

Uma heurística VNS baseada em memória e *Path-Relinking* para o problema de sequenciamento em uma máquina com data de entrega comum

Vinícius Leandro de Rezende Alves

Graduando em Engenharia de Sistemas, Universidade Federal de Minas Gerais
Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, Minas Gerais, Brasil
viniciusra83@ufmg.br

Lucas S. Batista

Departamento de Engenharia Elétrica, Universidade Federal de Minas Gerais
Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, Minas Gerais, Brasil
lusoba@ufmg.br

RESUMO

Este trabalho propõe uma heurística *Variable Neighborhood Search* (VNS) com estrutura de memória e busca local intensificada pelo método *Path-Relinking*. Essa estratégia é empregada para a otimização do problema de sequenciamento de tarefas em uma máquina com data de entrega comum. Nesse problema, um conjunto de tarefas, com data de entrega comum, devem ser realizadas em uma única máquina; e todas as tarefas concluídas antes ou após a data de entrega geram custos adicionais que devem ser minimizados. Para validar a abordagem proposta, considerou-se um conjunto de instâncias disponibilizado na literatura. Os resultados obtidos mostram que a técnica proposta não foi inferior à literatura corrente em nenhuma das instâncias avaliadas; além disso, obteve resultados mais promissores em 30% das instâncias testadas quando comparado à abordagem VNS apenas com estrutura de memória, destacando assim o impacto do refinamento via *Path-Relinking*.

PALAVRAS CHAVE. VNS, *Path-Relinking*, data de entrega comum.

Tópicos. Otimização Combinatória, Metaheurísticas.

ABSTRACT

This paper proposes a Variable Neighborhood Search (VNS) heuristic with a memory structure and local search intensified by a Path-Relinking method. This strategy is used to optimize the problem of scheduling jobs on a single machine against common due dates. In this problem, a set of jobs, with common due dates, must be processed on a single machine; and all completed jobs before or after the due date generate additional penalties that should be minimized. To validate the proposed approach, we considered a set of instances available in the literature. The results show that the proposed technique was not dominated by the current literature in any of the evaluated instances; in addition, it obtained more promising results in 30% of the tested instances when compared to the VNS approach with only memory structure, thus highlighting the impact of the local search refinement while using Path-Relinking.

KEYWORDS. VNS, *Path-Relinking*, common due date.

Paper topics. Combinatorial Optimization, Metaheuristics.

1. Introdução

O advento da terceira revolução industrial trouxe uma forma de administração de produção inovadora, o *just in time*. A aplicação desta filosofia instaurou um pensamento de que tudo deve ser produzido em “sua hora certa” [Slack et al., 2002]. Afinal, a preocupação em um meio de produção não deve ser apenas com o prazo final de entrega de um produto, uma vez que não se pode negligenciar custos com estocagem ou perdas para o caso de se concluir uma produção antes do prazo de entrega. Baseando-se nesse contexto, é notório como a nova forma de produção minimizou custos nos meios em que ela foi aplicada; alguns exemplos reais ocorridos na indústria chinesa são apresentados em [Chen e Tan, 2011]. No entanto, implantar a ideia trazida pelo *Just in Time* no contexto produtivo torna-se muito complexo à medida em que o tamanho da linha produtiva cresce. É de extrema importância, portanto, o estudo de metodologias capazes de lidar com esta complexidade e com os problemas a ela relacionados.

[Biskup e Feldmann, 2001] descrevem um exemplo de problema, abordado neste artigo, que surgiu ao se implantar a ideologia do *just in time*, i.e., o *Common Due-Date* (CDD) ou problema da data de finalização comum. Esse problema ocorre quando se tem um conjunto de tarefas, processadas em uma única máquina, com data de entrega comum, como acontece em linhas de produção. O cenário ideal seria aquele em que todas as tarefas ficassem prontas exatamente no dia estipulado pelo cliente, o que não acontece na realidade. O problema, portanto, envolve custos adicionais (penalizações) para casos de atraso ou adiantamento na entrega das tarefas concluídas. Dessa forma, objetiva-se determinar um cronograma de produção que gere um somatório mínimo de penalizações. Por só se conhecer algoritmos de complexidade fatorial para resolver esse problema, o CDD pertence a classe NP-Difícil [Hall et al., 1991]. Como alternativa a esses métodos computacionalmente caros, o uso de heurísticas eficientes, que aproximam a solução ótima, têm sido bastante eficaz para lidar com problemas desta classe [Rego, 2013; Hansen et al., 2017]. Essas heurísticas apresentam métodos computacionalmente competitivos para se chegar próximo ou até mesmo encontrar a solução ótima do problema.

