LVI SBPO SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL
DE 04 ATÉ 07 NOVEMBRO 2024
FORTALEZA | CEARÁ

REVENIMENTO PARALELO APLICADO AO SEQUENCIAMENTO DA PRODUÇÃO EM SISTEMAS DE MANUFATURA FLEXÍVEL

André Luís Barroso Almeida
Universidade Federal de Ouro Preto
Campus Morro do Cruzeiro, s/n, CEP 35400-000 - Ouro Preto/MG
andre.almeida@ifmg.edu.br

Joubert de Castro Lima
Universidade Federal de Ouro Preto
Campus Morro do Cruzeiro, s/n, CEP 35400-000 - Ouro Preto/MG
joubert@ufop.edu.br

Marco Antonio Moreira de Carvalho
Universidade Federal de Ouro Preto
Campus Morro do Cruzeiro, s/n, CEP 35400-000 - Ouro Preto/MG
mamc@ufop.edu.br

RESUMO

Este artigo relata uma aplicação da metaheurística revenimento paralelo (ou *parallel tempering*, PT). Aborda-se o problema de sequenciamento de tarefas e trocas de ferramentas em sistemas de manufatura flexível, o que envolve a determinação da sequência ótima de tarefas e a minimização de operações de *setup* em cada estágio da produção, sendo um problema NP-difícil. Este estudo desenvolve uma implementação paralela da metaheurística PT para resolver o problema abordado, obtendo uma redução substancial no tempo computacional em comparação com o algoritmo estado da arte.

PALAVRAS CHAVE. Sequenciamento. Manufatura Flexível. Revenimento Paralelo.

Tópicos (POI, MH, ADG&GP)

ABSTRACT

This paper reports an application of the parallel tempering metaheuristic (PT). The job sequencing and tool switching problem in flexible manufacturing systems is addressed, which involves determining the optimal sequence of jobs and minimizing setup operations at each stage of production, being an NP-hard problem . This study develops a parallel implementation of the PT metaheuristic to solve the addressed problem, obtaining a substantial reduction in computational time compared to the state-of-the-art algorithm.

KEYWORDS. Sequencing. Flexible Manufacturing. Parallel Tempering.

Paper topics (POI, MH, ADG&GP)

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

1. Introdução

Este estudo é um relato de aplicação da metaheurística revenimento paralelo a problemas de sequenciamento da produção em sistemas de manufatura flexível (SMF). Este tipo de sistema produtivo é caracterizado pela utilização de máquinas de comando numérico configuráveis e um sistema automatizado de fluxo de matéria prima, sendo comumente encontrado na indústria microeletrônica e metalúrgica. O problema de sequenciamento de tarefas e trocas de ferramentas (ou *job sequencing and tool switching problem*, SSP) é um problema fundamental neste tipo de sistema produtivo, originando diversas variantes práticas.

O SSP é constituído por um conjunto de tarefas $J = \{1, 2, 3, ..., n\}$, um conjunto de ferramentas $F = \{1, 2, 3, ..., m\}$ e de somente uma máquina de comando numérico capaz de armazenar, no máximo, C ferramentas de forma simultânea. Para cada tarefa $j \in J$ existe um subconjunto $F_j \in T$ contendo todas as ferramentas necessárias para seu processamento, com $F_j \leq C$. Durante o processo produtivo, diversas tarefas são produzidas em sequência, sendo necessário substituir as ferramentas que se encontram no *magazine*. Esse processo de troca de ferramentas, segundo Van Hop [2005], consome mais tempo do que qualquer outra atividade de configuração operacional em um SMF. O tempo de troca entre as ferramentas, considerada uma operação de *setup*, é idêntico, além de cada espaço do *magazine* comportar qualquer uma das ferramentas do conjunto F. O objetivo do SSP é determinar a sequência de tarefas e o plano de trocas de ferramentas que minimize o número total final de tais trocas.

De maneira geral, o SSP é composto por dois subproblemas: i) determinar a sequência ótima de execução das tarefas; e ii) determinar quais ferramentas devem ser trocadas antes de uma nova tarefa ser processada. O primeiro subproblema pertence a classe *NP-difícil*, não se conhecendo algoritmo em tempo determinístico polinomial capaz de resolvê-lo, enquanto o segundo subproblema pertence à classe *P*, devido a existência de algoritmos com tempo determinístico polinomial para sua solução. A Figura 1 apresenta um exemplo de instância do SSP e duas diferentes soluções.

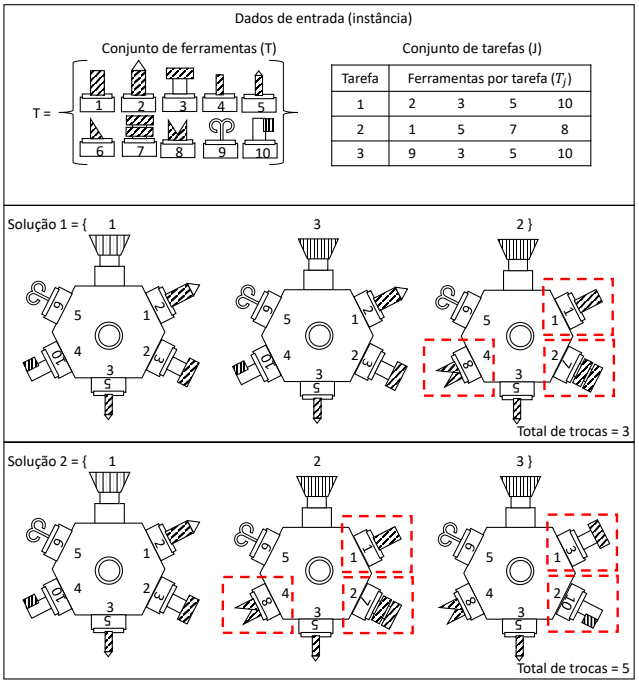


Figura 1: Exemplo com 10 ferramentas, 3 tarefas e uma capacidade máxima do *magazine* igual a 5.

