Pesquisa Operacional para o Desenvolvimento
Vol. 11, n. 3, p. 159–182, 2019

ISSN 1984-3534

ABORDAGEM EXATA E HEURÍSTICAS PARA O PROBLEMA DE PLANEJAMENTO DE ORDENS DE MANUTENÇÃO DE LONGO PRAZO: UM ESTUDO DE CASO INDUSTRIAL DE LARGA ESCALA[1]

Roberto D. Aquino[a*], Jonatas B. C. Chagas[a,b], Marcone J. F. Souza[a]

[a] Departamento de Computação
Universidade Federal de Ouro Preto - UFOP, Ouro Preto-MG, Brasil

[b] Departamento de Informática
Universidade Federal de Viçosa - UFV, Viçosa-MG, Brasil

Artigo premiado SBPO 2019: Mestrado - 1o Lugar
Recebido 05/10/2019, aceito 17/12/2019

RESUMO

Este trabalho tem seu foco em um problema real de planejamento de manutenção de longo prazo para uma planta de beneficiamento de minério de ferro no Brasil. Este é um problema complexo de programação de ordens de manutenção preventiva, para o qual é necessário atribuir ordens de manutenção preventiva para as equipes de trabalho disponíveis em um horizonte de 52 semanas. Para resolvê-lo, foi desenvolvido um modelo de programação linear inteira mista, bem como algoritmos metaheurísticos baseados nos métodos *Simulated Annealing*, *Variable Neighborhood Search* e *Biased Random-Key Genetic Algorithm*. O modelo exato serviu para validar os resultados dos algoritmos heurísticos aplicados a instâncias de dimensões menores. Os algoritmos metaheurísticos foram capazes de produzir soluções melhores do que aquelas empregadas pela empresa, e em um tempo de execução adequado para a tomada de decisão.

Palavras-chave: Planejamento de manutenção de longo prazo, Escalonamento, Otimização combinatória, Metaheurísticas.

ABSTRACT

This work has its focus on a real long-term maintenance programming problem of an iron ore processing plant of a company in Brazil. This is a complex problem of maintenance programming, where preventive programming orders have to be assigned to the available work teams within 52-week planning. In order to solve it, we developed a Mixed Integer Linear Programming (MILP) model as well as metaheuristic algorithms based on *Simulated Annealing*, *Variable Neighborhood Search*, and *Biased Random-Key Genetic Algorithm*. The MILP model was used to validate the results of the metaheuristic algorithms applied to smaller instances. The metaheuristic algorithms were able to find better solutions than those employed by the company, at an execution time adequate for decision making.

Keywords: Long-term maintenance programming, Scheduling, Combinatorial Optimization, Metaheuristics.

* Autor para correspondência. E-mail: aquinord@yahoo.com.br
DOI: 10.4322/PODes.2019.012

[1] *Todos os autores assumem a responsabilidade pelo conteúdo do artigo.*

1. Introdução

Uma prática muito frequente em grandes indústrias para manter o sistema de produção em boas condições de funcionamento é a realização de manutenções preventivas em seus equipamentos. O objetivo é corrigir eventuais falhas antes que elas ocorram, de forma a evitar a interrupção da produção. Essa prática é adotada visto que o custo das manutenções corretivas, aquelas realizadas após falhas dos equipamentos, é significativamente maior que as manutenções preventivas.

As manutenções preventivas são realizadas periodicamente baseando-se na experiência técnica ou nos manuais dos fabricantes. A realização delas consome uma parcela considerável do recurso humano disponível nas indústrias e, por consequência, é uma parte do orçamento anual que deve ser otimizada. Visto tal cenário, é fácil concluir que um bom planejamento das ordens de manutenções preventivas a serem realizadas pelas equipes de trabalho contribui para a redução dos custos de operação das indústrias.

Em grandes empresas é comumente realizado um planejamento de todas as manutenções preventivas a serem executadas durante todo o ano. Todas essas manutenções são compiladas em um documento chamado de mapa de 52 semanas, no qual cada manutenção, ou, mais especificamente, cada ordem de manutenção, é relativa a uma atividade específica de manutenção. A manutenção deve ser executada em uma determinada janela de tempo em um equipamento por uma equipe especializada naquela atividade e com uma duração pré-determinada.

Este trabalho faz um estudo de caso das manutenções preventivas de uma unidade de beneficiamento de minério de ferro, localizada no Estado de Minas Gerais. Nessa unidade, o mapa de 52 semanas é realizado por uma equipe de engenharia de manutenção especializada com o auxílio de um software de programação de manutenção. O produto final entregue pela equipe é o mapa de 52 semanas.

Neste cenário, quando todas as ordens de manutenção são mapeadas em um único documento, é possível notar a sobreposição de algumas delas em um único equipamento na mesma janela de tempo. Isso ocorre porque cada área de manutenção (ex.: mecânica, elétrica, lubrificação) faz um planejamento separado para o que seria ideal para cada equipamento (o quê, quando, onde, como e por qual especialidade). Além disso, como o planejamento é focado no equipamento, não é verificada a disponibilidade de mão de obra necessária para a realização das manutenções, apenas a área. Dessa forma, é de responsabilidade da equipe de manutenção de curto prazo (a responsável pelo planejamento mensal), a alocação efetiva das manutenções. O problema é que, atualmente, na unidade estudada, menos de 50% das ordens de manutenção são realizáveis sem alocação de mão de obra extra ou contratação de mão de obra externa.

Tendo em vista que um melhor planejamento das ordens de manutenção preventiva é necessário para maximizar o número de manutenções executadas, assim como minimizar a mão de obra necessária, este trabalho propõe apresentar métodos de solução para o problema. No caso, propõe-se uma formulação de programação matemática, assim como algoritmos heurísticos, com o objetivo de encontrar o melhor ou bons planejamentos das ordens de manutenção em cenários reais de produção. Com essas técnicas, espera-se aumentar o percentual de manutenções planejadas que sejam efetivamente realizadas e, assim, reduzir a probabilidade de ocorrência de paradas para manutenções corretivas. Em consequência, contribui-se para reduzir os custos de produção da empresa, melhorando sua competitividade.

Por outro lado, há também um grande interesse de pesquisa em métodos de solução para o problema, visto que ele é da classe NP-difícil. De fato, esse problema se reduz ao de sequenciamento em máquinas paralelas não-relacionadas, que é NP-difícil (Brucker, 2007), ao considerarmos as ordens de serviço como tarefas e cada grupo de equipes de trabalho com a mesma especialidade como um conjunto de máquinas paralelas.

2. Planejamento de Ordens de Manutenção Preventiva de Longo Prazo

O Problema de Planejamento de Ordens de Manutenção Preventiva de Longo Prazo (PPOMPLP) consiste na atribuição das ordens de manutenção para as equipes de manutenção em um horizonte de planejamento, aqui considerado de 52 semanas. Na construção desse mapa de 52 semanas, diversas restrições devem ser consideradas, sendo que a maioria delas está diretamente relacionada a problemas de escalonamento e sequenciamento. Uma ordem de manutenção tem que ser alocada simultaneamente a uma equipe de trabalho e a um equipamento, sendo que a equipe de trabalho não pode executar mais de uma ordem de manutenção simultaneamente e cada equipamento não pode ter mais de uma ordem de manutenção sendo executada simultaneamente. Estas duas alocações simultâneas são as principais restrições que caracterizam o problema. Além disso, há um limite de horas que cada equipe de trabalho pode trabalhar, cada ordem de manutenção deve ser executada dentro de uma janela de tempo, cada ordem de manutenção deve ser executada por um tipo específico de mão de obra. No cenário estudado, é comum haver mais de uma equipe de trabalho capaz de executar a mesma ordem de manutenção.

Se considerarmos uma simplificação do PPOMPLP, ele pode ser comparado ao problema de sequenciamento em máquinas paralelas. Nesta comparação, as ordens de manutenção seriam as tarefas e as equipes de trabalho seriam as máquinas. Cada grupo de equipes com a mesma especialidade seria um conjunto de máquinas paralelas. Nessa analogia, o problema abordado neste trabalho seria definido, de acordo com Pinedo (2008), como $P_m \mid r_j M_j \mid \gamma$, no qual o campo γ seria definido de forma a minimizar a mão de obra necessária para executar o maior número de tarefas.

O conceito de multiprocessamento de tarefas pode ser aplicado ao PPOMPLP se considerarmos dois processamentos: um seria pela equipe de trabalho, e o outro pelo equipamento no qual a manutenção deve ser executada. Uma adaptação necessária seria a consideração de que um dos processadores seria um conjunto de máquinas paralelas, pois várias equipes de trabalho são compatíveis para a execução de uma tarefa. Além disso, é necessário mudar a função objetivo. Nessa analogia, o problema pode ser definido como $MPT2_m \mid r_j M_j \mid \gamma$.

Na empresa em estudo, o mapa de 52 semanas é feito por uma gerência de engenharia de manutenção utilizando o módulo de manutenção preventiva do software SAP da empresa alemã SAP SE (https://www.sap.com/corporate/en.html). O problema é que esse módulo de manutenção não leva em consideração as restrições do problema e, assim, pode gerar soluções inviáveis. Para tentar executar o máximo de manutenções possíveis, a equipe de programação de manutenção de curto prazo, ou seja, aquela que trabalha com um horizonte de planejamento de 30 dias, refaz a programação, incluindo horas extras e a utilização de mão de obra de empresas terceirizadas. Apesar disso, e mesmo com a elevação de custo decorrente dessas medidas, nem todas as manutenções preventivas são realizadas. Portanto, o desafio deste trabalho é justamente a proposição de uma forma mais eficiente de alocação dessas manutenções às equipes já existentes na empresa estudada.

3. Revisão da Literatura

Os problemas de otimização de manutenção não são novos. Apesar de não ter sido encontrado na literatura um problema igual ao do foco deste trabalho, diferentes problemas relacionados à melhoria ou otimização da manutenção foram objetos de estudo em diversos trabalhos. Uma revisão de trabalhos nesse tema é feita por Simões et al. (2011) e Sharma et al. (2011). Entretanto, poucos foram os trabalhos revisados que estão relacionados diretamente à minimização do custo de manutenção.

