

Proposta de um algoritmo para o problema de sequenciamento em máquina única com tempos de *setup* dependentes buscando minimizar o atraso ponderado total através do *Iterated Local Search*

Monique Tamara de Lima (UTFPR) monique_tamara@hotmail.com

Pedro Antonio de Albuquerque Felizola Romeral (UTFPR) pedroromeral@bol.com.br

Rafael Henrique Palma Lima (UTFPR) rafaelhlma@utfpr.edu.br

Resumo:

Este artigo propõe um algoritmo baseado na metaheurística *Iterated Local Search* (ILS) para o problema de sequenciamento da produção em uma máquina, com tempos de *setup* dependentes e com o objetivo de minimizar o tempo de atraso ponderado total. Na literatura, há autores que desenvolveram métodos exatos para a resolução deste problema, mas os tempos computacionais ainda são muito elevados. Por se tratar de um problema operacional, algoritmos para o sequenciamento de produção devem obter soluções mais rapidamente. O algoritmo proposto neste trabalho foi testado com duas configurações de parâmetros usando 10 instâncias propostas por Tanaka e Araki (2013), com um tempo médio de execução de 1 hora. Apesar de as soluções encontradas ainda não estarem próximas da solução ótima, o algoritmo foi considerado satisfatório, pois foi capaz de encontrar boas soluções em um curto espaço de tempo.

Palavras chaves: Sequenciamento da produção, Máquina única, Tempo de *setup* dependente; *Iterated Local Search*

Proposal of an *Iterated Local Search* algorithm for the single machine scheduling problem with sequence-dependent *setup* times to minimize total weighted tardiness

Abstract:

This paper proposes an *Iterated Local Search* (ILS) algorithm to tackle the production scheduling problem in a single machine with sequence-dependent *setup* times. The objective function seeks the minimization of the total weighted tardiness. In the literature, there are authors who have developed exact methods to solve this problem, but computational times are still very high. Because production scheduling is a problem that needs to be dealt with at the operational level, algorithms must obtain solutions quickly. The algorithm proposed in this work was tested with two parameter settings using 10 instances proposed by Tanaka and Araki (2013), with an average execution time of 1 hour. Although the solutions found are not yet close to the optimal solution, the algorithm was considered satisfactory, because it was able to find good solutions in a short time.

Keywords: Production scheduling, single-machine problems, sequence-dependent *setup* times, *Iterated Local Search*

1. Introdução

O advento da globalização permitiu comunicações mais eficientes e trocas de informações mais ágeis. Isto exigiu das indústrias um planejamento estratégico direcionado para a melhoria de processos, principalmente no Planejamento e Controle da Produção. Autores como Slack et al. (2015) e Velez-Gallego et al. (2016) salientam que investimentos em otimização podem trazer melhorias diretas nas atividades de programação da produção, demonstrando um grande potencial para a redução de custos e para diminuição do tempo de fabricação de um determinado produto.

Segundo Sule (2008), quando o preço do produto é determinado pela concorrência, a principal maneira de aumentar o lucro é reduzir os custos de produção e distribuição. Por isso, gerenciar e operar a organização de forma eficiente é essencial. Para o autor, o planejamento e programação da produção visam aumentar a eficiência na fabricação e melhorar a eficácia no atendimento ao cliente, propiciando melhorias na eficiência operacional.

Apesar do tempo de *setup* estar presente em vários processos produtivos, de acordo com Gupta e Smith (2006), grande parte dos algoritmos e técnicas de resolução de problemas de sequenciamento pressupõem que os tempos de *setup* são independentes da sequência de tarefas em uma máquina. Assume-se que os tempos de configuração são insignificantes ou são adicionados aos tempos de processamento das tarefas. Para os autores, no entanto, em muitos casos é necessário um gasto de tempo significativo para configurar máquinas conforme trocasse o produto a ser fabricado. Desta forma, a máquina processa muitos trabalhos diferentes e o tempo de configuração para um trabalho depende daquele que acabou de processar anteriormente.

Diante deste contexto, este trabalho tem como objetivo propor um novo algoritmo inspirado na técnica *Iterated Local Search* (ILS) para o problema de sequenciamento de produção em máquina única com tempos de *setup* dependentes da sequência. Para isso, são propostas estruturas de vizinhança e métodos de perturbação para obter soluções satisfatórias para o problema em um curto espaço de tempo. A qualidade do algoritmo foi testada usando 10 instâncias propostas por Tanaka and Araki (2013), para as quais se conhece a solução ótima. Ao final do artigo são feitas discussões sobre a qualidade do algoritmo e propostas para trabalhos futuros.

2. Referencial Teórico

Para Pinedo (2016), a programação da produção (*scheduling*) é um processo de tomada de decisão que é usado regularmente em muitas indústrias de todos os setores. Ela trata da alocação de recursos para tarefas em determinados períodos de tempo e seu objetivo é otimizar uma ou mais metas. O autor salienta que esta programação, como um processo de tomada de decisão, desempenha um papel importante na maioria dos sistemas de produção, bem como na maioria dos ambientes de processamento de informações.

Framinan (2014) afirma que as condições industriais são normalmente muito variáveis, podendo dificultar ou impedir o cumprimento de um cronograma, por mais detalhado e completo que seja, possivelmente causando revisão e modificação de toda a programação existente. O autor sugere que as decisões de programação da produção possuem curto intervalo de tempo a serem tomadas uma e outra vez, podendo ser considerado como um processo de agendamento contínuo.

Em suas pesquisas, Sule (2008) verificou que muitos pesquisadores contribuíram para o desenvolvimento de técnicas de programação e alguns temas e terminologias comuns foram

desenvolvidas como: Máquina única, *Flow shop*, Máquinas paralelas, *Job shop*, *Open shop*, Loja dependente, Processamento do lote, Tempos de configuração dependentes da sequência, Linha de montagem, Linha de montagem de modo misto, Modelos de planejamento de mão-de-obra.

