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

Resolução do Problema de Minimização de Trocas de Ferramentas com uma Busca Local Iterada Baseada em Chaves Aleatórias

Humberto Gimenes Macedo[a,b]
Camyla Ferreira Moreno[a,b]
[a]Universidade Federal de São Paulo (UNIFESP)
[b]Instituto Tecnológico de Aeronáutica (ITA)
[a]Av. Cesare Mansueto Giulio Lattes, 1201 - Eugênio de Melo, São José dos Campos - SP
[b]Praça Marechal Eduardo Gomes, 50 - Vila das Acácias, São José dos Campos - SP
gimenes.humberto@unifesp.br camyla.moreno@unifesp.br

RESUMO

A resolução de problemas práticos em otimização combinatória muitas vezes esbarra na intratabilidade computacional, requerendo o emprego de meta-heurísticas para encontrar soluções próximas da ótima em tempo viável. Dentre elas, tem-se aquelas baseadas em processos biológicos da natureza, como é o caso do algoritmo genético. Em um algoritmo genético, uma das formas para codificar a solução do problema consiste em utilizar chaves aleatórias, permitindo que a busca pela solução seja feita indiretamente no espaço de chaves aleatórias. Essa abordagem permite dividir a implementação em uma parte dependente e outra independente do problema. Recentemente, essa abordagem promissora foi estendida para outras meta-heurísticas. Nesse contexto, o objetivo deste trabalho é resolver o Problema de Minimização de Trocas de Ferramentas com uma busca local iterada implementada segundo o formalismo de chaves aleatórias. Os resultados mostraram a eficácia dessa abordagem alternativa em relação a usual em termos de tempo computacional e robustez.

PALAVRAS CHAVE. Problema de Minimização de Trocas de Ferramentas, Chaves Aleatórias, Busca Local Iterada.

Tópicos (Otimização Combinatória, Meta-heurísticas, PO na Indústria)

ABSTRACT

Solving practical problems in combinatorial optimization often encounters computational intractability, requiring the use of meta-heuristics to find solutions close to the optimum within feasible time frames. Among these, there are those based on biological processes in nature, such as the genetic algorithm. In a genetic algorithm, one way to encode the problem solution is to use random keys, allowing the search for the solution to be indirectly conducted in the space of random keys. This approach enables the implementation to be divided into parts that are dependent and independent of the problem. Recently, this promising approach has been extended to other meta-heuristics. In this context, the objective of this work is to solve the Job Sequencing and Tool Switching Problem with an iterated local search based on random keys. The results showed the effectiveness of this alternative approach compared to the usual one in terms of computational time and robustness.

KEYWORDS. Job Sequencing and Tool Switching Problem, Random Keys, Iterated Local Search.

Paper topics (Combinatorial Optimization, Meta-heuristics, OR in Industry)

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

1. Introdução

A maioria dos problemas relevantes de otimização combinatória são computacionalmente intratáveis, ou seja, são classificados como problemas NP-difíceis [Ausiello et al., 1999]. Na prática, isso significa que não há algoritmos conhecidos que sejam eficientes para resolvê-los. Assim, para essa classe de problemas, o usual é empregar métodos capazes de encontrar uma solução ao menos próxima da solução ótima em um tempo computacional razoável, como é o caso das meta-heurísticas. Meta-heurística é um termo cunhado por Glover [1986] e trata-se, formalmente, de uma estrutura algorítmica de alto nível e independente do problema que oferece um conjunto de diretrizes para desenvolver um algoritmo de otimização heurístico [Sörensen e Glover, 2013]. Em geral, uma meta-heurística apresenta três características fundamentais, a saber: (i) procura por uma solução próxima da ótima em vez de tentar encontrar a solução ótima; (ii) usualmente não apresenta uma prova rigorosa de convergência para a solução ótima; e (iii) geralmente é mais rápida do ponto de vista computacional que uma busca exaustiva. Atualmente, é possível encontrar uma miríade de meta-heurísticas na literatura. Uma parcela dessas meta-heurísticas foi projetada com o intuito de tentar imitar processos biológicos que ocorrem na natureza, como é o caso dos algoritmos evolutivos.

Os algoritmos evolutivos foram desenvolvidos com base em ideias de biólogos do século XIX, como Darwin e Mendel, que introduziram a teoria da evolução [Taillard, 2023]. A fim de resolver problemas complexos de otimização combinatória, esses algoritmos tentam reproduzir artificialmente a evolução de seres vivos por meio de operadores matemáticos de aptidão, seleção do mais apto, cruzamento e mutação. Esses operadores atuam em cromossomos (vetores) que representam as soluções do problema ao longo de várias gerações até a convergência. Entre os algoritmos evolutivos existentes, o algoritmo genético (GA, do inglês *Genetic Algorithm*) proposto por Holland [1975] é um dos mais proeminentes. Em um GA, a solução do problema de otimização precisa ser codificada em um cromossomo (vetor), onde a forma de codificação influencia significativamente o desempenho do algoritmo. Cada cromossomo consiste em uma cadeia de genes, cujos valores são chamados de alelos. Esses alelos podem ser, por exemplo, números binários, inteiros ou reais. Em particular, caso eles sejam valores reais e aleatórios no intervalo $[0,1]$, tem-se um GA com soluções codificadas por chaves aleatórias.