Na Figura 1 pode-se observar no primeiro retângulo os dados do SSP. Para cada tarefa, o

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

respectivo conjunto de ferramentas necessárias é apresentado. No segundo retângulo é apresentada uma solução para o problema, em que a ordem de execução das tarefas foi definida como $S_1 = [1, 3, 2]$. Pode-se observar que para a execução da tarefa 3 não foi necessária nenhuma troca de ferramentas, pois todas já se encontravam no *magazine*. Porém, para executar a tarefa 2 foi necessário trocar três ferramentas. Assim, ao final da execução das três tarefas foram necessárias três trocas de ferramentas. No terceiro retângulo, uma solução alternativa é apresentada ($S_2 = [1, 2, 3]$). Contudo, após a execução da tarefa 1 foi necessária a troca de três ferramentas (2 por 1, 3 por 7 e 10 por 8) e após a execução da tarefa 2 foi necessária a troca de outras duas ferramentas (1 por 3 e 7 por 10), totalizando 5 trocas. É possível observar que a ordem de execução das tarefas influencia diretamente a quantidade de trocas necessárias.

Os estudos relacionados ao SSP tiveram início na década de 80, sendo formalmente apresentado por Tang e Denardo [1988]. Algum tempo depois, Crama et al. [1994] provaram que o SSP pertence a classe NP-*Difícil*. Em seguida, diversos estudos surgiram, relacionando o SSP a inúmeros problemas industriais em várias áreas. Na indústria eletrônica, encontram-se os estudos de Tzur e Altman [2004], Raduly-Baka et al. [2005] e Hirvikorpi et al. [2006]. Em sistemas computacionais os estudos de Djellab et al. [2000], Privault e Finke [2000] e Ghiani et al. [2007] e na indústria de bens de consumo encontram-se os estudos de Shirazi e Frizelle [2001] e Mecler et al. [2021].

Diversos estudos adotaram métodos baseados na programação matemática para solucionar o SSP [Laporte et al., 2004; Yanasse et al., 2009; Catanzaro et al., 2015]. Outros focaram na utilização de heurísticas [Salonen et al., 2006; Burger et al., 2015] e metaheurísticas. Dentre os métodos baseados em metaheurísticas, alguns adotaram abordagens baseadas em trajetória, incluindo a busca tabu [Al-Fawzan e Al-Sultan, 2003; Konak et al., 2008] e a busca local iterativa [Chaves et al., 2012; Paiva e Carvalho, 2017]. Outros adotaram métodos populacionais como o algoritmo genético [Jäger e Molitor, 2008; Mecler et al., 2021] ou algoritmos meméticos [Amaya et al., 2008, 2013; Ahmadi et al., 2018]. Atualmente, o algoritmo reconhecido como o estado da arte para solucionar as maiores instâncias do SSP é o denominado *hybrid genetic search*, baseado em algoritmos genéticos e proposto por [Mecler et al., 2021].

Como apresentado, diferentes abordagens sequenciais foram utilizadas durante os anos para solucionar o SSP, desde algoritmos exatos a métodos heurísticos. Todavia, com o aumento do tamanho das instâncias do problema, o tempo computacional tem se tornado um gargalo considerável. Assim, buscando obter melhores tempos de execução e melhores soluções, o presente estudo desenvolveu uma implementação paralela da metaheurística *revenimento paralelo* para resolver as maiores instâncias do SSP presentes na literatura. Os resultados obtidos demonstram que o revenimento paralelo é um método capaz de resolver o SSP com uma redução significativa do tempo computacional, obtendo uma redução de até 92,98% quando comparado com o algoritmo considerado o atual estado da arte, mantendo a qualidade das soluções geradas.

O restante deste estudo é estruturado em quatro seções. Na Seção 2, o método utilizado para solucionar o SSP é apresentado. A Seção 3 apresenta a implementação paralela do método. Logo depois, na Seção 4, os experimentos, comparando a implementação proposta com o estado da arte para solucionar o SSP, são apresentados. Por fim, na Seção 5, são apresentadas as conclusões.

2. Revenimento paralelo

O revenimento paralelo (ou *parallel tempering*, PT) é um método de amostragem de espaços de busca proposto por Geyer [1991] e formalizado por Hukushima e Nemoto [1996], obtendo a denominação de *replica exchange*. Esse método, como o *simulated annealing*, foi proposto no intuito de mitigar o problema de estagnação em ótimos locais do algoritmo de Metropolis, sendo atualmente um método vastamente utilizado nas áreas de física, química e biologia para resolver

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

problemas de simulação.