A literatura apresenta algumas abordagens que utilizam heurísticas para tratar o CDD. Em um trabalho posterior ao desenvolvido por [Biskup e Feldmann, 2001], [Feldmann e Biskup, 2003] utilizam Estratégias Evolutivas e uma heurística *Simulated Annealing* para resolver 140 instâncias do problema. [Nearchou e Omirou, 2013] utilizam um algoritmo baseado em enxame de partículas, i.e., *Particle Swarm Optimization*, a partir do qual obtiveram soluções promissoras. Por fim, mais recentemente, [Coelho, 2016] aborda a resolução do problema por meio de um *Variable Neighborhood Search* (VNS) e propõe duas variações dessa heurística em seu trabalho, incluindo um VNS baseado em memória.

Dando sequência a pesquisa de [Coelho, 2016], esse trabalho propõe uma heurística VNS para otimização do problema de sequenciamento de tarefas, em uma máquina única, com data de entrega comum. Essa heurística emprega uma estrutura de memória, com o intuito de evitar a reavaliação de soluções já visitadas; além disso, é utilizada também uma estratégia de refinamento baseada no *Path-Relinking*. Para a validação da abordagem, considerou-se um conjunto de instâncias proposto em [Biskup e Feldmann, 2001]. Na análise realizada, a heurística proposta é avaliada tanto na presença (VNS-MS-PATH) quanto na ausência (VNS-MS) do método *Path-Relinking*, mantendo-se a estrutura de memória (MS) em ambas as versões. Os resultados obtidos não são inferiores à literatura corrente [Biskup e Feldmann, 2001; Nearchou e Omirou, 2013; Coelho, 2016] em nenhuma das instâncias; além disso, o VNS-MS-PATH foi melhor do que o VNS-MS em 30% dos problemas teste, exigindo um menor número médio de avaliações de função até a convergência.

O restante deste artigo está organizado da seguinte forma: na Seção 2 apresenta-se a modelagem do problema utilizada no corrente trabalho, adaptada de [Biskup e Feldmann, 2001]; na Seção 3 discute-se a heurística proposta baseada em memória e *Path-Relinking*; na Seção 3 os resultados obtidos são contrastados com os disponíveis na literatura; por fim, na Seção 5 são apresentadas as considerações finais e possibilidades de continuidade desta pesquisa.

2. Descrição do Problema

A formulação empregada neste trabalho foi elaborada por [Biskup e Feldmann, 2001]. Suponha que um conjunto de n tarefas localizadas temporalmente no instante zero, com data de entrega comum d , devem ser processadas em uma única máquina. Cada tarefa $i = 1, \dots, n$ necessita apenas uma operação para ser finalizada e possui um tempo de processamento p_i já conhecido. Como discutido anteriormente, se o tempo de conclusão c_i for maior que d , a tarefa foi concluída com um atraso $T = c_i - d$. Além disto, como o tempo de antecipação E é considerado no problema, se c_i for menor que d , $E = d - c_i$. Entretanto, como é impossível saber c_i por antecipação, as tarefas atrasadas e antecipadas são caracterizadas por $T_i = \max\{0, c_i - d\}$ e $E_i = \max\{0, d - c_i\}$. A cada tarefa é atribuída uma penalidade por antecipação, i.e., *earliness* (α), ou atraso, i.e., *tardiness* (β). Um detalhamento mais completo do problema é apresentado em [Biskup e Feldmann, 2001]. A função objetivo do problema envolve a minimização conjunta da soma das penalidades:

$$f(s) = \sum_{i=1}^n \alpha_i E_i + \sum_{i=1}^n \beta_i T_i \quad (1)$$

em que s denota um cronograma viável de processamento das tarefas.

Na função objetivo, a data de vencimento comum d pode ser modelada como uma variável cujo valor deve ser determinado, ou pode ser fornecido a priori. Se $d > \sum p_i$, o problema se torna irrestrito, já que em todas as configurações as tarefas são concluídas antes do prazo. Se $d = 0$, o problema se torna trivial, pois as tarefas estarão todas atrasadas e a solução seria configurá-las de forma decrescente de acordo com as penalidades de atraso.

Esse problema, portanto, possui diferentes complexidades de acordo com a definição de seus parâmetros. Para $\alpha_i = \beta_i = 1$ e data d suficientemente pequena para que o problema seja irrestrito, o mesmo é polinomialmente solucionável no tempo $O(n \log n)$. Mesmo quando α é diferente de β , mas estes se mantêm constantes para todas as tarefas, o problema ainda pode ser solucionado em tempo polinomial. Ele se torna NP-Difícil quando α e β variam para cada tarefa. Ademais, quando o *common due-date* possui d grande o suficiente para ser considerado um problema restrito, ele é NP-Difícil em qualquer situação [Hall et al., 1991].

A data de entrega d utilizada neste trabalho foi proposta em [Biskup e Feldmann, 2001]:

$$d = \text{round} \left(h \sum_{i=1}^n p_i \right) \quad (2)$$

em que h é uma constante que controla o nível restritivo do problema. De forma geral, um baixo valor de h torna o problema mais restritivo e, conseqüentemente, mais difícil de ser resolvido.

