

## Metaheurísticas Evolutivas para o Problema do Caixeiro Viajante com um Ajudante Voador\*

Matheus de Almeida Mazieiro

Pedro Guilherme Torres das Neves

Mário César San Felice

Universidade Federal de São Carlos

Rod. Washington Luiz, km 235, São Carlos - SP, Brasil

matheus.mazieiro@hotmail.com

pedrogtneves@gmail.com

felice@ufscar.br

Pedro Henrique Del Bianco Hokama

Universidade Federal de Itajuba

Av. BPS, 1303 - Pinheirinho, Itajuba - MG, Brasil

hokama@unifei.edu.br

### RESUMO

Neste artigo abordamos o problema do Caixeiro Viajante com um Ajudante Voador, onde um caminhão e um drone colaboram para realizar entregas no menor tempo possível. Para resolvê-lo, utilizamos duas metaheurísticas evolutivas, a *Hybrid Genetic Search with Adaptive Diversity Control* e o *Multi-Parent Biased Random-Key Genetic Algorithm*, ambas utilizando decodificadores baseados no algoritmo *SplitLazy*. Também propusemos uma busca local baseada na 2-opt, e uma família de heurísticas, chamada Bolhas Elípticas. Nossos métodos melhoraram os resultados da literatura para 96% das categorias de instâncias avaliadas, sendo que cada categoria tem 10 instâncias com mesmo tamanho e tipo de distribuição.

**PALAVRAS CHAVE.** Caixeiro Viajante, Metaheurística, Algoritmo Genético, Drone.

**Tópicos (Metaheurísticas, Otimização combinatória)**

### ABSTRACT

In this paper we address the Flying Sidekick Traveling Salesman Problem, where a truck and a drone collaborate to make deliveries in the shortest possible time. To tackle it, we employed two evolutionary metaheuristics, the *Hybrid Genetic Search with Adaptive Diversity Control* and the *Multi-Parent Biased Random-Key Genetic Algorithm*, both using decoders based on the *SplitLazy* algorithm. We also proposed a local search based on the 2-opt, and a family of heuristics, called *Elliptic Bubbles*. Our methods improved the literature results for 96% of the categories of instances evaluated, where each category has 10 instances with the same size and distribution type.

**KEYWORDS.** Traveling Salesman Problem, Metaheuristic, Genetic Algorithm, Drone.

**Paper topics (Metaheuristics, Combinatorial Optimization)**

\*O presente trabalho foi realizado com apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), Brasil. Processos 2023/10948-3 e 2023/10946-0

## 1. Introdução

Problemas de roteamento de veículos são relevantes para inúmeras áreas da indústria, em particular em setores de logística, e incluem o clássico Problema do Caixeiro Viajante (TSP, do inglês *Traveling Salesman Problem*). De modo informal, no TSP temos que encontrar um ciclo mais curto que um caixeiro viajante pode utilizar para visitar cada cidade uma única vez e retornar à origem. Problemas de roteamento apresentam grande dificuldade computacional, com a maioria sendo NP-Difícil, *i.e.*, problemas que não possuem algoritmo exato com complexidade de tempo polinomial a não ser que  $P = NP$ . Assim, esses problemas têm recebido grande atenção das comunidades científicas das áreas de algoritmos, otimização combinatória e pesquisa operacional.

Com o avanço tecnológico recente, veículos aéreos não-tripulados, também chamados de drones, tem se tornado cada vez mais usados em diversas áreas da sociedade, como segurança pública, monitoramento de tráfego, logística, entre outros. Nesse contexto, empresas como Amazon e Google têm desenvolvido projetos que usam drones juntamente de caminhões para auxiliar na realização de entregas Macrina et al. [2020]. Esses projetos indicam o interesse e relevância de problemas de roteamento envolvendo drones.

O primeiro problema deste tipo foi proposto por Murray e Chu [2015], que introduziram o Problema do Caixeiro Viajante com um Ajudante Voador (FSTSP, do inglês *Flying Sidekick Traveling Salesman Problem*). No FSTSP, que é o foco deste trabalho, um caminhão e um drone percorrem ciclos parecidos coordenando as entregas e sincronizando os reencontros. Murray e Chu [2015] também propuseram o Problema do Caixeiro Viajante com Escalonamento Paralelo de Drones, no qual um caminhão, juntamente com uma frota de drones realizam entregas, mas nessa variante o caminhão fica responsável principalmente por entregas distantes, enquanto os drones entregam os pacotes próximos ao centro de distribuição. Murray e Chu [2015] também apresentam uma formulação em Programação Linear Inteira para o FSTSP.

Agatz et al. [2018] propõe uma variante do FSTSP chamada de Problema do Caixeiro Viajante com Drone (TSP-D, do inglês *Traveling Salesman Problem with Drone*), o qual permite que um nó seja repetido a fim de sincronizar o drone e o caminhão, *i.e.*, no TSP-D o ponto de partida e de encontro do drone com o caminhão pode ser o mesmo. Além de propor esse problema, Agatz et al. [2018] apresenta uma formulação em Programação Linear Inteira compacta para o mesmo, bem como heurísticas que permitem encontrar soluções para instancias maiores que as apresentadas por Murray e Chu [2015].

Ha et al. [2020] aborda o FSTSP à partir da metaheurística evolutiva *Hybrid Genetic Search with Adaptive Diversity Control* (HGSADC), que foi apresentada por Vidal et al. [2012]. Essa metaheurística combina mecanismos da seleção natural, com buscas locais e gerenciamento da diversidade da população. Para decodificar as soluções desse algoritmo genético, Ha et al. [2020] aplica o algoritmo *Split* (Ha et al. [2018]), que obtém uma solução para o FSTSP a partir de um ciclo Hamiltoniano. Outra metaheurística evolutiva central neste trabalho é a *Multi-Parent Biased Random-Key Genetic Algorithm* (MP-BRKGA) [Andrade et al., 2021], uma versão mais moderna do *Biased Random-Key Genetic Algorithm* (BRKGA) [Gonçalves e Resende, 2013], que inclui um método de cruzamento utilizando vários pais.