Nagano et al (2004) caracterizam os problemas de programação nos sistemas produtivos por meio de variáveis como K (número de estágios da produção) e M_s (número de máquinas do estágio s), que caracterizam a complexidade para cada forma dos problemas. A figura a seguir mostra que para um problema de máquina única tem-se $K = 1$ e $M_1 = 1$, existindo apenas uma máquina e um estágio de produção.

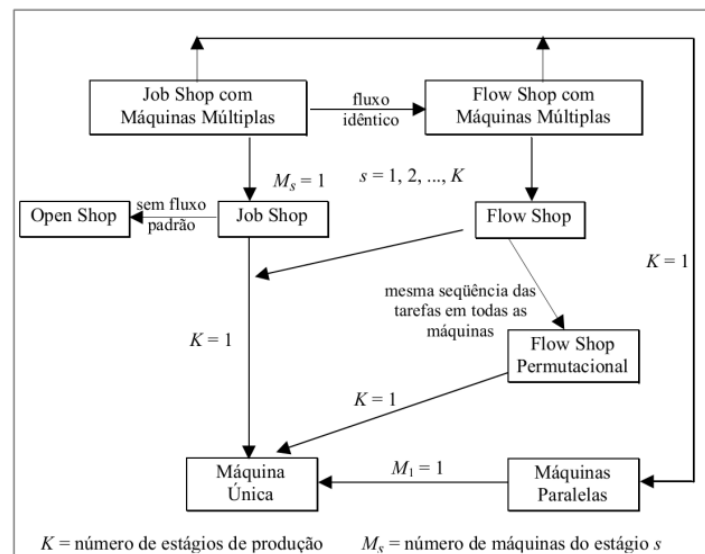


Figura 1 – Relação entre as classes de problemas de programação de operações em máquinas (Fonte: Nagano et al 2004)

Tanaka e Araki (2013) propuseram um algoritmo exato para resolver o problema de máquina única com atraso ponderado total e sequência dependente. É uma extensão do outro algoritmo dos autores que se baseia no método sucessivo de programação dinâmica de sublimação (SSDP) para o problema. É essencial que um esquema de ramificação seja incorporado no algoritmo exato para lidar com as instâncias difíceis. O algoritmo exato proposto é capaz de encontrar soluções ótimas para todas as 120 instâncias em tempos razoáveis, havendo dois casos que demoraram duas semanas ou mais para resolver.

Xu, Lü e Cheng (2013) utilizaram o algoritmo *Iterated Local Search* (ILS) que apresentava características distintivas, incluindo uma nova estrutura de vizinhança chamada Block Move e uma técnica de avaliação incremental rápida, para avaliação de soluções de vizinhança. Este estudo usou como referências as 120 instâncias do algoritmo exato de Tanaka e Araki (2013), conseguindo resultados altamente competitivos com os já obtidos.

Feili, Haddad, Ghanbari (2012) trabalharam com um modelo matemático para resolver o problema e, em seguida, duas metaheurísticas; Algoritmo Genético (GA) e *Simulated Annealing* (SA), obtendo resultados satisfatórios e sugerindo a utilização do *Hill Climbing* para futuras pesquisas.

Rodrigues e Pizzolato (2014) buscaram minimizar o *makespan* dos sequenciamentos de um problema de máquina única a partir da minimização dos Tempos de Preparação de Máquina, utilizando a metodologia do Problema do Caixeiro Viajante. Eles afirmam que ao tentar

minimizar algumas penalidades de atraso, pode-se incorrer em maiores *setups* e reduzir o aproveitamento da máquina.

Ferreira et al (2007) aplicaram a heurística *relax and fix* para otimizar problemas de gargalo no envase em uma fábrica de refrigerantes de pequeno porte e obtiveram redução de 37% do custo total, gerando soluções melhores que as fornecidas pela fábrica, mas ressaltam que o método possui limitações, citando a necessidade de se buscar métodos específicos de solução.

Uma heurística construtiva chamada ATCS (*Apparth Tardiness Cost with Setups*), foi desenvolvida para este problema por Lee et al. (1997), e teve um tempo de execução razoável para problemas de pequeno porte, porém para instâncias maiores a sua eficiência não foi adequada.

3. Descrição do algoritmo proposto

O problema tratado neste trabalho pode ser descrito da seguinte forma: uma máquina deve processar um conjunto de n tarefas, sendo que cada tarefa possui um tempo de processamento e um prazo máximo definido para o término do processamento; uma única máquina executa no máximo uma tarefa por vez não sendo permitida a interrupção do processamento; todas as tarefas estão disponíveis para processamento no início da programação; quando uma tarefa é sequenciada imediatamente após outra, é necessário um tempo para a preparação da máquina, o qual é dependente da ordem das tarefas; cada tarefa possui um peso representando sua importância; os trabalhos devem ser sequenciados com o objetivo de minimizar o tempo de atraso ponderado total (*total weighted tardiness*).

Para a resolução do presente estudo foi criado um algoritmo baseado nos conceitos do *Iterated Local Search* (ILS), que faz uso de perturbações na solução corrente com a finalidade de fugir de ótimos locais e explorar a vizinhança com maior profundidade (GLOVER; KOCHENBERGER, 2003). A Figura 2 apresenta o pseudocódigo do algoritmo proposto.

O primeiro passo do algoritmo foi criar um sequenciamento de produção em ordem de crescimento de prazo de entrega, conhecido como MDE ou EDD (*Menor Data de Entrega/Earliest Due Date*), que tem como finalidade reduzir o atraso máximo. Depois de criado esse sequenciamento, foram analisados os pesos referentes às prioridades de fabricação, onde as ordens de fabricação que possuíam pesos zeros, ou seja, que não haviam nenhuma prioridade, foram deslocadas de maneira que se tornassem as últimas ordens a serem realizadas, não levando em consideração os seus prazos. Esse sequenciamento foi denominado “Sequenciamento inicial”.