Essa formulação foi adotada por caracterizar o mesmo modelo considerado na literatura revista, base de comparação deste artigo. O problema geral de otimização é apresentado a seguir:

Minimize $f(s)$

Sujeito a:

$$T_i \geq s_i + p_i - d, \quad i = 1, \dots, n, \quad (3)$$

$$E_i \geq d - s_i - p_i, \quad i = 1, \dots, n, \quad (4)$$

$$s_i + p_i \leq s_k + R(1 - x_{ik}), \quad i = 1, \dots, n-1; \quad k = i+1, \dots, n, \quad (5)$$

$$s_k + p_k \leq s_i + Rx_{ik}, \quad i = 1, \dots, n-1; \quad k = i+1, \dots, n, \quad (6)$$

$$T_i, E_i, s_i \geq 0, \quad i = 1, \dots, n, \quad (7)$$

$$x_{ik} \in \{0, 1\}, \quad i = 1, \dots, n-1; \quad k = i+1, \dots, n, \quad (8)$$

em que s_i e x_{ik} são as variáveis de decisão do problema, que juntamente com a data de entrega, d , determinam um sequenciamento; s_i é o momento de início da tarefa i e x_{ik} é 1 se a tarefa i está sequenciada antes da tarefa k e $x_{ik} = 0$ caso contrário; R é um número suficientemente grande.

Os valores de T_i e E_i são calculados considerando-se as restrições (3) e (4). As restrições (5) e (6) determinam o momento inicial de cada tarefa: se a tarefa i for sequenciada antes da tarefa k , a restrição $s_i + p_i \leq s_k$ só é válida se $x_{ik} = 1$. Devido a adição de R , (6) não é restritiva para $x_{ik} = 1$. Entretanto, para $x_{ik} = 0$, a restrição (6) resulta em $s_k + p_k \leq s_i$ e (5) torna-se não restritiva. As restrições (7) e (8) definem o domínio dos parâmetros T_i e E_i e das variáveis s_i e x_{ik} .

3. Abordagem Proposta

Nesta seção apresenta-se a abordagem proposta neste artigo. Primeiramente, define-se a estratégia VNS com estrutura de memória (VNS-MS) e uma heurística construtiva para geração inteligente da solução inicial do problema. Posteriormente, propõe-se a heurística VNS-MS-PATH com técnica de refinamento baseada no *Path-Relinking*.

3.1. VNS-MS

O método VNS-MS representa uma adaptação da heurística VNS [Mladenovic e Hansen, 1997], em que se considera a inclusão de uma estrutura de memória (MS) para evitar a reavaliação de soluções já visitadas, diminuindo assim o esforço computacional.

O VNS é uma heurística que explora o espaço de soluções em busca de mínimos locais. Uma vez encontrados, um artifício de perturbação é utilizado para que haja a possibilidade da descoberta de novos mínimos locais. O pseudo-código do VNS está representado no Algoritmo 1. As entradas do algoritmo são s_0 , que representa a solução inicial, e r que indica a quantidade de estruturas de vizinhança utilizadas. Enquanto o critério de parada escolhido não for satisfeito, são realizados procedimentos de perturbação e busca local. Entre as linhas 9 e 14, a solução incumbente (s) é comparada à nova solução obtida (s''): caso s'' seja melhor, esta torna-se a solução incumbente; caso contrário, apenas a estrutura de vizinhança é atualizada. Por fim, retorna-se a melhor solução encontrada, s .

A linha 8 apresenta a fase de busca local do VNS. Neste trabalho, essa busca foi realizada com o algoritmo *Variable Neighborhood Descent* (VND) [Mladenovic e Hansen, 1997]. O VND é uma heurística de refinamento que utiliza trocas sistemáticas de estruturas de vizinhança. Cada vizinhança de uma solução está associada a um tipo de movimento. Assim, realiza-se determinado movimento na solução até que não se consiga mais melhorá-la. Neste momento, troca-se o tipo de movimento realizado, configurando uma troca de vizinhança. Partindo-se de uma solução inicial, portanto, caminha-se de vizinho em vizinho, de acordo com o movimento escolhido, para tentar encontrar uma solução melhor que a corrente. Optou-se por utilizar no VND o método de descida *First Descent* (FD), o qual retorna o primeiro mínimo local encontrado.