Neste trabalho atacamos o FSTSP através de heurísticas evolutivas projetadas usando o HGSADC e o MP-BRKGA. Ambas utilizam uma versão mais eficiente do *Split*, chamado *SplitLazy* [Hokama et al., 2024], para decodificar as soluções dos cromossomos. Além disso, para a heurística do HGSADC são propostos uma busca local baseada na 2-opt, busca local clássica do TSP, e um algoritmo de crossover. Também é proposta uma família de heurísticas para o FSTSP, chamada Bolhas Elípticas, cujas soluções são usadas como *warm start* para a heurística do MP-BRKGA. Por fim, apresentamos resultados computacionais melhorando a literatura.

## 2. Descrição do Problema

Neste estudo abordamos o FSTSP, onde um conjunto de clientes precisa receber uma encomenda cada. Para tanto, temos um caminhão equipado com um drone, e o objetivo é coordená-los de forma que todas as entregas sejam realizadas eficientemente, sem visitar o mesmo cliente mais de uma vez. O caminhão e o drone começam juntos no depósito, sendo que o drone pode se separar do caminhão, permitindo que estes façam entregas em paralelo. Em particular, por limitações de capacidade de carga, após partir do caminhão o drone pode atender a no máximo um cliente, enquanto o caminhão segue atendendo a outros. Após realizar a entrega, o drone deve reencontrar o caminhão. Ao final, tanto o caminhão quanto o drone devem retornar ao depósito.

De modo formal, temos um grafo completo e não orientado  $G = (V, E)$  com  $n + 1$  nós, onde  $v_0$  representa o depósito e os  $n$  nós  $v_1, \dots, v_n$  representam clientes. A cada arco  $e \in E$  está associado um custo  $c(e)$ . Definimos uma operação como um par  $o = (t, d)$ , em que  $t$ , é uma sequência de nós visitados pelo caminhão, enquanto  $d$  é a sequência de nós visitados pelo drone. Note que o caminhão e o drone devem estar no mesmo nó  $l$  no início e  $r$  no final de cada operação, e que em cada operação o drone visita no máximo um cliente. Assim,  $d$  pode ser vazia, ou corresponder a uma tripla  $(l, v, r)$ . O custo  $c(t)$  é a soma dos custos das arestas do caminho  $t$ , e  $c'(d) = c(d)/\alpha$ , sendo  $\alpha$  o fator de velocidade do drone e  $c(d) = c(l, v) + c(v, r)$ .

Seja  $O$  o conjunto de operações possíveis,  $c_o = \max\{c(t), c'(d)\}$  é o custo da operação  $o \in O$ , e  $x_o$  é uma variável que indica se a operação  $o$  foi escolhida ou não. Para  $v \in V$ , temos que  $O^-(v) \subset O$  é o conjunto de operações que terminam no nó  $v$ ,  $O^+(v) \subset O$  é o conjunto de operações que começam no nó  $v$ , e  $O(v) \subset O$  é o conjunto de operações que contêm o nó  $v$ . Para  $S \subset V$ , temos que  $O^-(S)$  é o conjunto de operações que começam em  $S$  e terminam em  $V \setminus S$  e  $O^+(S)$  é o conjunto de operações que terminam em  $S$  e começam em  $V \setminus S$ . Por fim,  $y_v$  é uma variável auxiliar que indica se  $v$  é o começo de uma operação.

Usando essas definições e variáveis, para definir formalmente o FSTSP, podemos modelá-lo com o seguinte Programa Linear Inteiro, baseado na formulação de Agatz et al. [2018] para o TSP-D.

$$\min \sum_{o \in O} c_o x_o \quad (1)$$

$$\text{s.a.} \quad \sum_{o \in O(v)} x_o \geq 1 \quad \forall v \in V, \quad (2)$$

$$\sum_{o \in O^+(v)} x_o \leq y_v \quad \forall v \in V, \quad (3)$$

$$\sum_{o \in O^+(v)} x_o = \sum_{o \in O^-(v)} x_o \quad \forall v \in V, \quad (4)$$

$$\sum_{o \in O^+(S)} x_o \geq y_v \quad \forall S \subset V \setminus \{v_0\}, v \in S, \quad (5)$$

$$\sum_{o \in O^+(v_0)} x_o = 1 \quad (6)$$

$$y_{v_0} = 1, \quad (7)$$

$$x_o \in \{0, 1\} \quad \forall o \in O, \quad (8)$$

$$y_v \in \{0, 1\} \quad \forall v \in V. \quad (9)$$

A Função (1) minimiza a soma dos custos das operações escolhidas. As Restrições (2) garantem que todos os nós serão cobertos por ao menos uma operação. As Restrições (3) evitam que exista

mais de uma operação partindo de um nó, impossibilitando que um nó seja visitado em duas ou mais operações, e ao mesmo tempo garantem que  $y_v$  será igual a 1 para todo  $v$  que é extremo de uma operação. As Restrições (4) asseguram que a quantidade de operações que começam em um nó é a mesma quantidade de operações que terminam no mesmo nó. As Restrições (5) evitam que surjam sub-ciclos isolados do depósito. A Restrição (6) assegura que exista uma operação que comece no depósito. Por fim, a Restrição (7) garante que a operação inicial parte do depósito.

### 2.1. O Problema h-FSTSP

Vários métodos da literatura para resolver o FSTSP utilizam uma abordagem em dois estágios: primeiro encontram um ciclo Hamiltoniano, e então utilizam um algoritmo que insere entregas por drone neste ciclo. Para formalizar esta abordagem, Hokama et al. [2024] definiram o problema h-FSTSP, que recebe a entrada do FSTSP junto de um ciclo Hamiltoniano  $h$ , e obtém a melhor solução para o FSTSP que respeita a ordem de  $h$ , *i.e.*, que só utiliza uma entrega via drone  $d = (l, v, r)$  se  $v$  está entre  $l$  e  $r$  em  $h$ . O h-FSTSP não é NP-difícil, e tanto Agatz et al. [2018] quanto Ha et al. [2018] apresentaram algoritmos  $O(n^3)$  para o mesmo. O primeiro algoritmo é uma programação dinâmica e o segundo é um algoritmo em grafos chamado *Split*, sendo que ambos consideram todas as  $\binom{n}{3}$  triplas possíveis como entregas via drone.

