LII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2020

UMA META-HEURÍSTICA PARA O PROBLEMA DE ESCALONAMENTO DE TAREFAS EM UMA MÁQUINA COM DATAS DE LIBERAÇÃO

Rafael Morais, Lenildo L. Carlos, Teobaldo Bulhões, Anand Subramanian
Universidade Federal da Paraíba
Centro de Informática, Universidade Federal da Paraíba (UFPB) – R. dos Escoteiros, s/n – Mangabeira, João Pessoa – PB, CEP 58055-000
rafaelsobralm@gmail.com, lenildoluan@gmail.com,
tbulhoes@ci.ufpb.br, anand@ci.ufpb.br

RESUMO

O sequenciamento de tarefas que utilizam um ou mais recursos é uma atividade corriqueira em ambientes industriais que deve ser realizada eficientemente. Por isso, há na literatura uma vasta gama de problemas que buscam modelar tais atividades. Dentre eles, destaca-se o problema de escalonamento de tarefas em uma única máquina, sujeito a datas de liberação e tempos de *setup* dependentes da sequência, com objetivo de minimizar o tempo da última tarefa executada (*makespan*). Este trabalho propõe uma meta-heurística para tal problema que utiliza uma estrutura de dados que permite a avaliação de movimentos em tempo constante amortizado durante a busca local. Experimentos computacionais realizados com um grupo de 1800 instâncias demonstram a competitividade do algoritmo proposto frente aos três melhores algoritmos encontrados na literatura. Em particular, o método desenvolvido foi capaz de melhorar a melhor solução conhecida em 1123 das 1800 instâncias testadas.

PALAVRAS CHAVE. ***Scheduling*. Tempos de *Setups* dependentes da sequência.** ***Iterated Local Search*.**

Meta-heurísticas, Otimização Combinatória

ABSTRACT

The scheduling of jobs that use one or more resources is a common activity in industrial environments that should be performed efficiently. As a consequence, one can find a wide range of problems that seek to model such activities in the literature. Among them, we highlight the problem of scheduling jobs on a single machine, subject to release dates and sequence-dependent setup times, in order to minimize the makespan. We propose a metaheuristic for this problem that uses a data structure that allows the evaluation of movements in amortized constant time during the local search. Computational experiments over 1800 instances attest the competitiveness of the proposed algorithm when compared to the three best algorithms found in the literature. In particular, our method was capable of improving the best known solution for 1123 out of the 1800 instances tested.

KEYWORDS. Scheduling. Sequence-dependent setup times. Iterated Local Search.

Metaheuristics, Combinatorial Optimization

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

1. Introdução

Em diversas situações é necessário alocar recursos para a realização de alguma tarefa durante certo intervalo de tempo. Isso pode ser visto em processos de manufatura, nos quais diferentes serviços devem ser feitos. Muitas vezes, recursos como maquinário e ferramentas são limitados, devendo portanto serem compartilhados. Desta forma, o sequenciamento e agendamento das tarefas que irão utilizar um recurso específico pode ser feito visando maior eficiência na execução ou conclusão do processo. Por este motivo, vários trabalhos visam modelar diversas situações práticas possíveis.

Em um real processo de manufatura, é possível que nem todas as tarefas estejam disponíveis para realização a qualquer momento. Por exemplo, a execução de uma tarefa pode ser limitada a um profissional específico. Sendo assim, essa tarefa só pode ser realizada a partir do momento que esse funcionário se faz presente. Neste caso, o problema de agendamento apresenta datas de liberação (*release dates*), isto é, cada tarefa possui um horário mínimo de inicio, não podendo ser processada antes desse horário. Outra característica que se faz presente em processos em que diversas tarefas compartilham de um mesmo recurso para serem executadas é o tempo de preparação (*setup time*). Supondo que vários trabalhos diferentes são executados em mesma uma máquina em diferentes momentos, pode ser necessário um tempo de preparação para configurar o ambiente de trabalho para a tarefa a ser executada, sendo possível que tal tempo varie de acordo com a tarefa imediatamente anterior (*sequence dependent setup times*). Caso a preparação para uma tarefa possa ser realizada antes do lançamento, é dito que o *setup* é antecipatório. O caso contrário é referido como *setup* não antecipatório. Muitos trabalhos assumem que os tempos de preparação são irrelevantes, o que facilita a resolução dos problemas tratados, porém a incorporação deles nas decisões de escalonamento representa grande economia em várias situações práticas em serviços e ambientes industriais [Allahverdi et al., 2008].