O algoritmo genético com chaves aleatórias (RKGA, do inglês *Random-Key Genetic Algorithm*) foi introduzido por Bean [1994] para resolver problemas de otimização combinatória envolvendo sequenciamento. Segundo ele, a abordagem de chaves aleatórias é adequada porque elimina o problema de produção de soluções infactíveis durante o cruzamento de soluções factíveis. No RKGA, as soluções codificadas com chaves aleatórias são fornecidas a um decodificador para serem transformadas em soluções factíveis. O decodificador é um algoritmo determinístico que tem como entrada um vetor de chaves aleatórias (cromossomo) e associa a ele uma solução factível do problema de otimização cujo valor da função objetivo pode ser prontamente calculado. Recentemente, Gonçalves e Resende [2011] aprimoraram o RKGA ao introduzir elitismo na forma como os indivíduos da população, representados por cromossomos, são escolhidos para cruzamento. Isso deu origem ao algoritmo genético com chaves aleatórias enviesadas (BRKGA, do inglês *Biased Random-Key Genetic Algorithm*). Assim, no RKGA os indivíduos são escolhidos aleatoriamente a partir da população como um todo, enquanto no BRKGA um deles deve advir, necessariamente, da elite da população. O sucesso do RKGA e BRKGA na solução de diversos problemas ratifica a eficiência de se utilizar chaves aleatórias para a codificação das soluções dos problemas.

Nas meta-heurísticas mencionadas, a busca pela solução do problema ocorre indiretamente no espaço de chaves aleatórias, o qual consiste de um hipercubo unitário cuja dimensão é igual ao tamanho do cromossomo. Durante a busca, o decodificador é invocado conforme ne-

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

cessário para mapear um cromossomo no espaço de chaves aleatórias para uma solução no espaço das soluções factíveis do problema, tornando possível, a partir disso, avaliar a função objetivo. A implementação da busca no espaço de chaves aleatórias pode ser feita de forma genérica e reutilizável, pois ela não depende do problema sendo resolvido. Por outro lado, a implementação do decodificador é altamente dependente do problema e deve ser realizada de forma bastante eficiente, devido à necessidade de mapeamentos frequentes. Assim sendo, tanto o RKGA quanto o BRKGA são meta-heurísticas compostas por uma componente independente do problema para realizar a busca, e uma componente dependente para realizar os mapeamentos. Visto que a componente independente só precisa ser implementada uma única vez, o esforço encontra-se no decodificador, que deve ser acoplado a ela para resolver um problema específico. Schuetz et al. [2022] sugerem uma generalização do formalismo de chaves aleatórias do BRKGA e de sua distinta separação de módulos dependentes e independentes do problema para meta-heurísticas alternativas que operam diretamente sobre o espaço de solução, como é o caso do recozimento simulado.

Dada a natureza inovadora e promissora do uso de meta-heurísticas alternativas com o formalismo de chaves aleatórias, este estudo visa reexaminar o problema de minimização de trocas de ferramentas previamente abordado por Macedo e Bueno [2023]. A proposta é aplicar uma busca local iterada baseada em chaves aleatórias em vez da usual que opera diretamente no espaço de solução. O objetivo é demonstrar a viabilidade e a eficiência dessa abordagem em termos de tempo computacional e robustez na resolução do problema em questão. Para alcançar esse propósito, este artigo está organizado da seguinte forma: a Seção 2 apresenta uma descrição detalhada do problema de otimização em análise, juntamente com o funcionamento do decodificador correspondente. Na Seção 3, é fornecida uma visão geral da busca local iterada e uma descrição de como ela foi implementada para o problema em consideração. Em seguida, na Seção 4, são apresentados e discutidos os resultados computacionais obtidos usando a abordagem proposta para resolver as instâncias de tamanho moderado apresentadas por Macedo e Bueno [2023]. Por fim, na Seção 5, são apresentadas as conclusões finais e delineadas possíveis direções para pesquisas futuras.

2. Descrição do Problema

Considere um conjunto $J = \{1, 2, \ldots, N\}$ de N tarefas que devem ser processadas sequencialmente e sem preempção por uma única máquina flexível. Seja $T = \{1,2, \ldots, M\}$ o conjunto das M ferramentas que precisam ser utilizadas pela máquina para o processamento de todas as tarefas em J. A máquina contém uma caixa de ferramentas capaz de armazenar $C < M$ ferramentas de cada vez, onde a ordem de armazenamento é irrelevante. Cada tarefa $j \in J$ requer um subconjunto não vazio de ferramentas $T_j \subset T$ para ser processada, as quais devem estar na caixa de ferramentas no momento em que a tarefa é processada. Assuma que $C \geq \max_{j \in J} |T_j|$ a fim de que a máquina possa processar todas as tarefas em J. Como a máquina não é capaz de armazenar todas as ferramentas de uma vez, algumas trocas podem ser necessárias para o processamento de tarefas consecutivas e distintas. Uma troca consiste da remoção de uma ferramenta da caixa e a subsequente inserção de outra em seu lugar. Por simplicidade, suponha que a máquina tenha um sistema de troca automática de ferramentas e que cada troca seja feita em um tempo desprezível.