A abordagem consiste na coordenação da constante T do algoritmo de Metropolis, denominada temperatura, e na utilização da distribuição de *Boltzmann* para coordenar a diversificação e a intensificação da busca. Assim, em temperaturas altas, há mais diversificação que intensificação, e em temperaturas baixas, há mais intensificação do que diversificação.

No algoritmo PT, κ réplicas ou cópias do sistema ($R = \{r_1, r_2, r_3, ..., r_\kappa\}$) são simuladas em diferentes temperaturas, ou seja, diferentes valores de T. Isto corresponde à amostragem do espaço de busca, realizada por cadeias de Markov homogêneas. Neste processo, um mecanismo transforma uma solução r_i em uma nova solução r_i', dependendo unicamente da solução r_i e da probabilidade de transição entre as soluções. A homogeneidade das cadeias implica na não alteração da probabilidade de se atingir uma nova solução dada uma solução anterior. Cada réplica, a uma temperatura fixa e utilizando o algoritmo de Metropolis, simula o sistema via uma cadeia de Markov de comprimento pré-determinado. Após esse processo, uma troca de temperatura entre duas réplicas r_i e r_j é proposta. A probabilidade de efetivação da troca é dada pela equação

$$P(r_i \rightarrow r_j) = \min[1, exp(\Delta\beta\Delta E)] \tag{1}$$

em que $P(r_i \rightarrow r_j)$ corresponde a probabilidade de troca de temperaturas entre as réplicas r_i e r_j; $\Delta\beta = \frac{1}{T_j} - \frac{1}{T_i}$ corresponde à diferença entre o inverso das temperaturas e $\Delta E = E(r_j) - E(r_i)$ corresponde a diferença de energia entre as réplicas, i.e., a diferença entre os valores de avaliação das soluções correspondentes. Esse processo é ilustrado pela Figura 2, em que cada cor corresponde a uma réplica do sistema e cada quadrado corresponde a uma cadeia de Markov homogênea a uma temperatura T_i fixa. O processo de troca de temperatura entre as réplicas é representado pelas setas na diagonal na Figura 2 e o processo de manutenção da temperatura é representado pelas setas na horizontal.

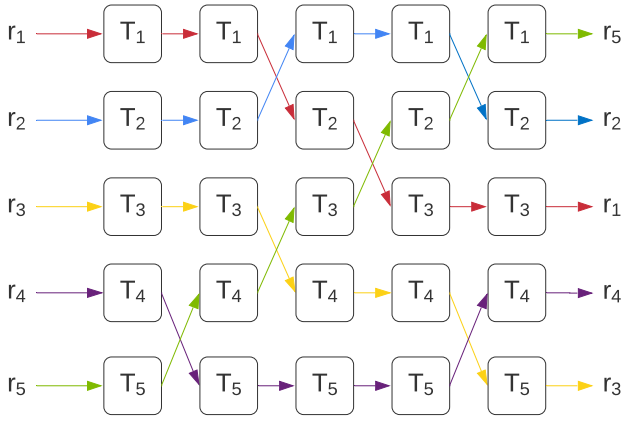


Figura 2: Representação de réplicas e cadeias de Markov no método PT.

Segundo Atchadé et al. [2011]; Rozada et al. [2019], a correta definição das temperaturas de cada réplica desempenha um papel crucial no desempenho do PT. Atualmente, os métodos existentes para definição da distribuição de temperaturas podem ser classificados como estáticos (temperaturas definidas antes do início do algoritmo) ou dinâmicos (temperaturas atualizadas durante a execução do algoritmo, a uma taxa pré-determinada). Os métodos estáticos mais populares incluem as distribuições i) linear; ii) linear-inverso; iii) exponencial; e iv) progressão geométrica. Entre os métodos dinâmicos, destacam-se i) o *feedback-optimized*; ii) a determinação de temperaturas que

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

iguale a probabilidade de taxas de aceitação de trocas; e iii) determinação de temperaturas que gere uma taxa de aceitação de 23%, de acordo com valor recomendado na literatura [Syed et al., 2019].

2.1. Codificação

Comumente, a codificação utilizada para representar o SSP é baseada em um vetor de n itens, em que n representa o número máximo de tarefas. Cada solução corresponde a uma permutação entre os n itens e sua representação pode ser observada na Figura 3. Embora, essa representação seja a mais usual, para o cálculo da função de avaliação é necessário uma representação mais completa.

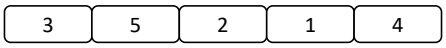


Figura 3: Representação de uma solução do SSP.

A representação mais completa, além de indicar a permutação das tarefas, exibe quais ferramentas são necessárias para a realização de cada uma das tarefas. Para tal, o solução é representada através de uma matriz binária, em que cada linha representa uma ferramenta e cada coluna representa uma tarefa. Se uma determinada ferramenta é necessária para realizar uma tarefa, a célula correspondente possui valor igual a 1, caso contrário, possui valor igual a 0. O mesmo exemplo da Figura 3 é representado de forma completa na Tabela 1.

Tabela 1: Matriz de ferramentas de uma instância do SSP.