Este trabalho lida com o problema de escalonamento de tarefas em uma única máquina com datas de liberação, tempos de preparação dependentes da sequência e com o objetivo de minimizar o tempo de conclusão da última tarefa. Considera-se que os *setups* são não antecipatórios. O problema em questão pertence à classe NP-Díficil [Pinedo, 2012] e, de acordo com a clássica notação de Graham et al. [1979], é denotado por $1|r_j, s_{ij}|C_{max}$. Para resolvê-lo, propõe-se um algoritmo heurístico baseado na meta-heurística ILS (do Inglês *Iterated Local Search*) e no algoritmo de descida em vizinhança variável com escolha aleatória RVND [Subramanian et al., 2010] (do Inglês *Randomized Variable Neighborhood Descent*). Experimentos computacionais realizados com um conjunto de 1800 instâncias disponíveis na literatura [Ovacikt e Uzsoy, 1994] demonstram que o algoritmo foi capaz de encontrar melhores soluções para diversas instâncias.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta alguns trabalhos relacionados ao problema tratado. A Seção 3 detalha a definição do problema. A Seção 4 descreve o algoritmo proposto. Na Seção 5 são apresentados os experimentos realizados e os resultados alcançados. Finalmente, a Seção 6 apresenta algumas considerações finais.

2. Trabalhos Relacionados

Os primeiros trabalhos envolvendo tempos de *setup* dependentes da sequência [Gilmore e Gomory, 1964; Presby e Wolfson, 1967] propuseram soluções baseadas no problema do caixeiro viajante (PCV), aproveitando as semelhanças entre os dois problemas, uma vez que os tempos de *setup* podem ser comparados às distâncias entre pontos no PCV. Os trabalhos de [Allahverdi et al., 1999, 2008; Allahverdi, 2015] apresentam revisões de diversos trabalhos na área dos problemas de *scheduling* com *setup times*.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

Potts [1980] propôs duas abordagens heurísticas para o problema de *scheduling* com única máquina visando minimizar o tempo de conclusão da última tarefa. O trabalho de Farn e Muhlemann [1979] apresenta várias heurísticas com objetivo de minimizar o tempo total de *setup*. Bianco et al. [1988] desenvolveram um algoritmo *Branch-and-Bound* e uma solução utilizando MILP (do Inglês *Mixed Integer Linear Programing*) para o problema considerando *setup* antecipatório, no qual a preparação para realização da tarefa pode ser iniciada antes de seu lançamento no sistema. Ovacikt e Uzsoy [1994] propuseram um algoritmo de decomposição para o problema de minimizar o atraso máximo ($1|r_j, s_{ij}|L_{max}$). O mesmo problema foi abordado por Asano e Ohta [1996] e Shin et al. [2002], que desenvolveram um algoritmo *Branch-and-Bound* e uma heurística de Busca Tabu, respectivamente.

O problema de minimizar o atraso ponderado ($1|s_{ij}|\Sigma w_j T_j$) foi abordado por Chang et al. [2004], que desenvolveram um algoritmo heurístico. O trabalho de Xu et al. [2013] lida com o problema através de um algoritmo ILS. Por sua vez, os trabalhos de Subramanian et al. [2014] e Subramanian e Farias [2017] apresentam a meta-heurística ILS, que se mostrou competitiva quando comparada a outros algoritmos presentes na literatura.

Montoya-Torres et al. [2012] apresentam uma heurística de inserções aleatórias para o problema com uma máquina e também com máquinas paralelas, mas considerando que o tempo de *setup* da primeira tarefa é inexistente, ou seja, inicialmente a máquina está preparada para processar qualquer tarefa. O trabalho de Velez-Gallego et al. [2016] apresenta uma solução com MILP e um algoritmo *Beam Search* (BS) para o problema de minimizar o *makespan* em um *scheduling* para máquina única, com *release dates* e tempos de *setup* dependentes da sequência, considerando *setup* não antecipatório, sendo o mesmo problema abordado neste trabalho. Recentemente, outra solução para o este problema foi apresentada por [Fan et al., 2019], que propõe uma heurística de inserção de bloco variável (*Variable Block Insertion Heuristic*, VBIH).

3. Definição do Problema

Seja $J = \{1, 2, \ldots, n\}$ um conjunto de n tarefas que devem ser executadas em uma única máquina. Cada tarefa $j \in J$ possui um tempo de processamento p_j e uma data de lançamento r_j, isto é, o momento em que a tarefa passa a estar disponível no sistema. Antes do início do processamento de uma tarefa, gasta-se um tempo configurando a máquina, sendo este tempo dependente da tarefa imediatamente anterior. A notação s_{ij} indica o tempo de preparação na transição da tarefa i para a j. Caso a tarefa j seja a primeira do escalonamento, considera-se um tempo de configuração inicial q_j. São considerados tempos de preparação *não antecipatórios*, isto é, a configuração que antecede o processamento de uma tarefa j só pode ser iniciada a partir da data de lançamento r_j.