O Problema de Minimização de Trocas de Ferramentas (MTSP, do inglês *Minimization of Tool Switches Problem*) visa determinar uma sequência de processamento de tarefas que minimize o número total de trocas de ferramentas. Vale destacar que uma instância do MTSP pode ser representada por uma quádrupla ordenada (N, M, C, A), em que $A \in \mathbb{R}^{M \times N}$ é uma matriz de valores lógicos chamada de matriz incidente. Essa matriz define as ferramentas necessárias para processar cada tarefa, ou seja, a entrada $A(p, q) = 1$ se, e somente se, a ferramenta $p + 1$ for requerida para processar a tarefa $q+1$[1]. Além disso, como as soluções do problema são permutações das N tarefas

[1]Neste trabalho, os índices que identificam a entrada de um vetor ou matriz começam em 0.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

em J, o espaço de solução $\mathcal{S}$ é formado por todas as $N!$ possibilidades de permutação, enquanto o espaço de chaves aleatórias é $\mathcal{X} = [0,1]^N$.

2.1. Decodificador

O decodificador para o MTSP empregado neste trabalho é o mesmo sugerido por Chaves et al. [2016], sendo constituído por duas partes. Na primeira, o vetor de chaves aleatórias (cromossomo) fornecido é mapeado em uma solução factível do problema, ou seja, em uma sequência de tarefas. Esse mapeamento é feito da seguinte forma: para cada chave aleatória no vetor dado associa-se uma única tarefa. Assim, a primeira chave aleatória é associada com a primeira tarefa, a segunda chave com a segunda tarefa e assim por diante. Por conseguinte, o vetor de chaves aleatórias é posto em ordem crescente. À medida que as chaves são colocadas em ordem crescente, a disposição das tarefas associadas muda em conformidade, dando origem à sequência de tarefas que o vetor de chaves aleatórias dado representa. A Figura 1 ilustra essa parte da decodificação para um exemplar com cinco tarefas.

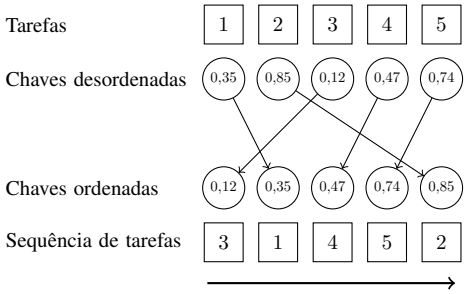

Figura 1: Mapeamento de um vetor de chaves aleatórias em uma solução factível do MTSP.

A segunda parte do decodificador consiste em calcular o número total de trocas de ferramentas necessárias para processar as tarefas na sequência determinada na primeira parte. Contudo, isso deve ser feito de forma otimizada, ou seja, é preciso que, durante o processamento de cada tarefa na sequência dada, as ferramentas na caixa sejam selecionadas de modo a minimizar o número total de trocas. Este é um subproblema do MTSP chamado de problema da substituição de ferramentas (TRP, do inglês *Tool Replacement Problem*). Para encontrar uma solução ótima para o TRP, Tang e Denardo [1988] propuseram um algoritmo denominado de política KTNS (do inglês *Keep Tool Needed Soonest*), que é baseado na ideia de sempre manter na caixa somente as ferramentas que serão necessárias em breve. A política KTNS percorre do início ao fim uma sequência de tarefas seguindo a duas regras simples, a saber: (i) para cada tarefa, apenas as ferramentas requeridas para processá-la são inseridas na caixa de ferramentas da máquina; e (ii) sempre que novas ferramentas são carregadas na caixa, mantêm-se nela somente aquelas que serão necessárias o mais breve possível. As ferramentas que serão necessárias o mais tarde possível são removidas conforme a necessidade para dar espaço às ferramentas que entram.

A Figura 2 ilustra o pseudocódigo da política KTNS, onde se verifica que seu custo computacional é da ordem de $O(MN)$. Observa-se que o algoritmo tem como entrada a matriz incidente A de uma instância e uma sequência de tarefas $s \in \mathcal{S}$. Além disso, sua implementação requer duas definições: (i) um vetor $W \in \{0,1\}^M$ cuja entrada $W(i)$ assume o valor 1 se a ferramenta $i+1$ está na caixa da máquina no instante n, e 0 caso contrário; e (ii) uma matriz $L \in \{0,1\}^{M,N}$ cuja entrada $L(i,n)$ é o primeiro instante, que pode ser o instante n ou algum instante após dele, que a ferramenta $i+1$ é requerida[2]. As linhas 1-20 carregam na caixa as ferramentas necessárias para

[2]A variável n que representa um instante começa em 0.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

processar a primeira tarefa, enquanto, se houver espaço, as linhas 21-31 preenchem a caixa com as ferramentas que serão requeridas mais cedo partir do instante de processamento da primeira tarefa. Por fim, para cada tarefa subsequente, as linhas 32-50 colocam na caixa as respectivas ferramentas necessárias, removendo sempre aquelas que serão requeridas o mais tarde possível a partir do instante corrente. Note que a caixa está sempre preenchida com as ferramentas requeridas no instante corrente e por aquelas que serão necessárias o mais breve possível a partir dele.

