Dissertação apresentada à Pró-Reitoria de Pós-Graduação e Pesquisa do Instituto Tecnológico de Aeronáutica e da Universidade Federal de São Paulo, como parte dos requisitos para obtenção do título de Mestra em Ciências no Programa de Pós-Graduação em Pesquisa Operacional, Área de Engenharia de Produção/Pesquisa Operacional.

Bárbara Lessa Vianna

MÉTODOS EXATO E HEURÍSTICO PARA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE EM FAMÍLIAS

Dissertação aprovada em sua versão final pelos abaixo assinados:

Prof. Dr. Antônio Augusto Chaves

Orientador

Dr. Tiago Tiburcio da Silva

Coorientador

São José dos Campos, SP - Brasil
2022

Dados Internacionais de Catalogação-na-Publicação (CIP)
Divisão de Informação e Documentação

Lessa Vianna, Bárbara
Métodos exato e heurístico para resolução do Problema do Caixeiro Viajante em Famílias / Bárbara Lessa Vianna.
São José dos Campos, 2022.
100f.

Dissertação de Mestrado – Curso de Pesquisa Operacional. Área de Engenharia de Produção/Pesquisa Operacional – Instituto Tecnológico de Aeronáutica e Instituto de Ciência e Tecnologia da Universidade Federal de São Paulo, 2022. Orientador: Prof. Dr. Antônio Augusto Chaves. Coorientador: Dr. Tiago Tiburcio da Silva.

1. Problema do Caixeiro Viajante. 2. Algoritmo Genético. 3. Aprendizagem por Reforço. I. Instituto Tecnológico de Aeronáutica. II. Universidade Federal de São Paulo. III. Título.

REFERÊNCIA BIBLIOGRÁFICA

LESSA VIANNA, Bárbara. **Métodos exato e heurístico para resolução do Problema do Caixeiro Viajante em Famílias**. 2022. 100f. Dissertação de Mestrado – Instituto Tecnológico de Aeronáutica e Universidade Federal de São Paulo, São José dos Campos.

CESSÃO DE DIREITOS

Bárbara Lessa Vianna
R. Dr. Jamil Cury, Vila Industrial, 679
12.220-281 – São José dos Campos–SP

MÉTODOS EXATO E HEURÍSTICO PARA RESOLUÇÃO DO PROBLEMA DO CAIXEIRO VIAJANTE EM FAMÍLIAS

Bárbara Lessa Vianna

Composição e Assinatura da Banca Examinadora:

Prof. Dr.	Mariá Cristina Nascimento	Presidente	-	UNIFESP
Prof. Dr.	Antônio Augusto Chaves	Orientador	-	UNIFESP
Dr.	Tiago Tiburcio da Silva	Coorientador	-	UNIFESP
Prof. Dr.	Luiz Leduino Salles Neto	Membro interno	-	UNIFESP
Prof. Dr.	Marco Antonio M Carvalho	Membro externo	-	UFOP

ITA/UNIFESP

Este trabalho é dedicado à minha família.

Agradecimentos

À Deus por ter me provido saúde à mim e minha família, sabedoria e força para enfrentar dois anos de pandemia e me permitir, mesmo com todas as dificuldades, seguir no mestrado. Ao meu coorientador, Dr. Tiago Tiburcio da Silva, por todo o suporte dado durante o mestrado.

Ao Prof. Dr. Antônio Augusto Chaves, pela orientação, apoio e conhecimento transmitido.

Ao meu coorientador, Dr. Tiago Tiburcio da Silva, por todo o suporte dado durante o mestrado.

Ao Prof. Dr. Marcos Schenekemberg, pelas contribuições dadas.

À minha família por sempre acreditar em mim e pelo apoio incondicional.

À minha família, em especial a minha mãe (Elane), meu pai (Arnaldo) e minha irmã (Gabriela), por sempre acreditarem em mim e por fazerem o possível e o impossível para que eu pudesse seguir nesse caminho.

À minha filha (Helena) que é o meu combustível diário e por compreender a minha limitação de tempo para brincar, por muitas vezes, durante o mestrado.

À meu marido (Felipe) pelo apoio, compreensão, e por ter suprido a minha ausência com nossa filha em todas as vezes em que foi necessário nestes últimos anos.

À todos os familiares e amigos que de forma direta ou até mesmo indireta me ajudaram com palavras de incentivo, cuidado na Helena para mim e torcendo por essa conquista.

Aos membros da banca pela disposição e análise deste documento e pelas contribuições propostas.

Aos demais professores e funcionários da UNIFESP e do ITA.

Aos colegas de mestrado pelos trabalhos realizados e conhecimentos adquiridos.

À todos que torceram por essa conquista.

E finalmente, à FAPESP pelo suporte recebido.

"If I have seen farther than others,
it is because I stood on the shoulders of giants."
— Sir Isaac Newton

Resumo

No campo da Pesquisa Operacional, uma tendência é o desenvolvimento de métodos híbridos exatos e heurísticos para obter resultados de boa qualidade em problemas de otimização combinatória. Existem diversas maneiras de hibridização. Uma possibilidade de hibridizar métodos exatos é através da integração de um método exato com heurísticas de busca local. Uma versão híbrida de metaheurísticas pode ser obtida com a integração de técnicas de Aprendizado de Máquinas.

Um dos problemas mais clássicos de otimização combinatória é o Problema do Caixeiro Viajante (TSP). Nesse problema considera-se a minimização apenas dos custos operacionais envolvidos no percurso do vendedor. Contudo, o TSP pode ser adaptado para diferentes problemas que empresas logísticas enfrentam, como, por exemplo, diferentes categorias de produtos, prioridades de entregas, e localização de produtos em armazéns. Este trabalho aborda o Problema do Caixeiro Viajante em Família (FTSP, do inglês *Family Traveling Salesman Problem*), em que os clientes são agrupados em famílias que correspondem a produtos de mesma similaridade e com a demanda de visitas predefinidas. O objetivo do FTSP é determinar a rota de custo mínimo visitando apenas um subconjunto de clientes de cada família. Assim como o TSP, trata-se de um problema de otimização combinatória pertencente a classe NP-Difícil.

Para solucionar o problema proposto foram desenvolvidos dois métodos: (*i*) um *branch-and-cut* paralelo com um procedimento de busca local eficiente para obter a solução ótima, e (*ii*) uma metaheurística adaptativa que combina o método *Biased Random-key Genetic Algorithm* (BRKGA) com um algoritmo de aprendizado por reforço, *Q-Learning* (QL). Neste caso, o algoritmo *Q-Learning* é utilizado para controlar os parâmetros do BRKGA durante o processo evolutivo.

Experimentos computacionais foram realizados em um conjunto de dados de referência bem conhecido, que possui 185 instâncias. O algoritmo P-B&C desenvolvido para o FTSP prova o valor ótimo para 179 instâncias e o BRKGA-QL encontrou os melhores limites superiores para as outras quatro instâncias. Os resultados foram comparados com os melhores resultados da literatura, e ambos os métodos mostram robustez e eficiência para resolver o FTSP.

Abstract

In the field of Operations Research, a trend is the development of hybrid exact and heuristic methods to obtain good quality results in combinatorial optimization problems. There are several ways of hybridization. One possibility to hybridize exact methods is through the integration of an exact method with local search heuristics. A hybrid version of metaheuristics can be obtained by integrating Machine Learning techniques.

One of the most classic combinatorial optimization problems is the Traveling Salesman Problem (TSP). In this problem, only the minimization of operating costs involved in the salesperson's route is considered. However, TSP can be adapted to different problems that logistics companies face, such as different product categories, delivery priorities, location of products in warehouses.

This paper addresses the Family Traveling Salesman Problem (FTSP), in which nodes are grouped into families that correspond to products of the same similarity and the goal is to determine the minimum cost route by visiting only a subset of nodes from each family. Like the TSP, it is a combinatorial optimization problem belonging to the NP-Hard class.

To solve the proposed problem, two methods were developed: (i) a parallel branch-and-cut algorithm with an efficient local search procedure to obtain the optimal solution, and (ii) an adaptive metaheuristic that combines the Biased Random-key Genetic Algorithm (BRKGA) with a reinforcement learning algorithm, *Q-Learning* (QL). In this case, the Q-Learning algorithm is used to control the parameters of the BRKGA during the evolutionary process.

Computational experiments were performed on a well-known benchmark data set, which has 185 instances. The branch-and-cut developed for FTSP proves the optimal value for 179 instances and the BRKGA-QL found the best upper bounds for the other four instances. The results were compared with the best results of the literature, and both methods show robustness and efficiency to solve the FTSP.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

B&B	*Branch-and-Bound*
B&C	*Branch-and-Cut*
BNT	*Bayesian Network Tuning*
BRKGA	*Biased Random-key Genetic Algorithm*
CTPP	*Capacitated Traveling Purchaser Problem*
DGA	*Diploid Genetic Algorithm*
evPR	*Evolucionary Path-relinking*
FTSP	*The Family Traveling Salesman Problem*
GA	*Genetic Algorithm*
GCSP	*Generalized Covering Salesman Problem*
GRASP	*Greedy Random Adaptive Search Procedure*
GTSP	*Generalized Traveling Salesman Problem*
IA	Inteligência Artificial
ILP	*Integer Linear Programming*
ILS	*Iterated Local Search*
LS	*Local Search*
MDP	*Markov Decision Process*
MH	Metaheurística
ML	*Machine Learning*
PCVF	Problema do Caixeiro Viajante em Famílias
PUX	*Parametrized Uniform Crossover*
QL	*Q-Learning*
RL	*Reinforcement Learning*
RKGA	*Random-keys Genetic Algorithm*
RPD	*Relative Percentage Deviation*
RVND	*Random Variable Neighborhood Descent*
TSP	*Traveling Salesman Problem*
VND	*Variable Neighborhood Descent*
WRS	*Wilcoxon Signed-Rank*

Lista de Símbolos

A	conjunto finito de ações.
a	ação.
c_{ij}	custo associado ao percurso de i para j.
df	fator de desconto.
E	estimativa do retorno total esperado.
F_l	Conjunto de famílias.
L	quantidade de famílias.
l	família.
lf	coeficiente de aprendizagem.
N	conjunto de nós.
p	tamanho da população.
p_e	partição elite.
p_m	partição de mutantes.
$Q(s, a)$	função valor-ação.
R	valor da recompensa obtido pelo aprendizado.
S	conjunto finito de estados.
s	estado.
T	conjunto de instantes de tempo.
V	demanda total de visitas.
v_l	número de visitas para cada família l.
$V(s)$	função valor-estado.
N	conjunto de nós.
π	política de controle.
ρ_e	probabilidade de herança do cromossomo do pai elite.

Sumário

1 Introdução

Nas últimas décadas, pôde-se observar um crescimento exponencial das tecnologias, o surgimento da indústria 4.0, e uma competitividade eminente entre diversos segmentos do mercado, como por exemplo, em empresas logísticas. Estes fatores evidenciaram a real necessidade do desenvolvimento e aprimoramento de ferramentas capazes de automatizar processos que demandam recursos tecnológicos e físicos além de promover maior eficiência e precisão, características indispensáveis atualmente (AIRES *et al.*, 2019). Neste sentido, e aliado ao aumento significativo de produção e armazenamento de dados, a área de Inteligência Artificial (IA) se tornou uma das áreas mais estudadas e promissoras devido à sua capacidade de resolução de diversos desafios, por exemplo, na otimização de recursos e aumento da qualidade dos serviços (GOMES, 2010).

Diante desse cenário, áreas como ciência da computação e pesquisa operacional estão investindo cada vez mais no desenvolvimento de métodos e dispositivos que integram técnicas de inteligência artificial na resolução de problemas de otimização combinatória (SONG *et al.*, 2019), garantindo assim desempenhos cada vez melhores, como maior produtividade, minimização robusta dos custos e consequentemente maximização dos índices de lucratividade.

Um dos problemas clássicos de otimização combinatória é o Problema do Caixeiro Viajante (TSP, do inglês, *Traveling Salesman Problem*) (DANTZIG *et al.*, 1954). O problema apresenta uma situação prática de empresas logísticas na otimização da cadeia de suprimentos. O desafio do TSP consiste em determinar a rota de custo mínimo, ou distância, para o transporte de produtos, partindo de um depósito, visitando todos os clientes, ou nós, uma única vez, e retornando ao ponto de partida. Este problema requer uma atenção especial, pois o custo de transporte representa a maior parcela dos custos totais e melhorias podem promover um impacto financeiro (SNYDER; SHEN, 2019).

O TSP incorpora apenas os parâmetros de custo ou distância na função objetivo, e tem como objetivo encontrar as melhores soluções que minimizem os custos, ou distâncias, com restrições de designação e conectividade relativas à construção da rota do caixeiro viajante. Contudo, em problemas reais de empresas logísticas, principalmente empresas de grande porte, com diferentes categorias de produtos, e complexos sistemas de armazenagem e

distribuição, necessitam de uma otimização englobando diversas variáveis que traduzam de forma robusta os desafios enfrentados na prática.

Uma das variantes do TSP consiste do Problema do Caixeiro Viajante em Famílias (FTSP, do inglês, *Family Traveling Salesman Problem*) introduzida por Morán-Mirabal *et al.* (2014). Nesta variante, o conjunto de clientes é particionado em famílias e o número de visitas a serem realizadas em cada família é predefinido. Sendo assim, o número de clientes visitados na rota é definido pela soma das visitas em cada família. Portanto, nem todos os clientes pertencentes ao problema estarão contidos na rota.

O FTSP foi motivado por um problema de otimização para coleta de produtos armazenados em locais diferentes, no mesmo armazém ou em armazéns diferentes (MORÁN-MIRABAL *et al.*, 2014). A estratégia do sistema de armazenamento caótico (CSS, do inglês, *Chaotic Storage System*) armazena cada produto aleatoriamente onde há espaço em um determinado armazém, desconsiderando os locais especificados. Os avanços tecnológicos permitem uma localização rápida de cada produto. Assim, o armazém fica mais flexível e otimizado, o que reduz os erros de seleção de produtos (WALTS, 2020). Esta situação permite uma aplicação prática do FTSP. Uma família é composta por produtos do mesmo tipo espalhados por todo o armazém. As visitas exigidas em cada família representam a demanda por pedidos de produtos. Assim, o problema de coleta dos produtos de acordo com sua demanda, pode ser modelado como um FTSP.

Morán-Mirabal *et al.* (2014) provou que o FTSP é um problema NP-Difícil. Portanto, não são conhecidos algoritmos que consigam resolver problemas de otimização combinatória desta classe na otimalidade e em tempo polinomial (GAREY; JOHNSON, 1990). Para a resolução de problemas de otimização combinatória bastaria fazer a enumeração de todas as soluções viáveis e então guardar aquela que retornar a melhor função objetivo. Contudo, em problemas de dimensões elevadas, o processo enumerativo pleno pode ser impraticável se o número de soluções viáveis crescer exponencialmente em virtude da complexidade do problema. Neste sentido, os métodos de otimização buscam formas de tornar o processo de enumeração mais inteligente, de modo a possibilitar uma redução do número de soluções viáveis e consequentemente a resolução de problemas de tamanhos reais.