Uma solução é definida por uma sequência $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ em que cada tarefa $j \in J$ é realizada exatamente uma vez. O tempo de conclusão de uma tarefa $j \in J$, denotado por C_j, é dado por $C_j = r_j + q_j + p_j$, se j for a primeira tarefa da sequência, ou seja, $j = \sigma_1$. Caso contrário, tem-se $C_j = \max(C_i, r_j) + s_{ij} + p_j$, sendo i a tarefa que antecede j no escalonamento. O problema consiste em encontrar uma solução que minimize o tempo de conclusão da última tarefa (*makespan*) dado por $C_{max} = \max_{j \in J}\{C_j\}$.

4. Algoritmo Proposto

Esta seção descreve o método proposto para resolução do problema $1|r_j, s_{ij}|C_{max}$ com *setups* não antecipatórios. O algoritmo é baseado na meta-heurística ILS, utilizando na fase de busca local o procedimento de descida em vizinhança variável com escolha aleatória (RVND), uma variação do procedimento proposto por Mladenovic e Hansen [1997]. Conforme detalhado no Algoritmo 1, em cada iteração gera-se uma solução inicial que é iterativamente melhorada através de procedimentos de busca local e de perturbação, sendo estes necessários para que o método escape de ótimos locais e, portanto, diversifique as soluções encontradas.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

Algoritmo 1 ILS

```
1:  procedimento ILS(I_R, I_ILS)
2:      f* ← ∞
3:      para i ← 1, ..., I_R faça
4:          α ← Valor aleatório ∈ {0, ..., 0.5}
5:          s ← Construção(α)
6:          s' ← s
7:          IterILS ← 0
8:          enquanto IterILS < I_ILS faça
9:              s ← RVND(s)
10:             se f(s) < f(s') então
11:                 s' ← s
12:                 IterILS ← 0
13:             s ← Perturbação(s')
14:             IterILS ← IterILS + 1
15:         se f(s') < f(s*) então
16:             s* ← s'
17:             f* ← f(s')
18:     retorne s*
```

4.1. Procedimento construtivo

O procedimento construtivo utilizado para gerar soluções iniciais é descrito no Algoritmo 2. Inicialmente, constrói-se uma lista de candidatos composta por todas as tarefas, a qual é mantida em ordem crescente de *release date*. Uma lista de candidatos restrita é criada considerando apenas os $\alpha\%$ melhores candidatos, ou seja, aqueles com os menores valores de *release date*. Uma tarefa dessa lista é selecionada aleatoriamente e adicionada à solução parcial como sendo a tarefa inicial. Após a primeira inserção, as listas passam a ser ordenadas de acordo com o tempo ocioso gerado pela inserção das tarefas na solução parcial, e todo o processo é repetido até a obtenção de uma solução completa. Ao fim da construção, cria-se a estrutura de subsequências abordada na Seção 4.2.2.

Algoritmo 2 Construção

```
1:  procedimento CONSTRUÇÃO(α)
2:      s ← {};
3:      CL ← Lista de todas as tarefas;
4:      enquanto CL ≠ ∅ faça
5:          se s = ∅ então
6:              Ordenar CL de forma crescente de acordo com o release date;
7:          senão
8:              Ordenar CL de forma crescente de acordo com o idle time;
9:          Criar uma lista RCL considerando apenas os α% primeiros candidatos;
10:         Escolher uma tarefa j ∈ RCL aleatoriamente;
11:         s ← s ∪ {j};
12:         CL ← CL − {j};
13:     retorne s
```

4.2. Busca Local

4.2.1. Estruturas de Vizinhança

A fase de busca local se dá pela exaustão de um conjunto de vizinhanças seguindo o método RVND [Subramanian et al., 2010]. Quando uma das vizinhanças não consegue melhorar a solução atual, outra vizinhança do grupo é selecionada aleatoriamente. Em caso de melhora, todas as vizinhanças passam a ser consideradas novamente. O conjunto é formado pelas seguintes vizinhanças:

- *Swap*: realiza a troca de posição entre duas tarefas tarefas.
- *l-block insertion*: um bloco com l tarefas adjacentes é realocado em outra posição da solução.