Algoritmo 1: POLÍTICA KTNS

Dados: Matriz incidente $A \in \{0,1\}^{M \times N}$ e sequência de tarefas $s \in \mathcal{S}$
Resultado: Número de trocas $p \in \mathbb{N}$
1 $n \leftarrow 0, c \leftarrow 0, p \leftarrow 0$. Defina $W \in \{0,1\}^{M}$ e $L \in \{0,1\}^{M \times N}$ e inicialize cada uma de suas entradas com 0
2 **para** $n = N - 1$ *até* 0 **faça**
3 **para** $i = 0$ *até* $M - 1$ **faça**
4 **se** $A(i, s(n)) == 1$ **então**
5 $L(i, n) \leftarrow n$
6 **senão**
7 **se** $n < N - 1$ **então**
8 $L(i, n) \leftarrow L(i, n + 1)$
9 **senão**
10 $L(i, n) \leftarrow N$
11 **fim**
12 **fim**
13 **fim**
14 **fim**
15 **para** $i = 0$ *até* $M - 1$ **faça**
16 **se** $L(i, n) == n$ **então**
17 $W(i) \leftarrow 1$
18 $c \leftarrow c + 1$
19 **fim**
20 **fim**
21 **enquanto** $c < C$ **faça**
22 $\ell \leftarrow \infty$
23 **para** $i = 0$ *até* $M - 1$ **faça**
24 **se** $W(i) == 0$ *e* $L(i, n) < \ell$ **então**
25 $t \leftarrow i$
26 $\ell = L(i, n)$
27 **fim**
28 **fim**
29 $W(t) \leftarrow 1$
30 $c \leftarrow c + 1$
31 **fim**
32 **para** $n = 1$ *até* $N - 1$ **faça**
33 **para** $i = 1$ *até* $M - 1$ **faça**
34 **se** $W(i) == 0$ *e* $L(i, n) == n$ **então**
35 $W(i) \leftarrow 1$
36 $c \leftarrow c + 1$
37 **fim**
38 **fim**
39 **enquanto** $c > C$ **faça**
40 $k \leftarrow n$
41 **para** $i = 1$ *até* $M - 1$ **faça**
42 **se** $W(i) == 1$ *e* $L(i, n) > k$ **então**
43 $t \leftarrow i$
44 $k \leftarrow L(i, n)$
45 **fim**
46 **fim**
47 $W(t) \leftarrow 0, c \leftarrow c - 1$
48 $p \leftarrow p + 1$
49 **fim**
50 **fim**

Figura 2: Pseudocódigo da política KTNS.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

A Figura 3 ilustra o funcionamento geral do decodificador para o mesmo exemplar da Figura 1, o qual reune o mapeamento e a política KTNS.

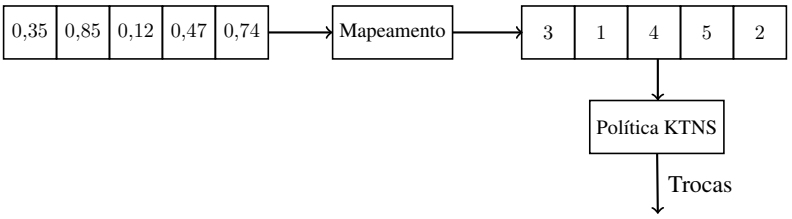

Figura 3: Representação pictórica do funcionamento do decodificador.

Doravante, o termo *decodificador* será empregado para designar uma chamada à função que implementa o decodificador e retorna o número de trocas.

3. Busca Local Iterada baseada em Chaves Aleatórias

Uma busca local iterada (ILS, do inglês *Iterated Local Search*) é uma meta-heurística que, partindo de uma solução inicial $x_0 \in \mathcal{X}$ gerada aleatoriamente ou por uma heurística construtiva, aplica reiteradamente as operações de perturbação, busca local e critério de aceitação para realizar a busca por uma solução melhor para uma instância do problema [Gendreau e Potvin, 2019]. A partir da melhor solução conhecida $x^* \in \mathcal{X}$, a ILS aplica uma perturbação sobre ela a fim de encontrar uma solução perturbada $x' \in \mathcal{X}$ em uma vizinhança próxima. Por conseguinte, uma busca local é feita sobre x' para aprimorá-la, atingindo-se, assim, uma solução ótima local $x^{*\prime}$. Por fim, no critério de aceitação, verifica-se se a solução ótima local recém descoberta ($x^{*\prime}$) aprimora a melhor solução conhecida até então (x^*). Em caso afirmativo, a melhor solução conhecida passa a ser a solução ótima local recém encontrada. Caso contrário, mantém-se a melhor solução conhecida. Em seguida, todas as etapas descritas são realizadas novamente a partir da melhor solução conhecida, atualizada ou não, até que o critério de parada seja atendido. A parada pode ocorrer quando um número máximo de iterações é atingido ou quando aprimora-se a melhor de todas as soluções conhecidas da instância. A Figura 4 ilustra o pseudocódigo da ILS baseada em chaves aleatórias utilizada para resolver o MTSP. Como é possível constatar, além do decodificador, três procedimentos precisam ser implementados, a saber: a heurística construtiva, a busca local e a perturbação. Estes serão descritos a seguir.