Algoritmo 1 VNS

```
1: entrada:  $r, s_0$ ;  
2: saída:  $s$ ;  
3:  $s \leftarrow s_0$ ;  
4: enquanto critério de parada não for satisfeito faça  
5:    $k \leftarrow 1$ ;  
6:   enquanto  $k \leq r$  faça  
7:     Gere um vizinho arbitrário  $s' \in N^k(s)$ ;  
8:      $s'' \leftarrow \text{BuscaLocal}(s')$ ;  
9:     se  $f(s'') < f(s)$  então  
10:       $s \leftarrow s''$ ;  
11:       $k \leftarrow 1$ ;  
12:     senão  
13:       $k \leftarrow k + 1$ ;  
14:     fim se  
15:   fim enquanto  
16: fim enquanto  
17: Retorne  $s$ ;
```

O VNS com estrutura de memória, VNS-MS, foi proposto em [Coelho, 2016] e sua única diferença em relação ao VNS original é a presença de uma estrutura de memória baseada em tabela *Hash*, responsável por armazenar as soluções já avaliadas na função objetivo, com seu respectivo valor de função.

Como os movimentos realizados pelas estruturas de vizinhança são aplicados iterativamente, é muito comum que a mesma solução seja gerada mais de uma vez. Desta maneira, sem a estrutura de memória, toda vez que uma solução for revisitada, ela será reavaliada. O objetivo da implementação de uma tabela *Hash* é, portanto, diminuir o esforço computacional empregado desnecessariamente para a avaliação da função objetivo.

Assim sendo, no algoritmo VNS-MS, toda vez que uma solução inédita é gerada, esta é salva em uma tabela *Hash* juntamente com seu valor de função objetivo. Desta maneira, a avaliação de soluções revisitadas é recuperada na tabela evitando um custo computacional extra. Detalhes de implementação desta estrutura de memória são apresentados em [Coelho, 2016].

3.2. Heurística construtiva para geração da solução inicial

Esta heurística foi proposta por [Coelho, 2016] e baseia-se na fase construtiva do GRASP [Feo e Resende, 1995]. O pseudo-código deste método é mostrado no Algoritmo 2.

Esse algoritmo tem como entradas n, p, α, β e d , em que n representa a quantidade de tarefas, p contém o tempo de processamento de cada tarefa, α e β são as penalidades por adiantamento e atraso, e d a data de entrega.

O algoritmo ordena as tarefas por ordem de processamento (linha 1) e, entre as linhas 4 e 13, insere as tarefas no vetor *in*, que representará a solução inicial. Na linha 5 é calculada a quantidade de tarefas que serão inseridas na construção. Esta quantidade é usada na linha 6 para a criação da lista de tarefas. As penalidades e os tempos de processamento são empregados nas linhas 8 e 9. Na linha 10 as tarefas começam a ser inseridas no vetor, por ordem de tempo de processamento. As listas contendo as penalidades e o tempo de processamento são atualizadas (linha 12) à medida que as tarefas são inseridas em *in*. A saída do algoritmo fornecerá *in*, que representa um vetor com a ordem das tarefas da solução inicial.

Algoritmo 2 ConstroiSolução

```
1:  $[ord] = sort(p)$ ;  
2:  $in = []$ ;  
3:  $c = 0$ ;  
4: para  $i \leftarrow 1$  até  $n$  faça  
5:    $TamLista = ceil(1 + rand(1) * (n - i - 1))$ ;  
6:    $JobsCand = ord(1 : TamLista)$ ;  
7:    $c = c + p(JobsCand)$ ;  
8:    $Fun = \alpha_i max(d - c, 0) + \beta_i max(c - d, 0)$ ,  $i = 1, \dots, TamList$ ;  
9:    $[custoMenor, jobs] = min(Fun)$ ;  
10:   $in(i) = jobs$ ;  
11:   $c = c + p(in(i))$ ;  
12:  atualiza os valores de  $p, \alpha, \beta$ ;  
13: fim para  
14: Retorne  $in$ ;
```

3.3. Estruturas de vizinhança

A solução do problema em questão é uma sequência das tarefas que representa uma ordem de execução na máquina. Desta maneira, um vizinho de uma solução é uma outra sequência destas mesmas tarefas. Assim sendo, o conjunto de todas as permutações possíveis de tarefas formam a vizinhança do problema. Como um problema de *Scheduling*, em geral, envolve muitas tarefas, a análise de apenas parte das soluções permite encontrar resultados satisfatórios, com um menor esforço computacional. Para isto, apenas alguns tipos de permutações são feitas, o que define as estruturas de vizinhança, que são construídas com subconjuntos do conjunto de todas as soluções. Portanto, ao invés de analisar todas as soluções geradas, permutando todas as tarefas, apenas alguns movimentos de troca são realizados. Neste trabalho, três estruturas de vizinhanças são utilizadas:

1. Troca de ordem entre duas tarefas arbitrárias consecutivas;
2. Troca de ordem entre duas tarefas arbitrárias quaisquer;
3. Troca de ordem entre três tarefas arbitrárias quaisquer.