Podemos classificar os métodos de otimização combinatória como exatos ou heurísticos. Nos métodos exatos, como forma de superar a limitação frente ao processo enumerativo, algumas restrições precisam ser geradas e adicionadas ao longo da otimização do problema. Uma abordagem exata que permite superar essa limitação é a técnica *Branch-and-Cut* (B&C) (MITCHELL, 1988), a qual integra a metodologia *Branch-and-Bound* (B&B) (LAWLER; WOOD, 1966) com algoritmos de planos de corte. O método B&C é um algoritmo muito eficiente para resolver uma ampla variedade de problemas de programação inteira e capaz de fornecer resultados na otimalidade (MITCHELL, 1988). Além disso, essa abor-

dagem apresenta vantagem frente a outros métodos de otimização para problemas de programação linear inteira, como, por exemplo, menor tempo computacional em relação ao B&B e maior confiabilidade em relação ao Planos de Corte de *Gomory* (do inglês, *Gomory Cutting Planes*) (GILMORE; GOMORY, 1961).

As heurísticas, por sua vez, consistem em ferramentas eficientes na obtenção de resultados aproximados, de boa qualidade em um tempo computacional satisfatório. As metaheurísticas (MHs) consistem em um método heurístico capaz de resolver problemas de otimização de alta complexidade computacional, com tempo computacional razoável (BLUM; ROLI, 2003). Como exemplo de MHs, têm-se os algoritmos genéticos (AGs), os quais simulam o processo evolutivo dos indivíduos, transformando a população através de operadores de reprodução, cruzamento e mutações, obtendo novas gerações (GOLDBARG, 2005). Nos AGs objetiva-se encontrar o indivíduo com o melhor material genético (*fitness*), e no caso do TSP, as informações genéticas codificadas e ordenadas representam a sequência de clientes, a serem visitados e, portanto, uma solução viável para o problema (LACERDA; CARVALHO, 1999). Portanto, as soluções viáveis de um problema são representadas por cada indivíduo da população nos algoritmos genéticos. Esta dissertação propõe a metaheurística *Biased Random-key Genetic Algorithm* (BRKGA), uma variante dos AGs, na qual as soluções são codificadas através de um vetor de chaves aleatórias. O BRKGA é um método relativamente novo, com resultados competitivos em comparação aos demais métodos da literatura na área de Pesquisa Operacional, tendo seus primeiros resultados em 2002 (ERICSSON *et al.*, 2002).

Segundo Talbi (2002) uma tendência na área de Pesquisa Operacional, em otimização combinatória, é o desenvolvimento de métodos híbridos exatos e heurísticos. Uma razão para o aumento de interesse entre os pesquisadores dessa área se diz respeito aos algoritmos híbridos apresentarem resultados de boa qualidade na resolução de problemas de otimização (JUNIOR *et al.*, 2007a). Existem diversos tipos de integrações possíveis. A integração de metaheurísticas com heurísticas de busca local é muito utilizada, como forma de intensificar a busca de melhores soluções em regiões promissoras. Outro exemplo de versão híbrida de metaheurísticas pode ser obtido com a integração de técnicas de Aprendizado de Máquinas (ML, do inglês *Machine Learning*). Neste caso, as técnicas de ML ajudam a definir ou controlar os parâmetros das MHs de forma autônoma e com maior eficiência (KARIMI-MAMAGHAN *et al.*, 2021). Pode-se citar também uma possibilidade de hibridizar métodos exatos, através da integração de um método exato com heurísticas de busca local, por exemplo.

Aprendizado de Máquinas é uma subárea da IA, que desenvolve máquinas capazes de pensar, raciocinar, aprender e resolver problemas altamente complexos (BISHOP, 2006). Pode também ser definida como um conjunto de algoritmos que são capazes de aprender assim como o cérebro humano. Os aprendizados e padrões extraídos pelas técnicas de ML

são incorporados nas MHs as tornando mais inteligentes, permitindo assim um melhor aproveitamento das informações obtidas e consequentemente um aumento significativo no desempenho dos algoritmos (Song *et al.* (2019), Talbi (2016)). Uma técnica de ML, *Q-Learning* (QL), consiste em um algoritmo de Aprendizado por Reforço (RL, do inglês *Reinforcement Learning*), capaz de aprender padrões e tendências a partir de dados e análises do problemas gerados em todo o processo de pesquisa (WATKINS; DAYAN, 1992).

Neste contexto, Chaves e Lorena (2021) desenvolveram o método BRKGA-QL, um método relativamente novo que tem obtido sucesso quando aplicado em problemas de otimização, em termos de qualidade da solução e tempo computacional. O BRKGA-QL consiste em uma meta-heurística híbrida: BRKGA com *Q-learning*. Neste contexto, o QL é utilizado para configuração *online* dos parâmetros do BRKGA, podendo minimizar significativamente os efeitos de más decisões tomadas durante o processo de ajuste e possibilitar resultados de melhor qualidade. O BRKGA-QL consiste em um método simples de ser implementado a problemas de otimização com o desenvolvimento de novos decodificadores e heurísticas de busca local. Através dos resultados já obtidos podemos afirmar que o método apresenta eficiência e competitividade frente aos algoritmos propostos na literatura para resolução de problemas similares. Como exemplo, no contexto do TSP simétrico, o método encontrou a solução ótima (conhecida na literatura) para todas as instâncias até 200 nós em poucos segundos e em 72% das instâncias testadas (CHAVES; LORENA, 2021).

Nesta dissertação foram propostos métodos exatos e heurísticos para resolver o FTSP. Foi desenvolvido um algoritmo paralelo B&C (P-B&C) com um procedimento de busca local eficiente para encontrar soluções ótimas com um limite de tempo de três horas. Em seguida, o BRKGA-QL (CHAVES; LORENA, 2021) foi adaptado para obter boas soluções, principalmente em instâncias FTSP de grande escala. Nesta versão do BRKGA-QL foram desenvolvidos novos decodificadores e heurísticas de busca local específicas para o FTSP.

1.1 Objetivos

1.1.1 Objetivo Geral

Esta dissertação tem como objetivo o estudo, desenvolvimento e aplicação de dois métodos híbridos para resolução do Problema do Caixeiro Viajante em Famílias (FTSP): **a**) Um método de programação Linear Inteira P-B&C e **b)** Uma metaheurística híbrida, BRKGA-QL, a qual combina a metaheurística (BRKGA), um algoritmo de aprendizagem por reforço (Q-Learning) e heurísticas de busca local.

1.1.2 Objetivos específicos

São objetivos específicos deste trabalho:

- Realizar a integração do algoritmo B&C com procedimentos de busca local compondo o método P-B&C para resolução do FTSP;
- Realizar a integração dos métodos BRKGA e QL para composição do BRKGA-QL bem como melhorias e adaptações para resolver o FTSP;
- Aplicar os métodos de otimização exato (P-B&C) e heurístico (BRKGA-QL) para resolução do FTSP;
- Implementar *decoder* e heurísticas de busca local eficientes para o FTSP;
- Analisar a eficácia e robustez dos métodos através de extensos testes computacionais com as instâncias da literatura;
- Analisar a robustez e competitividade dos métodos desenvolvidos através da comparação com os resultados obtidos pelos métodos encontrados na literatura para resolução do FTSP;
- Analisar a influência de cada componente e atributo do BRKGA-QL por meio de testes computacionais.

1.2 Organização do trabalho

Esta dissertação está organizada em cinco capítulos, sendo este o primeiro. O Capítulo 2 define o Problema do Caixeiro Viajante em Famílias apresentando seus conceitos e componentes. Neste é apresentado um modelo de programação inteira, a revisão bibliográfica a respeito dos métodos aplicados na resolução do FTSP e exemplos de problemas conhecidos na literatura que são correlatos ao FTSP.

Em seguida, no Capítulo 3 faz-se uma descrição detalhada dos dois métodos desenvolvidos para a resolução do FTSP: P-B&C com busca local e BRKGA-QL com heurísticas de busca local. São apresentando os conceitos básicos e as principais características de cada método, assim como as respectivas implementações para o FTSP.

No Capítulo 4 estão apresentadas as análises sobre os resultados dos experimentos computacionais realizados pelos métodos aplicados. Os resultados obtidos pelos métodos desenvolvidos neste trabalho são comparados com os resultados encontrados na literatura. São apresentados também os impactos de cada componente do BRKGA-QL.

O Capítulo 5 apresenta as considerações, sintetiza as principais conclusões a respeito dos resultados obtidos, apresenta um sumário das principais contribuições deste trabalho e por fim, apresenta os possíveis impactos científicos, sociais, econômicos e ambientais deste trabalho.

2 Definição do Problema

O Problema do Caixeiro Viajante em Famílias (FTSP) é uma variante do Problema do Caixeiro Viajante (TSP) e consiste em um problema de otimização combinatória da classe NP-Difícil (GAREY; JOHNSON, 1990). No TSP o objetivo é determinar uma rota de custo mínimo sendo obrigatório que o caixeiro viajante visite todos os clientes, uma única vez, e a rota deve iniciar e finalizar no mesmo ponto. Por outro lado, apesar do FTSP também ter como objetivo definir uma rota de menor custo, o caixeiro viajante deve iniciar e finalizar a rota em um ponto definido como depósito e somente é necessário visitar um número predefinido de clientes do problema. Em suma, nesta variante são necessárias duas tomadas de decisões importantes: (i) definir quais clientes serão visitados; e (ii) definir a rota de menor custo.

O FTSP foi desenvolvido a partir de um problema de separação de produtos de mesmo tipo, armazenados separadamente, no estoque ou até mesmo em armazéns diferentes. Neste contexto, surgiu a necessidade de otimizar a coleta dos produtos necessários de cada tipo, conforme a demanda, percorrendo a menor rota possível no armazém (MORÁN-MIRABAL *et al.*, 2014). Portanto, a motivação inicial para a elaboração do problema se enfatiza na escolha da rota de menor custo a realizar-se dentro de um armazém de modo a coletar produtos de diferentes categorias armazenados separadamente.

Neste sentido, o FTSP parte do princípio que dado um conjunto de produtos, aqueles pertencentes a uma mesma categoria são agrupados em um subconjunto denominado família. Mediante esse critério, o conjunto de nós é formado por diversas famílias que englobam produtos de mesma similaridade cada. A rota a ser construída é norteada conforme demanda dos produtos, isto significa que somente a quantidade de produtos requerida de cada categoria será selecionada. De forma análoga, as diferentes categorias de produtos correspondem às famílias do problema, a demanda por categoria representa a quantidade de visitas necessárias para cada família, e os produtos a serem coletados equivalem aos clientes.

A Figura 2.1 ilustra o exemplo de uma solução viável para uma instância do FTSP com três famílias. Compõem o exemplo selecionado, o nó 0, ou origem, que simboliza o depósito e treze produtos que representam o total de clientes contidos na instância. A

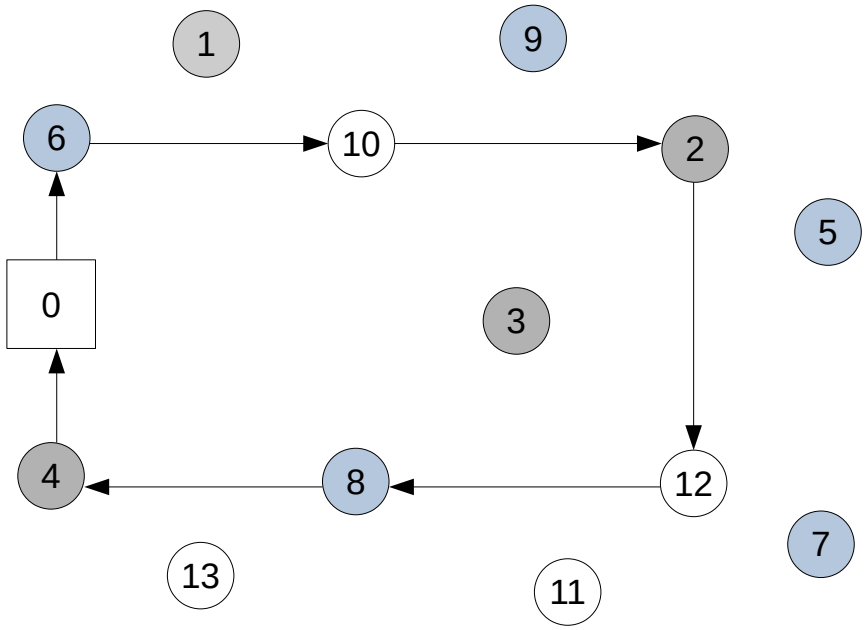

FIGURA 2.1 – Exemplo de uma solução viável para uma instância FTSP. Elaborada pela autora.

Família 1 é representada pela cor cinza e contém quatro membros (nós 1-4), a Família 2 indicada pela cor azul, tem cinco membros (nós 5-9) e a Família 3 representada pela cor branca contém quadro membros (nós 10-13). A demanda requerida para cada família é de apenas dois produtos cada, ou seja, somente dois clientes de cada família devem ser visitados. Não é necessário que os clientes de uma mesma família sejam visitados consecutivamente. Como pode ser observado, a rota inicia-se no depósito, e a rota construída foi: {(0,6),(6,10),(10,2),(2,14),(14,8),(8,4),(4,0)}.

2.1 Formulação Matemática

Como parâmetros de entrada do problema, considera-se:

- N: conjunto de clientes;
- A: conjunto de arestas;
- L: quantidade de famílias;
- F_l: conjuntos de famílias, $l = 1, \ldots, L$;
- n_l: número de membros de cada família l;

- v_l: quantidade de visitas requisitadas em cada família l, $l = 1, \ldots, L$;
- c_{ij}: custo relacionado ao percurso do cliente i para o cliente j;
- V: demanda total que designa a quantidade de visitas totais necessárias para construção da rota do caixeiro viajante. V pode ser expresso como:

$$V = \sum_{l=1}^{L} v_l. \tag{2.1}$$

O FTSP pode ser modelado usando um grafo completo e direcionado $G = (\{0\} \cup N, A)$, onde o elemento 0 representa o depósito, origem da rota. N indica o conjunto de clientes e A o conjunto de arestas. Um custo não negativo c_{ij} está associado a cada aresta $(i, j) \in E$. O conjunto N é particionado em famílias disjuntas L. Cada família $l \in \mathcal{L} = \{1, \cdots, L\}$ é representada por F_l e contém n_l membros. adota-se a ordenação de elementos consecutivos para composição de cada família e assume-se que $n_l \geq 1$. Sendo assim, a família 1 é dada pelos clientes $F_1 = \{1, \cdots, n_1\}$, a família 2 dada por $F_2 = \{n_1 + 1, \cdots, n_1 + n_2\}$, e assim por diante até que todas as famílias estejam completas, tal que $\cup_{l \in \mathcal{L}} F_l = N$ e $\cap_{l \in \mathcal{L}} F_l = \emptyset$. Além disso, cada família requer v_l visitas, totalizando V. Pode-se afirmar que uma família l recebeu o número de visitas necessárias se abranger v_l clientes da família l.