Algoritmo 2: BUSCA LOCAL ITERADA

Dados: número máximo de iterações $N_{\max}$, valor da função objetivo na melhor solução conhecida v, conjunto de ferramentas T e os subconjuntos $T_1, T_2, \ldots, T_N$.
Resultado: Solução ótima local $x^* \in \mathcal{X}$.

```
x_0 ← HeurísticaConstrutiva(T, T_1, T_2, ..., T_N)
x* ← BuscaLocal(x_0)
i ← 1
enquanto decodificador(x*) > v e i ≤ N_max faça
    x' ← Perturbação(x*)
    x*' ← BuscaLocal(x')
    se decodificador(x*') < decodificador(x*) então
        x* ← x*'
    fim
    i ← i + 1
fim
```

Figura 4: Pseudocódigo da ILS baseada em chaves aleatórias.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

Quanto a heurística construtiva, utilizou-se a proposta por Chaves et al. [2012], cujo pseudocódigo se encontra na Figura 5. Trata-se de um algoritmo guloso baseado em um grafo $G = (V, E)$ com múltiplas arestas, em que $V = \{1, 2, \ldots, M\}$ é o conjunto dos vértices e $E = \{(t_1, t_2, j) \in V^2 \times J \mid t_1, t_2 \in T_j\}$ é o conjunto das arestas. Dada uma instância do MTSP, constrói-se o grafo G cujos vértices são as ferramentas e cujas arestas entre um par de vértices representam as tarefas que requerem ambas as ferramentas para serem processadas (linha 1). Após a construção do grafo, tem-se uma estrutura de repetição que seleciona os vértices (ferramentas) em ordem crescente de grau e remove as arestas (tarefas) do grafo até que todos os vértices estejam isolados (linhas 2-14). Em cada iteração, escolhe-se o vértice de menor grau e, subsequentemente, remove-se a aresta do vértice escolhido que minimiza o número de trocas de ferramentas, em que cada remoção significa o processamento de uma tarefa. Então, remove-se todas as arestas do grafo associadas a essa tarefa, pois ela já foi processada. Por conseguinte, a caixa de ferramentas é atualizada para acomodar as ferramentas que foram requeridas para processá-la. No final do processo iterativo, todos os vértices estão isolados e todas as tarefas processadas em uma sequência que constitui a solução inicial do problema. Esta última é, então, convertida em um vetor de chaves aleatórias (linhas 15-18).

Fonte: Adaptado de Chaves et al. [2012].

Algoritmo 3: HEURÍSTICA CONSTRUTIVA

Dados: Conjunto de ferramentas T e subconjuntos $T_1, T_2, \ldots, T_N$.
Resultado: Solução candidata $x \in \mathcal{X}$.
1 $(V, E) \leftarrow$ ConstruirGrafo$(T, T_1, T_2, \ldots, T_N)$; $s \leftarrow \emptyset$; $C \leftarrow \emptyset$
2 **enquanto** $\exists v \in V$ *tal que* $grau(v) > 0$ **faça**
3 Escolha o vértice $v \in V$ com menor grau diferente de 0
4 **se** $grau(v) > 1$ **então**
5 Selecione a aresta $a \in E$ cuja tarefa associada minimiza o número de trocas.
6 Se houver empate, escolha a aresta cuja tarefa associada aparece mais vezes no grafo.
7 **senão**
8 Selecione a única aresta $a \in E$ conectada ao vértice v.
9 **fim**
10 $j \leftarrow$ tarefa associada à aresta a selecionada.
11 $s \leftarrow s \cup \{ j \}$
12 $E \leftarrow$ atualizarGrafo(E, j)
13 $C \leftarrow$ atualizarCaixa(C, T_j)
14 **fim**
15 $x \leftarrow$ vetor com N posições para armazenar as chaves aleatórias que representam s
16 **para** $i = 0$ *até* $N - 1$ **faça**
17 $x[s[i] - 1] \leftarrow i/N$
18 **fim**

Figura 5: Pseudocódigo da heurística construtiva.

Esta estratégia é conhecida como busca local de primeira melhoria. Apesar de sua simplicidade, essa escolha se justifica pelo fato de que a troca de chaves aleatórias entre dois índices de um vetor de chaves aleatórias tem o mesmo efeito no espaço de solução original do problema, não havendo a necessidade de escrever um código adicional. O pseudocódigo correspondente pode ser encontrado na Figura 6 e será explicado a seguir. A estrutura de repetição nas linhas 3-9 tem por objetivo construir uma sequência de pares ordenados que representam as possibilidades de troca entre duas tarefas, levando-se em conta a simetria dessa operação. Uma vez que há N tarefas, tem-se $N(N-1)/2$ possíveis pares de troca. A busca local de primeira melhora propriamente dita é reali-

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

zada na estrutura de repetição nas linhas 11-22. Inicialmente, uma troca entre duas chaves aleatórias é feita nas linhas 12-16. Caso ela melhore a melhor solução conhecida (x), esta última é atualizada com a solução recém encontrada x'. Caso contrário, mantém-se a melhor solução conhecida. Em todo caso, a busca prossegue a partir da melhor solução conhecida, estando ela atualizada ou não. Esse procedimento é feito até que todas as possibilidades de troca sejam consideradas.

Algoritmo 4: BUSCA LOCAL

Dados: Solução a ser aprimorada $x \in \mathcal{X}$.
Resultado: Solução ótima local $x' \in \mathcal{X}$.

```
1  Defina c ∈ ℕ^{2×N(N−1)/2}, a ∈ ℝ e k ∈ ℕ_0
2  k ← 0
3  para i = 1 até N faça
4      para j = i + 1 até N faça
5          c(0, k) = i
6          c(1, k) = j
7          k ← k + 1
8      fim
9  fim
10 x' ← x
11 para w = 1 até N(N − 1)/2 faça
12     i ← c(0, w − 1)
13     j ← c(1, w − 1)
14     a ← x'(i − 1)
15     x'(i − 1) ← x'(j − 1)
16     x'(j − 1) ← a
17     se decodificador(x') ≤ decodificador(x) então
18         x ← x'
19     senão
20         x' ← x
21     fim
22 fim
```

Figura 6: Pseudocódigo da busca local.