Yamayee et al. (1983) propuseram uma formulação matemática para a solução exata do problema de escalonamento de manutenções preventivas e desenvolveram um *framework* de programação dinâmica para sua resolução. O problema estudado consiste no escalonamento de 21 ordens de manutenção com diferentes tamanhos e diferentes custos. Assim como no presente

trabalho, a ordem de manutenção, a duração e a janela de tempo são conhecidos a priori. O objetivo considerado pelos autores, no entanto, é diferente, pois consiste em minimizar o custo esperado de produção e o custo da falta de confiabilidade. Foi observado por eles um grande esforço computacional para resolver o problema e identificado que o motivo é que a função objetivo tem dois componentes. Para estudar o efeito de cada componente da função objetivo, foram realizados dois estudos de caso, um com o custo esperado de produção e o outro com o custo da falta de confiabilidade como função objetivo. Os escalonamentos foram totalmente diferentes. A análise do estudo de caso mostrou que o custo de produção varia muito pouco nas proximidades do ponto ótimo. Foi observado também que a minimização do custo de falta de confiabilidade altera muito pouco o custo de produção. Além disso, o tempo computacional para a minimização do custo da falta de confiabilidade é muito menor e, portanto, é a mais recomendada.

Yao et al. (2004) estudaram um problema de manutenção na indústria de semicondutores. Para resolução do problema, foi proposto um *framework* com duas camadas hierárquicas, estando o planejamento de longo prazo na camada externa e o planejamento de curto prazo na camada interna. O foco do trabalho foi a resolução do problema de curto prazo e, para isso, foi proposto um modelo de programação inteira mista (MIP, das iniciais em inglês de *Mixed Integer Programming*) para a programação de 29 ordens de manutenção a serem executadas em 11 ferramentas diferentes, em uma janela de tempo pré-determinada. O escalonamento de várias ordens de manutenção para a mesma ferramenta é equivalente à programação de várias ordens de manutenção de equipamento simultaneamente no problema em estudo no presente trabalho. O objetivo foi maximizar a disponibilidade geral da ferramenta e minimizar a indisponibilidade nos períodos nos quais é esperada uma alta carga de trabalho para a ferramenta. Assim como no problema foco deste trabalho, a função objetivo não leva em consideração os custos da manutenção. Para a análise dos resultados foi feito um estudo de caso baseado em um modelo de simulação já utilizado e validado pela empresa. Os resultados foram, então, comparados com o planejamento atual da empresa. Das 11 ferramentas, 10 não tiveram um resultado estatisticamente diferente. Houve, então, uma ferramenta para a qual o modelo teve um resultado 13,9% melhor. Apesar de não ser estatisticamente diferente, houve uma melhora média observada de aproximadamente 1,6%, o que representou um ganho financeiro de U$39 milhões por ano.

Adhikary et al. (2016) propuseram um algoritmo genético multiobjetivo para o escalonamento de manutenções preventivas de uma caldeira a carvão. Os objetivos considerados foram a maximização do tempo disponível para operação e a minimização do custo de manutenção de uma planta de operação contínua em série. Foi realizado um estudo de caso de um gerador de uma unidade de geração de energia de 210 MW. Os resultados apresentados pelos autores mostraram que com o algoritmo genético a disponibilidade aumentou em 3% e houve uma redução do custo de manutenção de 437 milhões de rúpias indianas (aproximadamente U$6 milhões).

Saraiva et al. (2011) propuseram um modelo MIP e um algoritmo baseado na metaheurística *Simulated Annealing* (SA) para a programação de ordens de manutenção de geradores térmicos. O estudo de caso apresentado envolveu 29 geradores e 29 ordens de manutenção, sendo uma para cada gerador. Assim como neste trabalho, cada ordem de manutenção tem uma janela de tempo pré-determinada. O objetivo foi minimizar as perdas financeiras relacionadas às paradas dos geradores. Foi proposto o SA com uma penalização para cada restrição que pudesse ser violada. Algumas restrições foram consideradas fortes e não podiam ser violadas. Os resultados mostraram que o SA foi capaz de encontrar uma solução sem a violação de nenhuma restrição em um tempo de 15 minutos.

4. Formulação de Programação Matemática

O PPOMPLP pode ser descrito como a seguir. Considere $\mathcal{T} = \{1, 2, ...,n\}$ o conjunto de n manutenções a serem realizadas por um conjunto $\mathcal{W} = \{1, 2, ..., m\}$ de m equipes de trabalho. A cada manutenção $i \in \mathcal{T}$ é associado um conjunto $\mathscr{W}_i \subseteq \mathcal{W}$ de equipes de trabalho capazes

de realizá-la. A cada manutenção $i \in \mathcal{T}$ é associado um tempo p_i necessário para executá-la; um equipamento A_i no qual a manutenção deve ser prestada; e uma janela de tempo $[\, e_i, l_i \,]$, que especifica o intervalo de tempo em que a manutenção i pode ser realizada. A penalidade por não realizar a manutenção $i \in \mathcal{T}$ é dada por w_i. Cada equipe de trabalho $k \in \mathcal{W}$ tem disponibilidade de prestar h_k horas de mão de obra. O conjunto $\mathscr{T}_k \subseteq \mathcal{T}$ indica as manutenções que podem ser realizadas pela equipe de trabalho $k \in \mathcal{W}$, ou seja, $\mathscr{T}_k$ indica as manutenções que a equipe de trabalho k tem habilidade para executar. O objetivo é determinar o plano de execução das manutenções que minimiza o número de equipes de trabalho necessárias às manutenções, respeitando-se todas as restrições do problema, definidas formalmente no modelo matemático apresentado a seguir.

As variáveis utilizadas no modelo matemático são descritas abaixo:

- $x_{ij}^k = 1$, se a manutenção i precede imediatamente a manutenção j executada pela equipe de trabalho k;
- $y_{ik} = 1$, se a manutenção i será executada pela equipe de trabalho k;
- $z_k = 1$, se a equipe de trabalho k é utilizada;
- $c_{ik} \geq 0$, tempo de conclusão da manutenção i pela equipe de trabalho k.
- $r_{ij} = 1$, se a manutenção i precede imediatamente a manutenção j, para o caso em que ambas as manutenções referem-se ao mesmo equipamento ($A_i = A_j$) e que ambas são executadas por equipes diferentes.

Com essas variáveis é possível descrever a seguinte formulação de programação linear inteira mista (MILP, das iniciais em inglês *Mixed Integer Linear Programming*) para o PPOMPLP.

$$\min \sum_{k \in \mathcal{W}} z_k \; + \; \sum_{i \in \mathcal{T}_k} w_i \left(1 - \sum_{k \in \mathscr{W}_i} y_{ik} \right) \tag{1}$$

$$\sum_{k \in \mathscr{W}_i} y_{ik} \leq 1 \qquad \forall i \in \mathcal{T} \tag{2}$$

$$\sum_{i \in \mathscr{T}_k \cup \{0\} \setminus \{j\}} x_{ij}^k = y_{jk} \qquad \forall j \in \mathcal{T}, k \in \mathscr{W}_j \tag{3}$$

$$\sum_{j \in \mathscr{T}_k} x_{0j}^k = z_k \qquad \forall k \in \mathcal{W} \tag{4}$$

$$\sum_{i \in \mathscr{T}_k \cup \{0\} \setminus \{l\}} x_{il}^k = \sum_{j \in \mathscr{T}_k \cup \{0\} \setminus \{l\}} x_{lj}^k \qquad \forall k \in \mathcal{W}, l \in \mathscr{T}_k \tag{5}$$

$$c_{0k} = 0 \qquad \forall k \in \mathcal{W} \tag{6}$$

$$c_{jk} \geq c_{ik} + p_j - M'_{ij} \cdot (1 - x_{ij}^k) \qquad \forall k \in \mathcal{W}, i \in \mathscr{T}_k \cup \{0\}, \, j \in \mathscr{T}_k \tag{7}$$

$$c_{ik} \geq (e_i + p_i) \cdot y_{ik} \qquad \forall k \in \mathcal{W}, i \in \mathscr{T}_k \tag{8}$$

$$c_{ik} \leq l_i \qquad \forall k \in \mathcal{W}, i \in \mathscr{T}_k \tag{9}$$

$$c_{jk'} \leq c_{ik} - p_i + M''_{ij} \cdot r_{ij} \qquad \begin{array}{r} \forall k \in \mathcal{W}, k' \in \mathcal{W}, i \in \mathscr{T}_k, j \in \mathscr{T}_{k'}, \\ | \, k \neq k', i < j, A_i = A_j \end{array} \tag{10}$$

$$c_{jk'} \geq c_{ik} + p_j - M'_{ij} \cdot (1 - r_{ij}) \qquad \forall k \in \mathcal{W}, k' \in \mathcal{W}, i \in \mathscr{T}_k, j \in \mathscr{T}_{k'}, \mid k \neq k', i < j, A_i = A_j \qquad (11)$$

$$c_{ik} \leq h_k \qquad \forall k \in \mathcal{W}, i \in \mathscr{T}_k \qquad (12)$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall k \in \mathcal{W}, i \in \mathscr{T}_k \cup \{0\}, j \in \mathscr{T}_k \cup \{0\} \qquad (13)$$

$$y_{ik} \in \{0, 1\} \qquad \forall k \in \mathcal{W}, i \in \mathscr{T}_k \cup \{0\} \qquad (14)$$

$$z_k \in \{0, 1\} \qquad \forall k \in \mathcal{W} \qquad (15)$$

$$c_{ik} \geq 0 \qquad \forall k \in \mathcal{W}, i \in \mathscr{T}_k \qquad (16)$$

$$r_{ij} \in \{0, 1\} \qquad \forall i \in \mathcal{T}, j \in \mathcal{T} \mid i < j, A_i = A_j \qquad (17)$$

A função objetivo (1) busca reduzir a quantidade de mão de obra necessária para realizar o maior número possível de manutenções. O balanceamento entre os dois termos da função objetivo é realizado com base em uma penalidade w_i por não execução de cada manutenção. O conjunto de restrições (2) garante que cada manutenção será executada por no máximo uma equipe de trabalho. O conjunto de restrições (3) garante que toda manutenção executada terá uma mão de obra alocada. As restrições (4) garantem que uma determinada equipe de trabalho só será alocada se ela for utilizada para execução de alguma manutenção. As restrições (5) garantem o sequenciamento das ordens de manutenção executadas por cada equipe de trabalho. O conjunto de restrições (6) impõe que o tempo de conclusão de uma manutenção fictícia, criada para facilitar a modelagem, é nulo para todas as equipes de trabalho. As restrições (7), (10) e (11) garantem que a equipe de trabalho não pode executar mais de uma ordem de manutenção simultaneamente e cada equipamento não pode ter mais de uma ordem de manutenção sendo executada simultaneamente. Mais especificamente, as restrições (7) são ativadas apenas quando a manutenção j for realizada imediatamente após a manutenção i pela equipe k. Por sua vez, as restrições (10) são ativadas quando a manutenção j precede imediatamente a manutenção i para o caso em que ambas as manutenções referem-se ao mesmo equipamento ($A_i = A_j$) e executadas por equipes diferentes. As restrições (11) são similares às (10) quando a manutenção i precede imediatamente a manutenção j. Para ativar ou desativar essas restrições, as constantes M'_{ij} e M''_{ij} devem assumir valores maiores ou iguais à $l_i + p_j$ e $l_j + p_i$, respectivamente. As restrições (8) e (9) garantem que cada manutenção é executada dentro de sua janela de tempo, enquanto as restrições (12) garantem que o número de horas alocadas para uma equipe de trabalho seja menor ou igual ao número de horas disponíveis. Por fim, as restrições (13)-(17) indicam o domínio e escopo das variáveis de decisão.

