolução iterativa da formulação matemática por meio de um solver de programação inteira e em cortes introduzidos dinamicamente a cada iteração. Uma vez que os autores não conseguiram provar formalmente a validade de um dos conjuntos de cortes utilizados, as soluções resultantes do método iterativo não podem ser consideradas como ótimas. Para avaliar a qualidade das soluções, eles desenvolveram um modelo matemático para encontrar limites inferiores para o problema.

Considerando que o PSJP é um problema NP-difícil, este trabalho propõe uma solução baseada em uma meta-heurística híbrida e faz uma comparação com os resultados obtidos pelo método iterativo proposto por Pessoa et al. (2018).

O restante do artigo é organizado como segue. A Seção 2 apresenta uma definição formal do PJSP. A Seção 3 descreve a meta-heurística híbrida proposta. A Seção 4 discute os resultado obtidos. Por fim, a Seção 5 traz as considerações finais e propostas para a continuidade da pesquisa.

2. Definição do problema

Pessoa et al. (2018) definiram o problema como segue. Seja $X = \{x_1, x_2, \ldots, x_n\}$ um conjunto de símbolos, cada um deles com tamanho unitário, e $c : X \to \mathbb{Z}^+$ uma função que determina suas prioridades. Para um símbolo x_i, sua prioridade é denotada por $c_i =$

https://proceedings.science/p/85285?lang=pt-br

$c(x_i)$. Considere uma sequência $S = s_1, s_2, \ldots, s_T$ 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 $D_i = max\{max_{k=1,\ldots,l-1}\{d_{i,k,k+1}\}, d_{i,l,1}\}$ 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 $i = 1, \ldots, n$. Para maiores detalhes sobre a formulação matemática do problema, vide (Pessoa et al., 2018).

A Figura 1, extraída de (Pessoa et al., 2018), ilustra soluções distintas para um conjunto de símbolos $X = \{A, B, C, D, E, F\}$ com prioridades $c_A > c_B = c_C = c_D > c_E = c_F$. Elas representam duas soluções viáveis com diferentes distribuições de símbolos. Na Solução 1, as cópias de um dado símbolo são posicionadas uma após a outra. Já na segunda solução, as cópias são distribuídas de modo que a diferença entre as distâncias das cópias consecutivas de um mesmo símbolo seja minimizada. Comparando as soluções com base nas distâncias entre cópias do símbolo A, observa-se que há uma redução na maior distância de $D_A = 9$ na Solução 1 para $D_A = 3$ na Solução 2. Consequentemente, há também uma redução no valor do produto $D_A c_A$. Como a minimização da maior distância se estende a todos os símbolos, a Solução 2 consiste em uma solução melhor do que a Solução 1.

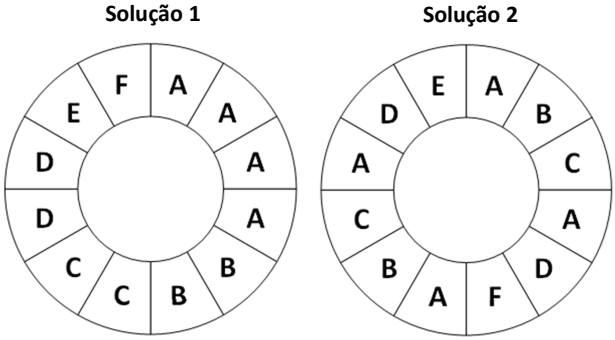


Figura 1. Distribuição uniforme e não-uniforme de símbolos.