Para iniciar uma iteração faz-se perturbação do “Sequenciamento Inicial”, que consiste em trocar a posição de uma Ordem de Fabricação atrasada por uma adiantada, gerando-se a “Solução Candidata”. A escolha de quais ordens de fabricação em específico serão trocadas é realizada de maneira aleatória, enquanto a quantidade de trocas realizadas pode ser definida pelo usuário, sendo um importante parâmetro do algoritmo. No estudo essas trocas foram denominadas de “Troca de Pontos”. Deve-se ressaltar que o número de trocas realizadas tem grande impacto no resultado obtido, pois quanto maior este número; maior será a aleatoriedade do algoritmo.

Após definir o número de trocas também é necessário definir parâmetros como o número de iterações, que representa reinícios do algoritmo, e o número de perturbações que ocorre a cada iteração, sendo tentativas de fugir dos ótimos locais. Se o número de reinícios for baixo, pode

se usar um número maior de perturbações; caso contrário, pode-se fazer muitos reinícios, então o número máximo de perturbações terá que ser menor.

```
Sub OtimizaILS
  \\Cria Solução Inicial para o início da Iteração que sofrerá uma perturbação
  SeqInicial = GeraSeqInicial (prazo, peso, pontos, seqOf)
  AtrasoPonderado = GeraAtrasoPonderado(pontos, SeqInicial)

  Para k = 1 Até qtdIteracoes
    \\zerar solução da perturbação anterior a cada nova iteração
    AtrasoSolMelhorViz1 = Infinito
    AtrasoSolMelhorViz2 = Infinito
    SolCandidata = GeraPerturbacao(pontos, AtrasoPonderado, TrocaPontos, SeqInicial)
    Para i = 1 Até qtdPerturbacoes
      atrasoCandidata = AvaliaAtrasoPonderadoTotal(pontos, solCandidata)
      solVizinha1 = MelhorVizinho(pontos, Nvizinho, solCandidata)
      atrasoVizinha1 = AvaliaAtrasoPonderadoTotal (pontos, solVizinha1)
      \\testa se a "Solução Vizinha" é melhor que a "Solução Candidata"
      Se atrasoVizinha1 < atrasoCandidata Então
        solEscolhida = solVizinha1
        atrasoEscolhida = atrasoVizinha1
      Fim
      Se atrasoVizinha1 > atrasoCandidata Então
        solEscolhida = solCandidata
        atrasoEscolhida = atrasoCandidata
      Fim
      \\testa se a "Solução Escolhida" é melhor que a "Solução Melhor Vizinha 1"
      Se atrasoEscolhida < AtrasoSolMelhorViz1Então
        AtrasoSolMelhorViz1 = atrasoEscolhida
        SolMelhorViz1 = solEscolhida
        solVizinha2 = MelhorVizinho2(pontos, SolMelhorViz1)
        atrasoVizinha2 = AvaliaAtrasoPonderadoTotal (pontos, solVizinha2)
        \\testa se a "Solução Vizinha 2 é melhor que a "Solução Melhor Vizinha 2"
        Se atrasoVizinha2 < AtrasoSolMelhorViz2 Então
          AtrasoSolMelhorViz2 = atrasoVizinha2
          SolMelhorViz2 = solVizinha2
        Fim
      Fim
      \\Encontra a Melhor Solução encontrada na presente iteração
      Se atrasoEscolhida > AtrasoSolMelhorViz1 Então
        SolucaoMelhor1Iteracao = SolucaoMelhor1Iteração
      Fim
      Se AtrasoSolMelhorViz2<= AtrasoSolMelhorViz1 Então
        SolucaoMelhor1Iteracao = SolMelhorViz2
      Fim
      Se AtrasoSolMelhorViz2 > AtrasoSolMelhorViz1 Então
        SolucaoMelhor1Iteração = SolMelhorViz1
      Fim
      AtrSolMelhot1It = AvaliaAtrasoPonderadoTotal(pontos, SoluçãoMelhor1Iteração)
      \\Encontra a melhor solução de TODAS iterações
      Se AtrSolMelhot1It <= AtrasoGlobal Então
        SolucaoGlobal = SolucaoMelhor1Iteracao
        AtrasoGlobal = AtrSolMelhot1It
      Fim
    solCandidata = GeraPerturbacao (pontos, AtrasoPonderado, TrocaPontos, SolucaoMelhor1Iteração)
  Fim
Fim
```

Figura 2 – Pseudocódigo do algoritmo proposto

Com os parâmetros definidos e com a “Solução Candidata”, inicia-se a primeira perturbação. Na perturbação é aplicada uma busca de Vizinhança na “Solução Candidata”. Essa vizinhança consiste em avaliar os atrasos e os atrasos ponderados (atraso multiplicado pelo seu peso de prioridade referente). Ao analisar os atrasos ponderados, são identificadas as ordens que possuem os 2 maiores atrasos ponderados. Já ao analisar os atrasos das Ordens de Fabricação que possuem pesos maiores que zero, são identificadas quais dessas são as 2 mais adiantadas.

Consecutivamente, serão realizadas as trocas de posições entre ordens de sequenciamento dos mais atrasados, com os mais adiantados. Para determinar como será realizada essa troca, sorteia-se um número aleatório: caso esse número seja menor ou igual a 0,7 escolhe-se o mais atrasado para a troca, caso contrário é escolhido o segundo mais atrasado. Após isso, sorteia-se novamente um número aleatório: caso esse número seja menor ou igual a 0,7 se escolhe o mais adiantado, caso contrário se escolhe o segundo mais adiantado. Após feita a escolha das duas Ordens Fabricação, é realizada a troca de posição entre estas. No entanto, também é realizada as trocas de posição entre as Ordens de Fabricação que não foram escolhidas através do sorteio dos dois números aleatórios.