3.4. Path-Relinking

O *Path-Relinking*, proposto por [Glover, 1996], é um método que visa mapear soluções de qualidade no caminho, definido no espaço de busca, entre duas soluções que fazem parte de um conjunto elite construído, por exemplo, ao longo do processo de otimização. Esse conjunto contém soluções que foram retornadas pela busca local e que são de alta qualidade. Alterações são realizadas sobre uma solução inicial, até que se torne idêntica a uma solução guia. Ao percorrer este caminho, a função objetivo é avaliada após cada alteração na solução, para verificar a ocorrência de uma melhora.

Os caminhos criados a partir de uma solução inicial X rumo a uma solução guia X' devem ser escolhidos de maneira a reduzir o número de opções a serem consideradas. Assim, a solução $X(i + 1)$ deve ser criada a partir de $X(i)$ e, a cada passo, deve ser escolhido um movimento que diminui o número de permutações faltantes para se obter X' [Glover, 1996].

O pseudo-código do *Path-Relinking* é apresentado no Algoritmo 3. Ele recebe como parâmetros as soluções inicial e guia. O ciclo iterativo, entre as linhas 3 e 8, é executado até que a distância de *Hamming* entre as duas soluções seja zero. A distância de *Hamming* mostra a distância simétrica entre a guia e a solução inicial [Resende e Ribeiro, 2010]. Um movimento específico é

Algoritmo 3 Path-Relinking (SolInicial, SolGuia)

```

1:  $\Delta \leftarrow DistanciaHamming(SolInicial, SolFinal)$ ;
2:  $Elite \leftarrow MenorArgumento\{f(SolInicial), f(SolGuia)\}$ ;
3: enquanto  $\Delta \neq 0$  faça
4:    $MelhorMovimento \leftarrow SeleccionaMelhorMovimento\Delta(Sol, SolFinal)$ ;
5:    $RetiraMovimentoListaMovimentos(MelhorMovimento)$ ;
6:    $MelhorSol \leftarrow MelhorSol + MelhorMovimento$ ;
7:    $AtualizaElite()$ ;
8: fim enquanto
9: Retorne  $MelhorSol$ 

```

escolhido e removido da lista de movimentos (linhas 4 e 5); então, uma nova solução é obtida na linha 6 e, caso seja promissora, o arquivo elite é atualizado (linha 7).

A diferença simétrica mínima considerada entre as soluções inicial e guia foi de 4 tarefas, pois observou-se empiricamente que valores menores que este, em geral, não geram combinações suficientes para se obter caminhos com possíveis soluções melhores para o problema. Este critério também foi considerado ao longo da construção do conjunto elite, o qual foi limitado a uma cardinalidade de 10 soluções. Na etapa de atualização, para que uma solução seja inserida nesse conjunto, a mesma deve possuir valor de função melhor que o da pior solução que o compõe; e deve também possuir uma distância de *Haming* mínima de 4 tarefas em relação as demais soluções que compõem o conjunto.

3.5. VNS-MS-PATH

O algoritmo final proposto neste artigo, envolvendo VNS, estrutura de memória e *Path-Relinking*, é mostrado no Algoritmo 4. Note que o método *Path-Relinking* é executado na linha 9, em que suas entradas são representadas pela solução s'' , retornada pela busca local, e por uma solução e , escolhida aleatoriamente a partir do conjunto elite. Na abordagem proposta, a solução guia é definida como a pior alternativa entre s'' e e .

Algoritmo 4 VNS-MS-PATH

```

1: entrada:  $r, s_0$ ;
2: saída:  $s$ ;
3:  $s \leftarrow s_0$ ;
4: enquanto critério de parada não for satisfeito faça
5:    $k \leftarrow 1$ ;
6:   enquanto  $k \leq r$  faça
7:     Gere um vizinho arbitrário  $s' \in N^k(s)$ ;
8:      $s'' \leftarrow BuscaLocal(s')$ ;
9:      $s'' \leftarrow PathRelinking(s'', e)$ 
10:    se  $f(s'') < f(s)$  então
11:       $s \leftarrow s''$ ;
12:       $k \leftarrow 1$ ;
13:    senão
14:       $k \leftarrow k + 1$ ;
15:    fim se
16:  fim enquanto
17: fim enquanto
18: Retorne  $s$ ;

```

4. Resultados

Nesta seção são apresentados e discutidos os resultados disponíveis na literatura [Biskup e Feldmann, 2001; Nearchou e Omirou, 2013; Coelho, 2016] e aqueles obtidos pelo emprego das heurísticas VNS-MS e VNS-MS-PATH, considerando-se quatro conjuntos de problemas teste.