3. Meta-heurística híbrida

A solução heurística aqui apresentada, denominada de Multi-start ILS-VND (MILS-VND), é resultado da combinação das meta-heurísticas *Iterated Local Search* (ILS) (Lourenço et al., 2003) e *Variable Neighborhood Descent* (VND) (Mladenović e Hansen, 1997), inseridas em um *framework multi-start*. A ideia geral da solução é executar o ILS um número específico de vezes, de maneira que as execuções construam sequências de símbolos de tamanhos distintos, com a busca local sendo determinada pelo VND. Com base em experimentos preliminares, constatou-se que as soluções ótimas raramente possuíam menos de $2n$ símbolos. Assim sendo, os tamanhos de sequências considerados pelo ILS variam no intervalo $[2n, TMAX]$. Quando $TMAX$ for igual a $2n$, utiliza-se o intervalo $[n, TMAX]$. A abordagem tratada é descrita pelo Algoritmo 1. Após a inicialização da melhor solução (s^*) com uma sequência em que há apenas uma cópia de

https://proceedings.science/p/85285?lang=pt-br

cada símbolo (linha 2), o método *multi-start* utilizado executa o ILS $nIter$ vezes, onde $nIter$ é definido pelo produto do parâmetro μ pela quantidade de tamanhos de sequências presentes no intervalo. A fase de inicialização do ILS, descrita pelas linhas 6-19, inicia-se com a construção de uma solução (linha 7) a partir de um tamanho de sequência escolhido aleatoriamente dentro do intervalo mencionado anteriormente. Em seguida, é realizada um busca local determinada pelo VND (linha 8). O ILS então prossegue aprimorando a solução oriunda da fase de inicialização por meio de uma série de buscas locais combinadas com um mecanismo de pertubação (linhas 11-18). Uma iteração do método *multi-start* se encerra quando o número de iterações do ILS sem melhora excede $nIterILS$.

Algoritmo 1: Algoritmo Multistart ILS-VND.

```
1  procedimento MILS-VND(X, n, TMAX, μ, nIterILS, φ)
2  s* ← x1x2...xn;
3  ν ← max{n, 2n}, tal que TMAX − ν > 0;
4  nIter ← μ × (TMAX − ν);
5  para it = 1 até nIter faça
6      Escolha t ∈ {ν, ..., TMAX} aleatoriamente;
7      s ← Construção(X, t);
8      s' ← VND(s);
9      j ← 1;
10     repita
11         s ← Pertubação(s', φ);
12         s ← VND(s);
13         se f(s) < f(s') então
14             s' ← s;
15             j ← 1;
16         senão
17             j ← j + 1;
18     até j > nIterILS;
19     se f(s') < f(s*) então
20         s* ← s';
21         f(s*) ← f(s');
22 retorna s*;
```

3.1. Procedimento construtivo

O procedimento construtivo, utilizado para construir soluções iniciais a cada iteração do método *multi-start* do algoritmo MILS-VND, é descrito pelo Algoritmo 2. O algoritmo é dividido em duas partes: a primeira estima o número ideal de cópias para cada símbolo (linhas 2-11), e a segunda aloca cada cópia em uma posição específica da sequência em construção (linhas 12-26). A estimativa referida é realizada com base no maior produto c_iD_i, onde D_i corresponde à distância ideal de cada símbolo tendo em vista o número total de cópias atual τ da sequência a ser construída. A cada iteração do laço nas linhas 6-11, o número de cópias do símbolo que possui o maior produto c_iD_i é incrementado. O laço é encerrado quando o número total de cópias atinge o tamanho t passado por

https://proceedings.science/p/85285?lang=pt-br

parâmetro. Baseando-se no número estimado de cópias para cada símbolo, a segunda parte do algoritmo aloca cada cópia na sequência S. Em cada iteração do laço interno (linhas 16-24), tenta-se alocar na posição p uma cópia do símbolo x_i, de modo que ela esteja separada da última cópia alocada de x_i por D_i posições. Caso a posição escolhida já esteja ocupada, é realizada uma busca circular pela primeira posição não ocupada. O laço é encerrado quando o símbolo x_i é alocado m_i vezes.