4.2.2. Avaliação Eficiente dos Movimentos

O processo de avaliação de uma vizinhança envolve calcular o *makespan* para cada movimento a fim de se determinar se houve melhora da solução atual. Calcular o custo diretamente envolve percorrer toda a sequência calculando os tempos de término de cada tarefa. Dessa forma, o procedimento de avaliação requer $\mathcal{O}(n)$ operações. Como cada estrutura de vizinhança utilizada possui $\mathcal{O}(n^2)$ movimentos a serem avaliados, a complexidade do algoritmo de busca em um dada vizinhança é $\mathcal{O}(n^3)$. Para evitar esse alto custo computacional, este trabalho propõe um método de avaliação dos movimentos através da concatenação de subsequências de tarefas, tendo como inspiração os métodos utilizados nos trabalhos de [Kindervater e Savelsbergh, 1997] e [Vidal et al., 2011]. Com isso, a avaliação de um movimento passa a ser realizada em tempo constante amortizado, o que implica reduzir a complexidade da busca em uma vizinhança para $\mathcal{O}(n^2)$.

Para cada subsequência $\sigma = (\sigma_1, \sigma_2, ..., \sigma_{|\sigma|})$ de uma solução, são armazenados os seguintes atributos:

- $P(\sigma) =$ primeira tarefa;
- $U(\sigma) =$ última tarefa;
- $E(\sigma) =$ tempo mínimo de ínicio para processar as tarefas sem tempo ocioso;
- $D(\sigma) =$ somas dos tempos de processamento e dos tempo de preparação.

Seja $\sigma = (j)$ uma subsequência unitária composta apenas pela tarefa $j \in J$. Logo, $P(\sigma) = j$, $U(\sigma) = j$, $E(\sigma) = r_j$ e $D(\sigma) = p_j$. Qualquer subsequência σ com $|\sigma| > 1$ pode ser formada pela concatenação de duas subsequências menores. Ou seja, é possível determinar σ^1 e σ^2 de modo que $\sigma = \sigma^1 \oplus \sigma^2$, sendo $\oplus$ o operador de concatenação detalhado no Algoritmo 3. Observe que o tempo mínimo de conclusão de uma subsequência — ou seja, seu *makespan* — é dado por $C(\sigma) = E(\sigma) + D(\sigma)$. Pode-se interpretar que todo o tempo ocioso é feito antes de ser iniciar o processamento das tarefas da subsequência. No entanto, o *makespan* de uma solução s representada pela subsequência $\sigma = (\sigma_1, \sigma_2, ..., \sigma_{|\sigma|})$ não necessariamente corresponde a $C(\sigma)$, uma vez que este valor não considera o *setup* da primeira tarefa. Na verdade, o *makespan* de s é igual a $C(\sigma' \oplus \sigma)$, sendo σ' uma subsequência unitária que contém uma tarefa artificial 0 tal que: $r_0 = 0$, $p_0 = 0$ e $s_{0j} = q_j$, para todo $j \in J$.

Na realização da operação $\sigma^1 \oplus \sigma^2$, existem diferentes casos em relação ao tempo ocioso. Na primeira situação (Figura 1), o menor tempo possível para conclusão da primeira subsequência ($C(\sigma^1)$) é maior ou igual ao tempo mínimo de início da segunda ($E(\sigma^2)$). Assim, não há tempo

ocioso além do necessário para se iniciar a primeira sequência. No segundo caso, a primeira subsequência é concluída em um tempo menor que $E(\sigma^2)$. É possível que todo o tempo entre $C(\sigma^1)$ e $E(\sigma^2)$ seja ocioso, conforme ilustrado na Figura 2. Porém, a ociosidade pode ser reduzida se o *setup* da primeira tarefa puder se iniciado antes de $E(\sigma^2)$, conforme ilustrado na Figura 3.

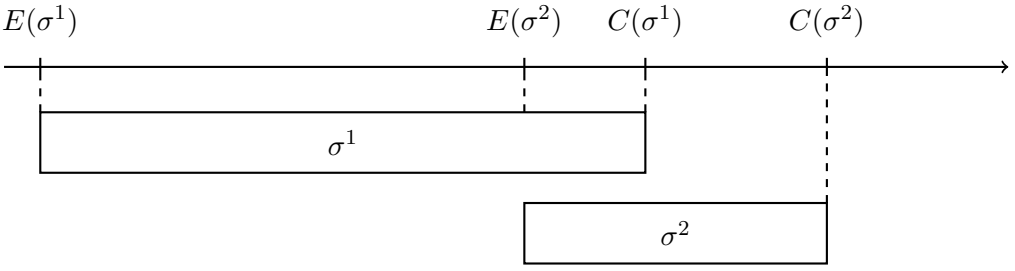


Figura 1: Concatenação em que $C(\sigma^1) \geq E(\sigma^2)$. O tempo ocioso da subsequência resultante deriva apenas de σ^1.