Para descrever este modelo, considere x_{ij} uma variável binária que indica se a aresta (i, j) foi percorrida:

$$x_{ij} = \begin{cases} 1, & \text{se o arco de } i \text{ para } j \text{ estiver no caminho;} \\ 0, & \text{caso contrário;} \end{cases}$$

Considera-se também y_i outra variável binária, em que:

$$y_i = \begin{cases} 1, & \text{se o cliente } i \text{ for visitado na rota;} \\ 0, & \text{caso contrário;} \end{cases}$$

O objetivo do FTSP é determinar uma rota de custo mínimo, partindo e retornando ao depósito, onde exatamente v_l clientes de cada família l são visitados na rota. Assim sendo, o FTSP foi formulado por (BERNARDINO; PAIAS, 2018) usando o seguinte modelo de programação linear inteira:

$$min \sum_{(i,j) \in E} c_{ij} x_{ij} \tag{2.2}$$

Sujeito à:

$$\sum_{j \in N} x_{0j} = 1 \tag{2.3}$$

$$\sum_{j \in \{0\} \cup N} x_{ij} = y_i \qquad \forall i \in N \tag{2.4}$$

$$\sum_{j \in \{0\} \cup N} x_{ji} - \sum_{j \in \{0\} \cup N} x_{ij} = 0 \qquad \forall i \in \{0\} \cup N \tag{2.5}$$

$$\sum_{i \in F_l} y_i = v_l \qquad \forall l \in \mathcal{L} \tag{2.6}$$

$$x(S', S) \geq y_k \qquad \forall S \subset N, \forall k \in S \tag{2.7}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i, j) \in E \tag{2.8}$$

$$y_i \in \{0, 1\} \qquad \forall i \in N \tag{2.9}$$

A equação (2.2) representa a função objetivo, que minimiza os custos da rota do caixeiro viajante. As restrições (2.3) garantem que o depósito seja o primeiro nó da rota. As restrições (2.4) têm a função de conectar as variáveis x e y e garantir que se o cliente i for visitado na rota, então deve haver um cliente j, tal que j é servido imediatamente após i. As restrições (2.5) restringem que os graus de entrada e saída dos vértices sejam iguais. Além disso, em conjunto com a restrição (2.3) asseguram a conectividade de uma rota. As restrições (2.6) garantem o número de visitas necessárias por família. Restrições (2.7), chamadas de desigualdades de corte de conectividade (CC), são baseadas nas conhecidas restrições de eliminação de sub-rotas propostas por Dantzig *et al.* (1954) para o TSP. Essas restrições exigem que pelo menos um arco no conjunto de corte $[S', S]$, com $S' = (\{0\} \cup N) \backslash S$, deve ser usado se houver um nó k sendo visitado no conjunto S. Finalmente, o domínio das variáveis x e y é representado respectivamente pelas restrições (2.8) e (2.9).

Para levar em conta as características do FTSP, Bernardino e Paias (2018) apresenta um novo conjunto de restrições para evitar sub-rotas, denominados pelos autores como desigualdades de visitas familiares arredondadas (RFV, do inglês *Rounded Family Visits*). Essas desigualdades garantem a conectividade da rota impondo que pelo menos um arco deve ser usado no corte $[S', S]$ se existir uma família l na qual o número de clientes em S' não é suficiente para atender todas as v_l visitas necessárias para esta família. Neste caso específico, podemos substituir a variável y_k no lado direito de (2.7) pelo valor 1, obtendo as seguintes desigualdades válidas:

$$x(S', S) \geq 1 \qquad \forall S \subset N : \exists l \in \mathcal{L} : |S \cap F_l| \geq n_l - v_l + 1 \tag{2.10}$$

2.2 Revisão da Literatura

Esta seção apresenta uma revisão bibliográfica acerca dos métodos propostos na literatura para a resolução do FTSP.

O FTSP foi introduzido em 2014 por Morán-Mirabal *et al.* (2014) e até o presente momento poucos artigos na literatura o abordam. De nosso conhecimento, há apenas quatro trabalhos abordando o FTSP na literatura. Ao final desta seção, a Tabela 2.1 apresenta um resumo deste trabalhos, com os respectivos métodos aplicados na resolução do FTSP.

Morán-Mirabal *et al.* (2014) desenvolveram métodos heurísticos na busca de soluções ótimas. Foram propostas duas metaheurísticas: um *Biased Random-key Genetic Algorithm* (BRKGA) e um *Greedy Random Adaptive Search Procedure* (GRASP) com *evolucionary path-relinking* (evPR). Através dos resultados obtidos pelos experimentos computacionais, os autores afirmam que ambas as heurísticas aplicadas se mostraram eficazes na solução do problema proposto pois foram capazes de encontrar soluções ótimas ou quase ótimas com qualidade satisfatória. A metaheurística GRASP + evPR teve melhor desempenho em instâncias de dimensões maiores, contudo apresentou um tempo computacional considerável para encontrar a melhor solução. Por outro lado, o BRKGA desempenhou melhor em instâncias com dimensões menores, sendo capaz de encontrar soluções iguais e até mesmo melhores com tempo computacional inferior em comparação ao GRASP + evPR. Com base nas análises e comparações entre os resultados obtidos por ambos métodos, os autores concluíram que conforme a dimensão das instâncias se torna maior, o GRASP + evPR supera consideravelmente o BRKGA.

Um modelo de programação linear inteira (ILP) também foi desenvolvido por Morán-Mirabal *et al.* (2014), cujas restrições de eliminação de sub-rotas foram baseadas no trabalho de Dantzig *et al.* (1954). O modelo ILP foi testado para instâncias com até 127 nós e apenas uma com 280 nós, e foi capaz de encontrar as soluções ótimas para as instâncias com até 48 nós. As instâncias foram criadas pela adaptação das instâncias do TSPLIB (REINELT, 1991).

Bernardino e Paias (2018) apresentaram seis modelos matemáticos diferentes e um algoritmo *branch-and-cut* (B&C) para resolver o melhor modelo, referente ao valor de relaxação de programação linear. As formulações matemáticas propostas apenas se diferenciam no conjunto de restrições de eliminação de sub-rotas, sendo assim aplicados diferentes modelos de fluxo para formular a conectividade da rota. O B&C resolveu instâncias com até 127 nós em menos de 70 segundos e uma instância com 280 nós em uma hora. De acordo com os autores, para melhorar a qualidade dos limitantes inferiores obtidos no B&C em instâncias de dimensões maiores devem ser derivadas novas desigualdades válidas e ainda mais eficazes. Também foi proposta uma metaheurística *Iterated*

Local Search (ILS) (STüTZLE, 1998) para resolver instâncias de grande escala, melhorando os limites superiores mais conhecidos.

A metaheurística ILS aplicada no estudo busca uma solução ótima através de um procedimento de busca local (LS) e um método de perturbação (PM). A primeira etapa do ILS consiste em determinar uma solução viável inicial para o FTSP. A partir da solução inicial aplica-se então a heurística do vizinho mais próximo (NN, do inglês *Nearest Neighbor*) respeitando os critérios de visitas por família, e tendo como critério de parada a conclusão de todas as famílias. No próximo passo aplica-se um procedimento de busca local (LS) na solução atual a fim de obter um valor ótimo local. O procedimento é executado até que se obtenha uma solução melhor do que a atual. Após obter um ótimo local, um método de perturbação é aplicado, o qual identifica e escapa de um ótimo local percorrendo todo espaço de solução de forma iterativa. Para instâncias de dimensões maiores com valor ótimo desconhecido, o algoritmo ILS foi capaz de melhorar os limitantes superiores já conhecidos na literatura com o tempo computacional razoável.

Bernardino e Paias (2021b) apresentaram três métodos de solução para o FTSP: um Algoritmo Genético (GA) (HOLLAND, 1975), um ILS aprimorado de Bernardino e Paias (2018) e um algoritmo híbrido que integra um algoritmo B&C com procedimento de busca local (LS).

O GA apresentado utiliza uma permutação dos nós para que a rota do FTSP seja construída. Neste caso, o nó correspondente ao depósito não é incluído no cromossomo já que é o ponto de partida como também o ponto final da rota. De todo modo, a permutação do conjunto N constitui os cromossomos que são usados no GA. Como o FTSP trata-se de um problema de minimização dos custos, e o valor de aptidão de um cromossomo neste caso é representada pelo custo associado à solução, então os indivíduos com os menores valores de aptidão são considerados os indivíduos mais aptos. Através dos resultados computacionais os autores concluíram que o GA apresentou-se como um método eficiente visto que foi capaz de solucionar instâncias com 1002 nós com uma média de tempo computacional de 243 segundos.

O ILS proposto consiste em uma adaptação com melhorias do Bernardino e Paias (2018). Neste, são consideradas duas vizinhanças adicionais e aplica-se um método de pertubação diferente baseado em uma busca dinâmica de grande vizinhança. Basicamente, foram introduzidas vizinhanças adicionais no processo de busca local como também foi alterada a política de busca. O procedimento de perturbação é orientado por um critério dinâmico que é atualizado ao decorrer da execução do algoritmo. Tais modificações resultaram no aumento da diversidade das soluções encontradas e consequentemente na melhor qualidade das soluções obtidas.

Finalmente, o algoritmo híbrido combina um B&C com procedimentos de busca local e

foi projetado para solucionar as instâncias *benchmark* de dimensões maiores que não foram solucionadas pelo B&C proposto no trabalho anterior dos autores (BERNARDINO; PAIAS, 2018). Dessa forma, o algoritmo híbrido é composto por duas fases, o B&C é responsável pela fase construtiva e o procedimento LS atua na fase de melhoria. A ideia principal do algoritmo consiste em construir uma solução inicial de boa qualidade através de um método exato, e posteriormente aplicar o procedimento de busca local com o objetivo de melhorar ainda mais a solução obtida inicialmente. Em sequência, aplica-se o B&C novamente, porém como parte de um método heurístico, com algumas modificações em relação ao modelo inicial e mais apto a alcançar soluções viáveis de qualidade superior. A prioridade do método é constituir a melhor solução possível para o que foi denominado de problema central pelos autores, que consiste na solução parcial responsável por estabelecer o esboço da rota. Como resultado, o algoritmo híbrido foi capaz de melhorar os limitantes superiores conhecidos para todas as instâncias *benchmark*. Este algoritmo híbrido é a heurística estado-da-arte para solução do FTSP.

Os autores concluíram também que ambos os métodos propostos foram capazes de melhorar os limitantes superiores conhecidos na literatura. Além disso, os autores desenvolveram um gerador de instâncias de modo a obter instâncias com características diversas.

Bernardino e Paias (2021a) abordou uma variante FTSP em que famílias com incompatibilidades entre si não podem ser visitadas na mesma rota. Isso acontece, por exemplo, com produtos alimentícios e produtos químicos, que não devem ser encaminhados juntos. Denominado pelos autores de FTSP com restrições de incompatibilidade (FTSP-IC). Os autores propuseram três modelos matemáticos, um algoritmo B&C, um algoritmo *Ant Colony Optimization* (ACO) (DORIGO, 1992) e um algoritmo ILS.

No algoritmo B&C, o processo de separação executa os algoritmos de forma sequencial em ordem de complexidade. Os algoritmos separam as desigualdades tanto de maneira exata, quanto heurística, garantindo que a solução obtida seja viável para o FTSP-IC. Para obter os limitantes superiores das instâncias estudadas, os autores desenvolveram uma heurística construtiva baseada na heurística do vizinho mais próximo (*Nearest Insertion*) (GOLDBARG, 2005) seguida de um procedimento de busca local (LS).

O algoritmo ACO projetado pelos autores para o FTSP-IC é baseado na extensão do sistema de colônia de formigas (ACS, do inglês*Ant Colony System)*) (DORIGO; GAMBARDELLA, 1997). A principal ideia do algoritmo ACO é usar formigas artificiais (por exemplo, agentes de simulação) para localizar soluções viáveis, a cada iteração, usando as informações de feromônio de soluções geradas anteriormente. As formigas simuladas registram as posições e qualidade das soluções, permitindo que em iterações seguintes mais formigas localizem soluções melhores (DORIGO; GAMBARDELLA, 1997). Portanto, o algoritmo utiliza funções probabilísticas levando em consideração as experiências acu-

muladas das formigas e as características da instância estudada. Emprega-se também uma técnica de diversificação de formigas, e um procedimento de busca local é aplicado à formiga melhor construtora e então as trilhas se feromônio são atualizadas.

O algoritmo ILS aplicado pelos autores utiliza a heurística do vizinho mais próximo e o mesmo procedimento de busca local adotado no algoritmo ACO. A ideia principal do método de perturbação é tornar a solução inviável removendo os nós e então restaurar sua viabilidade inserindo nós. Este critério é utilizado a fim de alcançar um conjunto mais diversificado de soluções.

Com base nos resultados obtidos através dos experimentos computacionais, os autores concluíram que ambas as metaheurísticas têm desempenho semelhante em termos de qualidade da solução.

TABELA 2.1 – Métodos aplicados na Literatura para resolução do FTSP.

Jornal/Conferência	Autores	Métodos aplicados
ITOR	Morán-Mirabal *et al.* (2014)	BRKGA e (GRASP + evPR)
EJOR	Bernardino e Paias (2018)	ILS (LS + PM)
ITOR	Bernardino e Paias (2021b)	GA, ILS, Algoritmo Híbrido(B&C + LS)
Networks	Bernardino e Paias (2021a)	B&C, ACO e ILS

2.3 Problemas Correlatos ao FTSP

Esta seção apresenta alguns problemas que podem ser modelados como um FTSP, tais como:

1. *Traveling Salesman Problem* (TSP) (DANTZIG *et al.*, 1954)
2. *Generalized Covering Salesman Problem* (GCSP) (GOLDEN *et al.*, 2012)
3. *Generalized Traveling Salesman Problem* (GTSP) (LAPORTE *et al.*, 1987)
4. *Capacitated Traveling Purchaser Problem* (CTPP) (BOCTOR *et al.*, 2003)

Conforme mencionado no Capítulo 1, o clássico problema do caixeiro viajante (TSP) (DANTZIG *et al.*, 1954) objetiva determinar a rota de custo ou distância mínima, em que todos os clientes do problema devem ser visitados uma única vez, e o nó inicial e final devem ser iguais. O FTSP, por sua vez, possui o mesmo objetivo, contudo somente precisa visitar um número predefinido de clientes em cada família. Dessa forma, um TSP pode ser facilmente adaptado para um FTSP e vice-versa.

O GCSP (GOLDEN *et al.*, 2012), ou problema do vendedor de cobertura generalizado, busca a determinação de uma rota com distância mínima, de modo que o subconjunto de nós visitados tenha cobertura máxima. Cada nó pode cobrir um determinado subconjunto de nós, e conta, além disso, com uma demanda preestabelecida.

Traduzindo para o FTSP, as famílias representam os nós i e compreendem o subconjunto de nós j, e a quantidade de visitas necessárias em cada família corresponde a demanda predefinida.

O FTSP pode ser visto com uma extensão do problema generalizado do caixeiro viajante (GTSP). No GTSP (LAPORTE *et al.*, 1987), o conjunto de nós é agrupado em *clusters*, equivalentes as famílias do FTSP, e o objetivo consiste em determinar a rota de menor custo de modo que cada *cluster* seja visitado ao menos uma vez e cada nó apenas uma única vez (MORÁN-MIRABAL *et al.*, 2014).

Em uma de suas variantes, denominado de igualdade GTSP (E-GTSP), cada *cluster* deve ser visitado uma vez. Sob a perspectiva do FTSP, em um caso onde apenas se faz necessária a visita de um único nó em cada família correspondente, as soluções viáveis encontradas para o problema também serão viáveis para o E-GTSP. Além disso, o GTSP é um caso especial do GCSP e, consequentemente, o FTSP também é uma variante deste problema.

Outro problema encontrado na literatura que pode ser modelado como o FTSP, segundo Bernardino e Paias (2018) é o problema do caixeiro comprador viajante capacitado (CTPP) (BOCTOR *et al.*, 2003). O objetivo é determinar uma rota que permita ao caixeiro comprador visitar todas as cidades-mercado, e adquirir com o menor custo possível os itens necessários. Cada item está disponível em um subconjunto de mercados-cidade, podendo ser ofertado por diferentes custos e quantidades. Em um exemplo, no qual cada mercado-cidade oferta apenas uma unidade de cada item, e que o custo associado ao item é igual em todos os mercados que vendem este item, então esse problema pode ser resolvido de forma equivalente ao FTSP.