Estudando o h-FSTSP, Hokama et al. [2024] demonstraram a Propriedade do Drone Preguiçoso, que apresenta critérios para ignorar certas triplas sem impactar a solução ótima do h-FSTSP. Usando esta propriedade obtiveram o algoritmo *SplitLazy*, que possui eficiência empírica linear, ainda que seu pior caso seja  $O(n^3)$ . Neste trabalho utilizamos o *SplitLazy* para encontrar boas soluções para o FSTSP a partir de ciclos Hamiltonianos. Isto simplifica a estrutura dos cromossomos em nossas heurísticas evolutivas, pois permite que estes armazenem apenas a ordem dos nós em um ciclo Hamiltoniano.

## 3. HGSADC

Nesta seção apresentamos nossa implementação do HGSADC para resolver o FSTSP. Esta metaheurística híbrida, proposta por Vidal et al. [2012], combina conceitos de seleção natural, com busca local e controle de diversidade da população, resultando em uma estratégia robusta para encontrar soluções de alta qualidade. A base do HGSADC reside na inspiração biológica dos algoritmos genéticos. Assim como na natureza, onde os organismos mais adaptados tendem a prosperar, neste algoritmo soluções promissoras são identificadas, reproduzidas e modificadas ao longo do tempo. Além disso, o HGSADC incorpora uma busca local para tentar encontrar melhores soluções mais rapidamente, bem como um controle de diversidade para evitar uma convergência prematura e uma estagnação em mínimos locais.

### 3.1. Representação da Solução

Representamos uma solução do FSTSP com um cromossomo, que guarda um ciclo Hamiltoniano necessariamente com o depósito como primeiro elemento. Dessa forma, sabemos que um cromossomo é uma permutação dos valores entre 0 e  $n$ , começando em 0, e no qual o  $i$ -ésimo alelo desse cromossomo representa o  $i$ -ésimo nó que será visitado. Uma solução do FSTSP pode ser extraída deste ciclo pelo uso do algoritmo *SplitLazy* (Seção 2.1). Essa representação permitirá realizar operações de crossover e busca local de forma simples e rápida.

### 3.2. Avaliação do Indivíduo

A avaliação de um indivíduo  $P$  depende do custo  $\phi(P)$  juntamente com um fator de contribuição à diversidade da população  $\Delta(P)$ . O cálculo da contribuição da diversidade pode se dar através da distância genômica média do indivíduo  $P$  aos seus vizinhos mais próximos.

A avaliação do indivíduo será chamada de **biased fitness** ( $BF(P)$ ), e é dada pela fórmula:

$$BF(P) = \text{fit}(P) + \left(1 - \frac{nbElite}{nbIndiv}\right) \text{dc}(P) , \quad (10)$$

onde  $\text{fit}(P)$  é a posição do indivíduo  $P$  na população com  $nbIndiv$  indivíduos ordenados com relação ao custo  $\phi(P)$  da solução que ele representa, e  $dc(P)$  é a posição do indivíduo  $P$  na mesma população ordenada com relação à contribuição à diversidade  $\Delta(P)$ . O parâmetro  $nbElite$  indica a quantidade de indivíduos que será passada para a próxima geração durante o processo de seleção de indivíduos.

A contribuição à diversidade  $\Delta(P)$  de um indivíduo  $P$  é calculada da seguinte forma:

$$\Delta(P) = \frac{1}{n_{close}} \sum_{P' \in N_{close}} \delta(P, P') , \quad (11)$$

sendo que  $N_{close}$  é o conjunto dos  $n_{close}$  indivíduos mais próximos de  $P$  considerando uma distância genômica  $\delta(P, P')$ . Esta é uma distância Hamming normalizada na qual um par de indivíduos é comparado alelo a alelo, e então a quantidade de alelos diferentes entre os indivíduos é dividida pela quantidade total de alelos de um cromossomo [Ha et al., 2020]. Mais precisamente,

$$\delta(P_1, P_2) = \frac{1}{n} \sum_{i=1, \dots, n} [\mathbf{1}(P_1(i) \neq P_2(i))] , \quad (12)$$

onde  $n$  é o número de alelos de um cromossomo, e a função de avaliação  $\mathbf{1}(P_1(i) \neq P_2(i))$  retorna 1 caso os alelos na  $i$ -ésima posição sejam diferentes, e 0 caso contrário.

O custo  $\phi(P)$  de um indivíduo  $P$  é obtido interpretando o cromossomo como descrito na Seção 3.1, *i.e.*, partimos do ciclo Hamiltoniano  $h$  codificado no cromossomo e, usando o algoritmo *SplitLazy*, inserimos entregas via drone de forma a minimizar o tempo total da solução sem desprezar a ordem dos nós em  $h$ .

### 3.3. Algoritmo

Sabendo representar e avaliar um indivíduo, podemos adentrar o funcionamento do HG-SADC, detalhado no Algoritmo 1. Este algoritmo opera em ciclos, cada um gerando um novo indivíduo a partir de um mecanismo de crossover, explicado mais adiante, e buscando aprimorar a qualidade das soluções geradas através de uma busca local, que também será descrita à frente.

---

#### Algoritmo 1: HGSADC

---

```

1  Inicializa População;
2  enquanto  $nIter < It_{NI}$  E  $tempo < T_{MAX}$  faça
3     $P_1 \leftarrow \text{SelecionaPai}()$ ;
4     $P_2 \leftarrow \text{SelecionaPai}()$ ;
5     $filho \leftarrow \text{Crossover}(P_1, P_2)$ ;
6     $filho \leftarrow \text{BuscaLocal}(filho)$ ;
7     $\text{AdicionaIndividuo}(filho)$ ;
8    se  $nIterSemMelhora = It_{IMPROV}$  então
9       $\text{Diversifica}()$ ;
10      $nIterSemMelhora \leftarrow 0$ ;
11  devolva melhor indivíduo;
```

---

Ao longo da execução do algoritmo, sempre que a população atinge  $\mu + \lambda$  indivíduos, ocorre uma seleção com base no *biased fitness*, que reduz o tamanho da população para  $\mu$ , sendo  $\mu$  e  $\lambda$  parâmetros do algoritmo. Essa seleção elimina clones antes de soluções ruins.