Ferramentas	Tarefas				
	3	5	2	1	4
1	0	1	1	0	0
2	0	0	0	1	0
3	0	0	0	1	1
4	0	1	0	0	0
5	0	0	1	1	1
6	1	0	0	0	0
7	0	1	1	0	0
8	1	0	1	0	0
9	1	0	0	0	0
10	1	1	0	1	1

No algoritmo proposto para solucionar o SSP, as soluções são armazenadas conforme ilustrado na Figura 3. Entretanto, para o cálculo da função objetivo, esta representação é expandida para a codificação em matriz binária apresentada na Tabela 1.

2.2. Soluções iniciais

O processo de geração das soluções iniciais envolve a aleatorização da ordem de processamento das tarefas. Esse processo, consiste inicialmente na criação de um vetor contendo um elemento para cada tarefa do problema. Após a criação desse vetor, seus elementos são reorganizados de maneira aleatória, dando origem a uma nova permutação. Esse procedimento é repetido para cada uma das temperaturas consideradas no PT.

2.3. Função de avaliação

Recentemente, Cherniavskii e Goldengorin [2022] propuseram um novo algoritmo para determinação do plano de trocas de ferramentas. Esse novo método, denominado *greedy pipe construction algorithm* (GPCA) possui complexidade de tempo amortizada limitada por $O(Cn)$.

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

O seu funcionamento está intimamente relacionado com uma nova interpretação da função de avaliação. Nesta interpretação, o algoritmo busca preencher os espaços vazios do *magazine* ao longo dos estágios de produção de forma a minimizar a quantidade de trocas de ferramentas entre dois estágios consecutivos. Assim, devido ao seu melhor desempenho, a presente implementação do PT utiliza a implementação paralela do GPCA apresentada em detalhes por Almeida et al. [2023] e disponibilizada para uso.

2.4. Movimentos

Nas cadeias de Markov, em cada temperatura, um movimento é aplicado de forma a gerar uma nova solução vizinha. Neste estudo, foram avaliados três tipos de movimentos para problemas permutacionais como o SSP: 2-opt, 2-*swap* e a inserção aleatória. Cada um desses movimentos é descrito abaixo.

2-opt: dada uma permutação inicial de tarefas, duas posições são aleatoriamente sorteadas. O intervalo de elementos entre as duas posições sorteadas é invertido dentro da permutação, gerando uma nova solução.

2-*swap*: na primeira etapa, duas posições são aleatoriamente geradas. Na segunda etapa, os elementos das posições geradas são trocados.

Inserção aleatória: duas posições são aleatoriamente geradas. Logo em seguida, o elemento da primeira posição é inserido na segunda posição.

2.5. Critério de parada

Na implementação proposta para o PT, o critério de parada utilizado foi o número de propostas de trocas entre as temperaturas do PT. O valor utilizado foi definido através de experimentos preliminares descritos na Seção 4.1.

3. Implementação paralela

É proposta uma implementação que utiliza o modelo de programação paralela *dataflow*. O algoritmo é representado como um grafo em que os nós representam segmentos de código e os arcos representam dependências de dados entre eles. Após a criação do grafo, o paralelismo é exposto naturalmente, com cada nó sendo executado conforme suas dependências são satisfeitas, permitindo que instruções independentes sejam executadas paralelamente. Além disso, o padrão *dataflow* proporciona um gerenciamento mais eficiente de seções críticas, contribuindo para melhorias de desempenho.

Internamente, cada nó cujas dependências foram satisfeitas é adicionado a uma fila que controla o fluxo de execução. À medida que os nós são adicionados, cada um deles é vinculado a uma *thread* consumidora, que é responsável por executar o código associado ao nó. No final da execução, a *thread* consumidora notifica todos os nós adjacentes sobre sua conclusão, portanto, se cada nó adjacente não tiver mais dependências, ele será adicionado à fila de execução. Este processo se repete até que não haja mais nós a serem processados. Focando no desempenho do algoritmo, o número total de *threads* consumidoras é igual ao número de núcleos da CPU, evitando assim o custo extra relacionado à troca de contexto entre os núcleos da CPU.

Primeiro, a implementação gera a distribuição de temperaturas iniciais. Simultaneamente, é construído o grafo, composto por três tipos de nós conforme ilustrado na Figura 4. O primeiro tipo de nó, denominado MC, representa o segmento de código relacionado ao cálculo da cadeia de Markov homogênea. Vários nós do tipo MC são executados em paralelo. Neste tipo de nó, uma estrutura de *loop* é usada para executar um procedimento composto por quatro componentes: *move*, *evaluation*, *acceptance* e *exchange*. No primeiro componente, uma solução vizinha é gerada de

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

acordo com um movimento pré-definido. A seguir, o valor da função de avaliação é calculado no segundo componente. Ambos os componentes anteriores dependem do problema, variando de um problema para outro. O próximo componente aplica um critério de aceitação, e se a solução vizinha for eventualmente aceita por este critério, o quarto componente a torna a solução atual do nó.