3 Métodos e Implementações

Neste capítulo estão detalhados todos os métodos aplicados neste trabalho para a resolução do FTSP. O método exato P-B&C desenvolvido para o FTSP é apresentado na Seção 3.1. A metaheuristica BRKGA e o algoritmo *Q-Learning* são descritos nas Seções 3.2 e 3.3 respectivamente. A Seção 3.4 descreve em detalhes o método BRKGA-QL proposto para FTSP. A Seção 3.5 apresenta a estrutura da busca local. Na Seção 3.6.1 estão descritos os *decoders* desenvolvidos e aplicados na resolução do FTSP, assim como as respectivas implementações. Por fim, a Seção 3.6.2 detalha as heurísticas de busca local aplicadas no FTSP.

3.1 Algoritmo *Branch-and-Cut* Paralelo (P-B&C)

Esta seção apresenta o algoritmo *Branch-and-Cut* Paralelo (P-B&C) desenvolvido para resolver o FTSP. Em contraste com os métodos sequenciais da literatura (veja Morán-Mirabal *et al.* (2014), Bernardino e Paias (2018) e Bernardino e Paias (2021b)), o algoritmo exato proposto é executado continuamente e independentemente em duas frentes, combinando a tradicional abordagem B&C com um procedimento de busca local (LS, do inglês *local search*) em uma estrutura de processamento paralelizado. No algoritmo P-B&C, a primeira frente é responsável por garantir a otimalidade de cada instância resolvida, além de identificar grupos interessantes de clientes, que podem ser atendidos na mesma rota, enquanto a segunda frente é responsável por gerar boas soluções de roteamento levando em consideração esses clientes selecionados.

3.1.1 Frente *Branch-and-Cut*

Branch-and-cut (B&C) é um algoritmo *branch-and-bound* (B&B) no qual planos de corte são usados para fortalecer as relaxações lineares de um determinado programa linear inteiro-misto ao longo da árvore B&B. O algoritmo começa definindo uma lista de nós ativos ($\mathcal{W}$) na árvore B&B, uma melhor solução $\mathcal{S}_{Melhor}$ e seu valor de função objetivo $z(\mathcal{S}_{Melhor})$. No início, a lista $\mathcal{W}$ contém apenas o problema relaxado obtido desconside-

rando a integralidade de todas as variáveis. A cada iteração, o algoritmo seleciona um nó $\mathcal{U}_k$ de $\mathcal{W}$, exclui-o da lista e resolve o problema relaxado $\mathcal{P}_k$ correspondente ao nó atual. Em seguida, uma etapa de poda é aplicada para podar $\mathcal{U}_k$ de acordo com três critérios: *i*) se $\mathcal{P}_k$ for inviável (poda por inviabilidade); *ii*) se a solução $\mathcal{S}_k$ de $\mathcal{P}_k$ for pior que a melhor solução conhecida $\mathcal{S}_{Melhor}$ (poda por limitante); *iii*) se a solução $\mathcal{S}_k$ for viável para o problema original (poda por integralidade). Sempre que a etapa de poda falha, ou seja, o nó atual não foi eliminado, desigualdades válidas podem ser adicionadas para fortalecer a relaxação linear atual e $\mathcal{P}_k$ é re-otimizado novamente (etapa de planos de corte). Por outro lado, é importante selecionar uma variável fracionária v_k de $\mathcal{S}_k$ para ramificação. A etapa de ramificação cria dois novos nós $\mathcal{U}_{k_1}$ e $\mathcal{U}_{k_2}$ impondo $v_k \leq \lfloor v_k \rfloor$ e $v_k \geq \lceil v_k \rceil$ para o problema relaxado $\mathcal{P}_k$, respectivamente. Esses nós são inseridos na lista $\mathcal{W}$ e um novo nó é escolhido para ser resolvido. O algoritmo é executado enquanto houver nós na árvore B&B a serem processados ou até que um limite de tempo ($limite_{tempo}$) seja excedido.

No FTSP, a relaxação linear é obtida desconsiderando a integralidade das variáveis y_i e x_{ij}, e uma solução $\mathcal{S}_k$ pode ser representada por todos os valores dessas variáveis após $\mathcal{P}_k$ ser resolvido na otimalidade. Além disso, o modelo matemático apresentado na Seção 2.1 pode ser totalmente gerado apenas para instâncias de tamanho muito pequeno. Para instâncias de dimensão grande, o número de cortes de conectividade e desigualdades arredondadas nas visitas familiares aumenta exponencialmente e sua enumeração completa é impraticável. Para superar essa limitação, é gerado um modelo no nó raiz sem as desigualdades (2.7) e (2.10) e aplicam-se procedimentos de separação para identificar desigualdades violadas sempre que uma solução é encontrada na árvore B&B. As desigualdades violadas identificadas são adicionadas dinamicamente ao longo da árvore do espaço de busca, garantindo a viabilidade da solução e fortalecendo a relaxação linear atual. Todos os procedimentos de separação são detalhados posteriormente na Subseção 3.1.1.1.

Uma versão modificada do algoritmo B&C tradicional foi projetada para abordar de forma exata o FTSP. O Algoritmo 1 introduz duas etapas extras (veja as linhas 5 e 12) que são usadas para atualizar a melhor solução ($\mathcal{S}_{Melhor}$) e uma lista global de clientes ($\mathcal{T}$), respectivamente. Este algoritmo descreve a frente B&C e é baseado na decomposição do FTSP em dois subproblemas sequenciais. No primeiro subproblema, selecionam-se potenciais clientes que serão visitados na rota, enquanto no segundo subproblema, esses clientes devem ser roteados de forma a gerar uma solução viável para o problema original. Seguindo essa ideia, o algoritmo B&C identifica grupos promissores de clientes usando as soluções $\mathcal{S}_k$ obtidas através da exploração da árvore B&B. Esses grupos de clientes são adicionados à lista $\mathcal{T}$ e devem ser otimizados na frente de busca local para gerar uma solução $\mathcal{S}_{LS}$ para o FTSP. Na Subseção 3.1.1.2 são apresentadas algumas estratégias desenvolvidas para atribuir clientes $\mathcal{T}$ e na Seção 3.5 é detalhado como esta lista de clientes

Algoritmo 1 Frente *Branch-and-Cut*

função B&C($limite_{tempo}$)
  Definir $\mathcal{W} \leftarrow \{\mathcal{U}_0\}$, onde $\mathcal{U}_0$ é o nó raiz da árvore B&B dado por (2.2)-(2.6) $\cup$ (2.8)-(2.9) (problema relaxado $\mathcal{P}_0$).
  Definir $\mathcal{T} \leftarrow \emptyset$, $\mathcal{S}_{Melhor} \leftarrow \mathcal{S}_{LS} \leftarrow \emptyset$, $z(\mathcal{S}_{Melhor}) \leftarrow z(\mathcal{S}_{LS}) \leftarrow \infty$, e $time \leftarrow 0$.
  enquanto $\mathcal{W} \neq \emptyset$ e $time < limite_{tempo}$ **faça**
    se $z(\mathcal{S}_{LS}) < z(\mathcal{S}_{Melhor})$ **então**
      Definir $\mathcal{S}_{Melhor} \leftarrow \mathcal{S}_{LS}$.
    Selecionar o nó $\mathcal{U}_k$ (problema relaxado $\mathcal{P}_k$) em $\mathcal{W}$ e remover este da lista.
    Resolver $\mathcal{P}_k$.
    se $\mathcal{P}_k$ for inviável **então**
      Ir para a etapa 4 (poda por inviabilidade).
    Seja $\mathcal{S}_k$ a solução ótima de $\mathcal{P}_k$ e $z(\mathcal{S}_k)$ seu valor de função objetivo.
    Atualizar lista $\mathcal{T}$ usando $\mathcal{S}_k$.
    se $z(\mathcal{S}_k) \geq z(\mathcal{S}_{Melhor})$ **então**
      Ir para a etapa 4 (poda por limitante).
    se $\mathcal{S}_k$ for uma solução viável para o FTSP **então**
      Definir $\mathcal{S}_{Melhor} \leftarrow \mathcal{S}_k$ e ir para a etapa 4 (poda por integralidade).
    se passo de planos de corte **então**
      Fortalecer $\mathcal{P}_k$ adicionando desigualdades violadas e ir para o passo 8.
    se passo de ramificação **então**
      Para $\mathcal{P}_k$, construir dois problemas lineares $\mathcal{P}_{k_1}$ e $\mathcal{P}_{k_2}$ com regiões menores cuja união não contém $\mathcal{S}_k$.
      Adicionar os novos nós correspondentes $\mathcal{U}_{k_1}$ e $\mathcal{U}_{k_2}$ em $\mathcal{W}$ e ir para a etapa 4.
  fim-enquanto
  retornar $\mathcal{S}_{Melhor}$.
fim função

é processada na frente de busca local.

3.1.1.1 Procedimentos de Separação

Ambas as desigualdades CC e RFV apresentadas na Seção 2.1 podem ser usadas para evitar sub-rotas no FTSP. A principal diferença entre elas está na criação da partição S e $S' = (\{0\} \cup N) \backslash S$, ou seja, como define-se o *cut-set* $[S', S]$. Em particular, para identificar desigualdades RFV que são violadas na solução atual $\mathcal{S}^* = (x^*, y^*)$, deve-se construir um conjunto S contendo pelo menos $n_l - v_l + 1$ membros de uma dada família l, garantindo que não seja possível obter uma solução viável usando apenas o conjunto S'. Além disso, para encontrar desigualdades CC violadas, deve-se criar um conjunto S com um ou mais clientes sendo visitados na solução $\mathcal{S}^*$, o que implica que o *cut-set* $[S', S]$ deve ser cruzado pelo menos uma vez para manter a conectividade da rota. Basicamente, dado que uma partição é gerada, o que determina se temos uma desigualdade CC ou RFV é o número de membros de cada família l no conjunto S. Assim, para todos os procedimentos que serão

descritos a seguir, inicialmente são identificados os conjuntos S e S', sem focar em um tipo específico de desigualdade, e só então verifica-se se uma desigualdade CC ou RFV foi violada.

Bernardino e Paias (2018) propuseram dois procedimentos de separação para identificar as desigualdades CC e RFV violadas. Esses procedimentos também são aplicados no P-B&C desenvolvido para o FTSP e funcionam da seguinte forma. O primeiro procedimento encontra heuristicamente um conjunto de componentes conectados $\{C_0, C_1, \cdots, C_p\}$ induzidos pela solução fracionária, $\mathcal{S}^*$, para os quais C_0 contém o nó 0 por padrão. Essencialmente, cada componente conectado inclui todos os nós i e j relacionados entre si, de modo que $x^*_{ij} > 0$. Usando esses componentes é possível determinar um total de $p+1$ desigualdades violadas. Inicialmente, assume-se $S' = C_0$ e $S = N\backslash(S'\backslash\{0\})$ para a primeira inequação, pois C_0 contém o depósito. Então, para os componentes p restantes, define-se $S = C_v$ e $S' = (\{0\} \cup N)\backslash S$, para todo $v = 1, \cdots, P$.

A heurística de separação descrita anteriormente às vezes falha em identificar desigualdades violadas existentes em $\mathcal{S}^*$. O segundo procedimento apresentado por (BERNARDINO; PAIAS, 2018) separa de forma exata as desigualdades CC (ou seja, é capaz de identificar todas as desigualdades violadas existentes em $\mathcal{S}^*$, se houver) e RFV heuristicamente resolvendo uma série de problemas de fluxo máximo como segue. Para cada cliente k tal que $y^*_k > 0$, o fluxo máximo (m) do nó 0 para o nó k é calculado usando o grafo $G^* = (\{0\} \cup N, E^*)$, onde a capacidade do arco é dada pelo valor das variáveis x^*_{ij} em $\mathcal{S}^*$. Se $m < y^*_k$, uma desigualdade violada é identificada, e os conjuntos S' e S são determinados tal que S' é o conjunto de nós alcançáveis a partir do nó 0 no grafo residual associado ao fluxo máximo.

Em seus experimentos computacionais, os autores começaram aplicando a heurística de separação baseada nos componentes conectados e, quando não conseguiram identificar nenhuma desigualdade violada, aplicaram o algoritmo baseado no problema de fluxo máximo. A heurística de separação foi utilizada para acelerar o processo de separação, uma vez que resolver uma série de problemas de fluxo máximo pode ser demorado, embora esses problemas possam ser resolvidos usando algoritmos de tempo polinomial. Os autores adicionaram um máximo de 20 de desigualdades violadas para cada problema relaxado (etapa dos planos de corte).

Além disso, para este trabalho, foi desenvolvido um algoritmo simples de Busca Tabu (TS) (GLOVER, 1986) para gerar mais desigualdades violadas sem aumentar significativamente o tempo computacional necessário para separar os cortes. Nesta implementação, primeiro todos os componentes conectados foram identificados. Se o número de componentes, ou seja, o número de desigualdades violadas, for menor que 25, busca-se encontrar mais desigualdades aplicando o algoritmo TS, completando as desigualdades restantes. O parâmetro de desigualdades violadas = 25 foi definido através de extensos testes com-

putacionais. Finalmente, se ambos os procedimentos falharem, então pode-se recorrer à resolução de uma série de problemas de fluxo máximo.

O algoritmo TS utilizado é inspirado no procedimento desenvolvido por (AUGERAT *et al.*, 1998) para separar cortes para o problema de roteamento de veículos capacitado. O algoritmo começa construindo dois conjuntos disjuntos $S = \{k\}$ e $S' = \{0\} \cup (N \backslash S)$ para algum cliente k tal que $y_k^* > 0$. Em seguida, aplica-se uma fase de expansão, movendo iterativamente os clientes de S' para S para a qual a função $f(S) = x(S', S) - y_k$ é minimizada. Esta função representa a diferença entre o lado esquerdo e o lado direito de (2.7) e um valor negativo de $f(S)$ indica uma desigualdade violada associada ao conjunto de corte $[S', S]$. A fase de expansão termina quando não há mais clientes em S' adjacentes, em G^*, a pelo menos um nó em S. Em seguida, aplicamos uma fase de troca para um número máximo de iterações para realocar clientes de S para S' e vice-versa. A cada iteração, verificamos a violação da função $f(S)$, armazenando a desigualdade violada para cada conjunto S identificado. Além disso, se um cliente v for adicionado (removido) a (de) S em qualquer iteração, o movimento reverso é definido tabu durante um número de iterações (comprimento da lista tabu), evitando trocas repetidas em iterações consecutivas.

3.1.1.2 Estratégias de atribuição de clientes

Foram desenvolvidas duas estratégias de atribuição de clientes que podem ser incorporadas à frente B&C para atualizar a lista $\mathcal{T}$. Ambas as estratégias usam os valores das variáveis y_i obtidos da solução $\mathcal{S}_k$ após cada nó da árvore B&B (problema $\mathcal{P}_k$) ter sido resolvido na otimalidade. A primeira estratégia só é aplicada quando $\mathcal{S}_k$ é uma solução inteira. Essa estratégia basicamente adiciona à lista $\mathcal{T}$ todos os clientes visitados na solução atual, ou seja, todos os clientes i tais que $y_i = 1$ em $\mathcal{S}_k$. Vale ressaltar que o número de visitas para cada família é diretamente satisfeito nesta estratégia devido às restrições (2.6) e às condições de integralidade.