O algoritmo inicializa a população gerando  $4\mu$  indivíduos aleatórios (Linha 1). Em seguida, entra no laço principal, que dura até atingir  $It_{NI}$  iterações ou tempo de execução  $T_{MAX}$ ,

com  $It_{NI}$  e  $T_{MAX}$  sendo parâmetros do algoritmos. Em cada iteração são selecionados dois indivíduos pais (Linhas 3 e 4). A seleção de cada pai é através de um torneio binário. Neste processo, dois indivíduos da população são selecionados com probabilidade uniforme, e aquele com maior *biased fitness* é escolhido. Com os pais escolhidos, é gerado um filho a partir de um operador de crossover (Linha 5). Após isso, o indivíduo filho é melhorado utilizando uma busca local (Linha 6) e é adicionado à população (Linha 7). Se o algoritmo passar muitas iterações sem atualizar a melhor solução, é acionado um mecanismo de diversificação para aumentar a variabilidade genética da população (Linha 9). Esse mecanismo elimina 2/3 da população, sendo removidos primeiro clones e depois indivíduos que têm os piores *biased fitness*. Então, novos indivíduos são gerados da mesma forma que ocorre na Linha 1.

No algoritmo apresentado por Vidal et al. [2012], existe um mecanismo de gerenciamento dos indivíduos em duas subpopulações considerando a viabilidade das mesmas. Entretanto, na versão do FSTSP abordada neste projeto não foram encontradas condições de inviabilidade, ou seja, qualquer permutação dos nós que precisam ser visitados corresponde a uma solução viável e não existe necessidade de trabalhar com uma subpopulação de inviáveis.

### 3.4. Crossover

Uma vez que os pais  $P_1$  e  $P_2$  foram selecionados, realiza-se o cruzamento, descrito no Algoritmo 2, para gerar um indivíduo filho.

---

#### Algoritmo 2: Crossover

---

**Entrada:** Pais  $P_1$  e  $P_2$   
**Saída:** Indivíduo *filho*  
 1 Sorteia  $a$  e  $b$  entre  $[1, n]$  de modo que  $a \leq b$ ;  
 2 Sorteia  $c$  entre  $[1, n - (b - a)]$ ;  
 3  $filho(0) \leftarrow 0$ ;  
 4  $i \leftarrow c$ ;  
 5 **para**  $j = a$  até  $b$  **faça**  
 6      $filho(i) \leftarrow P_1(j)$ ;  
 7      $i \leftarrow i + 1$ ;  
 8  $i \leftarrow 1$ ;  
 9 **para**  $j = 1$  até  $n$  **faça**  
 10    **se**  $P_2(j)$  ainda não existe em *filho* **então**  
 11     **se**  $i = c$  **então**  
 12        $i \leftarrow i + (b - a + 1)$ ;  
 13        $filho(i) \leftarrow P_2(j)$ ;  
 14        $i \leftarrow i + 1$ ;  
 15 **devolva** *filho*;

---

O algoritmo seleciona um intervalo  $[a, b]$  aleatoriamente (Linha 1) contido inteiramente entre  $[1, n]$  e, a partir da posição aleatória  $c$  entre  $[1, n - (b - a)]$  (Linha 2), copia o intervalo selecionado de  $P_1$  para o *filho* (Linhas 4 até 7). Então, para o cromossomo do filho ser uma solução válida, basta copiar os alelos faltantes na ordem que aparecem em  $P_2$  (Linhas 8 até 14).

### 3.5. Busca Local

O algoritmo de busca local implementado é inspirado pelo método 2-opt clássico do TSP, que busca melhorar a solução obtida ao remover cruzamentos entre pares de arestas [Potvin, 1996]. É central para o desempenho da 2-opt que ela consiga avaliar o impacto da troca de arestas em tempo constante, ao invés de ter que recalculer o custo de todo o ciclo, o que levaria tempo  $O(n)$ . Entretanto, para o FSTSP, avaliar o impacto da troca de um par de arestas pode ser mais complicado

caso as arestas envolvidas correspondam a trechos em que o drone e o caminhão não estão viajando juntos. Uma possibilidade seria, a cada troca de arestas recalculando a solução do FSTSP. Porém, mesmo usando o *SplitLazy* [Hokama et al., 2024] isso custaria, no melhor caso,  $O(n)$ .

Para reduzir o custo computacional do algoritmo, optamos por só avaliar a troca de arestas com o *SplitLazy*, se a solução para o TSP representada pelo indivíduo for melhorada pela mesma troca de arestas. Em outras palavras, é assumido que uma troca de arestas que piore o TSP provavelmente também piorará o FSTSP. Ainda em prol de reduzir o esforço computacional, a busca local considera a troca de arestas que estão a no máximo  $D_{MAX}$  de distância entre si, sendo este um parâmetro do algoritmo. O pseudocódigo do Algoritmo 3 descreve em alto nível a implementação.

---

**Algoritmo 3:** BuscaLocal

---

```

Entrada: Indivíduo indiv
Saída: Indivíduo indiv melhorado pela busca local
1 enquanto melhorar faça
2    $a \leftarrow$  valor aleatório entre  $[0, n - D_{MAX}]$ ;
3    $b \leftarrow a + D_{MAX}$ ;
4   para  $i$  de  $a$  até  $b - 1$  faça
5     para  $j$  de  $i + 1$  até  $b$  faça
6       Calcula custoEconomizadoTSP pela troca de  $(i, i + 1)$  e  $(j, j + 1 \bmod (n + 1))$ 
       por  $(i, j)$  e  $(i + 1, j + 1 \bmod (n + 1))$  em indiv;
7       se custoEconomizadoTSP  $> 0$  então
8         Realiza a troca das arestas em indiv;
9         se custo do SplitLazy sobre indiv melhorar então
10          Recomeça o laço principal com indiv alterado;
11        senão
12          Desfaz a troca de arestas da Linha 10 em indiv;
13 devolva indiv;
  
```

---

O loop da Linha 1 continua enquanto o algoritmo encontrar alguma troca de arestas que reduza o custo da solução do FSTSP. As Linhas 2 e 3 delimitam o intervalo de tamanho  $D_{MAX}$  em que as trocas de arestas serão buscadas. Os loops das Linhas 4 e 5 navegam pelas possíveis trocas, avaliando a melhoria no custo do TSP (Linhas 6 e 7). Se essa avaliação sugerir uma melhora então é realizada a troca no cromossomo (Linha 8) e é recalculada a solução do FSTSP, mantendo-a apenas se melhorar (Linhas 9 e 10). Caso contrário, a troca é desfeita (Linha 12) e a busca segue para o próximo par de arestas.