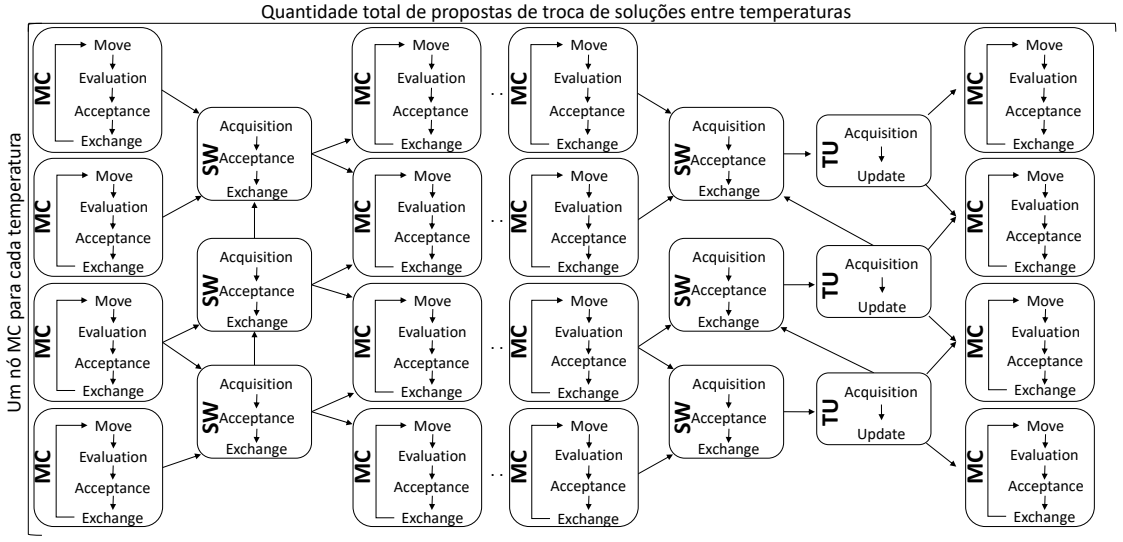


Figura 4: Modelo paralelo proposto.

O segundo tipo de nó, denominado SW, realiza a troca de soluções entre temperaturas que estão ligadas a nós do tipo MC. Este tipo de nó consiste em três componentes: *acquisition*, *acceptance* e *exchange*. Na fase *acquisition*, as soluções e temperaturas vinculadas aos nós MC conectados ao nó SW são recuperadas e transferidas para o componente *acceptance*. Neste componente, as soluções e temperaturas são submetidas a um critério probabilístico de aceitação de trocas, baseado na Equação 1. Se a troca for aceita, as soluções são trocadas entre os nós MC.

Por fim, o terceiro tipo de nó, denominado TU, é responsável por atualizar os valores das temperaturas. Este tipo de nó é gerado a uma taxa pré-definida, executando internamente dois componentes: *acquisition* e *update*. Inicialmente, o primeiro componente recupera informações vinculadas aos nós MC conectados ao nó TU. Esta informação é então passada para o segundo componente, que atualiza as temperaturas de acordo com o método definido.

Após a construção do grafo, o primeiro conjunto de nós MC é enviado para a fila de execução, iniciando uma reação em cadeia que culmina na execução de todos os nós. Além do modelo de programação paralela *dataflow*, é usado o conceito de *threadpool*. Neste conceito, *threads* são criadas apenas uma vez e reutilizadas para executar múltiplas tarefas. Assim, evita-se o custo computacional extra relacionado à criação de novas *threads* para cada nova tarefa. Além disso, este conceito permite um melhor controle das *threads* ativas.

A implementação do PT para problemas discretos descrita nesta seção é disponibilizada na forma de uma API em https://github.com/ALBA-CODES/PTAPI/ sob a licença *Creative Commons* BY-NC (CC BY-NC). Além do funcionamento básico do PT, a API inclui implementações da função de avaliação, dos movimentos e dos métodos estáticos e dinâmicos para determinação de temperaturas descritos na Seção 2.

4. Experimentos computacionais

Um conjunto de experimentos computacionais foi realizado com o objetivo de refinar e definir os parâmetros do PT, além de compará-lo, de forma justa, com o método recente considerado

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

o estado da arte para solução do SSP. O ambiente computacional adotado consiste em um computador com dois processadores Intel(R) Xeon(R) CPU E5-2660 v2 2.20GHz 10-Core 20 *threads* e 384 GB de memória RAM, sob o sistema operacional CentOS Linux 7.9.2009. O método PT foi implementado utilizando a linguagem de programação C++ e compilado utilizando o compilador GCC 10.1.0 e as opções -O3 e -lpthread.

As instâncias tradicionais da literatura possuem dimensões pequenas. Diante disto, Mecler et al. [2021] desenvolveram novas instâncias com o objetivo de estimular a pesquisa relacionada ao SSP, além de gerar uma base de comparação com instâncias mais desafiadoras. As novas instâncias foram divididas em três grupos (F_1, F_2 e F_3). Para cada combinação de número de tarefas, número de ferramentas e capacidade do *magazine*, cinco instâncias foram geradas. Um resumo das características destas instâncias pode ser observado na Tabela 2.

Tabela 2: Características das instâncias propostas por Mecler et al. [2021].

Grupo	Tarefas	Ferramentas	Capacidade	Número de instâncias
F_1	50	75	{25, 30, 35, 40}	20
F_2	60	90	{35, 40, 45, 50}	20
F_3	70	105	{40, 45, 50, 55}	20

4.1. Experimentos preliminares

Para definir os diversos parâmetros do PT, foi realizado um experimento preliminar baseado no método de configuração automática *irace* [López-Ibáñez et al., 2016]. Neste método, dado um conjunto de instâncias do problema e um conjunto de possíveis valores para cada parâmetro, o *irace* determina, de forma estatística, as melhores combinações de valores dos parâmetros. As instâncias utilizadas correspondem a 10% do total de instâncias, sendo estas selecionadas de forma aleatória. A Tabela 3 lista todos os parâmetros do PT e os valores selecionados pelo *irace*, destacados em negrito na referida tabela.