Algoritmo 2: Procedimento construtivo.

```
1  procedimento Construção(X, n, t)
2  Seja m_i o número de cópias estimadas para o símbolo x_i ∈ X;
3  Seja c_i a prioridade do símbolo x_i ∈ X;
4  m_i ← 1, para i = 1, ..., n;
5  τ ← Σ_{i=1}^{n} m_i;
6  enquanto τ < t faça
7      D_i ← τ / m_i, para i = 1, ..., n;
8      j ← i, tal que c_i D_i é o maior produto para i = 1, ..., n;
9      m_j ← m_j + 1;
10     τ ← Σ_{i=1}^{n} m_i;
11 Seja S uma sequência com t posições;
12 para i = 1 até n faça
13     k ← 0;
14     p ← 1;
15     enquanto k < m_i faça
16         se a posição p de S não estiver ocupada então
17             aloca uma cópia de x_i na posição p de S;
18             k ← k + 1;
19             p ← (p + D_i > t) ? (p + D_i) mod t : p + D_i;
20         senão
21             p ← (p + 1 > t) ? (p + 1) mod t : p + 1;
22 retorna S;
```

3.2. Busca local

O procedimento de busca local do algoritmo MILS-VND é realizado pela meta-heurística VND, cujo *framework* é descrito pelo Algoritmo 3. Em cada iteração do VND (linhas 4-13), escolhe-se uma estrutura de vizinhança de $\mathcal{N}$ e realiza-se uma busca pelo melhor vizinho da solução s passada por parâmetro. Quando o melhor vizinho encontrado não é melhor do que s, a busca local prossegue com a exploração da próxima estrutura de vizinhança de $\mathcal{N}$. Caso seja encontrado um vizinho melhor do que s em qualquer das estruturas de vizinhança, atualiza-se s e a busca local continua a partir da primeira vizinhança de explorada. O algoritmo termina quando a última estrutura de vizinhança de $\mathcal{N}$ é explorada e nenhuma solução melhor que s é encontrada.

https://proceedings.science/p/85285?lang=pt-br

Algoritmo 3: Algoritmo VND.

```
1  procedimento VND(s)
2  N ← {Trocar-Posição, Remover-Símbolo};
3  k ← 1;
4  enquanto k < |N| faça
5      N^(k) ← k-ésima vizinhança de N;
6      s' ← melhor vizinho de s em N;
7      se g(s') < g(s) então
8          s ← s';
9          k ← 1;
10     senão
11         k ← k + 1;
12 retorna s;
```

3.2.1. Estruturas de vizinhança

As seguintes estruturas de vizinhança são empregadas na busca local efetuada pelo VND:

- *Trocar-Posição*—$\mathcal{N}^{(1)}$: Troca a posição de dois símbolos na sequência;
- *Remover-Símbolo*—$\mathcal{N}^{(2)}$: Remove um símbolo da sequência.

Como ambas as estruturas de vizinhança são exploradas de forma exaustiva e utilizam a estratégia de busca *melhor aprimorante*, o número de movimentos realizados nas estruturas *Trocar-Posição* e *Remover-Símbolo* é da ordem de $\mathcal{O}(t^2)$ e $\mathcal{O}(t)$, respectivamente, onde t é o tamanho da sequência considerada.

3.2.2. Função de avaliação

A função de avaliação do VND é definida como segue:

$$g(s) = g'(s) + 10^{-\lambda} g''(s),$$

sendo λ uma constante inteira. A função $g'(s)$ é igual à função de avaliação $f(s)$ do Algoritmo 1, isto é, analisa o maior produto $c_i D_i, i = 1 \ldots, n$. A função complementar $g''(s)$, por sua vez, resulta no somatório de todos os desvios entre as distâncias parciais entre duas ocorrências consecutivas do símbolo x_i e sua distância ideal, $\forall x_i \in X$. A função $g''(s)$ pode ser definida da seguinte forma:

$$g''(s) = \sum_{i=1}^{n} \sum_{k=1}^{m_i - 1} (\frac{t}{m_i} - d_{i,k,k+1})^2,$$

onde t é o tamanho da sequência s, m_i é número de cópias do símbolo x_i e $d_{i,k,k+1}$ é a distância entre as cópias k e $k+1$ de x_i.