Na primeira estratégia, assume-se a suposição de integralidade para todas as variáveis na solução $\mathcal{S}_k$. Esta suposição não implica necessariamente que $\mathcal{S}_k$ seja viável para FTSP uma vez que a solução para o problema relaxado $\mathcal{P}_k$ pode conter sub-rotas sob as variáveis de roteamento x_{ij}. Isso ocorre porque as desigualdades (2.7) e (2.10) foram inicialmente removidas do modelo no nó raiz da árvore B&B. Por outro lado, a solução $\mathcal{S}_k$ pode ser uma rota sub-ótima viável para FTSP. Essas observações justificam o desenvolvimento da primeira estratégia. Isto é, o procedimento pode ser aplicado para recuperar a viabilidade de uma solução, bem como garantir a otimalidade de soluções sub-ótimas, pois os clientes enviados para a lista $\mathcal{T}$ serão roteados posteriormente na frente de busca local. Para exemplificar foi criada uma Subseção (veja subseção 3.1.1.3) com o mecanismo de viabilidade e melhoria.

A segunda estratégia leva em consideração as soluções de todos os nós, soluções fracionárias e inteiras, já otimizadas ao longo da árvore B&B para atualizar $\mathcal{T}$. Nesta estratégia, primeiro adiciona-se os valores das variáveis y_i de cada nova solução $\mathcal{S}_k$ em uma matriz cumulativa bidimensional (M) contendo $|N|$ linhas (indexadas de 1 a $|N|$) e $|N|$ colunas (indexadas de 1 a $|N|$), onde cada linha e cada coluna está relacionada a um cliente específico i. Para cada nova solução, atualiza-se a linha i de M somente se $y_i > 0$ na solução atual. Nesse caso, todas as colunas dessa linha são atualizadas adicionando os valores de suas variáveis de visita correspondentes.

Para exemplificar, considere uma instância com nove clientes ($N = \{1, \cdots, 9\}$), sendo 0 a origem da rota (depósito), três famílias ($\mathcal{L} = \{1, 2, 3\}$) com três membros cada um e duas visitas por família ($v_l = 2$, para todo $l \in \mathcal{L}$, e $V = 6$). Assim, temos: $F_1 = \{1, 2, 3\}$, $F_2 = \{4, 5, 6\}$ e $F_3 = \{7, 8, 9\}$. Para ilustrar como a matriz M é atualizada, considera-se que B&C resolveu dois nós ($\mathcal{U}_0$ e $\mathcal{U}_1$) na árvore B&B obtendo os seguintes valores para as variáveis de visitas. No primeiro problema relaxado ($\mathcal{P}_0$), assume-se que a solução $\mathcal{S}_0$ contém os valores $y_1 = 1, y_2 = 0,9, y_3 = 0,1, y_4 = 1, y_5 = 1, y_6 = 0, y_7 = 1, y_8 = 1$ e $y_9 = 0$, enquanto no segundo problema relaxado ($\mathcal{P}_1$), considera-se que $\mathcal{S}_1$ contém os valores $y_1 = 1, y_2 = 0,5, y_3 = 0,5, y_4 = 1, y_5 = 1, y_6 = 0, y_7 = 1, y_8 = 0,5$ e $y_9 = 0,5$.

A Figura 3.1 mostra a matriz M após cada atualização. Na Figura 3.1(a), são apresentados os valores da solução $\mathcal{S}_0$, em que as linhas correspondentes aos clientes 6 e 9 não foram atualizadas, pois esses clientes não foram visitados na solução de $\mathcal{P}_0$. A Figura 3.1(b) mostra os valores cumulativos obtidos somando os valores da solução $\mathcal{S}_1$ a M. Aqui, apenas a linha 6 não foi atualizada.

i/i	1	2	3	4	5	6	7	8	9
1	1,0	0,9	0,1	1,0	1,0	0,0	1,0	1,0	0,0
2	1,0	0,9	0,1	1,0	1,0	0,0	1,0	1,0	0,0
3	1,0	0,9	0,1	1,0	1,0	0,0	1,0	1,0	0,0
4	1,0	0,9	0,1	1,0	1,0	0,0	1,0	1,0	0,0
5	1,0	0,9	0,1	1,0	1,0	0,0	1,0	1,0	0,0
6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
7	1,0	0,9	0,1	1,0	1,0	0,0	1,0	1,0	0,0
8	1,0	0,9	0,1	1,0	1,0	0,0	1,0	1,0	0,0
9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0

(a) Matriz M após uma atualização.

i/i	1	2	3	4	5	6	7	8	9
1	2,0	1,4	0,6	2,0	2,0	0,0	2,0	1,5	0,5
2	2,0	1,4	0,6	2,0	2,0	0,0	2,0	1,5	0,5
3	2,0	1,4	0,6	2,0	2,0	0,0	2,0	1,5	0,5
4	2,0	1,4	0,6	2,0	2,0	0,0	2,0	1,5	0,5
5	2,0	1,4	0,6	2,0	2,0	0,0	2,0	1,5	0,5
6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
7	2,0	1,4	0,6	2,0	2,0	0,0	2,0	1,5	0,5
8	2,0	1,4	0,6	2,0	2,0	0,0	2,0	1,5	0,5
9	1,0	0,5	0,5	1,0	1,0	0,0	1,0	0,5	0,5

(b) Matriz M após duas atualizações.

FIGURA 3.1 – Exemplo de atualização da Matriz M com duas soluções

Pode-se destacar duas vantagens de atualizar a matriz M seguindo este procedimento. Primeiro, temos na diagonal de M, destacada em cinza, o valor total acumulado de cada variável de visita y_i. Por exemplo, observe que o cliente 3 recebeu visitas $\mathcal{S}_0$ e $\mathcal{S}_1$, totalizando em $0,6$ $(0,1 + 0,5)$ unidades, enquanto o cliente 1 também recebeu duas visitas, mas com valor total acumulado igual a 2 $(1 + 1)$. Esse valor total acumulado permite identificar quais clientes estão recebendo o maior número de visitas nas soluções obtidas ao longo da árvore B&B.

A segunda vantagem está relacionada ao agrupamento de clientes sempre que um cliente específico recebe uma visita. É notável, olhando para todos os valores acumulados em cada linha, que determinados clientes estão fortemente associados uns aos outros, indicando que quando um cliente específico é visitado, esses clientes também tendem a ser visitados. Por exemplo, suponha que a linha associada ao cliente 1 é escolhida na Figura 3.1(b). Assim, é possível identificar grupos de clientes usando um critério guloso, selecionando os valores acumulados mais altos para cada família. Seguindo essa ideia, os clientes selecionados para atualizar a lista $\mathcal{T}$ serão 1 e 2, da família $l = 1$, 4 e 5, da família $l = 2$, e os clientes 7 e 8, da família $l = 3$.

Na implementação deste trabalho, foi aplicado um procedimento semelhante para adicionar um grupo potencial de clientes à lista $\mathcal{T}$. Para manter algum aspecto de diversidade, foi adotado um mecanismo de roleta de dois estágios, em vez de um critério guloso. Primeiramente, uma linha candidata (cliente i) é escolhida, observando todos os valores na diagonal de M, de acordo com o valor total acumulado de cada cliente. Linhas com valores mais altos são mais propensas a serem escolhidas. Em seguida, aplica-se o mecanismo da roleta novamente para selecionar v_l clientes para cada família l, obtendo um total de V clientes selecionados para atualizar $\mathcal{T}$. Este procedimento de duas etapas é aplicado após cada novo nó ter sido resolvido na árvore B&B e a matriz M ter sido atualizada. Adota-se também adicionar o cliente correspondente à linha escolhida i no grupo de clientes selecionados, mesmo que este cliente tenha um valor total acumulado menor entre todos os membros de sua família. Além disso, por razões de economia de memória computacional, o tamanho de $\mathcal{T}$, foi limitado, permitindo adicionar novos elementos somente se o tamanho da lista não exceder 10.000 grupos ativos de clientes.

3.1.1.3 Mecanismo de viabilidade e melhoria

O mecanismo de viabilidade e melhoria opera na estrutura de roteamento de qualquer solução inteira do problema, viável ou não. No exemplo dado pela Figura 3.2, tem-se 12 clientes agrupados em três famílias l diferentes, de tal forma que: i) a Família 1 é representada pelo conjunto F1 = {1,2,3,4}, ii) a Família 2 é representada pelo conjunto F2 = {5,6,7} e iii) Família 3 é representada pelo conjunto F3 = {8,9,10,11,12}. Além disso, cada família possui um número de visitas necessárias v_l, e estas visitas são distribuídas de modo que: $v1 = 3$, $v2 = 2$, $v3 = 3$.

Para os casos 1 e 2, a sequência ótima de visita aos clientes é determinada após a otimização do TSP na frente de busca local. O mecanismo tem dois papéis essenciais no processo de busca. Primeiro, ele recupera a viabilidade quando a solução contém uma rota desconectada, o que necessariamente acontece no caso 1. Note que neste caso existe subrota entre os nós da Família 1, e seu conjunto associado é F1 $\leftarrow \{1, 2, 3\} \cup \{0\}$, na Família

2 possui CC gerando o conjunto F2 $\leftarrow \{5,6\} \cup \{0\}$ e a na Família 3, F3 $\leftarrow \cup\{10,11,12\}$. Portanto, após aplicar o procedimento de busca local é obtida a rota ótima: $\{0 \rightarrow 2 \rightarrow 3 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 11 \rightarrow 12 \rightarrow 10 \rightarrow 0\}$. O Segundo mecanismo do processo de busca, consiste na re-otimização da rota da solução atual, como no caso 2. Note que neste caso a rota é viável, sem cortes de conectividade e contém as visitas requeridas em cada família, contudo, não é a rota ótima pois apresenta trajetos não otimizados. Após resolver o TSP com o mecanismo de melhoria da busca local minimizando a distância percorrida, obtemos a rota de custo ótimo: $\{0 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 10 \rightarrow 9 \rightarrow 8 \rightarrow 0\}$.

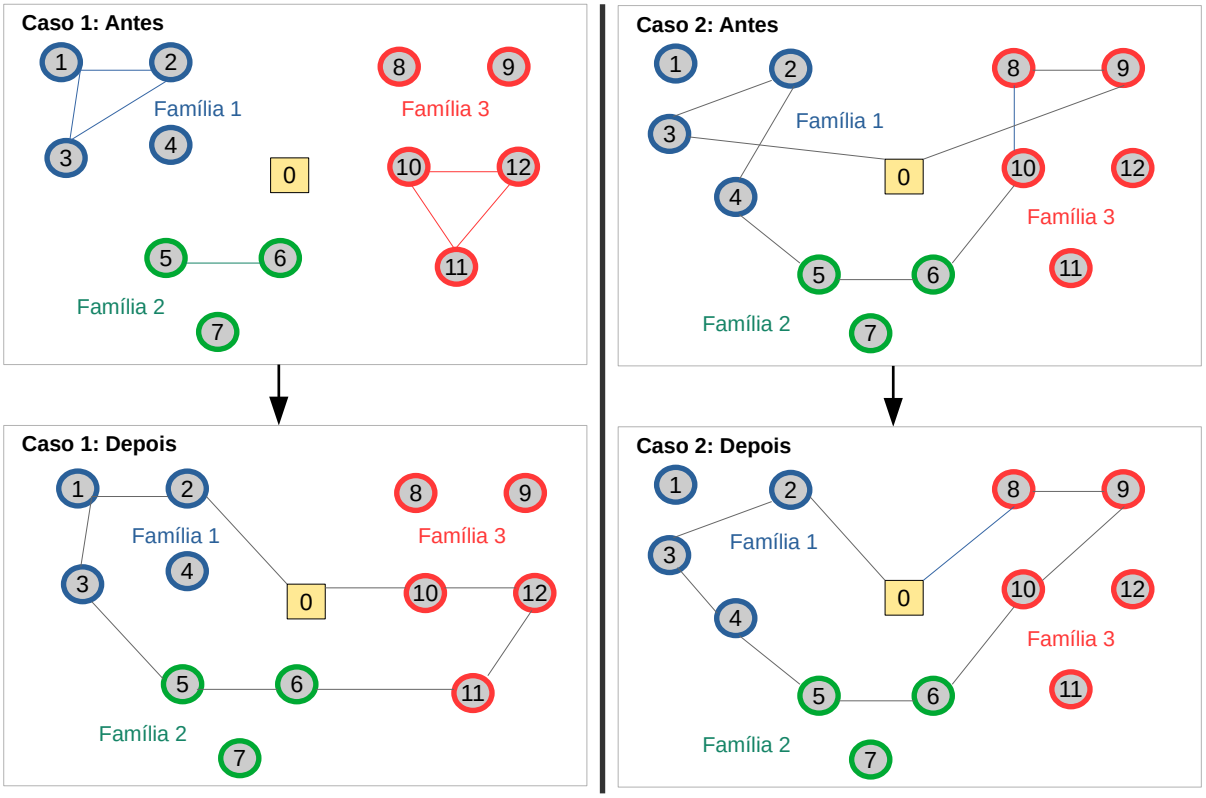

FIGURA 3.2 – Mecanismo de Viabilidade e Melhoria para o FTSP. Elaborado pela autora.

3.1.2 Frente de Busca Local

A frente de busca local é usada para gerar soluções de roteamento ótimas para o FTSP dado um grupo de clientes. O Algoritmo 2 apresenta esta frente, no início, foi definido o critério de parada (*executar*), um contador (*contagem*) e um conjunto vazio de clientes C_N. Em seguida, o algoritmo é executado iterativamente, desde que o B&C não tenha sido abortado. A cada iteração, verifica-se se a lista $\mathcal{T}$ não contém nenhum grupo de clientes a serem roteados ou se *contagem* é igual a um parâmetro chamado $modificar$, indicando que a solução mais conhecida deve ser modificada. Em ambos os casos, é identificado um conjunto de clientes sendo atendidos (C_N) na solução atual da frente B&C e é trocado aleatoriamente um cliente não visitado por um cliente visitado em uma

determinada família escolhida aleatoriamente. Caso contrário, é obtido um conjunto C_N armazenado em $\mathcal{T}$ e este é removido da lista. A seguir, é resolvido um TSP no subgrafo definido por $C_N \cup \{0\}$ obtendo uma solução $\mathcal{S}_{LS}$ para o FTSP. Se $\mathcal{S}_{LS}$ for melhor que $\mathcal{S}_{Melhor}$, $z(\mathcal{S}_{LS}) < z(\mathcal{S}_{Melhor})$, é definida esta nova solução para a frente B&C. Por fim, é verificado se o B&C está sendo executado e o parâmetro do critério de parada é atualizado (*executar*: verdadeiro ou falso).