Para analisar o mais adiantado, não foi considerado o atraso ponderado, pois poderia ocorrer a escolha de uma ordem de fabricação que possuía um pequeno adiamento, porém devido à sua grande prioridade, apresentaria um grande atraso ponderado. Ao fazer a troca de posições, essa ordem de fabricação poderia apresentar um grande atraso, e como sua prioridade é grande, poderia ocorrer uma piora na solução, já que ela apresentaria um atraso ponderado expressivo. Também não foram levadas em consideração as Ordens de Fabricação que possuíam peso zero, pois o objetivo é deixá-las nas últimas posições do sequenciamento, já que elas não interferem no atraso ponderado total da solução.

Neste algoritmo, são gerados 20 vizinhos da “Solução Candidata”. Essa quantidade foi definida após diversos testes do algoritmo. Após cada vizinhança, é avaliada qual solução apresenta o menor atraso ponderado total (soma de todas as Ordens de Fabricação que possuem atraso ponderado) e a que apresentar o menor valor será “Solução Vizinha”.

Ao terminar a vizinhança é feita a comparação entre a “Solução Candidata” e a “Solução Vizinha”, afim de analisar se a vizinhança foi promissora ou não. Escolhe-se aquela solução que apresentar o menor atraso ponderado total. Este atraso da “Solução Escolhida” também será comparado com o da “Solução Melhor Vizinha 1”, que é uma variável responsável por armazenar a melhor “Solução Escolhida” dentro de uma iteração.

Caso a “Solução Escolhida” seja melhor que a “Solução Melhor Vizinha 1”, isso significa que esta “Solução Escolhida” é uma solução promissora, sendo assim é aplicada uma segunda busca na vizinhança, desta vez mais aprofundada. Essa segunda vizinhança não é realizada para todas as soluções candidatas, pois ela exige grande esforço computacional, o que acarretaria em um maior tempo de execução.

Esta segunda vizinhança consiste em escolher a ordem de fabricação que possui o maior atraso ponderado e adiantá-la até que ela não possua mais atraso. Essa vizinhança é feita 60 vezes, que é exatamente a mesma quantidade de ordens de fabricação. Essas vizinhanças são avaliadas e aquela que apresentar o menor atraso ponderado total será “Solução Vizinha 2”. Similar o procedimento feito na “Solução Vizinha 1”, a “Solução Vizinha 2” é confrontada com uma variável chamada “Solução Melhor Vizinha 2” afim de armazenar a melhor solução obtidas da “Soluções Vizinha 2” em 1 Iteração.

A “Solução Melhor Vizinha 2” será confrontada com a “Solução Melhor Vizinha 1”, a que apresentar a melhor resultado, será escolhida, que será a “Solução Melhor de 1 Iteração”, que é uma variável responsável por armazenar a melhor solução encontrada em 1 Iteração. E ainda “Solução Melhor de 1 Iteração” é comparada com a “Solução Global”, que é aquela que possui a melhor solução de todas iteração e não mais de apenas de 1 iteração.

Para se iniciar uma nova perturbação a “Solução Candidata” passa a ser a “Solução Melhor da Iteração”, entretanto com uma perturbação. Essa perturbação é realizada com os mesmos

procedimentos usados para perturbar o “Sequenciamento Inicial”. Iniciando um Loop até finalizar o número de perturbações.

Após o término das perturbações, se inicia uma nova Iteração, onde as soluções: “Solução Melhor Vizinha 1”, “Solução Melhor Vizinha 2” são deletadas. E a primeira “Solução Candidata” passa a ser novamente o “Sequenciamento inicial” após sofrer perturbação. Iniciando assim a primeira perturbação da nova iteração, na qual será realizado o mesmo procedimento da iteração anterior. Esse método é realizado até que todas interações se finalizem. Após o término de todas as iterações, o algoritmo retorna o sequenciamento da melhor solução global encontrada juntamente com seu atraso ponderado total.

4. Resultados obtidos

O algoritmo proposto foi implementado utilizando a linguagem Excel VBA e foi executado em um computador com processador Intel i7 de 6ª Geração, com 8 GB de memória RAM. Para testar o desempenho do algoritmo foram utilizadas as 10 primeiras instâncias propostas por Tanaka e Araki (2013). Tais autores conseguiram obter as soluções ótimas dessas instâncias, porém com altos tempos de execução. Ao executar o algoritmo proposto neste trabalho, buscou-se limitar a quantidade de iterações e perturbações para que o tempo total girasse em torno de 1 hora. Para obtenção dos resultados, foram utilizadas duas configurações diferentes:

- **Configuração 1:** 3 iterações, com 1 “troca de pontos” e 420 perturbações por iteração;
- **Configuração 2:** 8 iterações, com 3 “troca de pontos” e 200 perturbações por iteração.

É possível notar que a Configuração 1 tem como característica uma busca mais aprofundada e menos aleatorizada por iteração, pois realiza menos trocas de pontos e tem mais perturbações por iteração. A Tabela 1 compara as soluções ótimas conhecidas com os resultados obtidos pelo algoritmo proposto com os parâmetros definidos de acordo com a primeira configuração.