3.3. Mecanismo de pertubação

O procedimento adotado para perturbar as soluções no Algoritmo 1 consiste em executar $\frac{\phi}{100} \times t$ movimentos na estrutura de vizinhança *Trocar-Posição*, escolhendo-se pares de símbolos aleatoriamente, onde t é o tamanho da sequência passada por parâmetro.

https://proceedings.science/p/85285?lang=pt-br

4. Resultados computacionais

Nesta seção, são apresentados os resultados referentes à meta-heurística híbrida MILS-VND e é realizada um comparação com os resultados obtidos pelo método iterativo proposto por Pessoa et al. (2018). Os experimentos foram realizados em uma máquina com processador Intel Core I7 com 1.9 GHz e 6 GB de memória RAM, executando o sistema operacional Ubuntu 16.04 LTS. Todos os tempos computacionais são apresentados em segundos.

As instâncias consideradas foram aquelas disponibilizadas por Pessoa et al. (2018) em `https://sites.google.com/site/weightedfairsequencesproblem/instances`. Os autores dividiram as 180 instâncias em 18 classes de acordo com o número de símbolos (n) e o tamanho máximo das sequências viáveis ($TMAX$). Foram consideradas instâncias com 5, 7, 9, 11, 13 e 15 símbolos, e $TMAX$ igual a $2n, 3n$ e $4n$. Para cada par $(n, TMAX)$, eles criaram 10 instâncias distintas atribuindo-se prioridades aleatórias para cada símbolo no intervalo$[1, 2n]$. Os nomes das classes seguem o seguinte formato: cls_n_TMAX. Assim, cls_5_15 denota a classe de instâncias com cinco símbolos e $TMAX = 15$.

Para cada uma das 180 instâncias, o MILS-VND foi executado 10 vezes, com tempo limite estabelecido em 60 segundos. Para manter a compatibilidade com os testes realizados pelos autores citados, o tempo limite das classes *cls*_13_52 e *cls*_15_60 foi ampliado para 300 segundos. Os autores estabeleceram o tempo limite de 1 hora para a maioria das classes e de 5 horas para as classes mencionadas.

Os parâmetros μ, $nIterILS$ e ϕ foram definidos, respectivamente, como 3, 10, 20, apoiados na análise de testes preliminares que considerou a melhor relação entre a qualidade das soluções e o tempo de execução.

Na Tabela 1, **Me** representa a média do valor das soluções entre todas as execuções, **Ma** denota a mediana, **DP** o desvio padrão, **MS** o melhor valor encontrado e **Tempo** é o tempo médio de execução. A sigla TL (tempo limite) é usada sempre que o MILS-VND não resolve pelo menos uma das instâncias de uma classe dentro do limite de tempo estabelecido.

As melhores soluções da meta-heurística atingiram os limites inferiores obtidos por Pessoa et al. (2018) em 72,22% das 180 instâncias consideradas, logo ela foi capaz de encontrar pelo menos 130 soluções ótimas. As medianas das soluções, por suas vezes, foram iguais às melhores soluções obtidas pela meta-heurística em 166 instâncias. Isso mostra que em pelo menos 50% das execuções realizadas sobre tais instâncias, o MILS-VND alcança seus melhores resultados, corroborando a robustez da abordagem empregada. A referida robustez também pode ser observada a partir do desvio padrão médio total de apenas 1,04.

A Figura 2 mostra o percentual de contribuição da busca local inicial (**BL_ini**) e da etapa que combina o mecanismo de pertubação e a buscal local (**Pert+BL**) no aprimoramento das melhores soluções encontradas. Por exemplo, na classe cls_11_33, **BL_ini** foi responsável por 73,98% do aprimoramento total da solução proveniente do método construtivo, ao passo que **Pert+BL** teve uma contribuição de 26,02%. Considerando todas as classes, as médias de contribuição de **BL_ini** e de **Pert+BL** foram de 83,18% e 16,82%, respectivamente.

https://proceedings.science/p/85285?lang=pt-br

Tabela 1. Resultados da meta-heurística MILS-VND.