Quanto à estratégia de perturbação adotada, o pseudocódigo correspondente pode ser encontrado na Figura 7. Este algoritmo oferece quatro opções de perturbação, e a seleção de qual aplicar a um vetor de chaves aleatórias $x \in \mathcal{X}$ é feita aleatoriamente, conforme indicado na linha 3. A primeira opção consiste em tomar o complemento em relação à unidade da chave aleatória armazenada na posição i de x (linha 6), onde i é um inteiro gerado aleatoriamente entre 0 e $N-1$ (linha 4). A segunda opção, por sua vez, troca de posição as chaves aleatórias nas posições i e j, com j sendo também um inteiro aleatório entre 0 e $N-1$ (linhas 9-12). Em seguida, a terceira opção simplesmente substitui a chave aleatória na posição i de x por uma outra chave aleatória (linha 15). Por fim, a quarta e última opção consiste na troca de duas chaves aleatórias consecutivas, cuja posição da primeira é escolhida de forma aleatória (linhas 17-20). Vale destacar que o número de perturbações que são feitas é controlado por um parâmetro $I \in \mathbb{N}$ que deve ser fornecido.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

Algoritmo 5: PERTURBAÇÃO

Dados: Solução a ser perturbada $x \in \mathcal{X}$ e intensidade I da perturbação.
Resultado: Solução perturbada $x' \in \mathcal{X}$.

$x' \leftarrow x$
para $k = 1$ *até* I **faça**
  $t \leftarrow$ inteiro aleatório entre 1 e 4
  $i \leftarrow$ inteiro aleatório entre 0 e $N-1$
  se t == *1* **então**
    $x'(i) = 1 - x'(i)$
  senão
    se t == *2* **então**
      $j \leftarrow$ inteiro aleatório entre 0 e $N-1$
      $a \leftarrow x'(i)$
      $x'(i) \leftarrow x'(j)$
      $x'(j) \leftarrow a$
    senão
      se t == *3* **então**
        $x'(i) \leftarrow$ um número real aleatório entre 0 e 1
      senão
        $z \leftarrow$ um inteiro aleatório entre 0 e $N-2$
        $a \leftarrow x'(z)$
        $x'(z) \leftarrow x'(z+1)$
        $x'(z+1) \leftarrow a$
      fim
    fim
  fim
fim

Figura 7: Pseudocódigo da perturbação.

4. Resultados Computacionais

A ILS baseada em chaves aleatórias descrita na seção 3, assim como o decodificador detalhado na subseção 2.1, foram implementados em `C++` e utilizados para resolver as 20 instâncias de tamanho moderado do MTSP de Macedo e Bueno [2023]. Vale ressaltar, entretanto, que por conta da facilidade de se construir e manipular grafos com múltiplas arestas em `Python`, para a heurística construtiva utilizou-se o mesmo código escrito por Macedo e Bueno [2023], o qual foi chamado a partir da linguagem `C++` por meio de uma interface entre as linguagens `Python` e `C++` escrita pelos autores deste texto. A partir dos resultados obtidos, uma comparação em termos de robustez e tempo computacional foi feita entre a abordagem de baseada em chaves aleatórias, escrita em `C++`, e a usual de Macedo e Bueno [2023] escrita em `Python`, a qual opera diretamente sobre o espaço de solução. Para uma comparação justa, os experimentos foram conduzidos no mesmo ambiente computacional, ou seja, em um computador com sistema operacional Windows® 11 de 64 *bits*, processador Intel Core® i5-11400H 2,70 GHz e 16 GB de RAM.

A Tabela 1 apresenta os resultados computacionais obtidos. A primeira coluna identifica as instâncias por meio de quádruplas ordenada (veja a seção 2), enquanto a segunda coluna contém o valor assumido pela função objetivo quando avaliada na melhor solução conhecida (BKS, *Best Known Solution*) de cada instância. Vale ressaltar que a BKS de cada instância já é a solução ótima, pois elas foram resolvidas até a otimalidade por Macedo e Bueno [2023] com o modelo de Tang e Denardo [1988]. Para cada instância, são reportados quatro valores nas colunas `BV`, `Gap`, `Média` e $\overline{\texttt{CPU (s)}}$, os quais estão associados a cada uma das abordagens de implementação da ILS. O primeiro valor refere-se ao melhor valor (BV, do inglês *Best Value*) que a função objetivo adquire ao longo de 10 execuções da respectiva ILS. O segundo valor mensura o erro relativo percentual entre o BV e aquele assumido pela função objetivo na BKS[3]. O terceiro valor é a média aritmética

[3]Em outras palavras, *Gap (%)* $= \left|\frac{\text{BV}}{\text{BKS}} - 1\right| \times 100$.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

dos valores assumidos pela função objetivo ao final de um conjunto de 10 execuções da respectiva ILS. Por fim, o quarto valor é a média do tempo de CPU requerido para a obtenção de uma solução melhor ou igual à BKS. Essa média temporal também é calculada ao final de um conjunto de 10 execuções.

Tabela 1: Resultados computacionais obtidos ao resolver as 20 instâncias de tamanho moderado do MTSP de Macedo e Bueno [2023] por meio de uma ILS usual, implementada em `Python`, e uma baseada em chaves aleatórias, implementada em `C++`.