Inst.	Solução Ótima	Solução Obtida 1	Tempo (hh:mm)	Sequenciamento
1	453	6.436	00:42	26, 36, 39, 56, 17, 6, 47, 8, 11, 58, 53, 52, 50, 44, 43, 4, 37, 1, 7, 30, 12, 31, 2, 28, 3, 22, 38, 27, 20, 25, 10, 9, 0, 18, 35, 48, 45, 55, 14, 24, 5, 32, 42, 49, 57, 34, 13, 29, 23, 46, 59, 41, 51, 21, 33, 40, 19, 15, 16, 54
2	4.794	17.322	00:59	0, 3, 43, 26, 59, 13, 44, 57, 5, 33, 30, 47, 49, 1, 23, 7, 51, 20, 55, 29, 31, 8, 15, 48, 21, 54, 40, 19, 53, 4, 45, 17, 52, 24, 38, 41, 35, 34, 16, 46, 9, 18, 37, 56, 25, 22, 36, 14, 27, 11, 39, 10, 42, 6, 2, 28, 12, 32, 50, 58
3	1.390	10.599	00:48	30, 58, 28, 27, 5, 32, 19, 43, 52, 31, 6, 9, 33, 59, 36, 44, 40, 0, 41, 1, 18, 54, 51, 11, 24, 47, 23, 53, 13, 26, 12, 48, 10, 14, 4, 17, 38, 16, 2, 37, 35, 39, 50, 3, 8, 7, 15, 25, 34, 22, 55, 42, 56, 21, 45, 57, 20, 29, 46, 49
4	5.866	20.683	00:51	1, 16, 7, 25, 5, 44, 0, 10, 4, 47, 34, 40, 3, 45, 41, 32, 38, 59, 24, 48, 6, 31, 14, 27, 12, 37, 9, 29, 26, 46, 49, 23, 11, 54, 43, 58, 30, 8, 35, 2, 21, 53, 56, 13, 50, 36, 57, 17, 55, 15, 18, 51, 22, 52, 28, 33, 42, 19, 20, 39
5	4.054	16.264	00:47	0, 55, 31, 35, 23, 3, 40, 1, 29, 12, 59, 45, 49, 7, 22, 19, 4, 32, 56, 51, 39, 52, 15, 5, 26, 34, 57, 27, 48, 18, 17, 9, 46, 20, 25,

				37, 33, 50, 38, 42, 43, 8, 36, 28, 16, 6, 30, 10, 41, 2, 24, 13, 21, 14, 47, 54, 44, 53, 58, 11
6	6.592	20.441	00:46	15, 22, 17, 11, 13, 26, 4, 8, 2, 21, 42, 59, 41, 1, 6, 3, 36, 24, 23, 47, 30, 20, 46, 58, 19, 9, 49, 48, 14, 18, 34, 50, 56, 28, 55, 25, 27, 52, 0, 53, 32, 33, 45, 51, 10, 5, 31, 7, 43, 16, 40, 44, 37, 29, 35, 12, 39, 57, 54, 38
7	3.267	12.957	00:40	46, 42, 47, 28, 55, 54, 39, 50, 20, 0, 6, 30, 23, 34, 11, 41, 16, 48, 18, 59, 49, 43, 35, 10, 1, 27, 5, 2, 4, 3, 37, 56, 52, 17, 53, 57, 14, 38, 24, 22, 44, 36, 26, 19, 29, 25, 12, 31, 33, 9, 15, 13, 58, 51, 8, 21, 32, 45, 7, 40
8	100	6.417	00:47	11, 17, 26, 50, 46, 49, 58, 56, 36, 54, 44, 60, 57, 29, 9, 14, 28, 47, 31, 10, 33, 59, 40, 23, 34, 18, 51, 42, 52, 61, 53, 39, 7, 22, 35, 30, 21, 45, 15, 25, 43, 19, 8, 6, 20, 2, 13, 38, 32, 4, 24, 16, 5, 48, 55, 3, 27, 12, 37, 41
9	5.660	20.016	00:51	56, 45, 25, 1, 3, 34, 5, 29, 58, 51, 20, 54, 2, 12, 8, 42, 17, 49, 44, 57, 43, 40, 59, 9, 26, 32, 53, 46, 7, 52, 16, 11, 41, 14, 15, 33, 38, 13, 21, 28, 23, 39, 50, 47, 10, 24, 36, 18, 4, 37, 48, 22, 6, 55, 30, 31, 60, 19, 27, 35
10	1.740	9.961	00:49	35, 39, 45, 29, 20, 27, 44, 46, 56, 47, 48, 25, 37, 28, 57, 5, 19, 23, 17, 59, 24, 33, 52, 38, 8, 10, 32, 34, 11, 26, 18, 54, 30, 7, 49, 15, 51, 42, 22, 21, 14, 41, 2, 60, 1, 9, 4, 53, 43, 31, 16, 50, 58, 55, 36, 12, 40, 3, 6, 13

Fonte: Os autores (2017)

Tabela 1 – Valores referentes aos atrasos ponderados totais e o tempo de execução do algoritmo

A Configuração 2 é caracterizada por uma busca mais aleatorizada, com uma maior quantidade de troca de pontos e iterações, porém com menor quantidade de perturbações por iteração para equilibrar o tempo total de execução do algoritmo. A Tabela 2 compara as soluções ótimas conhecidas com os resultados obtidos pelo algoritmo proposto com os parâmetros definidos de acordo com a segunda configuração.

Inst.	Solução Ótima	Solução Obtida 2	Tempo (hh:mm)	Sequenciamento
1	453	13.048	00:59	26, 24, 32, 56, 17, 18, 47, 46, 11, 4, 53, 13, 12, 44, 55, 2, 37, 1, 7, 30, 6, 31, 3, 5, 50, 22, 38, 45, 8, 9, 49, 23, 35, 20, 36, 42, 52, 10, 39, 58, 25, 57, 48, 29, 34, 43, 59, 14, 0, 28, 27, 41, 51, 21, 33, 40, 19, 15, 16, 54
2	4.794	24.385	01:03	26, 44, 0, 19, 45, 24, 9, 57, 53, 59, 38, 23, 33, 39, 43, 46, 21, 54, 40, 52, 1, 8, 41, 17, 51, 20, 10, 55, 4, 35, 29, 3, 7, 56, 16, 34, 49, 5, 48, 47, 18, 15, 31, 25, 22, 13, 30, 37, 27, 11, 36, 6, 42, 2, 58, 50, 14, 12, 32, 28
3	1.390	15.953	01:02	30, 27, 5, 51, 58, 28, 32, 12, 52, 33, 15, 9, 11, 37, 1, 18, 6, 3, 4, 54, 25, 39, 8, 31, 38, 47, 59, 14, 53, 40, 43, 34, 19, 48, 10, 24, 36, 44, 0, 35, 2, 16, 26, 7, 17, 23, 22, 50, 13, 41, 45, 55, 46, 42, 56, 21, 57, 20, 29, 49
4	5.866	26.508	01:02	1, 23, 5, 25, 44, 16, 31, 10, 4, 26, 34, 24, 3, 45, 53, 55, 38, 59, 41, 47, 54, 11, 21, 37, 12, 57, 9, 29, 22, 14, 2, 50, 32, 40, 8, 6, 30, 58, 48, 35, 27, 0, 56, 49, 43, 7, 36, 13, 46, 15, 17, 51, 18, 39, 52, 28, 33, 42, 19, 20