#### 4. MP-BRKGa

O BRKGa [Gonçalves e Resende, 2013] é uma metaheurística para criação de algoritmos genéticos, em que uma população de indivíduos evolui ao longo de várias gerações. Cada indivíduo é codificado em um cromossomo, que corresponde a um vetor de valores em  $[0, 1)$ , sendo que cada posição é um alelo. O ciclo principal do BRKGa passa pela classificação de cada cromossomo da população atual entre elite e não elite. Esta classificação é feita por um decodificador que devolve um valor de *fitness* do cromossomo. A população da próxima geração é composta por: (i) elites da geração anterior, (ii) mutantes gerados aleatoriamente, (iii) cromossomos fruto de cruzamentos entre pai elite e pai não-elite, escolhidos aleatoriamente. O cruzamento é feito de modo linear, com cada alelo sendo herdado de um pai via sorteio, com viés favorável aos alelos do pai elite. Vale destacar que o papel dos mutantes é manter a variabilidade genética.

Neste trabalho adotamos o MP-BRKGa, versão proposta por Andrade et al. [2021], em que o processo de cruzamento utiliza vários pais, em vez de apenas dois. Além disso, após todo o fim

do processo evolutivo, é selecionada uma quantidade de elites para passar pelo processo de *Implicit Path Relinking* (IPR), que busca conectar um par de soluções utilizando apenas seus cromossomos. Entretanto, desligamos o IPR nos testes computacionais reportados neste trabalho.

A adaptação do BRKGA para um problema específico depende do decodificador, que traduz um cromossomo para uma solução do problema. Nosso decodificador para o FSTSP tem duas etapas. Primeiro ele transforma o cromossomo em uma sequência dos nós (começada no depósito). Para tanto, ordena as posições de 1 a  $n$  do cromossomo de acordo com o valor de cada alelo, e verifica as posições em que os índices originais (correspondentes aos nós) terminam. Em seguida, interpreta esta sequência como um ciclo Hamiltoniano  $h$ , que é passado para o algoritmo *SplitLazy*. Este algoritmo, como explicado na Seção 2.1, insere entregas via drone que respeitam a ordem de  $h$ , resolvendo de modo ótimo o h-FSTSP.

#### 4.1. Warm Start

Introduzir soluções não triviais como parte da população inicial do BRKGA pode acelerar o processo de evolução dos indivíduos. Para realizar este procedimento, chamado *warm start*, primeiro usamos a Heurística Lin Kernighan (LKH) e o Concorde TSP solver para gerar dois ciclos para o TSP, um ótimo e outro muito próximo do ótimo. De fato, o ciclo produzido pela LKH também foi usado como solução primal (limitante superior) pelo resolvedor exato. Note que, a solução ótima para uma instância do FSTSP não necessariamente corresponde à solução ótima do h-FSTSP que recebe como entrada o ciclo ótimo do TSP. Por exemplo, para várias instâncias a solução do *SplitLazy* para o ciclo da LKH foi melhor que aquela sobre o ciclo ótimo do TSP.

Eliminar simetrias na representação de soluções é importante, pois evita gastar esforço computacional buscando em espaços de soluções repetidas. Uma simetria em circuitos Hamiltonianos ocorre em função do nó origem. Evitamos esta fixando a origem no depósito. Outra simetria, que ocorre quando os custos são simétricos, envolve o sentido em que o circuito é percorrido, como mostra a Figura 1. Isto é particularmente problemático pois o cruzamento entre soluções parecidas (ou mesmo iguais) pode gerar descendentes descaracterizados. Para evitar que isto acontecesse, tanto ciclo da LKH quando o inverso do mesmo foram comparados, considerando a distância em número de inversões, com o ciclo ótimo, sendo mantido aquele com menor distância. Procurando

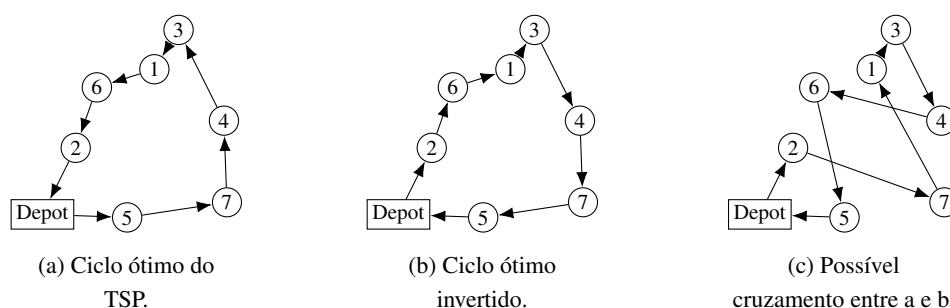


Figura 1: Consequências da simetria no cruzamento de circuitos Hamiltonianos.

encontrar melhores ciclos para serem usados no *warm start*, ou mesmo de modo independente, projetamos uma família de heurísticas para o FSTSP.

#### 4.2. Bolhas Elípticas

As heurísticas das Bolhas Elípticas foram inspiradas pelo algoritmo *SplitLazy*. Dado um ciclo  $h$  e uma tripla  $(i, j, k)$  candidata a entrega via drone, um algoritmo para o h-FSTSP só considera esta tripla caso o nó  $j$  esteja depois de  $i$  e antes de  $k$  em  $h$ . Assim, a heurística das Bolhas Elípticas busca modificar um ciclo Hamiltoniano, colocando nós interessantes para serem atendidos

pelo drone, entre nós originalmente consecutivos deste ciclo. Dados dois nós  $i$  e  $k$  consecutivos em  $h$ , considere uma região elíptica usando eles como focos, com o tamanho da região proporcional à distância focal e ao  $\alpha$  (fator velocidade do drone). Assim, consideramos interessante um nó  $j$  dentro desta elipse, *i.e.*, se  $c(i, j) + c(k, j) \leq \alpha \cdot c(i, k)$ . A motivação vem da Propriedade do Drone Preguiçoso, segundo a qual qualquer nó dentro dessa região que for atendido pelo drone saindo de  $i$  e retornando em  $k$  não fará o caminhão esperar, *i.e.*, o drone será rápido.