4.1. Projeto experimental

As instâncias do problema foram geradas utilizando-se a semente desenvolvida por [Biskup e Feldmann, 2001]. As instâncias adotadas possuem número de tarefas $n = 50, 100, 200$ e 500 , com o intuito de observar o comportamento dos algoritmos em instâncias pequenas, médias e grandes. Além disso, para o cálculo da data de entrega de cada tarefa, [Biskup e Feldmann, 2001] propuseram a utilização de valores de h iguais a $0.2, 0.4, 0.6$ e 0.8 . Neste trabalho, entretanto, optou-se por realizar os experimentos apenas com valores de h iguais a 0.2 e 0.4 , por caracterizarem as instâncias mais desafiadoras.

Todos os algoritmos foram implementados na linguagem *Python* e os testes computacionais foram realizados em um sistema operacional *MacOS High Sierra, Macbook pro*, com processador $2,3$ GHz, intel core i5, 8 Gigas de memória RAM.

4.2. Apresentação e análise dos resultados

Para a comparação entre as heurísticas, fixou-se um número máximo de avaliações de função objetivo e realizou-se 30 execuções dos algoritmos. O valor médio dos resultados finais é apresentado a seguir. O número máximo de avaliações de função, utilizado como critério de parada, foi definido a partir do algoritmo mais custoso. Esses resultados também são comparados com os melhores resultados conhecidos pela literatura [Biskup e Feldmann, 2001; Nearchou e Omirou, 2013; Coelho, 2016].

Nas Tabelas 1 a 4 são mostradas as médias dos resultados finais obtidos pela execução dos algoritmos. A coluna 1 apresenta as instâncias consideradas. As próximas colunas mostram, respectivamente, os resultados obtidos por [Biskup e Feldmann, 2001] (LS 1), por [Nearchou e Omirou, 2013] (LS 2), e pelas heurísticas VNS-MS e VNS-MS-PATH, tanto para $h = 0.2$ quanto para $h = 0.4$. Os resultados em negrito são aqueles com melhor valor de função objetivo.

Nota-se pelos resultados apresentados que, como a variação VNS-MS-PATH utiliza o VNS-MS como algoritmo base, as soluções retornadas por ele possuem, pelo menos, a mesma qualidade das soluções retornadas pelo VNS-MS. Além disso, observa-se que todos os resultados obtidos pelas heurísticas consideradas foram melhores, em média, que aqueles citados na literatura. Conforme as tabelas, o VNS-MS-PATH foi superior ao VNS-MS em 30% das instâncias testadas com $h = 0.2$ e em 43% das instâncias avaliadas com $h = 0.4$, sugerindo o impacto no desempenho do método devido à intensificação da busca local por meio da técnica *Path-Relinking*.

Tabela 1: Comparação dos resultados médios obtidos para as instâncias com 50 tarefas.

Instâncias	h = 0.2				h = 0.4			
	LS 1	LS 2	VNS-MS	VNS-MS-PATH	LS 1	LS 2	VNS-MS	VNS-MS-PATH
sch50k1	42.363	40.734	40.066	37.263	24.868	23.792	22.298	21.553
sch50k2	33.637	30.739	29.184	29.184	19.279	18.042	16.976	16.976
sch50k3	37.641	34.505	32.321	32.321	21.353	20.700	19.059	19.059
sch50k4	30.166	27.803	26.574	26.574	17.495	16.693	15.752	14.970
sch50k5	32.604	32.332	29.356	29.356	18.441	18.167	17.171	16.185
sch50k6	36.920	35.102	33.122	33.122	21.497	20.402	19.376	19.376
sch50k7	44.277	43.229	41.289	39.101	23.883	23.228	21.655	21.655
sch50k8	46.065	43.969	42.154	40.796	25.402	24.947	24.060	23.255
sch50k9	36.397	34.326	31.998	29.542	21.929	20.008	18.703	17.713
sch50k10	35.797	32.999	31.468	31.468	20.048	18.238	18.232	12.232

Tabela 2: Comparação dos resultados médios obtidos para as instâncias com 100 tarefas.

Instâncias	h = 0,2				h = 0,4			
	LS 1	LS 2	VNS-MS	VNS-MS-PATH	LS 1	LS 2	VNS-MS	VNS-MS-PATH
sch100k1	156.103	146.132	140.445	140.445	89.588	86.280	82.025	82.025
sch100k2	132.605	125.331	122.498	120.889	74.854	73.459	69.987	69.987
sch100k3	137.463	130.414	125.296	125.296	85.363	80.207	80.207	76.778
sch100k4	137.265	131.132	127.334	127.334	87.730	81.674	75.978	75.977
sch100k5	136.761	124.882	120.557	120.557	76.424	71.609	67.349	67.349
sch100k6	151.938	139.961	132.619	132.619	86.724	77.087	77.087	76.998
sch100k7	141.613	137.407	132.764	128.764	79.854	78.410	76.410	76.410
sch100k8	167.086	161.424	152.678	152.678	95.361	95.612	90.675	90.516
sch100k9	125.153	118.859	115.467	111.778	73.605	69.812	66.765	66.765
sch100k10	124.446	119.795	113.567	113.567	72.399	72.389	69.160	69.027

Tabela 3: Comparação dos resultados médios obtidos para as instâncias com 200 tarefas.