5	4.054	22.522	00:52	35, 40, 45, 3, 23, 17, 59, 22, 7, 12, 16, 1, 49, 14, 33, 56, 4, 32, 20, 51, 28, 52, 19, 29, 26, 34, 48, 0, 31, 50, 18, 15, 43, 42, 46, 13, 25, 57, 9, 39, 38, 37, 55, 36, 10, 27, 30, 5, 8, 6, 2, 41, 24, 54, 21, 11, 44, 53, 58, 47
6	6.592	24.432	00:52	17, 15, 50, 36, 8, 22, 4, 24, 58, 53, 42, 18, 41, 13, 26, 3, 52, 28, 27, 25, 14, 55, 46, 20, 11, 10, 49, 48, 0, 5, 2, 21, 23, 31, 1, 6, 56, 9, 51, 32, 30, 33, 47, 45, 19, 59, 16, 37, 38, 34, 44, 7, 40, 43, 29, 35, 12, 39, 57, 54
7	3.267	18.433	01:04	49, 56, 10, 20, 55, 54, 39, 1, 47, 0, 6, 5, 23, 19, 17, 53, 12, 13, 14, 37, 4, 42, 35, 33, 27, 46, 36, 2, 3, 15, 38, 34, 58, 25, 41, 16, 44, 43, 59, 30, 22, 57, 24, 48, 28, 31, 29, 51, 11, 26, 18, 52, 32, 50, 9, 21, 8, 45, 7, 40
8	100	11.284	00:51	42, 47, 48, 44, 38, 21, 56, 9, 34, 52, 23, 58, 55, 17, 15, 22, 7, 45, 28, 57, 26, 18, 41, 31, 49, 14, 40, 50, 43, 33, 51, 37, 54, 20, 36, 5, 27, 12, 0, 32, 19, 24, 13, 59, 29, 8, 11, 16, 4, 30, 46, 6, 2, 39, 3, 53, 1, 25, 10, 35
9	5.660	26.662	01:13	0, 19, 55, 25, 4, 15, 14, 47, 28, 50, 57, 9, 13, 11, 7, 37, 42, 31, 24, 56, 33, 46, 52, 53, 49, 22, 2, 12, 27, 6, 45, 44, 23, 3, 51, 41, 8, 1, 58, 32, 39, 20, 40, 35, 16, 48, 43, 10, 38, 17, 21, 36, 29, 54, 30, 59, 5, 18, 26, 34
10	1.740	14.572	00:58	31, 46, 16, 44, 28, 55, 34, 38, 26, 1, 8, 24, 36, 27, 47, 33, 21, 48, 10, 58, 45, 14, 56, 37, 7, 50, 41, 32, 52, 18, 30, 13, 43, 0, 29, 59, 4, 19, 23, 6, 40, 22, 20, 9, 42, 51, 53, 3, 17, 49, 25, 15, 12, 57, 54, 35, 11, 39, 2, 5

Fonte: Os autores (2017)

Tabela 2 – Valores referentes aos atrasos ponderados totais e o tempo de execução do algoritmo.

A execução do algoritmo nas duas configurações levou cerca de 1 hora para cada instância, o que foi visto como um tempo aceitável para situações reais de sequenciamento de produção com grandes quantidades de tarefas. Caso o algoritmo fosse executado por um tempo mais longo, haveria chances de obter soluções ainda melhores.

A Figura 3 busca facilitar a comparação das soluções apresentadas nas Tabelas 1 e 2 juntamente com a solução ótima fornecida por Tanaka e Araki (2013).

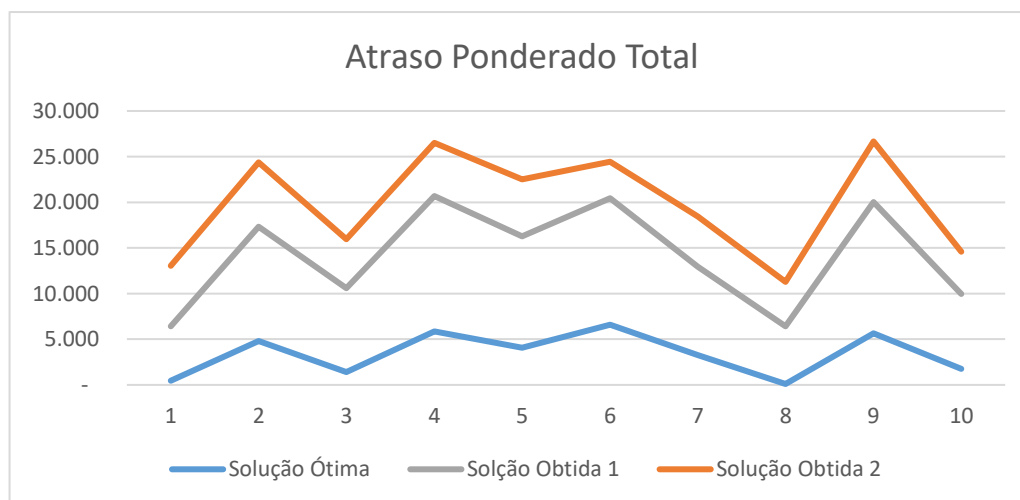


Figura 3 – Valores referentes aos atrasos ponderados totais das soluções ótima e soluções obtidas nas 7 instâncias. Fonte: Os autores, 2017

Analisando a figura acima é possível notar que as soluções ótimas e as soluções obtidas apresentam comportamento semelhantes, porém com amplitudes diferentes. No entanto, apesar da instância 4 ter uma solução ótima melhor que a da instância 6, ela foi a que apresentou o pior resultado nas duas soluções obtidas pelo algoritmo do estudo.