Classe	Me	Ma	DP	MS	Tempo
cls_5_10	37,50	37,50	0,00	37,50	0,05
cls_5_15	32,10	32,10	0,00	32,10	0,30
cls_5_20	34,82	34,70	0,30	34,70	1,09
cls_7_14	58,75	58,60	0,24	58,60	0,26
cls_7_21	65,16	65,10	0,19	65,10	1,88
cls_7_28	66,40	66,30	0,32	66,30	9,22
cls_9_18	111,50	111,20	0,91	111,20	1,02
cls_9_27	97,37	97,20	0,63	97,00	8,43
cls_9_36	94,34	94,30	0,59	93,60	45,84
cls_11_22	163,79	163,70	0,20	163,70	3,03
cls_11_33	140,21	140,10	0,70	139,30	30,59
cls_11_44	144,26	143,90	1,40	142,70	TL
cls_13_26	215,17	214,30	2,42	214,30	8,40
cls_13_39	215,54	214,80	1,60	214,80	TL
cls_13_52	179,01	178,40	1,27	177,90	TL
cls_15_30	323,84	323,30	1,71	323,30	18,73
cls_15_45	299,82	298,50	4,33	295,60	TL
cls_15_60	274,84	274,30	2,01	272,70	TL

Os resultados mostram claramente a eficiência da busca local, realizada pelo VND, na melhoria das soluções. Na maioria das instâncias, a solução resultante da busca local inicial já está bem próxima da melhor solução obtida. Além disso, também fica evidente a importância do mecanismo de pertubação do MILS-VND, o qual é responsável por fazer o algoritmo escapar de mínimos locais. Em algumas instâncias, praticamente todo o esforço de aprimoramento é realizado pela combinação da pertubação com a busca local.

4.1. Comparação entre o método iterativo e o MILS-VND

A Tabela 2 compara as melhores soluções encontradas pelo método iterativo de Pessoa et al. (2018) e o MILS-VND. As colunas **LI**, **MI** e **MILS-VND** contêm, respectivamente, limites inferiores obtidos por Pessoa et al. (2018) e as melhores soluções do método iterativo e do MILS-VND. Uma vez que os métodos ora analisados fazem parte de paradigmas de solução distintos (um se baseia em técnicas exatas e o outro é heurístico), não foi dado ênfase na análise dos tempos de execução.

Note que a qualidade das soluções encontradas pelo MILS-VND se aproxima bastante das soluções do método iterativo em instâncias pequenas e médias ($n \leq 11$ e $TMAX = 2n$ ou $3n$) e as supera em instâncias maiores. O MILS-VND é melhor em todas as classes com $n \geq 11$ e $TMAX = 4n$. No que diz respeito ao tempo de execução, o método iterativo é bastante competitivo em instâncias pequenas, porém é superado de forma contundente em instâncias maiores. Para exemplificar, considere a classe cls_15_60. Em apenas cinco minutos de execução, o MILS-VND encontrou soluções melhores que as obtidas pelo método iterativo com tempo de execução de cinco horas.