Tabela 3: Parâmetros considerados pelo *irace*. Valores selecionados destacados em negrito

Parâmetro	Opções
Temperatura inicial	{0,1; **0,2**; 0,3}
Temperatura final	{0,5; **1**; 5}
Número de temperaturas entre a mínima e a máxima	{12; **16**; 20; 24}
Tamanho da cadeia de Markov em cada temperatura	{300; **400**; 500; 600}
Quantidade de propostas de troca entre as temperaturas	{40000; 70000; **100000**; 130000; 160000}
Distribuição inicial das temperaturas[a]	{**1**; 2; 3; 4}
Tipo de movimento[b]	{**1**; 2; 3}
Tipo do ajuste dinâmico das temperaturas[c]	{0; 1; **2**; 3}
Taxa de atualização das temperaturas	{20000; 25000; 30000; **35000**}

[a] 1 - linear, 2 - linear-inverso, 3 - exponencial, 4 - progressão geométrica.
[b] 1 - 2-opt, 2 - 2-*swap*, 3 - inserção aleatória.
[c] 0 - desativado, 1 - 23%, 2 - iguala as taxas de aceitação, 3 - *feedback-optimized*.

4.2. Comparação com o estado da arte

Após a determinação da versão final do algoritmo, o PT foi comparado com o algoritmo reconhecido como o estado da arte, denominado *hybrid genetic search* (HGS), proposto por Mecler et al. [2021]. Para uma comparação justa, ambos métodos foram executados no mesmo ambiente computacional. Foram realizadas dez execuções independentes para cada instância utilizando todos os recursos disponíveis no ambiente computacional.

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

Os dados obtidos foram sumarizados na Tabela 4, em que cada linha corresponde a um conjunto com cinco instâncias. Na tabela, são apresentados o número de tarefas, o número de ferramentas e a capacidade do *magazine*. Para cada método, HGS e PT, são apresentados a média dos melhores valores das soluções geradas durante as dez execuções, o valor médio das soluções e o tempo computacional médio em segundos. Ademais, é apresentada a distância percentual (*gap*) entre as médias das melhores soluções e as médias de todas as soluções. O *gap* foi calculado como $gap = \frac{(PT-HGS)}{HGS} \times 100$. Os melhores resultados são destacados em negrito.

Tabela 4: Resultados para os grupos F_1, F_2 e F_3.

n	m	C	HGS			PT			*gap*	
			Melhor	μ	T(s)	Melhor	μ	T(s)	Melhor(%)	μ(%)
50	75	25	268,60	268,82	3302,42	**268,40**	**268,56**	**1004,01**	-0,07%	-0,10%
50	75	30	196,40	197,10	2594,11	**196,20**	**196,40**	**1107,12**	-0,10%	-0,36%
50	75	35	147,40	148,08	2731,85	**147,20**	**147,38**	**1223,64**	-0,14%	-0,47%
50	75	40	**109,80**	110,42	2310,42	**109,80**	**109,92**	**1342,78**	0,00%	-0,45%
60	90	35	**414,60**	415,60	12082,73	414,80	**415,40**	**1281,03**	0,05%	-0,05%
60	90	40	320,00	321,14	10979,21	**319,60**	**320,88**	**1414,80**	-0,12%	-0,08%
60	90	45	**247,20**	248,70	9304,33	247,60	**248,28**	**1558,12**	0,16%	-0,17%
60	90	50	**191,00**	192,84	8097,98	**191,00**	**191,94**	**1679,71**	0,00%	-0,47%
70	105	40	**576,60**	**577,44**	22535,20	577,00	577,94	**1580,93**	0,07%	0,09%
70	105	45	**459,00**	**459,90**	20405,51	459,20	460,32	**1728,86**	0,04%	0,09%
70	105	50	**369,60**	**371,08**	17456,87	370,00	371,12	**1844,73**	0,11%	0,01%
70	105	55	**298,60**	300,12	16468,95	**298,60**	**299,90**	**1967,96**	0,00%	-0,07%

Foram realizados testes estatísticos para fundamentar as conclusões das análises. Uma estrutura padrão de duas etapas para comparação estatística de algoritmos foi empregada. As amostras consideradas foram os valores médios da solução ou os melhores valores da solução gerados por cada algoritmo, conforme indicado na descrição de cada análise.