A Configuração 1 buscou realizar uma busca profunda na vizinhança e com menor aleatoriedade, enquanto a Configuração 2 buscou aumentar a aleatoriedade através do acréscimo do número de “trocas de pontos” e aumentou o número de iterações afim de procurar soluções candidatas promissoras. No entanto, comparando os resultados obtidos pelas duas configurações, é possível observar que a Configuração 1 alcançou melhores resultados em todas as 10 instâncias, concluindo que a busca aprofundada e com menor aleatoriedade é mais adequada para o algoritmo proposto.

O desempenho do algoritmo foi afetado pela quantidade de vizinhanças e perturbações por iteração, assim como o número total de iterações da otimização. Como as perturbações realizadas na “Solução Candidata” ocorrem através de trocas aleatórias entre uma ordem de fabricação atrasada com uma adiantada, não há garantias de que o aumento do número de iterações traga também melhores resultados. Isso também implica, para o aumento de perturbações, pois ao iniciar a iteração com uma “Solução Candidata” ruim, ou se o algoritmo convergir para um ótimo local não promissor, o algoritmo precisará de um número expressivamente maior de perturbações para conseguir explorar melhores vizinhanças, enquanto em outras situações, como por exemplo quando o algoritmo tende a um ótimo local, o número de perturbações necessárias é bem menor.

A Figura 4 foi elaborada para mostrar a evolução das melhores soluções em cada uma das 3 iterações do algoritmo para a instância 1 executada com a Configuração 1.

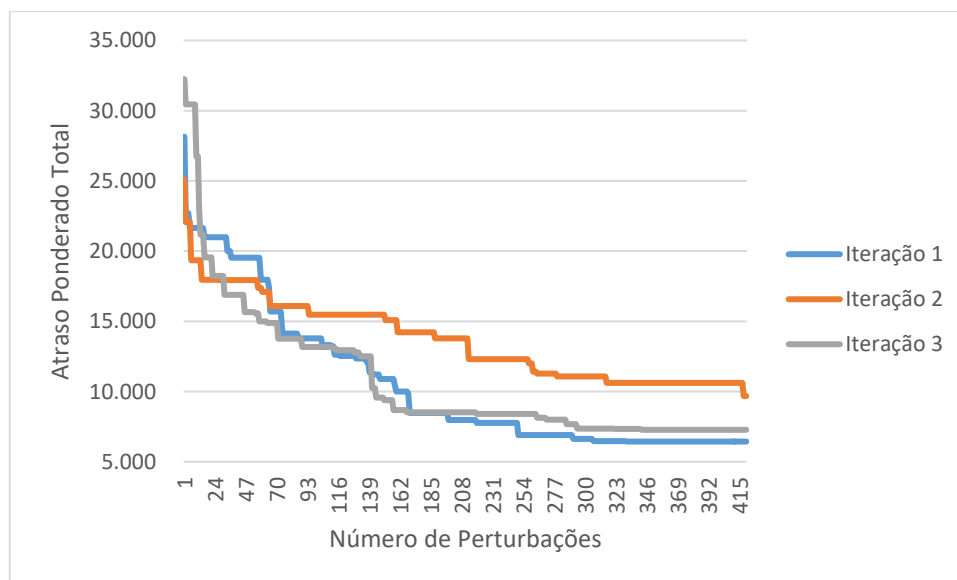


Figura 4 – Evolução dos atrasos ponderados totais para as 3 primeiras iterações da Instância 1 executada na configuração 1. Fonte: Os autores, 2017

Também foi construída a Figura 5 para mostrar a evolução dos atrasos ponderados totais da instância 1 ao ser resolvida pela Configuração 2, com 8 iterações.

Analisando as Figuras 4 e 5 é possível observar que ao decorrer das perturbações, o índice de melhoria do atraso ponderado total sofre decréscimos constantes nas duas configurações. Na iteração 1 executada na configuração 1 o algoritmo chega a rodar 114 perturbações, sem que ocorra nenhuma melhoria. Ao verificar o comportamento da Figura 5 percebe-se que há um potencial de melhorias caso o número de perturbações fosse aumentado.