Como seria muito custoso avaliar todos os nós que estão em uma certa área elíptica gerada a partir de 2 outros nós, foi decidido fazer uma escolha gulosa como pode ser visto no Algoritmo 4. Nele para cada par de nós  $(i - 1, i)$ , serão testados nós de  $i + 1$  até  $n$ , sendo que o primeiro que satisfizer a condição será o escolhido. Foi feita também uma variação deste algoritmo, nomeado Bolhas Elípticas Reversas, que começa a testar nós de  $n$  e vai até  $i + 1$ , priorizando assim nós mais próximos ao fim do ciclo.

#### Algoritmo 4: Bolhas Elípticas Diretas

**Entrada:** Ciclo Hamiltoniano  $h$   
**Saída:** Ciclo  $h$  modificado

```

1  $i \leftarrow 1$ ;
2 enquanto  $i < n$  faça
3   para  $j = i + 1$  até  $n$  faça
4     se  $c(v_{i-1}, v_j) + c(v_j, v_i) \leq \alpha_{relax} \cdot c(v_{i-1}, v_i)$  então
5       mova  $v_j$  para logo antes de  $v_i$  em  $h$ ;
6        $i \leftarrow i + 1$ ;
7       break;
8    $i \leftarrow i + 1$ ;
```

Outro fator a considerar no uso das Bolhas Elípticas é tratar tanto o ciclo  $h$  em sua ordem original, quanto em ordem inversa. Um exemplo dessa importância pode ser visto na Figura 2. Note que, para que a solução do h-FSTSP seja a solução ótima do FSTSP, é necessário um ciclo  $h$  em que o nó 6 esteja entre o depósito e o nó 4. No entanto, como o ciclo ótimo do TSP está indo primeiro para o nó 6, esta possibilidade nunca seria considerada pelas Bolhas Elípticas. Mas se usarmos o ciclo em ordem inversa, no algoritmo das Bolhas Elípticas Reversas a primeira tripla analisada será justamente  $v_{i-1} = \text{Depot}$ ,  $v_i = 4$  e  $v_j = 6$ , para  $i = 1$  e  $j = n$ .

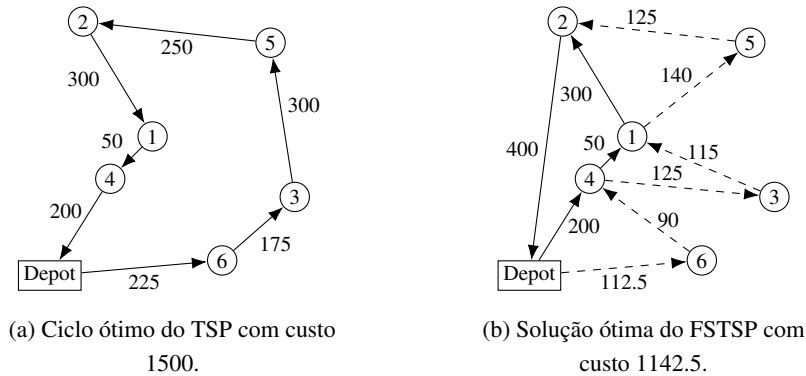


Figura 2: Comparação entre soluções ótimas do TSP e do FSTSP. Arcos sólidos são o trajeto do caminhão.

Arcos tracejados são o caminho do drone, com  $\alpha = 2$ . Exemplo baseado em Ha et al. [2018].

Por fim, note que uma entrega com drone que faça o caminhão esperar ainda pode levar a

uma boa (ou mesmo ótima) solução para o FSTSP. Assim, calculamos a elipse usando  $\alpha_{relax}$ , que é o produto do  $\alpha$  por um fator de relaxamento. Também percebemos que, caso  $\alpha_{relax} = 1$ , a elipse se degenera em um segmento de reta. Por isso não rodamos o algoritmo para  $\alpha_{relax} \leq 1$ .

## 5. Resultados Computacionais

A implementação do HGSADC foi feita em C++ e compilado com a flag “-O3”. Os experimentos do HGSADC foram realizados em um computador com processador Intel Core i5-12450H, 4.40GHz, com 8GBs de RAM, em um Windows 11, utilizando os atributos  $\mu = 25$ ,  $\lambda = 40$ ,  $nbElite = 4$ ,  $nClose = 5$ ,  $nIterSemMelhora = 500$ , e  $T_{MAX} = 240$ . Já para  $It_{NI}$  foi atribuído um valor suficientemente elevado, a fim de garantir que o algoritmo pare sua execução apenas em função do tempo máximo de 240 segundos.

Os experimentos do MP-BRKGA foram feitos em um computador com processador Intel i7-7500U 3.5GHz com 2 núcleos e 4 threads e com 16 GBs de RAM, em um Ubuntu 24.04 LTS, utilizando a API BRKGA-MP-IPR versão 3.0 [Andrade, 2023], com o Implicit Path-Relinking desligado e com apenas uma thread. O decodificador e a heurística das Bolhas Elípticas foram implementados na linguagem C++23 e compilados pelo g++ na versão 12.3.0, com flag “-O3”. Foi também utilizado o API discorde-tsp de Maravilha [2018] para calcular as soluções ótimas do TSP. O discorde-tsp usa o CPLEX-Studio, e a versão utilizada foi a 22.1.1.