Com essa formulação MILP, no entanto, não foi possível resolver instâncias reais do problema. Em vista disso, também foram desenvolvidos diferentes algoritmos metaheurísticos, os quais são apresentados a seguir, com o objetivo de tratar situações reais do problema em tempo de computação razoável.

5. Abordagens Heurísticas

A formulação MILP foi muito importante para o entendimento e definição formal do problema. Entretanto, dado o fato de o PPOMPLP ser da classe NP-difícil, também foram propostos cinco algoritmos metaheurísticos baseados em SA, VNS e BRKGA, objetivando encontrar soluções de boa qualidade dentro de um tempo computacional praticável para a tomada de decisão. Essas metaheurísticas têm sido utilizadas com sucesso na resolução de vários problemas combinatórios, como em Ferreira e Queiroz (2018), Cordone e Lulli (2016) e Zudio et al. (2018). Tais algoritmos, bem como seus componentes, são descritos em detalhes a seguir.

5.1. Representação da Solução

Em todos os cinco algoritmos propostos, uma solução s do problema é representada de forma indireta por uma permutação $s = \langle s_1, s_2, \cdots, s_n \rangle$ das n ordens de manutenção. A posição em que cada ordem de manutenção s_i aparece na permutação s define a prioridade da manutenção s_i a ser considerada no algoritmo de alocação, descrito na seção seguinte.

Optou-se por utilizar esta representação indireta, uma vez que dessa forma a geração de soluções iniciais, bem como a exploração do espaço de soluções viáveis por meio dos operadores de vizinhança, torna-se um processo muito simplificado, quando comparado à representação direta da solução, ou seja, uma lista de ordens de manutenção para cada equipe de trabalho.

5.2. Algoritmo de Alocação de Ordens de Manutenção

A partir de uma solução s do problema é aplicado um algoritmo simples, que busca alocar as ordens de manutenção às equipes. Este algoritmo serve, também, para avaliar o custo de uma solução. Os passos do referido processo de alocação são descritos em alto nível pelo Algoritmo 1.

Algoritmo 1: Alocação de Ordens de Manutenção

```
1  custo ← 0
2  para cada equipe k ← 1 até m faça
3      z_k ← false
4  para cada ordem de manutenção s_i da posição i ← 1 até n faça
5      alocou ← false
6      para cada equipe k ← 1 até m faça
7          se s_i ∈ T_k então
8              Construa um controle com as janelas de tempo consolidadas (interseção dos
                intervalos das tarefas, da equipe e do equipamento)
9              para cada intervalo faça
10                 se intervalo compatível então
11                     Aloque a ordem de manutenção s_i à equipe k
12                     Aloque a ordem de manutenção s_i em seu respectivo equipamento
13                     alocou ← true
14                     se z_k = false então
15                         z_k ← true
16                         custo ← custo + 1
17                     Vá para linha 4
18     se alocou = false então
19         custo ← custo + w_{s_i}
20 retorna custo
```

O Algoritmo 1 funciona como segue. Inicialmente (linhas 1 a 3), o custo e o controle de alocação das equipes são inicializados. Em seguida, as ordens de manutenção são percorridas sequencialmente, com o objetivo de tentar alocá-las (linha 4). A linha 5 inicializa um controle para verificação se a ordem de manutenção foi alocada. Para cada manutenção, as equipes de trabalho são percorridas sequencialmente (linha 6), da primeira até a última, até encontrar uma equipe de trabalho capacitada para a execução da ordem de manutenção. Na linha 7 é verificada se a equipe é capacitada a executar a manutenção. Caso ela seja capacitada, na linha 8 é construída uma janela de tempo consolidada, assim chamada uma janela que leva em consideração tanto a disponibilidade da equipe quanto a do equipamento no qual a manutenção será executada. Os intervalos disponíveis na janela de tempo consolidada são percorridos sequencialmente (linha 9). Se a ordem de manutenção puder ser alocada dentro do intervalo disponível (linha 10), a ordem de

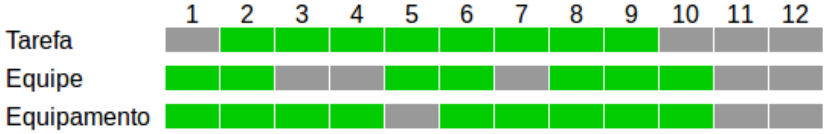
manutenção é alocada tanto no controle das equipes (linha 11), quanto no controle do respectivo equipamento (linha 12) e o controle é atualizado para indicar que a manutenção foi alocada (linha 13). Se a equipe de trabalho ainda não foi alocada (linha 14), o controle de alocação da equipe é atualizado (linha 15) e o custo é acrescido em uma unidade (linha 16). A linha 17 sai dos laços de tentativa de alocação da ordem de manutenção se ela já tiver sido alocada. Se, ao final de todas as tentativas de alocação, não for possível alocar a ordem de manutenção, ela não é executada e uma penalidade w_{s_i} por sua não execução é acrescida ao custo da respectiva solução, conforme mostra a linha 19 do Algoritmo 1. Pode ser facilmente verificado que o Algoritmo 1 tem uma complexidade de $\mathcal{O}(n^2m)$ operações, no pior caso, uma vez que o laço da linha 4 4 é executado n vezes e o da linha 6, m vezes, no máximo; enquanto o laço da linha 9 é aplicado n vezes, no máximo.

Note que, como o algoritmo de alocação trata todas as restrições do problema, qualquer solução s é uma solução viável para o problema. Essa é uma característica interessante para problemas com muitas restrições.

Para ilustrar o funcionamento do procedimento de alocação das ordens de manutenção deste Algoritmo, considere uma ordem de manutenção de duração igual a 2 unidades de tempo, que tenha uma janela de tempo de execução no intervalo [2, 9]. A Figura 1 representa na cor verde o controle das janelas de tempo da manutenção, assim como de uma dada equipe capacitada a executar essa ordem de manutenção e do equipamento em que ela será executada.

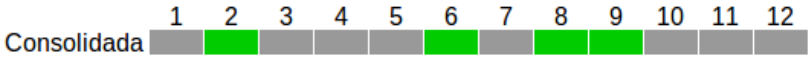
Para este exemplo, o procedimento verifica primeiramente que a equipe em teste possui uma disponibilidade de execução nos períodos [1, 2], [5, 6] e [8, 10]; o equipamento nos períodos [1, 4] e [6, 10]. A Figura 2 consolida todas essas disponibilidades de execução dessa ordem de manutenção, isto é, há disponibilidade nos períodos [2, 2], [6, 6] e [8, 9]. Como a manutenção tem uma duração de 2 unidades de tempo contínuas, não é possível alocá-la aos períodos [2, 2] e [6, 6]; neste caso, ela é alocada ao período [8, 9]. Se houver mais de um período de tempo que seja possível alocar a tarefa, ela é alocada no início do primeiro período de tempo disponível, isto é, foi considerado uma heurística *first-fit*.

Figura 1: Controle das janelas de tempo.



Fonte: Elaborada pelos autores.

Figura 2: Consolidação dos controles das janelas de tempo.



Fonte: Elaborada pelos autores.

Ao final da aplicação do algoritmo, ter-se-á um conjunto de ordens de manutenção alocadas a um conjunto de equipes, bem como, possivelmente, a existência de um conjunto de ordens de manutenção não alocadas.

5.3. *Simulated Annealing*

O primeiro algoritmo heurístico desenvolvido neste trabalho para abordar o PPOMPL foi baseado na metaheurística SA (Kirkpatrick et al., 1983), que é um algoritmo probabilístico inspirado no processo metalúrgico termodinâmico de recozimento de metais.

Algoritmos baseados na metaheurística SA simulam a dinâmica deste processo metalúrgico,

fazendo-se uma analogia entre o metal e a solução do problema em questão: quanto maior a temperatura, maior é a probabilidade de o algoritmo aceitar uma solução de piora.

Na simulação computacional, o sistema é iniciado com uma temperatura alta, a qual vai diminuindo gradativamente, de acordo com um fator de resfriamento, até atingir uma temperatura baixa pré-determinada. Para cada temperatura, busca-se o equilíbrio térmico gerando-se soluções vizinhas aleatórias da solução corrente durante um número de iterações pré-determinado. Em cada iteração de uma dada temperatura, se a solução vizinha for melhor que a solução corrente, ela é aceita; caso contrário, ela poderá ser aceita de acordo com uma probabilidade de se aceitar uma solução de piora, dada por $e^{-\Delta/t}$, sendo t a temperatura corrente e Δ a variação no valor da função objetivo ao se passar da solução corrente para a solução vizinha. Uma solução vizinha de piora é aceita apenas se for gerado um número aleatório $x \in [0,1]$, tal que $x < e^{-\Delta/t}$.