Instâncias	h = 0,2				h = 0,4			
	LS 1	LS 2	VNS-MS	VNS-MS-PATH	LS 1	LS 2	VNS-MS	VNS-MS-PATH
sch200k1	526.666	502.920	490.826	490.826	301.449	298.080	290.102	288.056
sch200k2	566.643	556.476	533.254	533.254	335.714	322.705	315.091	311.543
sch200k3	529.919	497.396	481.932	481.932	308.278	296.023	289.989	289.989
sch200k4	603.709	599.074	580.998	580.998	360.852	356.197	347.874	344.478
sch200k5	547.953	524.601	500.148	500.147	322.268	306.819	301.001	301.001
sch200k6	502.276	490.165	478.965	478.965	292.453	287.009	271.263	271.263
sch200k7	479.651	470.485	442.724	442.724	279.576	275.805	269.876	269.876
sch200k8	530.896	505.432	480.189	480.189	288.746	280.383	277.263	275.679
sch200k9	575.353	545.248	525.564	516.265	331.107	315.441	306.704	306.704
sch200k10	572.866	544.871	544.871	526.704	332.808	327.299	319.176	318.777

Tabela 4: Comparação dos resultados médios obtidos para as instâncias com 500 tarefas.

Instâncias	h = 0,2				h = 0,4			
	LS 1	LS 2	VNS-MS	VNS-MS-PATH	LS 1	LS 2	VNS-MS	VNS-MS-PATH
sch500k1	3.113.088	2.995.837	2.942.033	2.942.033	1.839.902	1.809.150	1.780.220	1.780.220
sch500k2	3.569.058	3.396.297	3.346.220	3.345.232	2.064.998	2.019.880	1.986.265	1.986.265
sch500k3	3.300.744	3.114.316	3.092.061	3.092.061	1.909.304	1.887.994	1.858.482	1.858.482
sch500k4	3.408.867	3.243.457	3.209.232	3.201.243	1.930.829	1.923.249	1.774.876	1.774.876
sch500k5	3.377.547	3.127.020	3.122.942	3.122.942	1.881.221	1.824.148	1.797.522	1.797.522
sch500k6	3.024.082	2.807.257	2.776.987	2.766.588	1.658.411	1.637.076	1.637.654	1.637.654
sch500k7	3.381.166	3.193.413	3.155.050	3.155.050	1.971.176	1.931.460	1.894.570	1.894.570
sch500k8	3.376.678	3.127.525	3.105.995	3.105.995	1.924.191	1.829.059	1.799.335	1.799.335
sch500k9	3.617.807	3.376.250	3.345.850	3.345.850	2.065.647	1.995.866	1.779.887	1.779.887
sch500k10	3.315.019	3.137.808	3.105.622	3.090.386	1.928.579	1.867.701	1.831.004	1.831.004

Com o intuito de complementar a análise dos resultados, estimou-se também o esforço computacional de cada heurística, necessário para se atingir um determinado valor de função objetivo. De forma geral, considerou-se os resultados apresentados em [Nearchou e Omirou, 2013] como limitantes superiores, e mediu-se o número médio de avaliações de função objetivo, considerando-se 30 execuções, que os métodos realizaram até atingir tais limitantes. Ambas as estratégias foram inicializadas a partir da mesma solução inicial, gerada pela heurística construtiva apresentada na Seção 3.2.

A Tabela 5 apresenta o número médio de avaliações de função obtido para as instâncias com 50, 100, 200 e 500 tarefas. Neste experimento considerou-se apenas o valor de $h = 0,2$, pois como caracteriza as instâncias mais difíceis, é possivelmente suficiente para a elaboração de discussões conclusivas sobre o custo e desempenho dos algoritmos.

O VNS-MS-PATH, apesar de incluir uma etapa adicional de refinamento, alcançou o limitante superior mais rapidamente que o VNS-MS em 72% dos testes. Em algumas instâncias,

como a *sch100k1* e a *sch100k4*, por exemplo, o VNS-MS-PATH encontrou uma solução melhor que o limitante superior com aproximadamente a metade das avaliações de função exigidas pelo VNS-MS. Assim, além do melhor desempenho sugerido anteriormente, o VNS-MS-PATH é em média menos custoso que o VNS-MS. Conforme observado, isso ocorre pois a cada solução gerada na busca local, o VNS-MS-PATH investiga novas soluções por meio da abordagem *Path-Relinking*, possibilitando uma exploração mais inteligente do espaço de busca do problema.