Os algoritmos foram testados com 270 instâncias selecionadas do *benchmark* de Bouman et al. [2018]. Elas podem ser divididas pelo número de vértices (50, 75 e 100), distribuição dos clientes (*uniform*, *singlecenter* e *doublecenter*), e pelo valor do  $\alpha$  (1, 2 e 3). Vale ressaltar que cada instância foi testada com 5 *seeds* diferentes, sendo reportados os valores médios.

| Tipo         | $\alpha$ | V   | OPT <sub>TSP</sub> | Agatz    | Freitas       | HGSADC          | MP-BRKGA | HBE      | MP-BRKGA <sub>ws</sub> |
|--------------|----------|-----|--------------------|----------|---------------|-----------------|----------|----------|------------------------|
| Uniform      | 1        | 50  | 596.42             | 504.58   | 551.49        | <b>488.06</b>   | 525.63   | 502.20   | 491.90                 |
|              |          | 75  | 696.29             | 585.56   | 624.26        | 578.29          | 654.84   | 582.12   | <b>568.53</b>          |
|              |          | 100 | 783.83             | 665.85   | 696.15        | 673.96          | 792.44   | 665.10   | <b>648.58</b>          |
|              | 2        | 50  | 596.42             | 461.39   | 420.80        | <b>402.06</b>   | 429.25   | 453.63   | 403.90                 |
|              |          | 75  | 696.29             | 528.95   | <b>459.71</b> | 465.63          | 533.59   | 522.49   | 473.88                 |
|              |          | 100 | 783.83             | 608.30   | 553.43        | 556.01          | 666.78   | 604.09   | <b>547.63</b>          |
|              | 3        | 50  | 596.42             | 460.05   | 377.28        | 369.38          | 389.93   | 444.64   | <b>368.70</b>          |
|              |          | 75  | 696.29             | 528.13   | 443.30        | <b>426.01</b>   | 499.69   | 516.26   | 435.55                 |
|              |          | 100 | 783.83             | 607.44   | 510.20        | 520.71          | 609.78   | 599.45   | <b>507.18</b>          |
| Singlecenter | 1        | 50  | 838.13             | 662.81   | 756.20        | 657.74          | 684.70   | 655.41   | <b>633.63</b>          |
|              |          | 75  | 1,120.18           | 894.34   | 978.32        | 889.73          | 965.63   | 886.68   | <b>863.52</b>          |
|              |          | 100 | 1,336.68           | 1,069.24 | 1,193.95      | 1,062.26        | 1,226.73 | 1,066.64 | <b>1,042.85</b>        |
|              | 2        | 50  | 838.13             | 582.41   | 553.53        | 508.77          | 533.33   | 565.16   | <b>495.75</b>          |
|              |          | 75  | 1,120.18           | 803.49   | 730.43        | 697.10          | 769.30   | 774.18   | <b>681.19</b>          |
|              |          | 100 | 1,336.68           | 947.76   | 908.29        | <b>829.97</b>   | 985.68   | 932.02   | 838.20                 |
|              | 3        | 50  | 838.13             | 579.06   | 478.32        | 443.80          | 459.36   | 545.42   | <b>429.80</b>          |
|              |          | 75  | 1,120.18           | 799.65   | 638.48        | 599.88          | 704.53   | 761.56   | <b>598.83</b>          |
|              |          | 100 | 1,336.68           | 941.89   | 798.54        | <b>727.02</b>   | 882.33   | 912.91   | 739.25                 |
| Doublecenter | 1        | 50  | 1,277.40           | 1,018.87 | 1,153.29      | <b>984.70</b>   | 1,054.80 | 1,015.75 | 990.38                 |
|              |          | 75  | 1,586.77           | 1,255.46 | 1,432.36      | <b>1,169.80</b> | 1,447.02 | 1,254.15 | 1,215.32               |
|              |          | 100 | 1,768.23           | 1,411.28 | 1,516.36      | 1,413.89        | 1,784.19 | 1,409.16 | <b>1,370.52</b>        |
|              | 2        | 50  | 1,277.40           | 901.57   | 811.08        | <b>790.55</b>   | 827.78   | 882.95   | 792.30                 |
|              |          | 75  | 1,586.77           | 1,126.87 | 1,109.53      | <b>948.03</b>   | 1,129.02 | 1,108.54 | 982.72                 |
|              |          | 100 | 1,768.23           | 1,273.08 | 1,184.76      | 1,133.79        | 1,416.02 | 1,262.49 | <b>1,113.95</b>        |
|              | 3        | 50  | 1,277.40           | 894.94   | 715.23        | <b>701.51</b>   | 730.29   | 856.53   | 720.32                 |
|              |          | 75  | 1,586.77           | 1,117.90 | 931.13        | <b>838.19</b>   | 971.53   | 1,081.79 | 882.86                 |
|              |          | 100 | 1,768.23           | 1,261.02 | 1,051.00      | 1,013.71        | 1,223.43 | 1,231.30 | <b>1,008.02</b>        |

Tabela 1: Comparação dos resultados com a literatura.

A Tabela 1 compara os resultados obtidos pelos nossos métodos com: [OPT<sub>TSP</sub>] ótimo do TSP; [Agatz] *exact partitioning* de Agatz et al. [2018] aplicado sobre o OPT<sub>TSP</sub>; e [Freitas] *Hybrid Tabu General Variable Neighborhood Search* de Freitas et al. [2023] aplicada sobre o OPT<sub>TSP</sub>. Até

onde apuramos, os resultados de Freitas et al. [2023] são os melhores da literatura para o FSTSP. Nossos métodos são: [HGSADC], [MP-BRKGA], [HBE] heurísticas das Bolhas Elípticas aplicadas sobre  $OPT_{TSP}$  e LKH, e [MP-BRKGA<sub>WS</sub>] MP-BRKGA recebendo as soluções produzidas por HBE como *Warm Start*. Destacamos que, embora alguns métodos da literatura considerem características de versões mais gerais do FSTSP, como restrições de carga e de duração da bateria do drone, para as instâncias testadas todos os métodos consideram as mesmas restrições.

Podemos observar que nossos métodos encontraram os melhores resultados médios para 96% das categorias de instâncias, sendo que cada categoria tem 10 instâncias. Mais precisamente, o HGSADC obteve os melhores resultados para 11 categorias, e o MP-BRKGA<sub>WS</sub> chegou nos melhores resultados para 15 categorias, enquanto Freitas manteve os melhores resultados para uma categoria. Notamos que o MP-BRKGA<sub>WS</sub> foi ligeiramente melhor nas instâncias *Uniform* e *Single-center*, enquanto o HGSADC foi levemente melhor para as *Doublecenter*. A divisão das melhores soluções obtidas entre nossas duas estratégias sugere uma complementariedade entre as mesmas, que pode ser mais explorada em trabalhos futuros.