Algoritmo 2: Simulated Annealing

```
1  s ← Solução inicial aleatória
2  melhorSolução ← s
3  t ← maxTemperatura
4  enquanto t > minTemperatura faça
5      iter ← 0
6      enquanto iter < maxIterações faça
7          s' ← Selecione um vizinho aleatório s' ∈ N1(s)
8          Δ ← f(s') − f(s)
9          se Δ < 0 então
10             s ← s'
11             se f(s') < f(melhorSolução) então
12                 melhorSolução ← s'
13         senão se rand(0, 1) < e^(−Δ/t) então
14             s ← s'
15         iter ← iter + 1
16     t ← t × (1 − α)
17 retorna melhorSolução
```

Para explorar o espaço de soluções foi definida uma estrutura de vizinhança simples, que consiste na troca de duas posições da permutação s. Todos os vizinhos de uma solução s são representados por $\mathcal{N}_1(s)$. É importante observar que a vizinhança usada é conexa e, assim, é possível explorar todo o espaço de soluções do problema partindo-se de qualquer solução inicial. Inicialmente, no Algoritmo 2, a solução inicial é criada de forma completamente aleatória, escolhendo-se uma permutação s qualquer das n ordens de manutenção (linha 1). A solução inicial é considerada a melhor solução até então (linha 2). Enquanto a temperatura mínima não é atingida, o algoritmo seleciona aleatoriamente um vizinho da solução corrente (linha 7). A solução vizinha s' é aceita como a nova solução corrente se a sua qualidade for melhor que a da solução corrente (linha 9) ou de acordo com uma probabilidade determinada em função da variação do valor da função objetivo ao se passar de s para s', e também da temperatura atual (linha 13). Durante todo o processo, a melhor solução encontrada é armazenada (linha 12). Depois de *maxIterações* iterações, a temperatura é diminuída por um fator de resfriamento $\alpha \in [0,1]$ (linha 16). O algoritmo termina quando a temperatura mínima é atingida, situação em que se retorna a melhor solução encontrada durante a busca (linha 17).

5.4. *Variable Neighborhood Search*

Algoritmos baseados no VNS (Mladenović e Hansen, 1997; Hansen et al., 2017) incluem uma fase de melhoria, denominada BUSCA-LOCAL, usada para refinar uma solução, e uma fase de perturbação, denominada SHAKE, usada para não se ficar preso em ótimos locais. Essas duas fases juntas com um passo de troca de vizinhança são aplicadas alternadamente até que um critério de parada seja atingido. Seja $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_{k_{\max}}\}$ um conjunto de operadores tais que cada operador $\mathcal{N}_k$, $1 \leq k \leq k_{\max}$, mapeia uma dada solução s para uma dada estrutura de vizinhança $\mathcal{N}_k$. A ordem dos operadores no conjunto $\mathcal{N}$ também define a ordem de exploração das estruturas de vizinhança de uma dada solução s. Algoritmos baseados no VNS tem sido usados, com sucesso, na resolução de vários outros problemas de natureza combinatória, como por exemplo, em Gao et al. (2008), Yazdani et al. (2010) e Chagas et al. (2020).

O Algoritmo 3 descreve o pseudocódigo do VNS proposto para tratar o PPOMPL. Nesse algoritmo, a solução inicial (linha 1) é gerada aleatoriamente. Em seguida (linha 2), é aplicada a ela uma busca local. Enquanto o tempo máximo de processamento $tempo_{\max}$ não for alcançado (linha 12), o algoritmo gera soluções intermediárias s', obtidas aleatoriamente da k-ésima estrutura de vizinhança ($\mathcal{N}_k$) pela aplicação do procedimento SHAKE (linha 5) à solução corrente s. Para cada solução intermediária s' é aplicada uma busca local (linha 6) usando a estrutura de vizinhança $\mathcal{N}_1$, resultando em um ótimo local s''. Se esse ótimo local for melhor que a solução s corrente (linha 7), ele é aceito como a nova solução corrente e a estrutura de vizinhança volta para a primeira (respectivamente, linhas 8 e 9). Caso contrário, passa-se para a próxima estrutura de vizinhança (linha 11). Quando o laço de repetição é interrompido pelo tempo de processamento, a melhor solução encontrada durante a busca é retornada (linha 13 do Algoritmo 3).

Algoritmo 3: Variable Neighborhood Search

```
1  s ← Solução inicial aleatória
2  s ← BUSCA-LOCAL(s, N_1)) // Algoritmo 5
3  k ← 1
4  repita
5  |  s' ← SHAKE(s, k, N) // Algoritmo 4
6  |  s'' ← BUSCA-LOCAL(s', N_1) // Algoritmo 5
7  |  se f(s'') < f(s) então
8  |  |  s ← s''
9  |  |  k ← 1
10 |  senão
11 |  |  k ← k + 1
12 até tempo > tempo_max;
13 retorna s
```

Os procedimentos SHAKE$(s, k, \mathcal{N})$ e BUSCA-LOCAL$(s', \mathcal{N}_1)$ do Algoritmo 1 são descritos pelos Algoritmos 4 e 5, respectivamente. O Algoritmo 4 é responsável por gerar as perturbações nos ótimos locais correntes e, assim, permitir que o algoritmo não fique preso neles. Para gerar um vizinho aleatório s' da k-ésima estrutura de vizinhança são realizadas k perturbações sucessivas na solução corrente s, utilizando-se a estrutura de vizinhança $\mathcal{N}_1$. Assim, à medida que o valor de k aumenta, o método se afasta gradativamente da região do espaço de soluções em que ele se encontra.

Algoritmo 4: SHAKE$(s, k, \mathcal{N})$

```
1 retorna um vizinho aleatório s' ∈ N_k(s)
```

O Algoritmo 5, por sua vez, refina a solução intermediária retornada pelo Algoritmo 4. A busca local, que utiliza a estrutura de vizinhança $\mathcal{N}_1$, analisa todos os vizinhos da solução corrente s e escolhe o melhor deles. Se esse vizinho for melhor que a melhor solução corrente s', então a solução é aceita. O procedimento é repetido até que não exista mais um vizinho capaz de melhorar a solução corrente, situação que configura a existência de um ótimo local com relação à estrutura de vizinhança $\mathcal{N}_1$ utilizada.

Algoritmo 5: BUSCA-LOCAL$(s, \mathcal{N}_1)$

```
1  s' ← s
2  repita
3      Encontre o melhor vizinho s'' ∈ N1(s')
4      se f(s'') < f(s') então
5          s' ← s''
6          otimo_local ← false
7      senão
8          otimo_local ← true
9  até otimo_local = true;
10 retorna s'
```

5.5. *Multi-Start Variable Neighborhood Search*

Algoritmos *Multi-Start* são iniciados em diversos pontos diferentes do espaço de soluções de um problema e retornam o melhor valor dentre as soluções refinadas (Martí et al., 2013). O *Multi-Start Variable Neighborhood Search* (MSVNS) é uma variante na qual o VNS é iniciado diversas vezes. Ele tem como vantagem ser facilmente paralelizável e, desta forma, pode explorar melhor o espaço de soluções de um problema, aproveitando a capacidade de multiprocessamento da geração atual de computadores. O Algoritmo 6 descreve o funcionamento do MSVNS implementado.

Algoritmo 6: Multi-Start Variable Neighborhood Search

```
1  f(s*) ← ∞
2  para cada núcleo de processamento faça
3      s ← execute o VNS // Algoritmo 3
4      se f(s) < f(s*) então
5          s* ← s
6  retorna s*
```

No Algoritmo 6, o laço mais externo (linha 2) é responsável por executar o algoritmo VNS (Algoritmo 3) diversas vezes, partindo de soluções iniciais diferentes. O número de execuções do laço externo é definido em função do número de núcleos de processamento existentes no computador utilizado nos testes. Ao final de todo o processamento (linha 6), o algoritmo MSVNS retorna a melhor solução encontrada dentre todas as execuções do VNS.

5.6. *Biased Random-Key Genetic Algorithm*

Motivados pela eficiência dos algoritmos evolucionários, um algoritmo baseado na metaheurística BRKGA (Martinez et al., 2011) também foi desenvolvido para tratar o PPOMPL. A metaheurística BRKGA é uma variante do *Random-Key Genetic Algorithm* – RKGA (Bean, 1994) e tem sido alvo de estudos na literatura desde 2004.

Os algoritmos evolucionários (Holland, 1992) fazem uma busca populacional baseada nos processos biológicos de evolução dos seres vivos. Nesses algoritmos são implementados operadores de evolução, como cruzamento e mutação, analogamente ao que acontece nos processos naturais de evolução. Neste paralelo, cada solução é considerada um indivíduo e, portanto, é codificada de forma similar a um cromossomo (conjunto de genes) de forma a permitir a aplicação de algoritmos que simulam o processo biológico.

Após a aplicação dos operadores, é feita uma seleção de quais indivíduos devem sobreviver, baseados em uma probabilidade de sobrevivência na qual os indivíduos melhores têm maiores chances de sobreviver e uma nova geração é criada, com características da geração anterior. Espera-se a reprodução de filhos cada vez melhores ao longo das gerações.

Bean (1994) introduziu os algoritmos genéticos de chaves aleatórias ou *Random-Key Genetic Algorithm* – RKGA para problemas de sequenciamento. No RKGA a representação de cada indivíduo é feita por um vetor de chaves aleatórias no intervalo contínuo [0,1). Devido a essa representação genérica, faz-se necessário aplicar um algoritmo de decodificação, a fim de traduzir um vetor de chaves aleatórias para uma solução propriamente dita.

A partir desse princípio de decodificação proposto, uma população inicial é facilmente gerada, simplesmente criando vetores de números aleatórios no intervalo contínuo [0,1). Para a reprodução, é realizado um *crossover* uniforme no qual dois pais são aleatoriamente selecionados e para cada gene do filho é feito um sorteio de qual pai o gene será herdado. A Figura 3 mostra um exemplo de reprodução nos quais os genes 1, 2 e 5 foram herdados do pai P1 e os genes 3, 4 e 6 foram herdados do pai P2.

Figura 3: Exemplo de reprodução.

$$
\begin{aligned}
P1 = &\ \langle\ \mathbf{0{,}12},\ \mathbf{0{,}22},\ 0{,}99,\ 0{,}68,\ \mathbf{0{,}88},\ 0{,}33\ \rangle \\
P2 = &\ \langle\ 0{,}52,\ 0{,}62,\ \mathbf{0.19},\ \mathbf{0.18},\ 0{,}22,\ \mathbf{0{,}01}\ \rangle \\
F = &\ \langle\ \mathbf{0{,}12,\ 0{,}22,\ 0{,}19,\ 0{,}18,\ 0{,}88,\ 0{,}01}\ \rangle
\end{aligned}
$$

Fonte: Elaborada pelos autores.