		ILS usual de Macedo e Bueno [2023]				**ILS baseada em chaves aleatórias**			
Instância	BKS	BV	*Gap (%)*	Média	$\overline{\text{CPU(s)}}$	BV	*Gap (%)*	Média	$\overline{\text{CPU(s)}}$
$(15, 20, 6, B_1)$	19	19	0,00	19,20	12,530	19	0,00	19,00	0,080
$(15, 20, 6, B_2)$	26	26	0,00	27,50	14,850	26	0,00	26,00	1,694
$(15, 20, 6, B_3)$	22	22	0,00	22,80	12,760	22	0,00	22,00	3,801
$(15, 20, 6, B_4)$	18	18	0,00	20,40	11,940	18	0,00	18,00	8,753
$(15, 20, 6, B_5)$	20	20	0,00	20,90	11,920	20	0,00	20,00	0,047
$(15, 20, 8, B_6)$	14	14	0,00	14,80	11,380	14	0,00	14,00	9,166
$(15, 20, 8, B_7)$	10	10	0,00	10,00	10,990	10	0,00	10,00	0,003
$(15, 20, 8, B_8)$	13	13	0,00	13,00	11,470	13	0,00	13,00	0,108
$(15, 20, 8, B_9)$	15	15	0,00	15,70	12,180	15	0,00	15,00	0,692
$(15, 20, 8, B_{10})$	14	14	0,00	14,60	11,290	14	0,00	14,00	0,083
$(15, 20, 10, B_{11})$	10	10	0,00	10,00	10,620	10	0,00	10,00	0,011
$(15, 20, 10, B_{12})$	8	8	0,00	8,00	10,530	8	0,00	8,00	0,002
$(15, 20, 10, B_{13})$	10	10	0,00	10,00	11,160	10	0,00	10,00	0,006
$(15, 20, 10, B_{14})$	8	8	0,00	8,10	10,710	8	0,00	8,00	0,002
$(15, 20, 10, B_{15})$	9	9	0,00	9,00	11,380	9	0,00	9,00	0,142
$(15, 20, 12, B_{16})$	8	8	0,00	8,00	10,360	8	0,00	8,00	0,006
$(15, 20, 12, B_{17})$	8	8	0,00	8,00	11,070	8	0,00	8,00	0,036
$(15, 20, 12, B_{18})$	7	7	0,00	7,00	11,020	7	0,00	7,00	0,006
$(15, 20, 12, B_{19})$	6	6	0,00	6,00	10,540	6	0,00	6,00	0,003
$(15, 20, 12, B_{20})$	8	8	0,00	8,00	11,110	8	0,00	8,00	0,006

Com base nos valores da Tabela 1, algumas observações podem ser realizadas. A primeira observação diz respeito ao fato da ILS baseada em chaves aleatórias ter demonstrado ser mais robusta que a ILS usual, ao menos para as instâncias em questão. De fato, os valores na coluna `Média` para a ILS baseada em chaves aleatórias são idênticos aos valores na coluna `BKS`, o que não acontece com a usual. Como todas as BKS's são ótimas, isso significa que, para cada instância, o valor ótimo foi atingido em cada uma das 10 execuções da ILS baseada em chaves aleatórias. Essa observação pode ser explicada pelo fato da perturbação implementada conforme a Figura 7 ser mais agressiva do que aquela utilizada por Macedo e Bueno [2023], promovendo uma maior diversificação durante a busca. Além disso, o fato da linguagem `C++` ser compilada permite cobrir uma região maior do espaço de busca em um tempo menor, aumentando a probabilidade de encontrar soluções melhores e, eventualmente, a solução ótima.

Uma segunda observação relevante é a redução significativa do tempo computacional em praticamente todas as instâncias analisadas. Para a ILS usual, o tempo médio de CPU ao longo das instâncias não varia muito, mesmo quando a capacidade da caixa C aumenta e o problema se torna mais trivial. Por outro lado, para a ILS baseada em chaves aleatórias há uma variação maior do tempo médio de CPU, pois ela foi capaz de capturar melhor a trivialidade das instâncias à medida que C aumenta, resolvendo-as na maioria dos casos em um tempo inferior a $0{,}1\,\text{s}$. Além disso, a média aritmética dos 20 valores na coluna $\overline{\texttt{CPU(s)}}$ para a ILS usual é $11{,}4905\,\text{s}$, enquanto para a baseada em chaves aleatórias ela é de apenas $1{,}2323\,\text{s}$. Isso indica uma redução percentual de $89{,}27\%$. É importante ressaltar, contudo, que para as instâncias associadas às matrizes de incidência

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

B_4 e B_6, o tempo médio de CPU é semelhante ao da ILS usual. Essa é uma constatação interessante, pois essas instâncias não requereram um tempo tão discrepante em relação as demais instâncias na ILS usual.