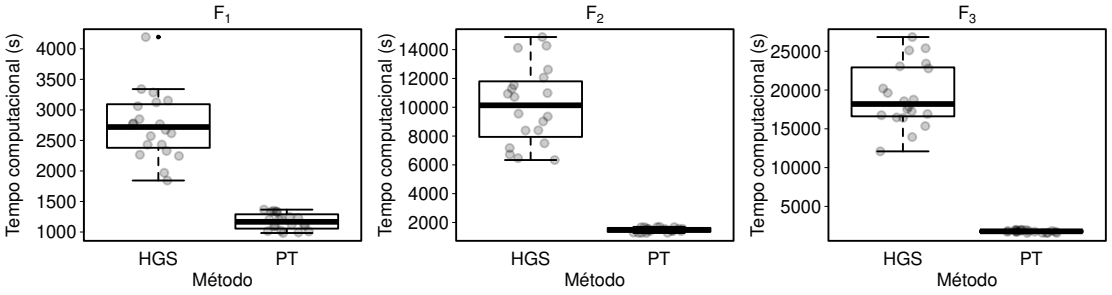
A primeira etapa testa se cada amostra pode ser modelada de acordo com a distribuição normal. O teste de normalidade *Shapiro-Wilk* foi utilizado para indicar com intervalo de confiança de 95% se um teste paramétrico (para amostras normalmente distribuídas) ou não paramétrico (para amostras não normalmente distribuídas) deveria ser utilizado na segunda etapa. Após o resultado da primeira etapa, um teste de hipótese estatística, seja o paramétrico *teste t de Student* ou o não paramétrico *Wilcoxon signed rank test*, foi aplicado para verificar se havia ou não, diferença significativa entre duas amostras pareadas comparadas com nível de significância de 0,05.

Considerando as amostras como as melhores soluções geradas por cada método, ambas puderam ser modeladas de acordo com a distribuição Normal. Posteriormente, o teste t indicou que não houve diferença significativa entre os resultados gerados. Considerando as amostras como as soluções médias geradas por cada método, novamente ambas as amostras puderam ser modeladas de acordo com a distribuição Normal. O teste t indicou que o PT obteve melhores resultados médios para as instâncias dos grupos F_1 e F_2, ao passo que as soluções de ambos os métodos para instâncias do grupo F_3 foram similares.

Ao avaliar o custo computacional, é possível observar claramente a superioridade do PT em relação ao HGS. O PT reduziu o tempo de execução em todos os doze conjuntos de instâncias, chegando a uma redução de até 92,98% quando comparado ao HGS. As Figuras 5 (a), (b) e (c) ilustram a comparação entre os tempos de execução.

Os dados indicam uma superioridade do PT em relação ao HGS. É possível observar ainda, uma maior dispersão dos valores obtidos pelo HGS. O PT demonstrou ser mais rápido que o HGS em

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879



(a) Tempo computacional. (b) Tempo computacional. (c) Tempo computacional.

Figura 5: Gráfico de *boxplot* da análise dos tempos computacionais.

todos os grupos de instâncias avaliadas. Notavelmente, o HGS demandou sete horas e trinta minutos para resolver uma das instâncias, um tempo impraticável em ambientes industriais. Em contraste, o PT necessitou apenas de trinta minutos para resolver a mesma instância. Outro indicador importante é a taxa de crescimento relativo ao tempo computacional entre os três grupos de instâncias. Ao comparar o menor tempo do grupo F_1 com o menor tempo do grupo F_2, observa-se que o tempo do PT cresceu a uma taxa de 27%, enquanto o tempo do HGS cresceu a uma taxa de 250%. Essas discrepâncias são reduzidas ao comparar o menor tempo do grupo F_2 com o menor tempo do grupo F_3, com o PT crescendo a uma taxa de 23%, enquanto o HGS cresceu a uma taxa de 100%.

5. Conclusão

Nesta aplicação do revenimento paralelo (ou *parallel tempering*, PT), abordou-se um problema derivado do sequenciamento de tarefas em máquinas flexíveis NP-Difícil, encontrado em ambientes industriais reais. Uma implementação paralela da metaheurística PT foi desenvolvida e aplicada. Os resultados mostram que o método proposto teve desempenho similar ao método considerado estado da arte em termos de qualidade da solução. Esse resultado evidencia a robustez do PT, que não possui nenhum componente específico ao problema abordado. Adicionalmente, o método possui a vantagem de requerer um tempo de execução significativamente menor, devido ao paralelismo empregado.

Os resultados obtidos neste estudo indicam que o PT emerge como um método promissor, destacando-se tanto em termos da qualidade das soluções alcançadas quanto no tempo computacional necessário. Esses achados não apenas validam a eficácia do PT no contexto abordado, mas também estimulam a condução de estudos de caso adicionais em problemas relacionados. Como trabalho futuro, sugere-se explorar novas estruturas de dados para codificar o SSP, a fim de aprimorar o tempo computacional do PT.

6. Agradecimentos

O presente estudo foi realizado com apoio do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) [Código de Financiamento 408341/2018-1]; Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) [Código de Financiamento 001]; Universidade Federal de Ouro Preto; e Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – Campus Ouro Preto.

Referências

Ahmadi, E., Goldengorin, B., Süer, G. A., e Mosadegh, H. (2018). A hybrid method of 2-tsp and novel learning-based ga for job sequencing and tool switching problem. *Applied Soft Computing*, 65:214–229.

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

Al-Fawzan, M. e Al-Sultan, K. (2003). A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. *Computers & industrial engineering*, 44(1):35–47.

Almeida, A., Lima, J., e Carvalho, M. (2023). Revisitando o algoritmo keep tools needed soonest: implementações seriais e paralelas. In *Simpósio Brasileiro de Pesquisa Operacional*, São José dos Campos, BR. SOBRAPO.

Amaya, J. E., Cotta, C., e Fernández, A. J. (2008). A memetic algorithm for the tool switching problem. In *International Workshop on Hybrid Metaheuristics*, p. 190–202, Berlin, Heidelberg. Springer Berlin Heidelberg.