Tabela 5: Comparação entre número médio de vezes que as heurísticas VNS-MS e VNS-MS-PATH avaliam a função objetivo até atingirem um limitante superior específico.

h = 0.2			h = 0.2		
Instâncias	VNS-CM	VNS-CM-PATH	Instâncias	VNS-CM	VNS-CM-PATH
sch50k1	74581	74583	sch200k1	96061	85498
sch50k2	75221	40301	sch200k2	94919	81943
sch50k3	75144	54594	sch200k3	95305	99117
sch50k4	81900	82334	sch200k4	96440	28324
sch50k5	73615	71615	sch200k5	97343	86635
sch50k6	75148	60119	sch200k6	95451	84951
sch50k7	73970	42755	sch200k7	94738	82402
sch50k8	45915	44078	sch200k8	90799	98527
sch50k9	46334	42777	sch200k9	86765	94430
sch50k10	46020	53421	sch200k10	95432	84934
sch100k1	56723	1890	sch500k1	248023	214118
sch100k2	55939	57057	sch500k2	110444	95346
sch100k3	55623	44498	sch500k3	111532	96285
sch100k4	55709	23954	sch500k4	113433	118653
sch100k5	55664	44532	sch500k5	114090	99046
sch100k6	55155	441240	sch500k6	118556	97044
sch100k7	55272	442176	sch500k7	114730	123298
sch100k8	55823	446584	sch500k8	112411	117868
sch100k9	56765	454120	sch500k9	100576	86827
sch100k10	55915	430541	sch500k10	113335	100868

5. Conclusão

Neste trabalho foram comparadas diferentes heurísticas empregadas na resolução do problema de sequenciamento com *Common Due Date*, considerando-se a modelagem proposta em [Biskup e Feldmann, 2001].

As heurísticas comparadas são variações do método VNS. A primeira, VNS-MS, utiliza uma estrutura de memória para armazenar todas as soluções geradas e evitar a reavaliação de soluções já visitadas. A segunda, VNS-MS-PATH, além da estrutura de memória, emprega também uma estratégia *Path-Relinking* para intensificação do refinamento de soluções ótimas locais. Ambos os algoritmos são inicializados a partir de uma mesma solução inicial, obtida por meio de uma heurística construtiva.

O VNS-MS-PATH mapeou melhores resultados se comparado ao VNS-MS e à literatura corrente. Além disto, os resultados do VNS-MS-PATH foram obtidos com um esforço computacional, em média, menor que o do VNS-MS. De forma geral, o VNS-MS-PATH converge, em 30% das vezes, para soluções em média melhores que as obtidas pelo VNS-MS, o que sugere o VNS-MS-PATH como um interessante *trade-off* entre qualidade final de solução obtida e tempo de execução.

Como continuidade deste projeto de pesquisa, sugere-se a investigação e incorporação de heurísticas matemáticas na estrutura do método proposto, possibilitando assim um refinamento ainda mais específico.

Referências

- Biskup, D. e Feldmann, M. (2001). Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates. *Computers & Operations Research*, 28(8):787–801.
- Chen, Z.-X. e Tan, K. H. (2011). The perceived impact of JIT implementation on operations performance: evidence from chinese firms. *Journal of Advances in Management Research*, 8(2): 213–235.
- Coelho, D. G. (2016). *Estruturas de memória e operadores adaptativos em meta-heurísticas*. PhD thesis, Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Minas Gerais, BH-MG, Brasil.
- Feldmann, M. e Biskup, D. (2003). Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers & Industrial Engineering*, 44(2):307–323.
- Feo, T. A. e Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133.
- Glover, F. (1996). Tabu search and adaptive memory programming – advances, applications and challenges. *Interfaces in Computer Science and Operations Research*, 6:1–75.
- Hall, N. G., Kubiak, W., e Sethi, S. P. (1991). Earliness-tardiness scheduling problems, ii: Deviation of completion times about a restrictive common due date. *Operations Research*, 39(5):847–856.
- Hansen, P., Mladenović, N., e Todosijević, R. (2017). Variable neighborhood search: basics and variants. *EURO J Comput Optim*, 5(3):423–454.
- Mladenovic, N. e Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.
- Nearchou, A. C. e Omirou, S. L. (2013). A particle swarm optimization algorithm for scheduling against restrictive common due dates. *International Journal of Computational Intelligence Systems, Taylor & Francis*, 6(4):684–699.
- Rego, M. F. (2013). Algoritmos multiobjetivos para o problema de sequenciamento de tarefas em uma máquina com tempo de preparação dependente da sequência e da família. Master's thesis, Universidade Federal de Ouro Preto, Ouro Preto, Brasil.
- Resende, M. G. e Ribeiro, C. C. (2010). *Greedy randomized adaptive search procedures: advances, hybridizations, and applications*, volume 146 of *Handbook of Metaheuristics, International Series in Operations Research & Management Science*. Springer.
- Slack, N., Chambers, S., e Johnston, R. (2002). *Administração da produção*. Atlas, São Paulo, 2 edition.