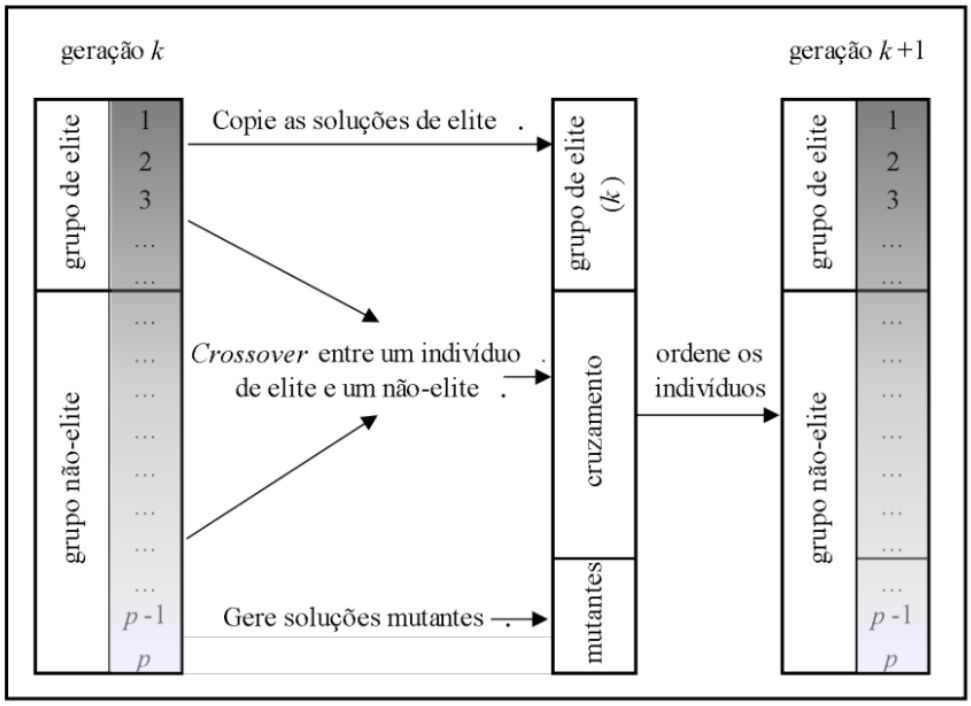
No RKGA o conceito original de mutação dos GAs não é utilizado. Ao invés de mutação é utilizado o conceito de mutantes. Os mutantes são novos indivíduos inseridos na população utilizando o mesmo processo de geração da população original, ou seja, são indivíduos que são criados codificados por um vetor de chaves aleatórias no intervalo contínuo [0,1).

Para realizar a evolução no RKGA são computados os custos de todas as soluções da geração atual e selecionado um grupo de elite com as melhores soluções e adicionados à próxima geração. Um outro grupo de mutantes é gerado e adicionado à próxima geração. Até que a população da próxima geração seja completada, é realizada a reprodução entre dois indivíduos aleatórios da geração atual e o filho é adicionado à nova geração.

Os algoritmos genéticos com chaves aleatórias tendenciosas ou *Biased Random-Key Genetic Algorithm* – BRKGA diferem do RKGA na forma em que os pais são selecionados para a reprodução e como a reprodução acontece (Gonçalves e Resende, 2011). No RKGA, todos os indivíduos da população têm a mesma probabilidade de serem escolhidos e a reprodução ocorre conforme explicado anteriormente. Já no BRKGA, um dos indivíduos é escolhido aleatoriamente sempre a partir do conjunto elite (conjunto formado com as melhores indivíduos) e o outro é escolhido aleatoriamente do conjunto não elite. A Figura 4 exemplifica o processo de evolução do BRKGA.

Na reprodução, para cada gene do filho a ser gerado é feito um sorteio tendencioso de qual pai ele herdará esse gene. Um parâmetro de probabilidade é então necessário para definir o quão tendenciosa será a escolha do grupo de elite. A Figura 5 mostra um exemplo de reprodução tendenciosa na qual a probabilidade de escolha do gene do pai proveniente do grupo de elite (P1) é 0,7. Observa-se nessa figura que, quando o número aleatório sorteado é menor que 0,7, o gene é herdado do pai P1 e quando é maior que 0,7, o gene é herdado do pai P2.

Figura 4: Evolução do BRKGA.



Fonte: Mainieri (2014).

Figura 5: Exemplo de reprodução tendenciosa.

P1	0,12	0,22	0,99	0,68	0,88	0,33
P2	0,52	0,62	0,19	0,18	0,22	0,01
Número aleatório	0,14	0,42	0,73	0,34	0,54	0,84
Parâmetro de probabilidade = 0.7	$<$	$<$	$>$	$<$	$<$	$>$
Herda de	P1	P1	P2	P1	P1	P2
F	0,12	0,22	0,19	0,68	0,88	0,01

Fonte: Elaborada pelos autores.

Neste trabalho, foi definida uma forma simples de decodificação, que é o ordenamento não decrescente do vetor de chaves aleatórias contínuas na qual a ordem de manutenção é representada pela posição relativa da chave no vetor. A Figura 6 mostra um indivíduo s' e sua decodificação s para a instância com 6 ordens de manutenção. No exemplo, a ordem de manutenção 3, representada pela chave 0,99 seria a última ordem a ser considerada pelo algoritmo de alocação, como pode ser observado na solução s.

Figura 6: Uma solução codificada s' e sua representação decodificada s.

$$s' = \left\langle 0{,}12, 0{,}22, 0{,}99, 0{,}68, 0{,}88, 0{,}33 \right\rangle \quad s = \left\langle 1, 2, 6, 4, 5, 3 \right\rangle$$

Fonte: Elaborada pelos autores.

O Algoritmo 7 mostra o pseudocódigo do BRKGA, baseado no código apresentado por De Faria Jr. et al. (2017). Na linha 1, a função é inicializada com um valor muito alto. O algoritmo segue gerando uma população de n chaves aleatórias, sendo n o número de manutenções (linha 2). Enquanto o critério de parada não for satisfeito (no caso, o limite de tempo do teste), o algoritmo continua evoluindo de acordo com os passos a seguir. Cada nova solução da população

é decodificada e avaliada (linha 4). A população é dividida em um conjunto elite e outro não elite (linha 5). A melhor solução é buscada (linha 6) e o seu índice armazenado (linha 7). Se o valor da solução encontrada é menor que o da melhor solução (linha 8), tanto o índice quanto a melhor solução são atualizados com os valores encontrados (linhas 9 e 10). Na linha 11 a próxima geração é inicializada com todos os indivíduos da população elite. Na linha 12 são gerados os conjuntos de população mutantes e adicionados à próxima geração (linha 13). A população é, então, completada até o número máximo de indivíduos de acordo com os passos a seguir. Um pai a é selecionado do grupo de elite (linha 15) e outro pai b do grupo não elite (linha 16). É, então, realizado um sorteio tendencioso para a escolha de cada gene da posição j do filho c a ser gerado, com a probabilidade maior de escolher o gene do pai do grupo de elite (linha 18). Esse gene escolhido da posição j de um dos pais é, então, atribuído ao filho c (linhas 20 e 22). Após a geração de um filho por esse processo, ele é adicionado à próxima população (linha 23). Na linha 24, a população é atualizada. Quando o critério de parada é satisfeito, a melhor solução é retornada (linha 25).

Algoritmo 7: Biased Random-Key Genetic Algorithm (BRKGA)

```
1  f* ← ∞
2  Gere uma população P com n vetores de chaves aleatórias
3  enquanto critério de parada não satisfeito faça
4      Decodifique e avalie o custo de cada nova solução em P
5      Divida P em dois conjuntos: P_e e P_ē
6      Encontre a melhor solução s+ de P_e
7      s+ ← argmin{f(s) | s ∈ P_e}
8      se f(s+) < f* então
9          s* ← s+
10         f* ← f(s*)
11     Inicialize a população da próxima geração: P+ ← P_e
12     Gere um conjunto P_m de mutantes, cada mutante com n chaves aleatórias
13     Adicione P_m à próxima geração: P+ ← P+ ∪ P_m
14     para cada i ← 1 a |P| − |P_e| − |P_m| faça
15         Selecione aleatoriamente um pai a de P_e
16         Selecione aleatoriamente um pai b de P_ē
17         para cada j ← 1 a n faça
18             Jogue uma moeda viciada com probabilidade ρ_e de sair cara
19             se cara então
20                 c[j] ← a[j]
21             senão
22                 c[j] ← b[j]
23         Adicione o filho c à população da próxima geração: P+ ← P+ ∪ {c}
24     Atualize a população: P ← P+
25 retorna s*
```

5.7. *Biased Random-Key Memetic Algorithm*

A fim de estabelecer um procedimento que combine boas características dos demais algoritmos apresentados anteriormente, foi desenvolvido um algoritmo baseado no BRKGA e também nos Algoritmos Meméticos (AMs), o *Biased Random-Key Memetic Algorithm* (BRKMA). De acordo com Neri e Cotta (2012), os AMs basicamente incluem operadores de busca local entre as etapas do processo evolucionário dos algortimos genéticos.

O BRKMA proposto neste trabalho aplica o procedimento de busca local definido

anteriormente (Algoritmo 5) a cada L ciclos evolucionários nas soluções decodificadas relativas aos indivíduos elite, ou seja, somente naqueles pertencentes à P_e. O objetivo da busca local é encontrar os ótimos locais da população elite, com isso, espera-se que o algoritmo encontre soluções que podem ser mais promissoras.

O Algoritmo 8 mostra o pseudocódigo do BRKMA, adaptado do código apresentado por De Faria Jr. et al. (2017). Este algoritmo difere do BRKGA (apresentado pelo Algoritmo 7) apenas pela adição da busca local (Algoritmo 5) aos elementos do grupo elite (linha 12) a cada L gerações (linha 11).

Algoritmo 8: Biased Random-Key Memetic Algorithm

```
1  f* ← ∞
2  Gere uma população P com n vetores de chaves aleatórias
3  enquanto critério de parada não satisfeito faça
4      Decodifique e avalie o custo de cada nova solução em P
5      Divida P em dois conjuntos: P_e e P_ē
6      Encontre a melhor solução s+ de P_e
7      s+ ← argmin{f(s) | s ∈ P_e}
8      se f(s+) < f* então
9          s* ← s+
10         f* ← f(s*)
11     se Número de gerações é múltiplo de L então
12         P_e ← {Busca-Local(e) ∀e ∈ P_e}
13     Inicialize a população da próxima geração: P+ ← P_e
14     Gere um conjunto P_m de mutantes, cada mutante com n chaves aleatórias
15     Adicione P_m à próxima geração: P+ ← P+ ∪ P_m
16     para cada i ← 1 a |P| − |P_e| − |P_m| faça
17         Selecione aleatoriamente um pai a de P_e
18         Selecione aleatoriamente um pai b de P_ē
19         para cada j ← 1 a n faça
20             Jogue uma moeda viciada com probabilidade ρ_e de sair cara
21             se cara então
22                 c[j] ← a[j]
23             senão
24                 c[j] ← b[j]
25         Adicione o filho c à população da próxima geração: P+ ← P+ ∪ {c}
26     Atualize a população: P ← P+
27 retorna s*
```

6. Experimentos e Resultados Computacionais

A formulação MILP foi programada na linguagem C++ utilizando-se o Concert Technology Library do CPLEX, versão 12.5 acadêmica, na configuração padrão. Os algoritmos SA, VNS, MSVNS, BRKGA e BRKMA foram implementados na linguagem C++.