https://proceedings.science/p/85285?lang=pt-br

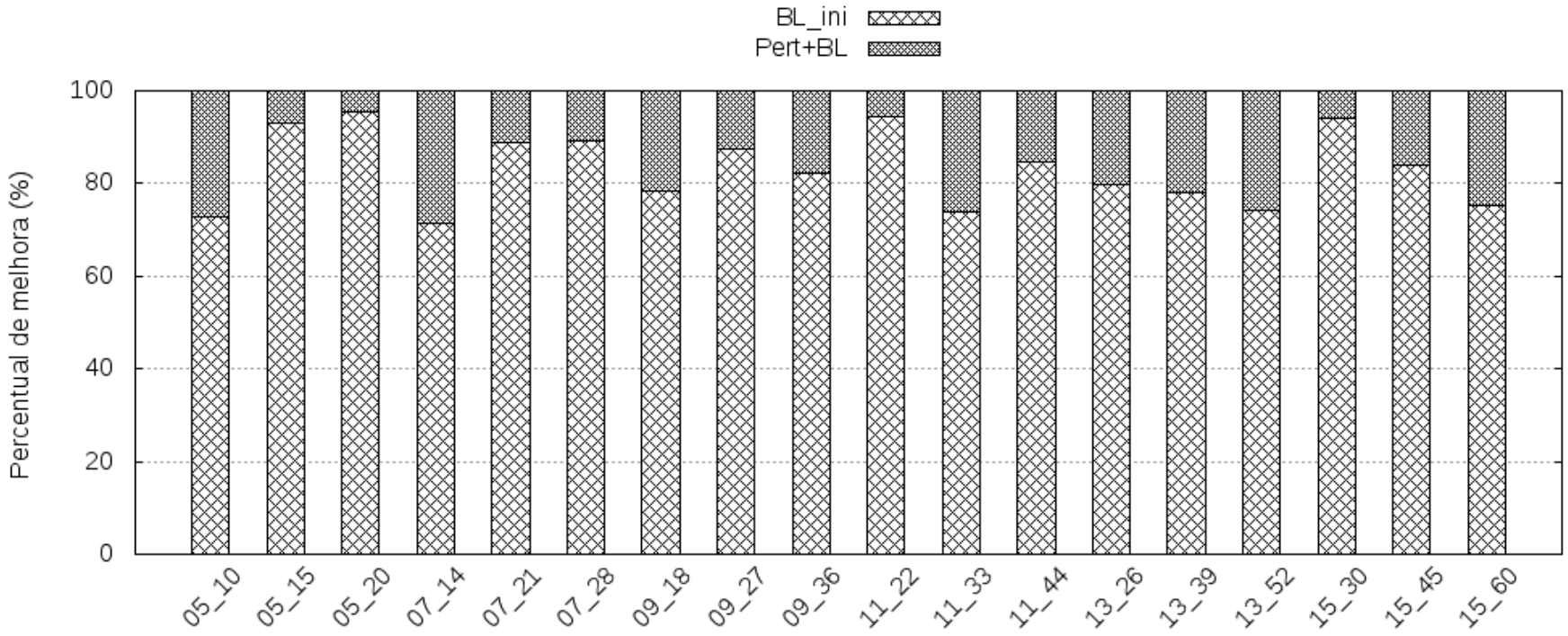


Figura 2. Percentual médio de contribuição das etapas do MILS-VND no aprimoramento das soluções.

https://proceedings.science/p/85285?lang=pt-br

Tabela 2. Comparação das melhores soluções obtidas pelo método iterativo, proposto por Pessoa et al. (2018), e o MILS-VND.

Classe	LI	MI	MILS-VND
cls_5_10	37,50	37,50	37,50
cls_5_15	30,60	**31,80**	32,10
cls_5_20	32,60	34,70	34,70
cls_7_14	58,60	58,60	58,60
cls_7_21	63,30	**64,90**	65,10
cls_7_28	63,20	**65,30**	66,30
cls_9_18	111,20	111,20	111,20
cls_9_27	96,20	**96,30**	97,00
cls_9_36	91,90	93,60	93,60
cls_11_22	163,70	163,70	163,70
cls_11_33	137,20	**137,20**	139,30
cls_11_44	140,10	143,10	**142,70**
cls_13_26	213,80	**213.80**	214.30
cls_13_39	213,50	214,80	214,80
cls_13_52	176,20	180,60	**177,90**
cls_15_30	322,80	**322,80**	323,30
cls_15_45	295,60	297,20	**295,60**
cls_15_60	268,60	281,80	**272,70**

5. Considerações Finais

O presente trabalho tratou do Problema das Sequências Justas Ponderadas (PSJP), que consiste em um novo problema de otimização introduzido recentemente na literatura. O PSJP surge sempre que há a necessidade de se minimizar a distância temporal entre execuções sucessivas de uma mesma tarefa ou atividade, em um contexto cíclico de execução. De modo geral, o PSJP ocorre sempre que produtos, tarefas ou clientes precisam ser sequenciados, a fim de minimizar o tempo que eles esperam pela próxima oportunidade de obterem os recursos desejados para prosseguirem. Assim, o PSJP abrange uma vasta gama de aplicações nas mais diversas áreas.