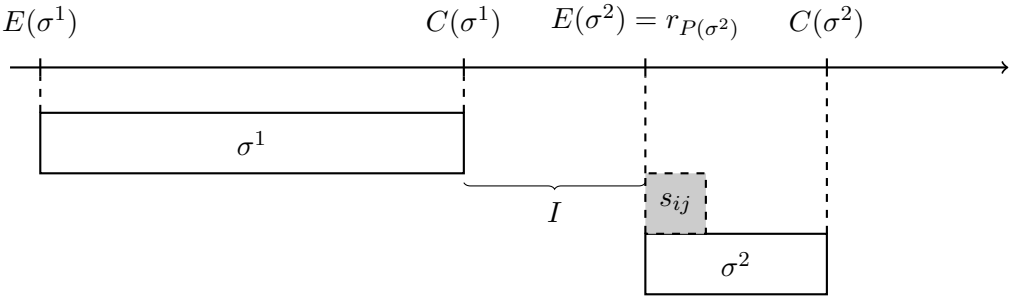


Figura 2: Um caso de concatenação em que $C(\sigma^1) \leq E(\sigma^2)$: o *setup* entre a última tarefa de σ^1 e a primeira de σ^2 não pode ser iniciado antes de $E(\sigma^2)$.

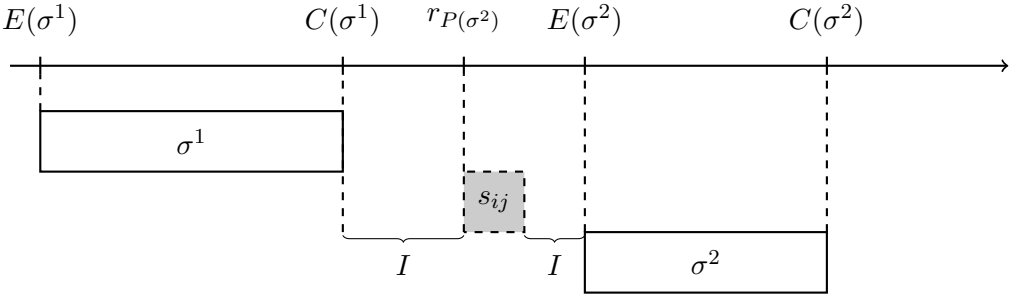


Figura 3: Outro caso de concatenação em que $C(\sigma^1) \leq E(\sigma^2)$: *setup* entre a última tarefa de σ^1 e a primeira de σ^2 pode ser iniciado antes de $E(\sigma^2)$, reduzindo o tempo ocioso. Note que ambos os tempos ociosos (antes e depois do *setup*) podem ser nulos.

4.3. Mecanismo de Perturbação

Quando a busca local não encontra uma solução melhor, um procedimento de perturbação é executado sobre a melhor solução da iteração correspondente. O procedimento utilizado, chamado *Double Bridge*, foi desenvolvido por Martin et al. [1991], e consiste na permutação de duas

Algoritmo 3 Concatenação $\sigma^1 \oplus \sigma^2$

procedimento CONCATENA SUBSEQUENCIAS(σ^1, σ^2)
  $P(\sigma^1 \oplus \sigma^2) \leftarrow P(\sigma^1)$;
  $U(\sigma^1 \oplus \sigma^2) \leftarrow U(\sigma^2)$;
  $I \leftarrow 0$;
  $i, j \leftarrow U(\sigma^1), P(\sigma^2)$;
  se $C(\sigma^1) < E(\sigma^2)$ **então**
    se $C(\sigma^1) < r_j$ **então**
      $I \leftarrow r_j - C(\sigma^1)$;
    se $C(\sigma^1) + I + s_{ij} < E(\sigma^2)$ **então**
      $I \leftarrow E(\sigma^2) - (C(\sigma^1) + s_{ij})$;
  $E(\sigma^1 \oplus \sigma^2) \leftarrow E(\sigma^1) + I$;
  $D(\sigma^1 \oplus \sigma^2) \leftarrow D(\sigma^1) + D(\sigma^2) + s_{ij}$;
  retorne $\sigma^1 \oplus \sigma^2$

subsequências escolhidas aleatoriamente, sendo comumente empregado no Problema do Caixeiro Viajante. Para o problema abordado neste trabalho, definiu-se $n/10$ como o tamanho máximo das subsequências a serem trocadas.