No endereço www.decom.ufop.br/prof/marcone/projects/LTPMSP.html, estão disponíveis os arquivos com as instâncias usadas para testar os métodos de solução desenvolvidos, assim como os resultados completos gerados por esses algoritmos.

A Seção 6.1 descreve as características das instâncias usadas. Na Seção 6.2 são apresentadas as comparações de desempenho entre os métodos MILP, SA e VNS, enquanto, na Seção 6.3, são

comparados os algoritmos MSVNS, BRKGA e BRKMA.

6.1. Descrição das Instâncias de Teste

A equipe de engenharia de manutenção da indústria estudada forneceu os dados referentes a todas as ordens de manutenção do ano de 2016. Esses dados contêm informações de 33484 ordens de manutenção preventivas, envolvendo 1032 equipamentos e 145 equipes de trabalho.

Nos testes iniciais foi verificado que a formulação MILP só era capaz de resolver instâncias de dimensões pequenas, no caso, com até 80 ordens de manutenção, 5 equipamentos e 14 equipes. Por esse motivo, foram geradas instâncias menores, que são decomposições dessa instância real, conforme especificado na Tabela 1. Nessa tabela, as instâncias estão agrupadas na primeira coluna em conjuntos (P, M, G e GG), de acordo com a sua dimensão. A instância real foi incluída no conjunto GG. Na segunda coluna é mostrado o número de instâncias de cada conjunto. Na terceira, quarta e quinta colunas são mostrados os números, mínimo e máximo, de ordens de manutenção, equipamentos e equipes, respectivamente, de cada conjunto de instâncias. Como pode ser observado, os conjuntos P a GG envolvem um total de 139 instâncias.

Tabela 1: Características das instâncias.

Conjunto	# Instâncias	# Ordens	# Equipamentos	# Equipes
P	100	20 a 80	2 a 5	2 a 14
M	18	150 a 600	57 a 256	91 a 388
G	18	1200 a 4800	88 a 283	252 a 1286
GG	3	9600 a 33484	132 a 145	816 a 1032

Fonte: Elaborada pelos autores.

6.2. Comparação entre MILP, SA e VNS

O primeiro conjunto de experimentos envolveu as 100 instâncias do conjunto P e foram utilizadas para a validação do MILP, assim como para comparar os resultados produzidos pelo MILP, SA e o VNS, descritos nas Seções 4, 5.3 e 5.4, respectivamente.

Todos os testes foram realizados em um computador Intel Xeon CPU E3-1225 v5 @ 3.30GHz $\times$ 4, 32GB RAM, sob sistema operacional Ubuntu 16.04.3 LTS 64 bits. Cada algoritmo foi aplicado 10 vezes em cada instância.

O algoritmo SA, dado pelo Algoritmo 2, tem quatro parâmetros: *maxTemperatura*: temperatura máxima, *minTemperatura*: temperatura mínima, *maxIterações*: número de iterações por temperatura e α: fator de resfriamento. Para calibrar esses parâmetros foi utilizada a ferramenta Irace (*Iterated Racing for Automatic Algorithm Configuration*), desenvolvida por López-Ibáñez et al. (2016). Essa ferramenta se propõe a retornar o conjunto mais apropriado de valores para os parâmetros de um algoritmo dentro de uma faixa pré-determinada de valores, baseando-se em uma série de testes automatizados realizados em um conjunto de instâncias de treinamento. O Irace é executado dentro de um pacote do software R (R Development Core Team, 2008).

Os experimentos para a determinação dos valores dos parâmetros do algoritmo SA foram realizados utilizando-se 20 instâncias de treinamento do conjunto P. Para garantir a representatividade dessas instâncias, foram selecionadas combinações com valores de parâmetros em toda a faixa de variação deles. Todos os parâmetros assumiram valores discretos nos testes; não foram utilizados valores contínuos devido ao alto custo computacional requerido. O critério de parada foi o parâmetro *budget*, configurado no valor 5000. Os conjuntos de valores dos parâmetros foram definidos conforme a seguir:

- *maxTemperatura* $\in \{10; 20; 50; 100; 500; 1000\}$
- *minTemperatura* $\in \{0{,}1; 1{,}0; 5{,}0\}$
- *maxIterações* $\in \{1n; 2n; 5n; 10n\}$
- $\alpha \in \{0{,}001; 0{,}002; 0{,}005, 0{,}01; 0{,}02; 0{,}05; 0{,}1\}$.

Após executar o Irace, foram retornados os seguintes valores para os parâmetros: *maxTemperatura*: 500, *minTemperatura*: 0,1, *maxIterações*: $2n$ e α: 0,002.

Visando a uma comparação mais justa com o algoritmo SA, o único parâmetro do algoritmo VNS (Algoritmo 3), $t_{\max}$, foi fixado no tempo médio de execução do SA para cada instância.

Os resultados encontrados pela formulação MILP e pelos algoritmos VNS e SA estão descritos em detalhes na tabela disponível em www.decom.ufop.br/prof/marcone/projects/LTPMSP/MILPxSAxVNS.ods.

Por esses resultados, observa-se que, com o limite de uma hora de processamento, a formulação MILP encontrou a solução ótima (coluna *Gap* com o valor 0) em 26 instâncias (26% do total). O algoritmo SA também alcançou os valores ótimos em 26 instâncias e o VNS, em 24. No conjunto das instâncias P, o SA obteve os melhores resultados em 95% das instâncias, enquanto o VNS e o MILP alcançaram em 91% e 53%, respectivamente. Destaca-se que o tempo médio para a obtenção dos resultados do SA e do VNS foi de 50,6 segundos; tempo muito inferior ao limite de uma hora que foi configurado para o MILP.

A Tabela 2, por sua vez, apresenta o desvio percentual médio dos resultados obtidos, calculado conforme a Equação (18). Nessa equação, f_i^{Medio} representa o valor médio da função objetivo em 10 execuções da i-ésima instância e f_i^{Melhor}, o melhor valor encontrado por todos os métodos de solução para essa instância i, enquanto |P| representa o número de instâncias do conjunto testado, no caso, 100 instâncias.

$$Desvio = \frac{1}{|P|} \sum_{i=1}^{|P|} \frac{f_i^{Medio} - f_i^{Melhor}}{f_i^{Melhor}} \times 100 \tag{18}$$

Tabela 2: Comparação MILP × SA × VNS nas instâncias P.

	MILP	**SA**	**VNS**
Desvio	158,38	**0,95**	2,17

Fonte: Elaborada pelos autores.

Como os resultados não seguiam uma distribuição normal, foi escolhido o teste de Friedman (Lowry, 2018), considerando um nível de confiança de 95%, para verificar se havia, de fato, diferença estatística entre os algoritmos. O teste de Friedman retornou um valor-p igual a $2{,}67 \times 10^{-13}$. Como esse valor é menor que 0,05, ficou comprovada a existência de diferença estatística entre os algoritmos. Para saber se havia diferença estatística entre todos os pares de algoritmos foi aplicado o teste *post-hoc* (Pohlert, 2014). Conforme mostrado na Tabela 3, o teste *post-hoc* mostrou a existência de diferença estatística entre todos os pares de algoritmos. Note que, considerando o desvio médio relativo ao MILP, os algoritmos heurísticos SA e VNS obtiveram, em geral, soluções com qualidade superior às das soluções encontradas pela formulação MILP. Pode-se, assim, concluir que tanto o SA quanto o VNS foram melhores que o MILP nas 100 instâncias do conjunto P, sendo que o SA foi o melhor deles.

Em seguida, foi realizado um teste com a instância real do conjunto GG, que contém mais de 33 mil ordens de manutenção. Como dito anteriormente, O MILP não foi capaz de resolvê-la. Por outro lado, com os parâmetros calibrados para a solução das instâncias do conjunto P, o SA demandaria mais de 100 dias de processamento, o que seria inviável. Foi, então, realizada uma calibração empírica dos valores dos parâmetros do SA para diminuir esse tempo de processamento.

Tabela 3: Resultados do teste de Friedman nas instâncias P.

	MILP	**SA**
SA	$6{,}05\times10^{-105}$	
VNS	$8{,}40\times10^{-95}$	$7{,}51\times10^{-8}$

Fonte: Elaborada pelos autores.

Os valores empíricos adotados foram os seguintes: *maxTemperatura* = 100, *minTemperatura* = 1, *maxIterações* = n e α = 0,01. Os resultados obtidos foram, então, comparados com o indicador que é controlado pela equipe de engenharia de manutenção da empresa. Este indicador é o número percentual de ordens de manutenção que são executadas em relação àquelas programadas. O valor médio histórico é próximo a 50%, mesmo após o uso de horas extras e contratação de mão de obra terceirizada. Com o SA, foi possível alocar 90% das ordens de manutenção em 162 horas de processamento e com o VNS, 95,5%, nesse mesmo tempo. Cabe destacar, no entanto, que este indicador não é a função objetivo proposta neste trabalho, dada pela Equação (1). O indicador da empresa não leva em consideração a prioridade das ordens de manutenção e nem o número de equipes alocadas. O custo da função objetivo da solução do SA foi de 199472 e o do VNS de 188308; ou seja, ao tratar a instância real com os parâmetros calibrados empiricamente, o SA foi pior que o VNS em 5,93%.

Os resultados desses testes validaram o uso dos algoritmos heurísticos SA e VNS para a resolução de problemas de interesse prático. A formulação MILP, apesar de só resolver instâncias muito pequenas, foi importante para validar o correto funcionamento e desempenho dos algoritmos heurísticos. Após a validação do SA e do VNS, foram estudadas alternativas e diversos testes preliminares foram executados. Optou-se, então, por utilizar algoritmos que pudessem ser executados em paralelo, explorando-se a capacidade de multiprocessamento encontrada nos computadores atuais. Devido aos bons resultados obtidos pelo VNS na instância real e como ele pode ser facilmente paralelizado utilizando-se sua versão multipartida, o MSVNS, optou-se por utilizá-lo. Algoritmos genéticos foram utilizados com bons resultados em Adhikary et al. (2016) para o escalonamento de manutenções preventivas. Assim, como os algoritmos genéticos BRKGA e BRKMA também podem ser paralelizados, optou-se por utilizá-los em todos os conjuntos de instâncias. Embora tenha alcançado bons resultados para o conjunto de instâncias P, o SA não foi utilizado para tratar instâncias maiores devido ao elevado tempo computacional requerido para a sua execução. Os resultados desses experimentos são apresentados e discutidos a seguir.