Este trabalho apresentou a primeira solução heurística para o problema, que certamente será usada como referência para trabalhos futuros. O PSJP tem uma conjunto de peculiaridades que dificultam sua solução via métodos exatos e heurísticos. O fato da quantidade de cópias de cada símbolo ser uma variável do problema foi um dos maiores entraves na elaboração da solução proposta. Por exemplo, foi constatado que, se o número de cópias de cada símbolo na fase de construção do algoritmo for muito diferente do número de cópias ótimo, as chances da fase de busca local encontrar boas soluções são muito pequenas. Por esse motivo, um *framework multistart* foi escolhido para produzir sequências de símbolos de tamanhos distintos em cada iteração, e uma heurística foi elaborada com a finalidade de encontrar as quantidades de cópias ideais. A função de avaliação também é um ponto importante a ser considerado. Uma vez que o máximo produto que se pretende minimizar não é sensível a qualquer troca de posições entre símbolos, optou-se por uma função de avaliação complementar mais sensível a movimentos de tal vizinhança, o que foi essencial para o bom desempenho da busca local. Desse modo,

a meta-heurística híbrida foi capaz de atingir praticamente o mesmo número de limites inferiores alcançados método iterativo proposto por Pessoa et al. (2018), em um tempo computacional muito menor.

Como o PSJP foi introduzido muito recentemente, há muito espaço para explorá-lo, tanto no que diz respeito a soluções exatas como heurísticas. Trabalhos futuros focarão na análise do politopo do problema, objetivando a elaboração de métodos *branch-and-cut*. No tocante a soluções heurísticas, meta-heurísticas baseadas em população serão elaboradas e comparadas com os resultados já obtidos.

Referências

Anily, S., Glass, C. A., e Hassin, R. (1998). The scheduling of maintenance service. *Discrete Applied Mathematics*, 82(1):27–42. ISSN 0166-218X.

Bar-Noy, A., Bhatia, R., Naor, J. S., e Schieber, B. (2002). Minimizing service and operation costs of periodic scheduling. *Math. Oper. Res.*, 27(3):518–544. ISSN 0364-765X.

Corominas, A., Kubiak, W., e Palli, N. M. (2007). Response time variability. *Journal of Scheduling*, 10(2):97–110. ISSN 1099-1425.

Corominas, A., Kubiak, W., e Pastor, R. (2010). Mathematical programming modeling of the response time variability problem. *European Journal of Operational Research*, 200 (2):347–357. ISSN 0377-2217.

García-Villoria, A. e Salhi, S. (2014). Scheduling commercial advertisements for television. *International journal of production research*, 53(4):1198–1215.

García-Villoria, A., Corominas, A., Delorme, X., Dolgui, A., Kubiak, W., e Pastor, R. (2013). A branch and bound algorithm for the response time variability problem. *Journal of Scheduling*, 16(2):243–252.

García-Villoria, A. e Pastor, R. (2010). Solving the response time variability problem by means of a genetic algorithm. *European Journal of Operational Research*, 202(2): 320–327. ISSN 0377-2217.

Han, C., Lin, K., e Hou, C. (1996). Distance-constrained scheduling and its applications to real-time systems. *IEEE Trans. Computers*, 45(7):814–826.

Kubiak, W. (2004). Fair sequences. In *In Handbook of Scheduling: Algorithms, Models and Performance Analysis, Leung, J.Y-T., editor, Chapman & Hall/CRC, Boca*.

Lourenço, H. R., Martin, O. C., e Stützle, T. (2003). *Iterated Local Search*, p. 320–353. Springer US, Boston, MA. ISBN 978-0-306-48056-0.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100. ISSN 0305-0548.

Morris, S. (2005). *Interactive TV standards: a guide to MHP, OCAP, and JavaTV*. Elsevier, San Diego, CA.

Pessoa, B. J. S., Aloise, D., e Cabral, L. A. (2018). The weighted fair sequences problem. *Computers and Operations Research*, 91(Supplement C):121 – 131. ISSN 0305-0548. URL `http://www.sciencedirect.com/science/article/pii/S0305054817302848`.

Waldspurger, C. A. e Weihl. W., E. (1995). Stride scheduling: Deterministic proportional-share resource management. Technical report, Cambridge, MA, USA.

https://proceedings.science/p/85285?lang=pt-br