5. Resultados Computacionais

O algoritmo foi implementado com a linguagem C++ (g++ 5.4.0) e executado em um PC Intel®Core™ i7-2600 3,40 GHz com 16 GB de memória RAM e sistema operacional Ubuntu 16.04 64 bits (kernel Linux 4.15.0-96). Foi utilizado o conjunto de instâncias descrito em [Ovacikt e Uzsoy, 1994], que consiste em 1800 instâncias com $n \in \{25, 50, 75, 100, 125, 150\}$. As instâncias também são divididas entre 5 diferentes fatores de dispersão de *release date* no tempo: $R \in \{0,2; 0,4; 1,0; 1,4; 1,8\}$. A fim de avaliar o desempenho do algoritmo, foram conduzidos testes e comparações com a heurística *Beam Search* [Velez-Gallego et al., 2016] e o VBIH desenvolvido por Fan et al. [2019], cujo trabalho também inclui a implementação de um algoritmo *Iterated Greedy* proposto por Ruiz e Stützle [2007], também utilizado para fins comparativos. Em todos os testes, o algoritmo ILS foi executado 10 vezes para cada instância e foram utilizados os seguintes parâmetros: $I_R = 20$ e $I_{ILS} = 100$.

5.1. Definição do tamanho dos blocos

Um dos movimentos utilizados pela busca local descrita na Seção 4.2 é o *l-block insertion*, que consiste na realocação de um bloco contendo l tarefas adjacentes. Para definir o tamanho máximo dos blocos, foram realizados testes em 45 instâncias com $n \in \{50, 75, 100\}$, variando entre todos os fatores de dispersão de *release date* presentes no conjunto de instâncias original. Os testes foram executados utilizando os mesmos parâmetros informados anteriormente. A Tabela 1 apresenta um resumo dos resultados obtidos.

Tabela 1: Resultados com diferentes tamanhos máximos de bloco (l)

l	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
#Melhor	7	5	7	8	15	13	18	21	24	27	25	28	25	29	30	29
Gap Médio (%)	0,49	0,44	0,33	0,19	0,12	0,09	0,06	0,01	0,00	-0,02	-0,03	-0,04	-0,02	-0,03	-0,06	-0,07
Tempo (50)	3,06	3,06	3,19	3,46	3,60	3,92	4,07	4,28	4,47	4,74	4,94	5,14	5,27	5,45	5,67	5,81
Tempo (75)	7,65	7,66	8,54	9,52	10,63	11,43	11,95	12,87	13,36	14,43	15,16	16,11	16,63	17,33	18,02	18,50
Tempo (100)	17,62	17,69	20,12	22,23	24,34	25,93	27,81	30,30	32,02	33,80	35,59	37,49	39,22	41,35	42,48	44,53

É possível notar que, aumentando-se o tamanho máximo do bloco, ocorre também o aumento na quantidade de vizinhanças no RVND, o que impacta diretamente no tempo de execução do algoritmo. Desta forma, apesar dos resultados indicarem que o *gap* em relação às melhores soluções conhecidas vai diminuindo conforme são adicionados novos tamanhos de bloco, foi definido que o tamanho máximo de bloco na versão final seria $l = 13$, considerando que nesta configuração o algoritmo foi, em média, 7 segundos mais rápido que a configuração com $l = 17$, e que foram encontradas melhores soluções para 28 das 45 instâncias, próximo do resultado obtido com os maiores blocos.

5.2. Comparação com algoritmos da literatura

Para as instâncias com até 75 tarefas, os resultados foram comparados aos dos três algoritmos mencionados anteriormente. Porém, para as instâncias maiores, a comparação é feita somente com o *Beam Search*, pois é o único trabalho que apresenta resultados para nesse caso. Para cada instância, foi computado o *gap* entre as melhores soluções encontradas pelo ILS e os algoritmos já mencionados. Os *gaps* médios são apresentados na Tabela 2, tendo cada grupo 60 instâncias.

Em média, o ILS conseguiu encontrar soluções de melhor qualidade para quase todo conjunto de instâncias com até 75 tarefas, sendo superado apenas pelo VBIH para as instâncias de 25 e 50 tarefas com menor dispersão de *release date*. No total, o ILS encontrou soluções melhores que o VBIH em 474 instâncias, e igualou em 329. Comparado ao *Beam Search*, foram 570 melhoras e 220 empates. Por fim, foram 594 melhoras e 300 empates na comparação com o *Iterated Greedy*. A Tabela 5 apresenta a quantidade de melhores soluções encontradas por cada um dos métodos envolvidos nas comparações.