6.3. Comparação entre MSVNS, BRKGA e BRKMA

O segundo conjunto de experimentos foi realizado em um computador Intel(R) Xeon(R) CPU E5-2660 v2 @ 2.20GHz $\times$ 40, com 384 GB de memória RAM sob o sistema operacional CentOS Release 6.8 (Final) Kernel Linux 2.6.32-642.1.1.el6.x86_64.

Todos os algoritmos foram configurados para serem executados em n segundos. Dessa forma, o parâmetro $tempo_{\max}$ foi definido como n. Todos os 40 núcleos do processador foram dedicados à execução em paralelo dos algoritmos. Após o limite de n segundos de execução em paralelo dos algoritmos, é computado o melhor resultado retornado. Na execução do MSVNS, cada núcleo processa um algoritmo VNS. Nas execuções dos algoritmos BRKGA e no BRKMA, o paralelismo dá-se ao decodificar os indivíduos em cada iteração do algoritmo.

Os algoritmos BRKGA e BRKMA considerados neste trabalho utilizam o *framework* desenvolvido por Toso e Resende (2015) que está disponível em http://mauricio.resende.info/src/brkgaAPI/. No BRKGA, foi necessário apenas implementar a função de codificação definida para o PPOMPLP, enquanto que para o BRKMA também foi necessário incluir a busca local definida e discutida na Seção 5.4.

Os algoritmos BRKGA e BRKMA têm quatro parâmetros em comum, descritos abaixo.

Além desses, o BRKMA faz uso do parâmetro L, que controla a frequência na qual a busca local é aplicada.

- $|P|$: tamanho da população;
- pe: fração da população P pertencente ao grupo de elite $= |P_e|/|P|$;
- pm: fração da população P a ser substituída por mutantes $= |P_m|/|P|$;
- ρ_e: probabilidade de herdar genes do grupo elite;

Note que o BRKMA é equivalente ao BRKGA quando $L = 0$, ou seja, quando a busca local não é aplicada. Dessa forma, a fim de determinar o melhor algoritmo (dentre o BRKGA e BRKMA) e os melhores valores de seus parâmetros, foi feita uma calibração utilizando a ferramenta Irace, já mencionada anteriormente.

Os experimentos para a determinação dos valores mais adequados para os parâmetros foram feitos utilizando-se 16 instâncias de treinamento. O tamanho da população foi definido para ser proporcional ao tamanho dos cromossomos, que neste caso é igual ao número de ordens de manutenção. Dessa forma, o tamanho da população P ficou definida como: $|P| = a \cdot n$. Apesar de ser aconselhado por Toso e Resende (2015) valores de $|P|$ maiores ou iguais ao do tamanho do cromossomo ($|P| \geq n$), testes preliminares mostraram que este valor permitia apenas algumas gerações para a instância real completa. Para verificar o desempenho com diferentes valores de $|P|$, primeiro o Irace foi executado com valores de $a \in \{1{,}0; 1{,}5; 2{,}0; 4{,}0\}$, seguindo as recomendações de Toso e Resende (2015). A melhor configuração sugerida pelo Irace foi com $a = 1$. Interessante observar que o valor de a está no limite inferior dos valores testados. Então, um segundo teste foi realizado com valores de $a \in \{1{,}00; 0{,}01; 0{,}005; 0{,}0025; 0{,}00125\}$. A melhor configuração sugerida pelo Irace foi com $a = 0{,}01$. Dessa forma, verificou-se que para as instâncias teste, o BRKGA teve um desempenho melhor para valores $|P| \leq n$. Com base nesses resultados, foi decidido utilizar uma fração do valor de n. Para que o tamanho da população não ficasse muito pequeno, quando instâncias menores fossem consideradas, o valor mínimo de $|P|$ foi definido como 20. Os valores finais testados para os parâmetros dos algoritmos BRKGA e BRKMA foram os seguintes:

- $a \in \{0{,}01; 0{,}005; 0{,}0025; 0{,}00125\}$
- $pe \in \{0{,}1; 0{,}15; 0{,}2; 0.25\}$
- $pm \in \{0{,}1; 0{,}15; 0{,}2; 0{,}25; 0{,}3\}$
- $\rho_e \in \{0{,}5; 0{,}55; 0{,}6; 0{,}65; 0{,}7; 0{,}75; 0{,}8\}$
- L: $\in \{0, 10, 20, 40, 80, 160\}$

Para o BRKGA, a melhor configuração de valores dos parâmetros retornada pelo Irace foi: $a = 0{,}01$, $pe = 0{,}15$, $pm = 0{,}15$ e $\rho_e = 0.8$. Já para o BRKMA, os melhores valores foram: $a = 0{,}00125$; $pe = 0{,}1$; $pm = 0{,}2$; $\rho_e = 0{,}7$ e $L = 80$. Dessa forma, tanto o BRKGA quanto o BRKMA foram testados em todas as instâncias usando-se esses valores retornados pelo Irace.

Os resultados completos desses testes estão disponíveis no endereço www.decom.ufop.br/prof/marcone/projects/LTPMSP/MSVNSxBRKGAxBRKMA.ods, assim como à página 55 de Aquino (2018). Uma análise quantitativa do número de melhores soluções encontrado por cada algoritmo no conjunto de todas as 139 instâncias mostra que os melhores resultados foram alcançados pelo MSVNS em 122 instâncias (87,8% do total), pelo BRKGA em 113 instâncias (81,3%) e pelo BRKMA em 112 instâncias (80,6%). Por esta análise, o MSVNS se mostrou superior aos demais algoritmos. No entanto, observa-se que, para muitas instâncias, os resultados eram muito próximos. Foi, então, feita uma análise estatística com relação ao desvio médio percentual, também calculado conforme a Equação (18), desta vez considerando todas as instâncias, para verificar se havia diferença estatística entre os algoritmos.

A Tabela 4 mostra o desvio médio percentual dos algoritmos em todos os conjuntos de instâncias. Como pode ser observado, o MSVNS pareceu ter melhor desempenho com relação

a esse indicador. Para verificar se havia diferença estatística entre os algoritmos, tal como anteriormente, verificou-se, inicialmene, que os resultados não seguiam uma distribuição normal. Ao aplicar o teste não paramétrico de Friedman, foi retornado um valor-p de $0{,}69$, que não comprovou haver diferença estatística entre os algoritmos.

Tabela 4: Desvio médio global dos algoritmos MSVNS, BRKGA e BRKMA no conjunto das instâncias.

MSVNS	**BRKGA**	**BRKMA**
4,64	6,36	5,04

Fonte: Elaborada pelos autores.

Em seguida, foi realizada uma outra análise para verificar se havia alguma diferença entre os algoritmos em função do tamanho da instância.

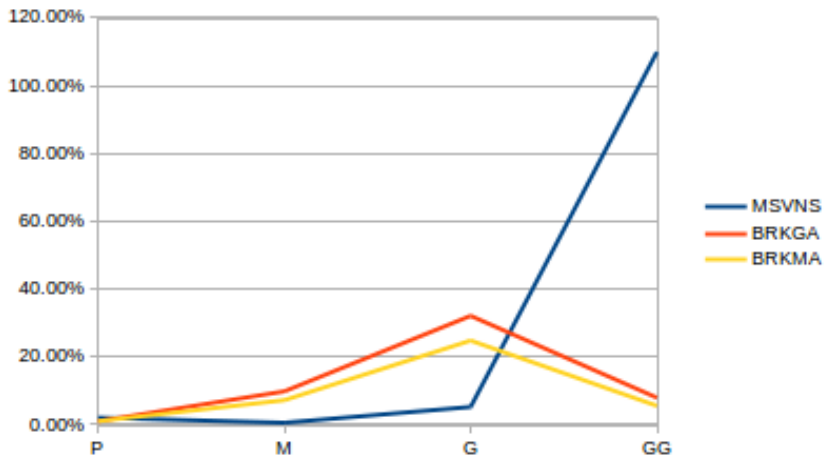
A Tabela 5 e a Figura 7 mostram os resultados dos desvios para cada conjunto de instâncias. Conforme será discutido a seguir, observa-se que para cada conjunto de instâncias há um algoritmo que possui um melhor desempenho.

Tabela 5: Desvio médio dos algoritmos MSVNS, BRKGA e BRKMA por conjunto de instâncias

Conjunto	**MSVNS**	**BRKGA**	**BRKMA**
P	2,09	**1,05**	1,07
M	**0,59**	9,85	7,25
G	**5,27**	32,14	24,85
GG	109,99	7,97	**5,59**

Fonte: Elaborada pelos autores.

Figura 7: Análise comparativa dos algoritmos por conjunto de instâncias.



Fonte: Elaborada pelos autores.

Nas instâncias de dimensões pequenas (linha P da Tabela 5), o teste de Friedman retornou um valor-p de $0{,}006572$, valor que confirma que os algoritmos são estatisticamente diferentes. Os testes *post-hoc* revelaram que todos os algoritmos possuem resultados estatisticamente diferentes. Assim, por essa análise, pode-se concluir que, dentre os algoritmos testados, o BRKGA é o melhor

deles (apesar de o BRKMA ter desvio médio percentual bem próximo) e o MSVNS é o pior nesse conjunto de instâncias.

Nas instâncias de dimensões médias (linha M da Tabela 5), o teste de Friedman retornou um valor-p de $0{,}009862$, confirmando que os algoritmos são estatisticamente diferentes. Os testes *post-hoc*, por sua vez, revelaram não haver diferença estatística entre os resultados dos algoritmos BRKGA e BRKMA, mas que há diferença estatística entre o MSVNS e os demais. Portanto, por essa análise pode-se concluir que o MSVNS é o melhor algoritmo no conjunto de instâncias médias, com desvio de 0,59%.