Algoritmo 2 - Frente de Busca Local

```
1: função LS(modificar)
2:   Definir executar ← verdadeiro, contagem ← 0, e C_N ← ∅.
3:   enquanto executar = verdadeiro faça
4:     se T estiver vazia ou contagem = modificar então
5:       Obter C_N usando S_Melhor para a frente B&C.
6:       Modificar C_N.
7:       Definir contagem ← 0.
8:     senão
9:       Obter C_N de T e remover este da lista.
10:      Definir contagem ← contagem + 1.
11:    se C_N não foi otimizado antes então
12:      Resolver o TSP usando C_N ∪ {0}, obtendo S_LS, e marcar C_N como otimizado.
13:      se z(S_LS) < z(S_Melhor) então
14:        Definir S_LS para a frente B&C.
15:    Atualizar executar.
16:  fim-enquanto
17: fim função
```

Na frente de Busca Local, além dos grupos de clientes gerados pela frente B&C (lista $\mathcal{T}$), foi decidido implementar a etapa de modificação (linha 6), permitindo alguma perturbação em $\mathcal{S}_{Melhor}$ e garantindo que um conjunto C_N seja criado quando $\mathcal{T}$ estiver vazio e uma solução viável for conhecida. Para isso, define-se $modificar = 30$ nos experimentos computacionais (veja a Seção 4.4). Este valor de parâmetro foi adotado após extensos testes preliminares. Além disso, o procedimento B&C descrito em (PADBERG; RINALDI, 1991) foi aplicado para resolver cada TSP, gerando uma rota ótima para cada conjunto C_N. A vantagem de usar um procedimento exato é poder abortar a otimização do TSP quando o valor do limite inferior for maior que o melhor limitante superior ($z(\mathcal{S}_{Melhor})$) para o FTSP.

3.1.3 Esquema geral do P-B&C

A Figura 3.3 apresenta o esquema geral que ilustra o algoritmo P-B&C. Ambas as frentes B&C e LS passam a funcionar simultaneamente, alimentando-se mutuamente sempre que um novo grupo potencial de clientes é identificado ou uma nova melhor solução é alcançada, respectivamente. Essa abordagem não sequencial permite a obtenção de boas

soluções mais rapidamente na frente de busca local, permitindo a poda antecipada de nós ativos na árvore B&B, potencialmente reduzindo o tempo total necessário para resolver o problema. No esquema apresentado, o B&C controla o critério de parada, finalizando a otimização do problema quando não houver mais nós no B&B para processar ou quando um limite de tempo for excedido, retornando uma solução ótima caso todos os nós tenham sido resolvidos na otimalidade.

3.2 Algoritmo Genético de Chaves Aleatórias Viciadas

Os algoritmos genéticos são metaheurísticas inspiradas no processo de seleção natural ou biologia evolutiva. Neste sentido, caracterizam-se como algoritmos evolutivos, e portanto, desenvolvem a população de indivíduos (soluções) através da aplicação de técnicas de simulação de processos do sistema natural de evolução ou princípio da sobrevivência dos indivíduos mais aptos (princípio de Darwin). Os princípios da hereditariedade são aplicados iterativamente às soluções por um conjunto de operadores estocásticos, sendo:

- Seleção: garante que os indivíduos mais aptos sejam selecionados/replicados, reproduzam-se e passem suas características aos seus descendentes;
- Cruzamento: operador que faz combinações entre duas soluções distintas para obter novas soluções de boa qualidade;
- Mutação: Acrescenta diversidade à busca através da perturbação de um indivíduo candidato, gerando indivíduos distintos na população.

O ciclo evolutivo tem início com a geração de uma população formada por um conjunto de indivíduos que representam as soluções codificadas de um problema. Durante o processo evolutivo, cada indivíduo desta população é avaliado e classificado de acordo com a sua aptidão. Após a classificação dos indivíduos, a população é ordenada de acordo com a aptidão de cada indivíduo e a nova geração é formada. O passo seguinte consiste na seleção dos indivíduos mais aptos para que então possam se reproduzir, passar suas características genéticas através do cruzamento ou sofrer modificações em suas características através de mutações gerando indivíduos distintos na população. Este processo é denominado de reprodução, e se repete até que uma condição seja satisfeita. Essa condição pode ser: até que seja encontrada uma solução satisfatória, por exemplo. O ciclo evolutivo do algoritmo genético é esquematizado na Figura 3.4, sendo:

- População: conjunto de indivíduos que representam as soluções codificadas de um determinado problema;

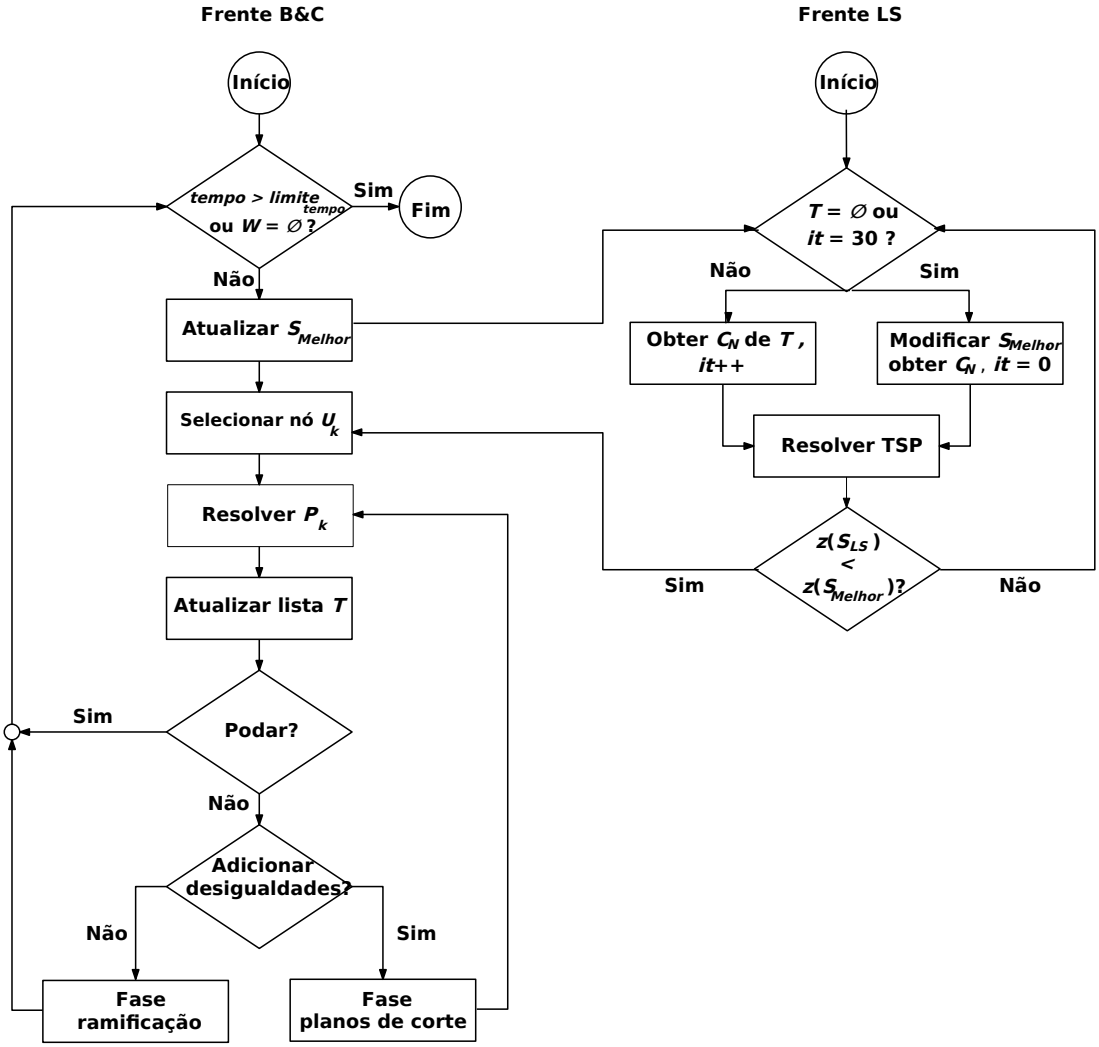

FIGURA 3.3 – Esquema geral do P-B&C.

- Indivíduo: um elemento da população, caracterizado pela sua aptidão;
- Geração: população durante as iterações do processo evolutivo do AG.

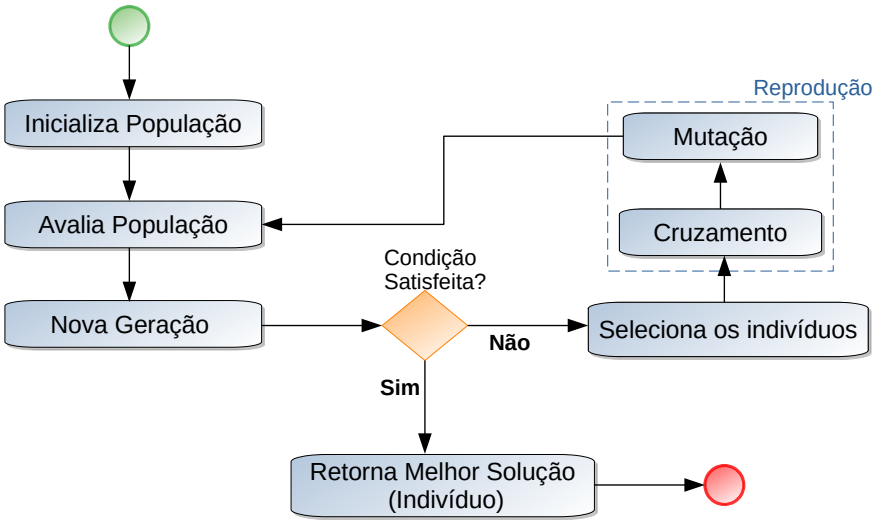

FIGURA 3.4 – Ciclo evolutivo do Algoritmo Genético (adaptado de Gonçalves e Resende (2011)).

Com o objetivo de solucionar problemas complexos de sequenciamento em otimização combinatória, foram introduzidos na literatura os algoritmos genéticos de chaves aleatórias, do inglês *Random-keys Genetic Algorithm* (RKGA) por Bean (1994). No RKGA a solução é codificada como um vetor de n elementos de números reais gerados aleatoriamente, denominados de chaves aleatórias, no intervalo [0,1). Ao decorrer do tempo, o RKGA foi estendido de diferentes maneiras para lidar com uma ampla classe de problemas de otimização combinatória. Um exemplo de variante do RKGA eficaz e amplamente estudada na literatura é o *Biased Random-Key Genetic Algorithm* (BRKGA), ou algoritmo genético de chaves aleatórias viciadas.

Introduzido por Gonçalves e Resende (2011), no BRKGA, o cruzamento é operado segundo um mecanismo tendencioso para selecionar os indivíduos da população. Tanto no RKGA quanto no BRKGA, o cruzamento é aplicado pelo *parametrized uniform crossover* (PUX) (SPEARS; JONG, 1991). A diferença entre o BRKGA e o RKGA se evidencia na forma de selecionar os pares para o cruzamento. Enquanto no RKGA os pares são selecionados aleatoriamente dentre a população disponível, no BRKGA um dos indivíduos pertence à partição da população atual com maior aptidão, selecionado aleatoriamente, e o outro pertence ao grupo dos indivíduos menos aptos.

No BRKGA, assim como no RKGA, uma solução viável é obtida por meio da decodificação de uma solução de chaves aleatórias através de um algoritmo determinístico

denominado *decoder* (BEAN, 1994). Um exemplo proposto por (BEAN, 1994) é a decodificação dos vetores de chaves aleatórias com base na ordenação numérica crescente para produzir a sequência dos cromossomos, conforme Figura 3.5. Para tanto, a representação cromossômica do decodificador deve ser devidamente apropriada, garantindo dessa forma um *framework* capaz de solucionar problemas de otimização combinatória de diferentes naturezas (TOSO; RESENDE, 2015).

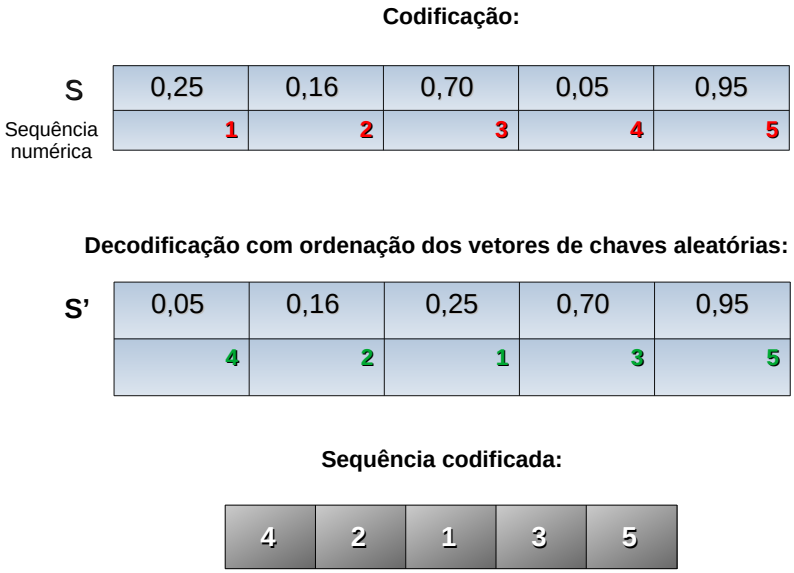

FIGURA 3.5 – Codificação e Decodificação de uma Solução (adaptado de Bean (1994)).

No BRKGA, definem-se gerações como a evolução da população obtida por meio das iterações. A população inicial P é formada por p cromossomos (indivíduos) gerados aleatoriamente. Na primeira geração, são avaliados cada um dos cromossomos através da função objetivo associada ao problema, e partir de então classificam-se os indivíduos em dois grupos com base em suas aptidões. O primeiro grupo, $P_e \in P$, é composto pelos indivíduos que apresentam os melhores valores de aptidão segundo a função objetivo, classificados como população elite. No segundo grupo, o conjunto $P - P_e$, são remanejados os indivíduos com menor aptidão, classificados como população não-elite. A avaliação da aptidão dos indivíduos é concebida com base na função objetivo do problema (GOLDBERG; HOLLAND, 1988; LACERDA; CARVALHO, 1999; GONÇALVES; RESENDE, 2011). No problema abordado nesse trabalho, a função objetivo é de minimização, portanto os indivíduos que apresentam menores custos são considerados mais aptos.

A Figura 3.6 apresenta o processo evolutivo do BRKGA. A cada geração, uma nova população é criada copiando a partição elite P_e da geração atual e inserindo dois novos conjuntos: o conjunto mutante P_m e o conjunto descendentes P_c. Os mutantes são gerados iguais aos indivíduos da população inicial. O conjunto de descendentes P_c, por outro lado,

é formado pelo cruzamento uniforme parametrizado (PUX) (SPEARS; JONG, 1991) entre os pais da elite e os não-elite. Portanto, a nova população é formada por três conjuntos: *i)* partição elite P_e, *ii)* partição mutante P_m e *iii)* conjunto descendentes P_c.

No PUX, considera-se o parâmetro ρ_e a probabilidade do filho herdar o gene (componente do vetor) do pai elite e 1-ρ_e do pai não-elite. Um número aleatório é gerado para cada chave aleatória. Caso este número for inferior ao ρ_e, os filhos recebem a chave do cromossomo elite, caso contrário recebem a chave do cromossomo não-elite. A Figura 3.7 apresenta um exemplo do funcionamento do PUX.

O *decoder* é aplicado a cada indivíduo, mapeando o cromossomo em uma solução do problema. Os indivíduos são avaliados e ordenados de acordo com a aptidão. O processo evolutivo é repetido até que algum critério de parada seja satisfeito. O critério de parada do BRKGA geralmente é o número máximo de gerações (max_{gen}). Os outros parâmetros do BRKGA são o tamanho da população $|P| = p$, a porcentagem de indivíduos na partição elite p_e ($|P_e| = p \times p_e$), e a porcentagem de indivíduos mutantes p_m ($|P_m| = p \times p_m$). O pseudocódigo do BRKGA é apresentado pelo Algoritmo 3.