Tabela 2: *Gap* médio do ILS comparado ao VBIH,BS e IG

	$n = 25$			$n = 50$			$n = 75$		
R	**VBIH**	**BS**	**IG**	**VBIH**	**BS**	**IG**	**VBIH**	**BS**	**IG**
0,2	0,00	-0,12	-0,11	0,02	-0,52	-0,89	-0,04	-0,50	-1,44
0,6	-0,17	-0,10	-0,09	-0,16	-0,30	-0,95	-0,30	-0,25	-1,46
1	-0,27	-0,08	-0,06	-0,48	-0,29	-0,70	-0,54	-0,19	-1,14
1,4	-0,23	-0,14	-0,04	-0,57	-0,69	-0,55	-0,65	-0,45	-0,93
1,8	-0,46	-0,64	-0,02	-0,63	-0,70	-0,60	-0,71	-0,35	-0,82

Tabela 3: *Gap* médio do ILS comparado ao BS

R	$n = 100$	$n = 125$	$n = 150$
0,2	-0,37	-0,29	-0,06
0,6	-0,09	0,08	0,26
1	0,09	0,16	0,49
1,4	-0,26	-0,15	0,18
1,8	-0,22	0,05	0,27

Os tempos de execução médios para cada tamanho de instância são apresentados na Tabela 4, juntamente com os tempos computacionais do algoritmo de *Beam Search* reportados por Velez-Gallego et al. [2016]. Estes tempos foram obtidos em um PC Intel®Xeon™ octa-core 2,40 GHz com 16 GB de memória RAM e sistema operacional Linux 64 bits.

Tabela 4: Tempos de execução médios em segundos

n	**ILS**	**BS**
25	0,48	0,12
50	4,00	0,75
75	13,68	2,24
100	33,43	4,99
125	66,47	10,17
150	97,58	19,24

Com relação as instâncias com mais de 75 tarefas, o ILS conseguiu melhorar a solução encontrada pelo *Beam Search* em 417 das 900 instâncias desse grupo, e apenas 7 empates ocorreram. Contudo, quando analisado o *gap* médio, observa-se que o BS obtém vantagem em 9 dos 15 subconjuntos de instâncias. O *Beam Search* se mostra mais eficiente que o ILS principalmente nas instâncias com $n = 150$. As Tabelas 3 e 6 apresentam o *gap* médio em relação ao BS e a quantidade de melhores soluções encontradas, respectivamente.

Tabela 5: Quantidade de instâncias em que a melhor solução conhecida foi encontrada ($n <= 75$)

	n = 25				n = 50				n = 75			
R	**ILS**	**VBIH**	**BS**	**IG**	**ILS**	**VBIH**	**BS**	**IG**	**ILS**	**VBIH**	**BS**	**IG**
0,2	60	60	33	39	43	55	2	0	34	24	3	0
0,6	60	47	40	44	40	32	11	0	38	12	12	0
1	60	40	42	50	48	19	8	3	34	11	15	0
1,4	59	37	44	51	49	11	10	15	42	9	18	10
1,8	59	39	40	54	45	15	20	21	32	10	27	18

Tabela 6: Quantidade de instâncias em que a melhor solução conhecida foi encontrada ($n > 75$)

	n = 100		n =125		n = 150	
R	**ILS**	**BS**	**ILS**	**BS**	**ILS**	**BS**
0,2	56	4	55	5	41	21
0,6	41	21	31	29	11	49
1	23	38	19	41	6	54
1,4	28	32	21	40	21	39
1,8	28	33	21	38	21	39

6. Considerações Finais

O presente trabalho aborda o problema de *scheduling* para única máquina com *release dates* e tempos de *setup* dependentes da sequencia $(1|r_j, s_{ij}|C_{max})$, propondo uma abordagem baseada na meta-heurística ILS (*Iterated Local Search*) combinada ao *Randomized Variable Neighborhood Descent* (RVND). Foram realizados testes em um grupo de 1800 instâncias, e os resultados foram comparados a outros 3 métodos encontrados na literatura. O algoritmo proposto se mostrou competitivo, sendo capaz de encontrar a melhor solução dentre os três métodos para 1123 instâncias. No entanto, o ILS teve dificuldade em encontrar soluções para as maiores instâncias do grupo, nas quais o algoritmo *Beam Search* se mostrou mais eficiente.