Amaya, J. E., Cotta, C., e Fernández-Leiva, A. J. (2013). Cross entropy-based memetic algorithms: An application study over the tool switching problem. *International Journal of Computational Intelligence Systems*, 6(3):559–584.

Atchadé, Y. F., Roberts, G. O., e Rosenthal, J. S. (2011). Towards optimal scaling of metropolis-coupled markov chain monte carlo. *Statistics and Computing*, 21:555–568.

Burger, A. P., Jacobs, C., van Vuuren, J. H., e Visagie, S. E. (2015). Scheduling multi-colour print jobs with sequence-dependent setup times. *Journal of Scheduling*, 18(2):131–145.

Catanzaro, D., Gouveia, L., e Labbé, M. (2015). Improved integer linear programming formulations for the job sequencing and tool switching problem. *European journal of operational research*, 244(3):766–777.

Chaves, A. A., Senne, E. L. F., e Yanasse, H. H. (2012). Uma nova heurística para o problema de minimização de trocas de ferramentas. *Gestão & Produção*, 19:17–30.

Cherniavskii, M. e Goldengorin, B. (2022). An almost linear time complexity algorithm for the tool loading problem. *arXiv preprint arXiv:2207.02004*.

Crama, Y., Kolen, A. W. J., Oerlemans, A. G., e Spieksma, F. C. R. (1994). Minimizing the number of tool switches on a flexible machine. *International Journal of Flexible Manufacturing Systems*, 6(1):33–54.

Djellab, H., Djellab, K., e Gourgand, M. (2000). A new heuristic based on a hypergraph representation for the tool switching problem. *International Journal of Production Economics*, 64(1-3):165–176.

Geyer, C. J. (1991). Markov chain monte carlo maximum likelihood. In *Computing Science and Statistics, Proceedings of the 23rd Symposium on the Interface*, p. 156–163. Interface Foundation of North America.

Ghiani, G., Grieco, A., e Guerriero, E. (2007). An exact solution to the tlp problem in an nc machine. *Robotics and Computer-Integrated Manufacturing*, 23(6):645–649.

Hirvikorpi, M., Nevalainen, O., e Knuutila, T. (2006). Job ordering and management of wearing tools. *Engineering optimization*, 38(2):227–244.

Hukushima, K. e Nemoto, K. (1996). Exchange monte carlo method and application to spin glass simulations. *Journal of the Physical Society of Japan*, 65(6):1604–1608.

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879

Jäger, G. e Molitor, P. (2008). Algorithms and experimental study for the traveling salesman problem of second order. In *International Conference on Combinatorial Optimization and Applications*, p. 211–224, Berlin, Heidelberg. Springer.

Konak, A., Kulturel-Konak, S., e Azizoğlu, M. (2008). Minimizing the number of tool switching instants in flexible manufacturing systems. *International journal of production economics*, 116(2): 298–307.

Laporte, G., Salazar-Gonzalez, J. J., e Semet, F. (2004). Exact algorithms for the job sequencing and tool switching problem. *IIE transactions*, 36(1):37–45.

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

Mecler, J., Subramanian, A., e Vidal, T. (2021). A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, 127:105153.

Paiva, G. S. e Carvalho, M. A. M. (2017). Improved heuristic algorithms for the job sequencing and tool switching problem. *Computers & Operations Research*, 88:208–219.

Privault, C. e Finke, G. (2000). k-server problems with bulk requests: an application to tool switching in manufacturing. *Annals of Operations Research*, 96(1):255–269.

Raduly-Baka, C., Knuutila, T., e Nevalainen, O. S. (2005). Minimising the number of tool switches with tools of different sizes. Technical Report 690, Turku Centre for Computer Science, Finland.

Rozada, I., Aramon, M., Machta, J., e Katzgraber, H. G. (2019). Effects of setting temperatures in the parallel tempering monte carlo algorithm. *Physical Review E*, 100(4):043311.

Salonen, K., Smed, J., Johnsson, M., e Nevalainen, O. (2006). Grouping and sequencing pcb assembly jobs with minimum feeder setups. *Robotics and Computer-Integrated Manufacturing*, 22(4):297–305.

Shirazi, R. e Frizelle, G. (2001). Minimizing the number of tool switches on a flexible machine: an empirical study. *International Journal of Production Research*, 39(15):3547–3560.

Syed, S., Bouchard-Côté, A., Deligiannidis, G., e Doucet, A. (2019). Non-reversible parallel tempering: A scalable highly parallel mcmc scheme. *arXiv preprint arXiv:1905.02939*.

Tang, C. S. e Denardo, E. V. (1988). Models arising from a flexible manufacturing machine, part i: minimization of the number of tool switches. *Operations research*, 36(5):767–777.

Tzur, M. e Altman, A. (2004). Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. *IIE Transactions*, 36(2):95–110.

Van Hop, N. (2005). The tool-switching problem with magazine capacity and tool size constraints. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 35(5): 617–628.

Yanasse, H. H., Rodrigues, R. d. C. M., e Senne, E. L. F. (2009). Um algoritmo enumerativo baseado em ordenamento parcial para resolução do problema de minimização de trocas de ferramentas. *Gestão & Produção*, 16(3):370–381.

DOI: 10.59254/sbpo-2024-193879
http://live-proceedings.pantheonsite.io/p/193879