Como algoritmos genéticos apresentam melhoria com o passar do tempo, foram registrados os resultados com 15, 30, 60, 120 e 240 segundos de execução. A Figura 3 ilustra o desenvolvimento da média de todas as melhores soluções de ambas as técnicas com o passar do tempo.

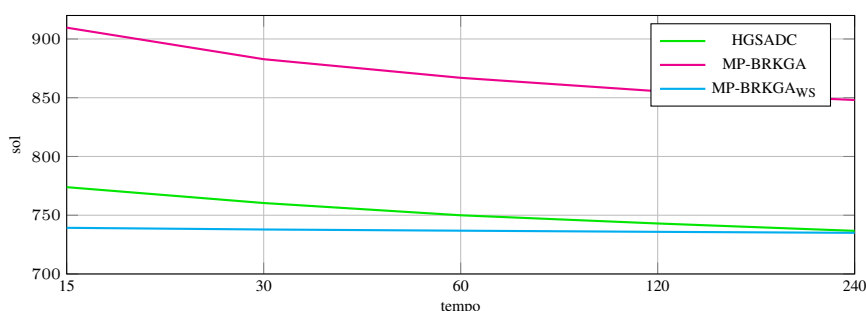


Figura 3: Evolução das soluções do HGSADC, do MP-BRKGA e do MP-BRKGA<sub>WS</sub> no decorrer do tempo.

Nesse cenário, fica nítida a importância de boas soluções no início do BRKGA, melhorando consideravelmente a qualidade das soluções desde as primeiras iterações do algoritmo. Além disso, com o decorrer do tempo, as soluções do HGSADC se aproximam e, em alguns casos, até melhoram as soluções do BRKGA com *warm start*. Conjecturamos que isso se deve à combinação da busca local com um *crossover* que preserva trechos dos ciclos. O mesmo não acontece com o MP-BRKGA, que possui um *crossover* que favorece fragmentação de trechos.

## 6. Conclusão

Neste trabalho apresentamos as metaheurísticas HGSADC, amparado por um algoritmo de crossover tal como uma busca local baseada na 2-opt do TSP, e BRKGA, juntamente com a família de heurísticas de Bolhas Elípticas, para a solução do FSTSP. Ambos os algoritmos genéticos utilizam o *SplitLazy* Hokama et al., 2024 para calcular o custo das soluções obtidas.

Os testes empíricos mostram que os resultados obtidos melhoram valores atuais da literatura. Entretanto, mostra-se fundamental realizar testes de forma mais intensiva e abrangente. Em particular, realizar testes com as demais instâncias de Bouman et al. [2018], além de instâncias usadas em outros trabalhos, como Freitas et al. [2023] e Ha et al. [2020]. Como os algoritmos se mostraram eficientes, também queremos realizar testes com instancias maiores, bem como adaptá-los para versões do FSTSP com restrições adicionais, como carga e duração da bateria do drone.

Por fim, pretendemos aprofundar nossa pesquisa em duas frentes com relação às metaheurísticas. No HGSADC, inserir um *warm start* baseado em heurísticas primais e de busca local,

e adaptar as bolhas elípticas em um algoritmo de busca local. Já no MP-BRKGa, pretendemos utilizar o *Implicit Path Relinking*, além de outras funcionalidades presentes no *framework*, como várias populações, *shakes*, entre outras. Também desejamos generalizar as heurísticas das Bolhas Elípticas para considerar vértices não consecutivos.

## Referências

- Agatz, N., Bouman, P., e Schmidt, M. (2018). Optimization approaches for the traveling salesman problem with drone. *Transportation Science*, 52(4):965–981.
- Andrade, C. E., Toso, R. F., Gonçalves, J. F., e Resende, M. G. (2021). The multi-parent biased random-key genetic algorithm with implicit path-relinking and its real-world applications. *European Journal of Operational Research*, 289(1):17–30. ISSN 0377-2217.
- Andrade, C. E. d. (2023). Ceandrade/BRKGa\_MP\_IPR\_CPP: The multi-parent biased random-key genetic algorithm with implicit path relink - C++ version. URL [https://github.com/ceandrade/brkg\\_a\\_mp\\_ipr\\_cpp](https://github.com/ceandrade/brkg_a_mp_ipr_cpp).
- Bouman, P., Agatz, N., e Schmidt, M. (2018). Instances for the tsp with drone (and some solutions). *Zenodo: London, UK*, p. v1.
- Freitas, J. C., Penna, P. H. V., e Toffolo, T. A. (2023). Exact and heuristic approaches to truck–drone delivery problems. *EURO Journal on Transportation and Logistics*, 12:100094. ISSN 2192-4376.
- Gonçalves, J. F. e Resende, M. G. (2013). A biased random key genetic algorithm for 2d and 3d bin packing problems. *International Journal of Production Economics*, 145(2):500–510. ISSN 0925-5273.
- Ha, Q. M., Deville, Y., Pham, Q. D., e Hà, M. H. (2020). A hybrid genetic algorithm for the traveling salesman problem with drone. *Journal of Heuristics*, 26:219–247.
- Ha, Q. M., Deville, Y., Pham, Q. D., e Hà, M. H. (2018). On the min-cost traveling salesman problem with drone. *Transportation Research Part C: Emerging Technologies*, 86:597–621. ISSN 0968-090X.
- Hokama, P. H. D. B., Lintzmayer, C. N., e San Felice, M. C. (2024). Sobre o Problema do Caixeiro Viajante com Drone. In *Anais do 9º Encontro de Teoria da Computação (to appear)*.
- Macrina, G., Di Puglia Pugliese, L., Guerriero, F., e Laporte, G. (2020). Drone-aided routing: A literature review. *Transportation Research Part C: Emerging Technologies*, 120:102762. ISSN 0968-090X.
- Maravilha, A. L. (2018). discorde-tsp. URL <https://github.com/andremaravilha/discorde-tsp/tree/master>.
- Murray, C. e Chu, A. (2015). The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, 54.
- Potvin, J.-Y. (1996). Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:337–370.
- Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N., e Rei, W. (2012). A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60:611–624.