Nas instâncias de dimensões grandes (linha G da Tabela 5), o teste de Friedman retornou um valor-p de $0{,}000064$, mostrando que os algoritmos são estatisticamente diferentes. Os testes *post-hoc* mostraramm que todos os pares de algoritmos possuem resultados estatisticamente diferentes. Dessa forma, pode-se concluir que o MSVNS é o melhor algoritmo no conjunto de instâncias grandes, com desvio de 5,27%.

Nas instâncias de dimensões gigantes (linha GG da Tabela 5), entre as quais está a instância real completa com 33484 ordens de manutenção, a variação nos resultados foi expressiva. O teste de Friedman retornou um valor-p igual a $0{,}000006$, indicando haver diferença estatística entre os resultados dos algoritmos. A aplicação dos testes *post-hoc* mostrou que todos os pares de algoritmos têm resultados estatisticamente diferentes. Em função disso, conclui-se que o melhor algoritmo foi o BRKMA, com desvio de 5,59%.

Por fim, na comparação com o indicador que é controlado pela equipe de engenharia de manutenção da empresa (% de ordens de manutenções que são executadas em relação às programadas), observou-se que o BRKMA conseguiu atender a 95% das ordens de manutenção da instância real.

7. Conclusões

Este trabalho tratou o problema de alocação de ordens de manutenção preventiva. Para resolvê-lo, inicialmente foi desenvolvida uma formulação de programação linear inteira. Entretanto, com essa formulação, somente foi possível resolver instâncias de pequeno porte, com até 80 tarefas, 5 equipamentos e 14 equipes. Para tratar as instâncias reais, que envolviam mais de 33 mil ordens de serviço, foram, então, desenvolvidos algoritmos baseados nas metaheurísticas SA, VNS, MSVNS, BRKGA e BRKMA.

Os experimentos computacionais com esses algoritmos foram agrupados em dois conjuntos, em função do uso ou não de processamento paralelo. No primeiro, em que não foi feito uso de processamento paralelo, foram comparados os desempenhos dos algoritmos SA e VNS para tratar instâncias pequenas, assim como para tratar uma instância real. Nesses experimentos comprovou-se, estatisticamente, a superioridade do algoritmo SA em relação ao VNS e à formulação matemática no tratamento das instâncias pequenas. Ao se resolver a instância real, no entanto, verificou-se que o tempo computacional demandado pelo algoritmo SA, com seus parâmetros devidamente calibrados para as instâncias pequenas, era extremamente alto, tornando proibitivo seu uso na prática. Assim, foi necessário ajustar seus parâmetros para resolver a instância real. Entretanto, com esse ajuste, o desempenho do algoritmo SA foi inferior ao do VNS. Assim, constatado um melhor desempenho do VNS na resolução da instância real desse primeiro conjunto de experimentos e visando a reduzir seu tempo de processamento ou, em um mesmo tempo de processamento, melhorar a qualidade da solução final gerada por ele, decidiu-se utilizar uma versão facilmente paralelizável do VNS, que é o MSVNS. Além disso, decidiu-se comparar o desempenho dele com o de outros algoritmos de busca populacional, o BRKGA e o BRKMA.

Assim, no segundo conjunto de experimentos foram testados os algoritmos MSVNS, BRKGA e BRKMA, explorando-se processamento paralelo. Para esses experimentos foram incluídas mais 38 instâncias, todas derivadas da instância real, com dimensões classificadas como

pequenas, médias, grandes e gigantes. Considerando todas as instâncias nesse segundo conjunto de experimentos, os resultados mostraram que não houve um algoritmo estatisticamente melhor para a resolução do problema. No entanto, dependendo da dimensão da instância, um algoritmo diferente tem o melhor desempenho. Para as instâncias de dimensões pequenas, o algoritmo BRKGA obteve os melhores resultados. Nas instâncias de dimensões médias e grandes, o MSVNS teve o melhor desempenho. O BRKMA, por sua vez, foi o melhor nas instâncias de dimensões gigantes. Tendo em vista o interesse prático de resolução do mapa de 52 semanas, este resultado é o que deve ser considerado na empresa em estudo. Desta forma, recomenda-se o uso do algoritmo BRKMA para a resolução de instâncias reais do problema de ordens de manutenção preventiva.

Como trabalhos futuros, sugere-se considerar que o tempo de execução de uma ordem de serviço seja dependente da equipe que a executa. Assim, o plano de execução das ordens pode ser mais assertivo com aquele que foi efetivamente programado. Sugere-se, também, levar em consideração o custo com a perda financeira pela não execução de uma ordem de manutenção. Apesar de este trabalho ter tratado um caso específico de uma indústria mineradora, a mesma abordagem pode ser utilizada para tratar outros problemas de planejamento similares.

Agradecimentos. Os autores agradecem à Universidade Federal de Ouro Preto (UFOP) e aos órgãos de fomento Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Fundação de Amparo a Pesquisa do Estado de Minas Gerais (FAPEMIG) e Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelo apoio ao desenvolvimento deste trabalho. Os autores registram, também, seus agradecimentos aos revisores anônimos pelas valiosas sugestões, que contribuíram para a melhoria deste trabalho.

Referências

Adhikary, D. D., Bose, G. K., Jana, D. K., Bose, D. e Mitra, S. Availability and cost-centered preventive maintenance scheduling of continuous operating series systems using multi-objective genetic algorithm: a case study. *Quality Engineering*, v. 28, n. 3, p. 352–357, 2016.

Aquino, R. D. *Abordagem exata e heurísticas para o problema de planejamento de ordens de manutenção de longo prazo: um estudo de caso industrial de larga escala*. Dissertação (Mestrado em Ciência da Computação) – Programa de Pós-graduação em Ciência da Computação, Universidade Federal de Ouro Preto, Ouro Preto-MG, 2018.

Bean, J. C. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, v. 6, n. 2, p. 154–160, 1994.

Brucker, P. *Scheduling Algorithms*. Springer, 2007.

Chagas, J. B. C., Silveira, U. E. F., Santos, A. G. e Souza, M. J. F. A variable neighborhood search heuristic algorithm for the double vehicle routing problem with multiple stacks. *International Transactions in Operational Research*, v. 27, n. 1, p. 112–137, 2020.

Cordone, R. e Lulli, G. Multimode extensions of combinatorial optimization problems. *Electronic Notes in Discrete Mathematics*, v. 55, p. 17–20, 2016.

De Faria Jr., D., H, Resende, M. G. C. e Ernst, D. A biased random key genetic algorithm applied to the electric distribution network reconfiguration problem. *Journal of Heuristics*, v. 23, n. 6, p. 533–550, 2017.

Ferreira, K. M. e Queiroz, T. A. Two effective simulated annealing algorithms for the location-routing problem. *Applied Soft Computing*, v. 70, p. 389–422, 2018.

Gao, J., Sun, L. e Gen, M. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, v. 35, n. 9, p. 2892–2907, 2008.

Gonçalves, J. F. e Resende, M. G. C. Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, v. 17, n. 5, p. 487–525, 2011.

Hansen, P., Mladenović, N., Todosijević, R. e Hanafi, S. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization*, v. 5, n. 3, p. 423–454, 2017.

Holland, J. H. *Adaptation in Natural and Artificial Systems*. An introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. MIT Press, 1992.

Kirkpatrick, S., Gelatt, C. D. e Vecchi, M. P. Optimization by simulated annealing. *Science*, v. 220, n. 4598, p. 671–680, 1983.

Lowry, R. *Concepts & applications of inferential statistics*. 2018. Disponível em: http://vassarstats.net/textbook/. Acesso em: 06/11/2018.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M. e Stützle, T. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, v. 3, p. 43–58, 2016.

Mainieri, G. B. *Meta-heurística BRKGA aplicada a um problema de programação de tarefas no ambiente flowshop híbrido*. Tese (Doutorado em Engenharia de Produção) – Escola Politécnica, Universidade de São Paulo, São Paulo-SP, 2014.

Martí, R., Resende, M. G. C. e Ribeiro, C. C. Multi-start methods for combinatorial optimization. *European Journal of Operational Research*, v. 226, n. 1, p. 1–8, 2013.

Martinez, C., Loiseau, I., Resende, M. G. C. e Rodriguez, S. BRKGA algorithm for the capacitated arc routing problem. *Electronic Notes in Theoretical Computer Science*, v. 281, p. 69–83, 2011.

Mladenović, N. e Hansen, P. Variable neighborhood search. *Computers & Operations Research*, v. 24, n. 11, p. 1097–1100, 1997.

Neri, F. e Cotta, C. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, v. 2, p. 1–14, 2012.

Pinedo, M. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.

Pohlert, T. *The pairwise multiple comparison of mean ranks package*, 2014. Disponível em: https://CRAN.R-project.org/package=PMCMR. Acesso em: 06/11/2018.

R Development Core Team. *R: A language and environment for statistical computing*, 2008. Disponível em: www.R-project.org. Acesso em: 06/11/2018.

Saraiva, J. T., Pereira, M. L., Mendes, V. T. e Sousa, J. C. A simulated annealing based approach to solve the generator maintenance scheduling problem. *Electric Power Systems Research*, v. 81, n. 7, p. 1283–1291, 2011.

Sharma, A., Yadava, G. S. e Deshmukh, S. G. A literature review and future perspectives on maintenance optimization. *Journal of Quality in Maintenance Engineering*, v. 17, n. 1, p. 5–25, 2011.

Simões, J. M., Gomes, C. F. e Yasin, M. M. A literature review of maintenance performance measurement: A conceptual framework and directions for future research. *Journal of Quality in Maintenance Engineering*, v. 17, n. 2, p. 116–137, 2011.

Toso, R. F. e Resende, M. G. C. A C++ application programming interface for biased random-key genetic algorithms. *Optimization Methods and Software*, v. 30, n. 1, p. 81–93, 2015.

Yamayee, Z., Sidenblad, K. e Yoshimura, M. A computationally efficient optimal maintenance scheduling method. *IEEE Transactions on Power Apparatus and Systems*, v. PAS-102, n.2, p. 330–338, 1983.

Yao, X., Fernández-Gaucherand, E., Fu, M. C. e Marcus, S. I. Optimal preventive maintenance scheduling in semiconductor manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, v. 17, n. 3, p. 345–356, 2004.

Yazdani, M., Amiri, M. e Zandieh, M. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, v. 37, n. 1, p. 678–687, 2010.

Zudio, A., da Silva Costa, D. H., Masquio, B. P., Coelho, I. M. e Pinto, P. E. D. BRKGA/VND hybrid algorithm for the classic three-dimensional bin packing problem. *Electronic Notes in Discrete Mathematics*, v. 66, p. 175–182, 2018.