Para trabalhos futuros, podem ser considerados métodos para melhorar o desempenho do algoritmo em instâncias maiores, reduzindo o tempo de execução e melhorando a qualidade de solução. Também é possível considerar uma adaptação do algoritmo para o problema envolvendo mais de uma máquina.

Referências

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Allahverdi, A., Gupta, J. N. D., e Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, 27(2):219–239.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., e Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187:985–1032.

Asano, M. e Ohta, H. (1996). Single machine scheduling using dominance relation to minimize earliness subject to ready and due times. *International Journal of Production Economics*, 44(1): 35–43.

Bianco, L., Ricciardelli, S., Rinaldi, G., e Sassano, A. (1988). Scheduling tasks with sequence-dependent processing times. *Naval Research Logistics*, 35(2):177–184.

Chang, T.-Y., Chou, F.-D., e Lee, C.-E. (2004). A heuristic algorithm to minimize total weighted tardiness on a single machine with release dates and sequence-dependent setup times. *Journal of the Chinese Institute of Industrial Engineers*, 21(3):289–300.

Fan, J., Öztop, H., Tasgetiren, M., e Gao, L. (2019). A variable block insertion heuristic for single machine with release dates and sequence dependent setup times for makespan minimization. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, p. 1676–1683, Xiamen, China. IEEE.

Farn, C. e Muhlemann, A. (1979). The dynamic aspects of a production scheduling problem. *Int. J. Prod. Res.*, 17(1):15–21.

Gilmore, P. e Gomory, R. (1964). Sequencing a one state-variable machine: A solvable case of the traveling salesman problem. *Operations Research*, 12(5):655–679.

Graham, R., Lawler, E., Lenstra, J., e Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E. J. e Korte, B., editors, *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, volume 5 of *Annals of Discrete Mathematics*, p. 287 – 326. Elsevier.

Kindervater, G. e Savelsbergh, M. (1997). Vehicle routing: handling edge exchanges. *In: Aarts, E., Lenstra, J. (Eds.), Local Search in Combinatorial Optimization. Wiley, New York*, p. 337–360.

Martin, O., Otto, S., e Felten, E. (1991). Large-step markov chains for the traveling salesman problem. *Complex Systems*, 5(3):299–326.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings

Mladenovic, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Montoya-Torres, J., González-Solano, F., e Soto-Ferrari, M. (2012). Deterministic machine scheduling with release times and sequence-dependent setups using random-insertion heuristics. *International Journal of Advanced Operations Management*, 4(1/2):4–26.

Ovacikt, I. e Uzsoy, R. (1994). Rolling horizon algorithms for a single-machine dynamic scheduling problem with sequence-dependent setup times. *International Journal of Production Research*, 32(6):1243–1263.

Pinedo, M. L. (2012). *Scheduling: Theory, Algorithms, and Systems*. Springer-Verlag, Nova Iorque.

Potts, C. (1980). Analysis of a heuristic for one machine sequencing with release dates and delivery times. *Operations Research*, 28(6):1436–1441.

Presby, J. e Wolfson, M. (1967). An algorithm for solving job sequencing problems. *Management Science*, 13(8):454–464.

Ruiz, R. e Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Shin, H., Kim, C.-O., e Kim, S. (2002). A tabu search algorithm for single machine scheduling with release times, due dates, and sequence-dependent set-up times. *The International Journal of Advanced Manufacturing Technology*, 19(11):859–866.

Subramanian, A., Battarra, M., e Potts, C. N. (2014). An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. *Int. J. Prod. Res.*, 52(9):2729–2742.

Subramanian, A. e Farias, F. (2017). Efficient local search limitation strategy for single machine total weighted tardiness scheduling with sequence-dependent setup times. *Computers & Operations Research*, 79:190–206.

Subramanian, A., Drummond, L. M. d. A., Bentes, C., Ochi, L. S., e Farias, R. (2010). A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37(11):1899–1911.

Velez-Gallego, M. C., Maya, J., e Montoya-Torres, J. (2016). A beam search heuristic for scheduling a single machine with release dates and sequence dependent setup times to minimize the makespan. *Computers & Operations Research*, 73:132–140.

Vidal, T., Crainic, T., Gendreau, M., e Prins, C. (2011). A unifying view on timing problems and algorithms. Cirrelt tech.rep. 2011-43, CIRRELT.

Xu, H., Lü, Z., e T., C. (2013). Iterated local search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness. *J Sched*, 17:271–287.

Galoá { Este trabalho foi publicado utilizando o **Galoá** proceedings