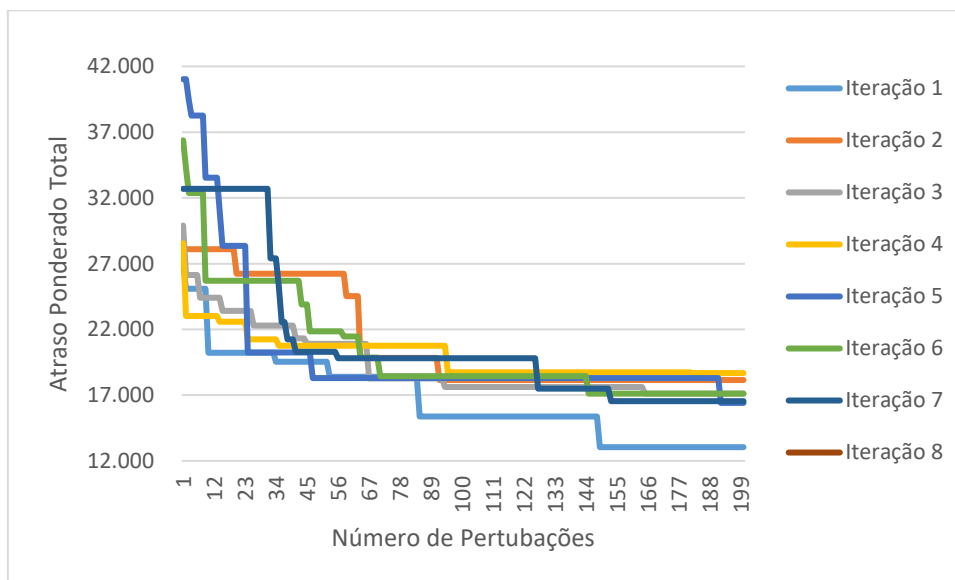


Figura 5 – Evolução dos atrasos ponderados totais para as 8 primeiras iterações da Instância 1 executada na configuração 2. Fonte: Os autores, 2017

5. Considerações Finais

O algoritmo proposto busca otimizar o atraso ponderado total de ordens de fabricação, na situação em que há máquina única e tempos de *setup* dependentes. Da forma como foi estruturado, é possível notar que existem grandes possibilidades de se obter resultados melhores caso sejam aumentados o número de iterações e principalmente o número de perturbações.

Como os problemas de sequenciamento exigem respostas rápidas, priorizou-se a resolução das instâncias em um tempo aproximado de 1 hora, pois este é um valor aceitável no estudo para se realizar programações da produção sem prejudicar outras atividades de uma empresa. Acredita-se que este algoritmo seja útil para encontrar boas soluções para o problema estudado, especialmente se for utilizada a Configuração 1, que favorece uma busca mais aprofundada em cada iteração.

A forma como o método *Iterated Local Search* foi implementada neste artigo não permitiu que soluções próximas às ótimas fossem obtidas em um curto espaço de tempo. Para aprimorar o algoritmo em pesquisas futuras, recomenda-se o uso de outras estratégias para avaliação da vizinhança e realização de perturbações. É possível que a simples troca de tarefas muito atrasadas por tarefas muito adiantadas não seja suficiente para explorar o espaço de soluções de maneira eficiente. Por isso, novos mecanismos de vizinhança poderiam ser testados em versões futuras do algoritmo.

Referências

- FEILI, H.; HADDAD, H.; GHANBARI, P.** *Two Hybrid Algorithms for Single-Machine Total Weighted Tardiness Scheduling Problem with Sequence-Dependent Setup*. American Journal of Scientific Research, Volume 64, p. 22-29.
- FRAMINAN, J. M.; LEISTEN R.; GARCÍA R. R.** *Manufacturing Scheduling Systems: An integrated view on models, methods and tools*. 2014, Ed. Springer, 404p.
- FERREIRA, D.; MORABITO, R.; RANGEL, S.** *Um modelo de otimização inteira mista e heurísticas relax and fix para a programação de fábricas de refrigerantes de pequeno porte*. Disponível em: <https://www.dcce.ibilce.unesp.br/~socorro/resultados/producao_2007_41.pdf> Acesso em 12/07/2017
- GLOVER, F.; KOCHENBERGER, G.** *Handbook of metaheuristics*. New York: Kluwer Academic, 2003.
- GUPTA, S. R.; SMITH, J. S.** *Algorithms for single machine total tardiness scheduling with sequence dependent setups*. European Journal of Operational Research, 2006, Volume 175, Issue 2, p.722-739
- XU, H.; LÜ, Z.; CHENG, T. C. E.** *Iterated Local Search for single-machine scheduling with sequence-dependent setup times to minimize total weighted tardiness*. 2013, Ed. Springer, p.271-287.
- LEE, Y.H.; BHASKARAM K.; PINEDO, M.** *A heuristic to minimize the total weighted tardiness with sequence-dependent setups*. Disponível em: <<http://dx.doi.org/10.1080/07408179708966311>> Acesso em 10/07/2017
- LIN S.W.; YING K.C.** *Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics*. The International Journal of Advanced Manufacturing Technology, 2007, Volume 34, Issue 11-12, p.1183-1190.
- NAGANO, M. S.; MOCCELIN, J. V; LORENA L. A. N.** *Programação da Produção Flow Shop Permutacional com Minimização do Tempo Médio de Fluxo*, 2004. Anais do XXXVI SBPO, v. 36, p. 114-123, 2004.
- PINEDO, M. L.** *Scheduling: Theory, Algorithms and Systems*. 5ª edição, 2016, Ed. Springer, 674p.
- RODRIGUES, G.S.; PIZZOLATO, N. D.** *Sequenciamento de um lote de peças em uma única máquina com datas de entrega e penalidades: um estudo de caso*. Disponível em: <<http://pdf.blucher.com.br/s3-sa-east-1.amazonaws.com/marineengineeringproceedings/spolm2014/126523.pdf>> Acesso em 07/07/2017
- SLACK, N., CHAMBERS, S. e JOHNSON, R.** *Administração da produção*. São Paulo: Atlas, São Paulo, 3ª edição, 2015.
- SULE, D. R.** *Production Planning and Industrial Scheduling: Examples, case studies and applications*. 2ª edição, 2008, Ed.CRC Press, 556p.
- TANAKA, S.; ARAKI, M.** *An exact algorithm for the singlemachine total weighted tardiness problem with sequence-dependent setup times*. Computers & Operations Research, 2013, Volume 40, Issue 1, p.344-352
- VELEZ-GALLEGO, M. C., MAYA, J., e MONTOYA-TORRES, J. R.** *A beam search heuristic for scheduling a single machine with release dates and sequence dependent setup times to minimize the makespan*. Computers and Operations Research, 73:132 – 140. ISSN 03050548, 2016. Disponível em: <<http://dx.doi.org/10.1016/j.cor.2016.04.009>> Acesso em 06/07/2017