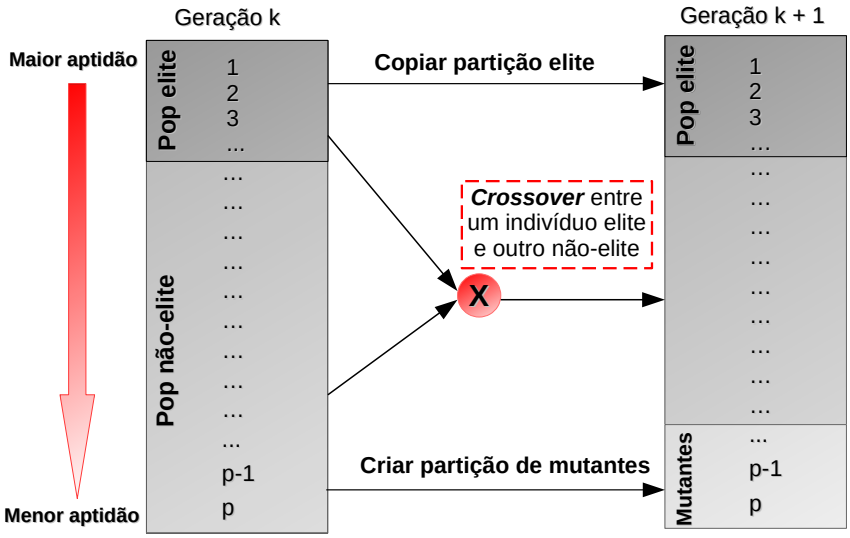

FIGURA 3.6 – Evolução de uma população no BRKGA (adaptado de Gonçalves e Resende (2011)).

3.3 Algoritmo *Q-Learning* (QL)

A inteligência artificial (IA) constitui um campo da computação que estuda o desenvolvimento de máquinas capazes de reproduzir a inteligência humana (RUSSELL; NORVIG,

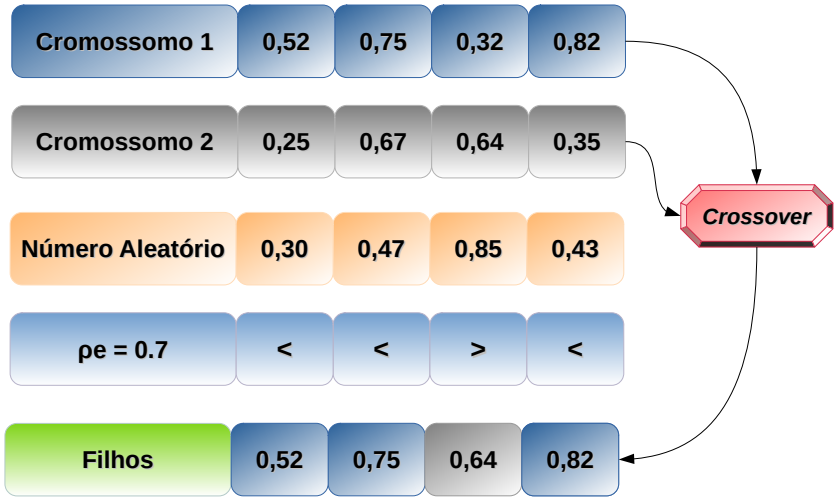

FIGURA 3.7 – *Parametrized Uniform Crossover* (adaptado de Gonçalves e Resende (2011)).

Algoritmo 3 BRKGA

função BRKGA(p,p_e,p_m, max_{gen})
    Gerar população Pop com p vetores de Chaves Aleatórias
    Avaliar e ordenar os indivíduos de acordo com a aptidão
    Armazenar a melhor solução sol^*
    Para max_{gen} **repetir**
        Criar partição elite P_e ($|P_e| = p \times p_e$) e copiar para Pop^+
        Gerar filhos P_c usando o PUX e copiar para Pop^+
        Criar partição de mutantes P_m ($|P_m| = p \times p_m$) e copiar para Pop^+
        Pop = Pop^+
        Avaliar e ordenar os indivíduos de acordo com a aptidão
        Armazenar a melhor solução sol^*
    fim-para
    retornar melhor solução sol^*
fim função

2010). Aprendizado de Máquinas (ML) é uma área da IA que estuda a construção de sistemas que, a partir de dados, geram modelos de aprendizado (MONARD; BARANAUSKAS, 2003). Aprendizagem por Reforço (RL) é um modelo de aprendizado de AM, onde um agente treina a execução de tarefas com um sistema de atribuição de recompensas para melhorar o próprio desempenho em tarefas futuras (RUSSELL; NORVIG, 2010). O *Q-Learning*, por sua vez, é um algoritmo de aprendizagem por reforço, que busca aprender uma política que maximize a recompensa total (WATKINS; DAYAN, 1992).

Segundo Russell e Norvig (2004), a inteligência artificial confere ordem, método, estrutura e autonomia às tarefas intelectuais, caracterizando-se assim, como um campo universal. A IA é um campo da ciência, cuja finalidade é estudar a construção de entidades inteligentes, através do desenvolvimento de sistemas capazes de aprender, raciocinar e até mesmo solucionar problemas de alta complexidade. Em outras palavras, a IA busca reproduzir o entendimento da inteligência humana na computação (RUSSELL; NORVIG, 2010). Os primeiros trabalhos de IA foram elaborados por Mcculloch e Pitts (1943), nos quais evidenciavam os neurônios artificiais, que sucederiam no desenvolvimento das máquinas capazes de aprender. Desde então, é uma área amplamente estudada e em desenvolvimento no mundo inteiro.

O Aprendizado de Máquinas ou *Machine Learning* (ML), é um paradigma da Inteligência Artificial, composto por algoritmos de aprendizagem e abordagens estatísticas capazes de possibilitar que os computadores aprendam novas tarefas através de dados, assim como o cérebro humano (BISHOP, 2006). Seu objetivo consiste no desenvolvimento de sistemas especializados, treinados a partir de grandes volumes de dados e capacitados para prever tendências, padrões e resultados (MONARD; BARANAUSKAS, 2003).

Os métodos de Aprendizagem por Reforço consistem em um agente adquirir aprendizado de forma iterativa através de tentativas e erros sobre um ambiente dinâmico (SUTTON; BARTO, 2018). Portanto, a experiência do agente obtida através das interações fornece o aprendizado necessário para que o mesmo tome as melhores decisões. As melhores decisões, neste contexto, são aquelas que maximizam as recompensas ou, de outra perspectiva, minimizem os riscos. O objetivo destes métodos é garantir que o agente defina uma política de ações que visem maximizar os ganhos adquiridos no decorrer do aprendizado, ou seja, o agente deve maximizar o desempenho geral dado um comportamento ideal (ações) e um *feedback* das recompensas.

O problema (ou tarefa) de aprendizagem por reforço deve ter o seu domínio modelado como um processo de decisão de Markov. Neste sentido, trata-se de um problema de decisão de Markov (MDP - *Markov Decision Process*) (SUTTON; BARTO, 2018). Um MDP é um processo de decisão sequencial definido por um conjunto de cinco elementos (T, S, A, P, R) tal que: T é um conjunto discreto de instantes de tempo, S um conjunto finito de estados do sistema, A um conjunto finito de ações, P o modelo de transição dos

estados e R a função de recompensa.

A interação do processo é realizada da seguinte forma: no instante inicial (t), o agente seleciona uma ação (a), define um novo estado (s), recebe uma recompensa (r) e então o ciclo se reinicia no próximo instante $(t+1)$. O próximo estado s no instante $t+1$ é representado por s'. O agente interage com o ambiente em uma sequência discreta de tempo $t = 0,1,2,...,T$. Conforme representado pela Figura 3.8.

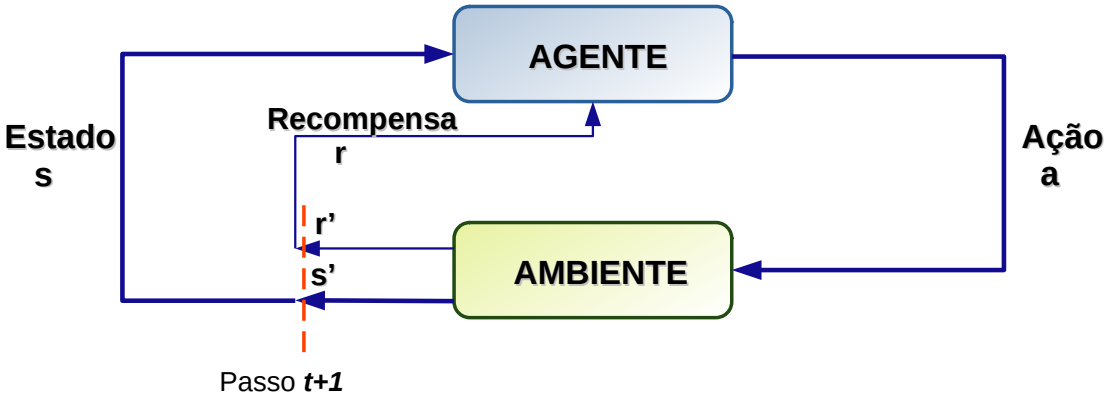

FIGURA 3.8 – Processo de interação agente-ambiente em AR (adaptado de Sutton e Barto (2018)).

Portanto, os componentes básicos da aprendizagem por reforço são: ambiente, agente, ação, estado, recompensa e objetivo. E a interação dos mesmos acontece da seguinte forma:

- Ambiente: trata-se de um ambiente dinâmico e parcialmente observável;
- Agente: o agente de aprendizagem aplica ações sobre o ambiente;
- Conjunto de ações (A): consideram-se todas as ações (a) possíveis de serem executadas;
- Estado (s): o estado é decorrente das ações executadas no ambiente;
- Conjunto de estados (S): considera-se todos os estados que descrevem o ambiente;
- Recompensa (r): a recompensa é o retorno do ambiente em consequência da ação escolhida pelo agente. O agente baseia as tomadas de decisões subsequentes no valor das recompensas obtidas no processo de aprendizagem.

- Política de controle (π): expressa pela função $\pi(s, a)$, mapeia estados em ações. Tem a função de definir o comportamento do agente para atingir o objetivo. Conforme o agente ganha experiência, a probabilidade de uma ação ser escolhida no estado s sofre alterações. Portanto, a busca pela política ideal expressa o processo de aprendizagem.

- Função Valor: a função valor é obtida através do mapeamento do par estado-ação, das recompensas atuais e futuras, e indica uma estimativa do ganho total durante o processo de aprendizagem partindo de s e abrangendo os estados seguintes. Há dois tipos de função valor: (i) Função valor-estado $V_(s)$ e (ii) Função valor-ação $Q(s, a)$.

A somatória das recompensas obtidas durante o processo de aprendizagem ou o retorno acumulado pode ser expressa pela expressão (3.1) (KAELBLING *et al.*, 1996):

$$r = \sum_{k=0}^{T} df^k r_{t+k}. \tag{3.1}$$

A taxa de desconto df determina o peso referente as recompensas, evitando que o valor de retorno não fique demasiado, diminuindo gradativamente os reforços conseguintes, tal que: $0 \leq df \leq 1$.

A função valor-estado $V(s)$ é expressa por:

$$V(s) = E\{R_t | S_t = s\}, \tag{3.2}$$

em que E representa a estimativa do retorno total esperado pelo agente seguindo a política π a partir de um estado $s_t = s$ no instante t.

O algoritmo *Q-Learning* (QL), consiste no método mais popular em Aprendizagem por Reforço. O QL é amplamente estudado por tratar-se de um algoritmo sem modelo (*model-free*) que estabelece de forma autônoma a política de ações interativamente (*off-policy)* (WATKINS; DAYAN, 1992) (ver Figura 3.9). O QL dispensa uma modelagem completa do ambiente para determinar qual política ótima aplicar, visto que é capacitado para realizar o aprendizado a partir das experiências obtidas. Desse modo, converge para valores ótimos de Q independente da política selecionada. A função Q expressa o valor agregado ao par estado-ação (s, a) e evidencia quão boa é a aplicação dessa ação na otimização do retorno. Em outras palavras, o Q representa a qualidade das ações escolhidas para obtenção das recompensas futuras.

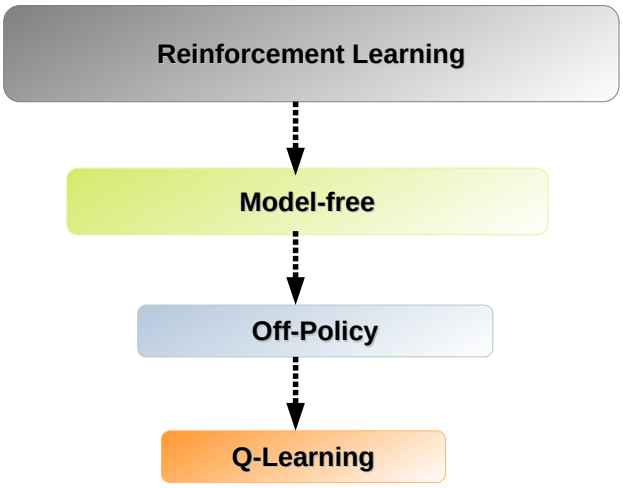

FIGURA 3.9 – Modelo *Q-Learning*. Elaborada pela autora.

A expressão matemática para atualizar a matriz dos Q-valores é definida pela Equação 3.3:

$$Q(s,a) \leftarrow Q(s,a) + lf \left[r + df * \max_{a'} Q(s',a') - Q(s,a) \right] \tag{3.3}$$

em que o objetivo a cada etapa consiste em maximizar o valor da função $Q(s,a)$ e, como consequência das atualizações dos pares estado-ação, aproxima-se da função estado-ação ótimo (JUNIOR *et al.*, 2007b). Nesta expressão s representa o estado atual, a a ação executada no estado atual, r a recompensa recebida após a aplicação de a em s, s' é o próximo estado, df o fator de desconto e lf o coeficiente de aprendizagem.

Como forma da assegurar o retorno total esperado, são estabelecidas estratégias que podem ser intensificação (*exploitation*) e diversificação (*exploration*). As visitas realizadas às boas ações são intensificadas e em paralelo são empregadas técnicas de diversificação para identificar novas ações que possibilitem a descoberta de políticas com maiores convergências, garantindo dessa maneira que todos os pares estado-ação sejam efetivamente explorados.

Uma boa estratégia que garante um equilíbrio entre a intensificação e diversificação é a política ϵ-*Greedy* (MNIH *et al.*, 2015). Esta pode aderir a duas estratégias, das quais uma executa a ação com maior valor em Q com probabilidade $1 - \epsilon$ e a outra seleciona uma ação de forma aleatória caso contrário.

O algoritmo *Q-Learning* apresenta boa convergência dado um procedimento de controle ótimo, visto que o processo de aprendizagem dos pares estado-ação é representado por uma tabela completa (*Q-table*) com o valor de cada par. A *Q-table* pode ser iniciada com seus valores zerados e os mesmos são atualizados incrementalmente durante o pro-

cesso de aprendizagem. Na *Q-table*, as linhas correspondem aos estados $S = s_1, s_2, \cdots, s_n$ e as colunas às ações $A = a_1, a_2, \cdots, a_n$.