5. Conclusões e Trabalhos Futuros

Este trabalho apresenta uma ILS que realiza a busca pela solução do MTSP no espaço de chaves aleatórias, uma abordagem alternativa e promissora que é tradicionalmente empregada com sucesso nas meta-heurísticas RKGA e BRKGA para resolver problemas variados. Por conta disso, a ILS baseada em chaves aleatórias proposta foi implementada em `C++` e aplicada para resolver as 20 instâncias de tamanho moderado de Macedo e Bueno [2023], que originalmente as resolveram com uma ILS usual que opera diretamente sobre o espaço de solução. Dessa forma, o objetivo era aprimorar os resultados obtidos em termos de robustez e tempo computacional, além de explorar a viabilidade de integrar chaves aleatórias em uma meta-heurística que normalmente opera apenas com soluções pertencentes ao espaço de busca original do problema. Os resultados computacionais revelaram que a ILS baseada em chaves aleatórias foi capaz de reduzir em $89{,}27\%$ a média aritmética do tempo médio de CPU de todas as 20 instâncias. Essa redução significativa está relacionado com o fato da linguagem `C++` ser compilada e também com o fato de se ter utilizado uma perturbação mais agressiva que permitiu uma maior diversificação durante a busca. Além disso, a ILS baseada em chaves aleatórias conseguiu capturar melhor a trivialidade das instâncias com as maiores caixas de ferramenta. Nessas instâncias, o tempo médio de CPU é muito menor que nas instâncias mais complexas. Isso também teve sua parcela de contribuição na redução temporal mencionada. Por fim, a ILS baseada em chaves aleatórias não apresentou variabilidade ao longo das execuções em nenhuma das instâncias, consistentemente alcançando a solução ótima em todas elas. Esses resultados evidenciam sua robustez em comparação com a ILS convencional.

Para trabalhos futuros, a abordagem proposta será aplicada em instâncias encontradas em estudos acadêmicos de revistas científicas internacionais. Isso nos permitirá avaliar sua eficácia e desempenho em cenários de maior complexidade e escala, frequentemente considerados nessas publicações [Mecler et al., 2021]. Essa extensão do escopo de aplicação contribuirá para a generalização dos resultados obtidos até o momento, consolidando a abordagem em questão como uma ferramenta viável e eficiente para que possa ser aplicada na resolução de outros problemas de otimização tão desafiadores quanto o MTSP.

Agradecimentos

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), processos 88887.669708/2022-00 e 88887.644322/2021-00.

Referências

Ausiello, G., Marchetti-Spaccamela, A., Crescenzi, P., Gambosi, G., Protasi, M., e Kann, V. (1999). *Complexity and Approximation*. Springer, Berlin, Heidelberg. ISBN 978-3-642-63581-6 978-3-642-58412-1. URL `http://link.springer.com/10.1007/978-3-642-58412-1`.

Bean, J. C. (1994). Genetic Algorithms and Random Keys for Sequencing and Optimization. *ORSA Journal on Computing*, 6(2):154–160. ISSN 0899-1499, 2326-3245. URL `https://pubsonline.informs.org/doi/10.1287/ijoc.6.2.154`.

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. ISSN 0104-530X, 1806-9649. URL `http://www.scielo.br/j/gp/a/PyN9C4V8GDczsxzQndQhX4w/?lang=pt`. Publisher: Universidade Federal de São Carlos.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en

Chaves, A., Lorena, L., Senne, E., e Resende, M. (2016). Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. *Computers & Operations Research*, 67: 174–183. ISSN 03050548. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054815002403.

Gendreau, M. e Potvin, J.-Y., editors (2019). *Handbook of Metaheuristics*, volume 272 of *International Series in Operations Research & Management Science*. Springer International Publishing, Cham. ISBN 978-3-319-91085-7. URL http://link.springer.com/10.1007/978-3-319-91086-4.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549. ISSN 03050548. URL https://linkinghub.elsevier.com/retrieve/pii/0305054886900481.

Gonçalves, J. F. e Resende, M. G. C. (2011). Biased random-key genetic algorithms for combinatorial optimization. *Journal of Heuristics*, 17(5):487–525. ISSN 1572-9397. URL https://doi.org/10.1007/s10732-010-9143-1.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. U Michigan Press, Oxford, England. ISBN 978-0-472-08460-9. Pages: viii, 183.

Macedo, H. G. e Bueno, L. F. (2023). Estudo Comparativo de Diferentes Métodos de Solução para o Problema de Minimização de Trocas de Ferramentas. In *Anais do Simpósio Brasileiro de Pesquisa Operacional*, volume 55, São José dos Campos, SP. Publisher: Galoá.

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. ISSN 0305-0548. URL https://www.sciencedirect.com/science/article/pii/S0305054820302707.

Schuetz, M. J., Brubaker, J. K., Montagu, H., Van Dijk, Y., Klepsch, J., Ross, P., Luckow, A., Resende, M. G., e Katzgraber, H. G. (2022). Optimization of Robot-Trajectory Planning with Nature-Inspired and Hybrid Quantum Algorithms. *Physical Review Applied*, 18(5):054045. ISSN 2331-7019. URL https://link.aps.org/doi/10.1103/PhysRevApplied.18.054045.

Sörensen, K. e Glover, F. W. (2013). Metaheuristics. In Gass, S. I. e Fu, M. C., editors, *Encyclopedia of Operations Research and Management Science*, p. 960–970. Springer US, Boston, MA. ISBN 978-1-4419-1153-7. URL https://doi.org/10.1007/978-1-4419-1153-7_1167.

Taillard, D. (2023). *Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem*. Graduate Texts in Operations Research. Springer International Publishing, Cham. ISBN 978-3-031-13713-6. URL https://link.springer.com/10.1007/978-3-031-13714-3.

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. ISSN 0030-364X, 1526-5463. URL http://pubsonline.informs.org/doi/10.1287/opre.36.5.767.

DOI: 10.59254/sbpo-2024-193916
https://proceedings.science/p/193916?lang=en