O pseudocódigo do *Q-Learning* é apresentado pelo Algoritmo 4:

Algoritmo 4 *Q-Learning* com política ϵ-*Greedy*

```
1: função Q-LEARNING(lf, df, ϵ)
2:     Inicializar valores da Q-table
3:     para toda iteração repetir
4:         Escolher o estado inicial s
5:         para cada passo da iteração repetir
6:             Escolher uma ação a para A(s) usando a política ϵ-Greedy
7:             Observar a recompensa r e o novo estado s'
8:             Atualizar Q(s, a) usando a Equação 3.3
9:             s ← s'
10: fim função
```

Informações mais detalhadas sobre Aprendizagem por Reforço e *Q-learning* podem ser encontradas em Kaelbling *et al.* (1996) e Sutton e Barto (2018), respectivamente.

3.4 BRKGA-QL

O Método BRKGA-QL proposto por Chaves e Lorena (2021), integra o BRKGA (Seção 3.2) com o algoritmo *Q-Learning* (Seção 3.3). Portanto, de forma mais abrangente, há a integração de uma metaheurística (MH) com uma técnica de *Machine Learning* (ML). Ainda que as técnicas de MHs e ML apresentem diferentes finalidades, ambas executam tarefas em comum, como, por exemplo, a resolução de problemas de otimização combinatória (KARIMI-MAMAGHAN *et al.*, 2021). Integram-se as técnicas de ML e MHs como forma de aumentar a convergência para boas soluções.

Na literatura, pode ser observado tanto a aplicação de MHs em tarefas de ML (MH-*in*-ML) (XUE *et al.*, 2016) quanto a aplicação de técnicas de ML às MHs (ML-*in*-MH)(KARIMI-MAMAGHAN *et al.*, 2021). Nesta última, evidencia-se tanto a melhoria da autonomia quanto da inteligência do processo de pesquisa (QUEIROZ DOS SANTOS *et al.*, 2014). Neste sentido, técnicas de aprendizagem por reforço (RL), como, por exemplo, o *Q-Learning*, conseguem selecionar os operadores mais eficientes das MHs durante o processo de busca. Um panorama robusto sobre integrações de MHs e ML pode ser observado em Karimi-Mamaghan *et al.* (2021).

Destacam-se na Figura 3.10 os componentes que correspondem ao método aplicado neste trabalho (cor verde), uma vez que o método BRKGA-QL utiliza a integração ML-*in*-MH. Neste caso, o algoritmo *Q-Learning* tem como função a configuração dos parâmetros. Os usuários precisam configurar os parâmetros do BRKGA clássico em uma fase de ajuste. No entanto, ao estabelecer esses parâmetros, eles não são modificados ao longo do processo

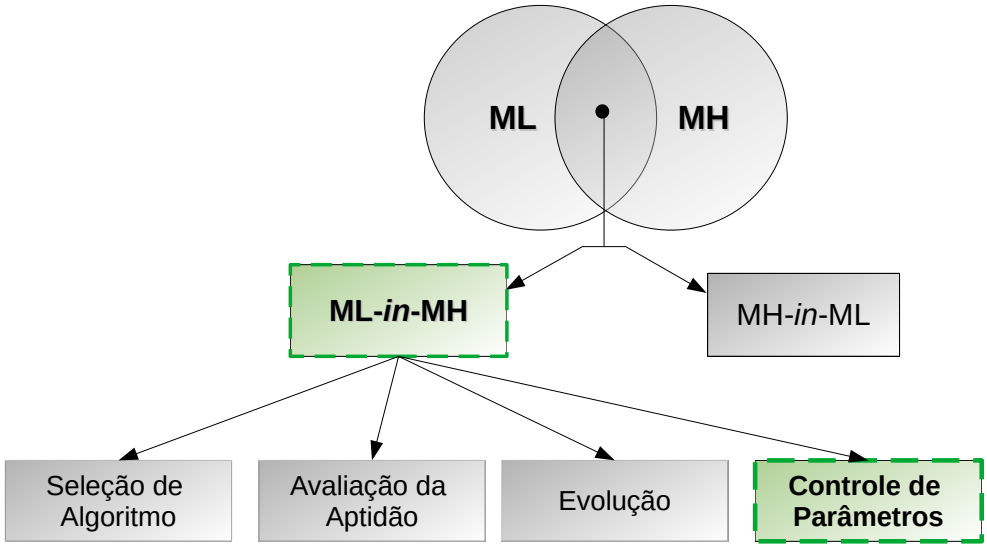

FIGURA 3.10 – Taxonomia para integração de ML e MH (adaptado de Karimi-Mamaghan *et al.* (2021)).

evolutivo do método. Portanto, essa configuração pode não ser a ideal, pois os valores dos parâmetros podem apresentar desempenhos diferentes de acordo com a etapa de busca, bem como a instância testada (CHAVES; LORENA, 2021).

A tarefa de definir os valores ótimos para cada parâmetro pode ser uma tarefa difícil. Neste caso, o *Q-Learning* é utilizado para superar esta limitação através de um controle *online* dos parâmetros. No método BRKGA-QL, o QL é aplicado para controlar os parâmetros do BRKGA durante o processo de busca. Os parâmetros do BRKGA controlados pelo QL são: tamanho da população p, porcentagem de indivíduos elites p_e e mutantes p_m, e a probabilidade de herdar uma chave de um pai elite ρ_e.

A Tabela 3.1 fornece a lista de ações para cada parâmetro , correspondente aos valores recomendados na literatura para cada parâmetro do BRKGA (PRASETYO *et al.*, 2015). O parâmetro p (tamanho da população) é determinado pela sequência de *Fibonacci*, a qual caracteriza-se por um comportamento de crescimento da população a uma taxa controlável (EVANS *et al.*, 2020). O *Q-Learning* é formulado de modo que para cada parâmetro existe apenas um único estado.

Em cada geração do BRKGA-QL, utilizamos o método Q-Learning, com a política ϵ-Greedy em Q, para escolher os valores de cada parâmetro. Como forma da assegurar o retorno total esperado, o agente precisa estabelecer estratégias que podem ser intensificação (*exploitation*) e diversificação (*exploration*). Dessa forma, as visitas realizadas às boas ações são intensificadas e em paralelo são empregadas técnicas de diversificação

TABELA 3.1 – Lista de ações dos parâmetros (CHAVES; LORENA, 2021).

Parâmetro (estado)	Lista de Ações
p	{233; 377; 610; 987; 1597; 2584}
p_e	{0,30; 0,25; 0,20; 0,15; 0,10}
p_m	{0,25; 0,20; 0,15; 0,10; 0,05}
ρ_e	{0,80; 0,75; 0,70; 0,65; 0,60; 0,55}

para identificar novas ações que possibilitem a descoberta de políticas com maiores convergências, garantindo dessa maneira que todos os pares estado-ação sejam efetivamente explorados. Portanto, a política ϵ-*Greedy* garante um equilíbrio entre a intensificação e diversificação (MNIH *et al.*, 2015). Um valor de ϵ próximo a 1 resulta em uma diversificação maior, enquanto um valor próximo a 0 resulta em maior intensificação. É necessário haver diversificação no início do processo evolutivo, para permitir que o agente obtenha informações suficientes sobre o ambiente. Uma vez que o agente tenha as informações necessárias, pode haver uma interação mais gulosa (intensificação). Usamos o decaimento exponencial ϵ-Greedy (HARIHARAN; PAAVAI, 2021) para obter esse comportamento. Assim, temos um decaimento gradual em ϵ de acordo com o processo evolutivo. O valor de ϵ é definido em cada geração por:

$$\epsilon = \epsilon_{inicial} * (\epsilon_{min})^{t/run_{time}} \tag{3.4}$$

onde $\epsilon_{inicial} = 1$, $\epsilon_{min} = 0.01$, t é o tempo de execução atual e run_{time} é o tempo de execução máximo.

O parâmetro lf, taxa de aprendizado, também é atualizado a cada geração de acordo com a Equação (3.5). Inicialmente, as informações recém-adquiridas têm prioridade mais alta. Assim, o algoritmo decrementa lf e tem maior prioridade para as informações existentes em *Q-Table*. O parâmetro df, taxa de desconto, é fixo e igual a 0,8. Portanto, buscamos uma recompensa maior e de longo prazo (SAMMA *et al.*, 2020). Essas estratégias de atualização dos parâmetros QL são uma novidade desenvolvida por nós para melhorar o BRKGA-QL.

$$lf = 1 - (0,9 * t/run_{time}) \tag{3.5}$$

No BRKGA-QL podem ser implementados diferentes decodificadores. Um parâmetro autoadaptativo é usado para definir qual decodificador será aplicado a um indivíduo. Assim, usamos uma chave aleatória na posição $n+1$ para definir qual decodificador realizará o mapeamento entre o espaço de chave aleatória e o espaço de solução do problema. Suponha que existam k decodificadores implementados, o intervalo [1,0] é dividido em

k partes iguais. Portanto, o decodificador id de um indivíduo é definido pelo intervalo correspondente ao valor da sua chave aleatória. Por exemplo, se $k = 2$, teremos dois intervalos $id_1 = [0{,}0\ ,\ 0{,}5]$ e $id_2 = [0{,}51\ ,\ 1]$. Neste caso, se a chave aleatória do indivíduo é 0,81 então será aplicado o id_2.

Os valores de Q são iniciados como zero (Seção 3.3). A cada geração do BRKGA-QL seleciona-se uma ação (valor) da *Q-table* para cada estado (parâmetro) através da política ϵ-*Greedy*. O processo evolutivo do BRKGA-QL é o mesmo do BRKGA clássico (subseção 3.2, Figura 3.6). A população P é particionada em dois grupos (elite e não-elite), e então a próxima geração é criada de modo que a partição elite é copiada para a próxima geração. Os indivíduos mutantes são gerados aleatoriamente e os filhos gerados pelo PUX. A nova população é novamente ordenada em ordem crescente de aptidão (ver exemplo de decodificação na Figura 3.5).

Verificamos se o algoritmo encontrou um indivíduo melhor na população P a partir das configurações dos parâmetros selecionados nesta geração. Se verdadeiro, podemos atualizar a melhor solução encontrada (sol^*), definir o valor da recompensa para 1 ($r = 1$) e atualizar Q para todos os pares de estado/ação selecionados usando a Equação (3.3).

O BRKGA-QL também integra um módulo de busca local para intensificar o processo de busca quando $r = 1$ (ou seja, foi encontrada uma nova melhor solução na população) ou em intervalos de gerações (por exemplo, a cada $n/5$ gerações). Chaves *et al.* (2018) propôs este componente que consiste em identificar comunidades dentro da partição Elite (P_e) usando o método Label Propagation (LP) (RAGHAVAN *et al.*, 2007). Assim, aplicamos a busca local apenas no melhor indivíduo de cada comunidade que ainda não foi explorado. Este processo ocorre em paralelo com o processo evolutivo e não interfere na população BRKGA.

O método de busca local utilizado no BRKGA-QL é o *Random Variable Neighborhood Search* (RVND) (PENNA *et al.*, 2013). Portanto, o usuário não precisa definir a melhor ordem das heurísticas de busca local. O RVND seleciona aleatoriamente qual heurística será aplicada em cada iteração. O método é executado enquanto alguma heurística melhora a função objetivo da solução atual.

Além do componente de busca local, também desenvolvemos um componente de perturbação em indivíduos semelhantes de uma comunidade. Com base no Algoritmo de Evolução Caótica (LI; PEI, 2019), geramos um indivíduo caótico usando o PUX para realizar o cruzamento entre indivíduos Elite e mutantes. Assim, intensificamos a busca em regiões promissoras do espaço de busca e aumentamos a diversificação da partição Elite. Esta também é uma novidade do BRKGA-QL desenvolvido neste trabalho.

A última etapa do algoritmo estabelece um parâmetro de parada na execução. O tempo computacional é o critério de parada do BRKGA-QL. O parâmetro run_{time} é

o único que precisa ser configurado pelo usuário no BRKGA-QL e consiste no tempo máximo de execução. Assim, quando o tempo computacional excede run_{time}, o algoritmo retorna a melhor solução (sol^*).

A Figura 3.11 esboça o fluxograma do BRKGA-QL e o Algoritmo 5 apresenta o seu pseudocódigo.

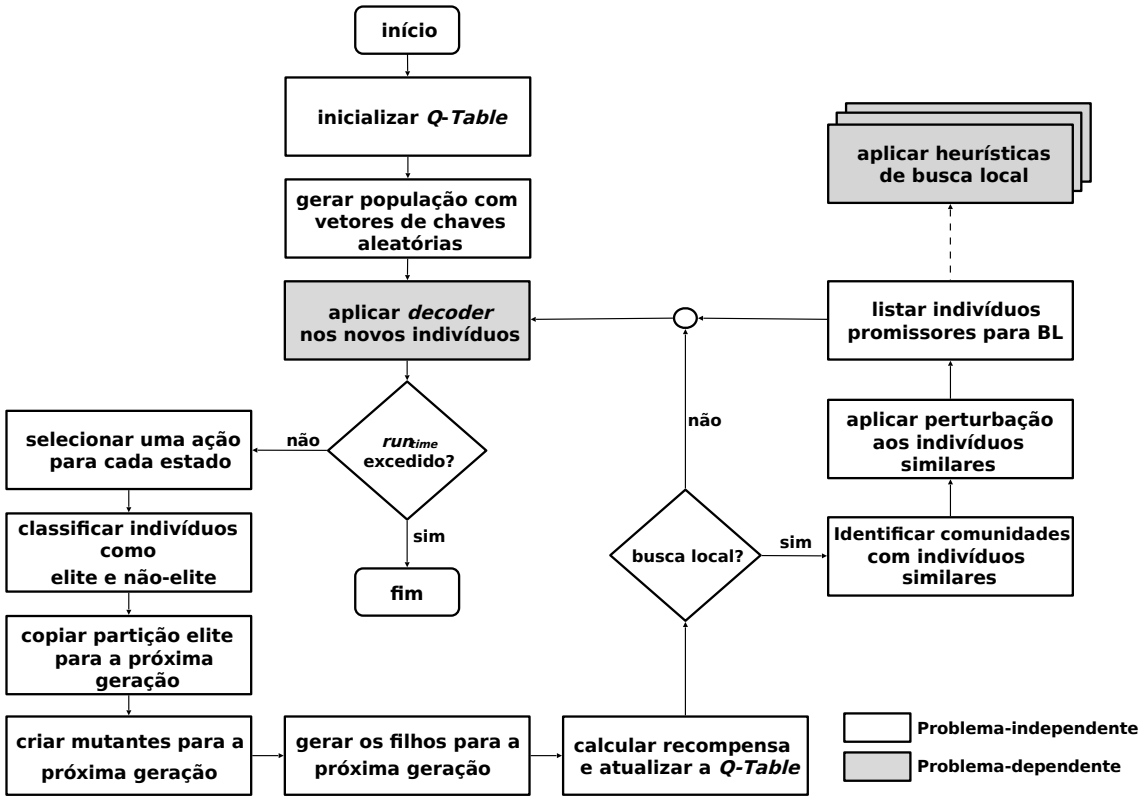

FIGURA 3.11 – Fluxograma BRKGA-QL (adaptado de Chaves e Lorena (2021)).

3.5 Componente de Busca Local

O procedimento de busca local realiza de forma sistemática a definição e construção de uma estrutura de vizinhança, ou seja, um conjunto de soluções viáveis com características similares à solução corrente. Portanto, a busca local é aplicada, a cada solução, para percorrer áreas promissoras, em busca de outra solução com valor melhor (menor, em problema de minimização) comparado à solução corrente. Caso uma solução melhor à solução corrente é encontrada, torna-se a nova solução corrente e o algoritmo continua a exploração. Se uma solução melhor não for encontrada na vizinhança, a solução corrente é um ótimo local da área considerada.

O uso de procedimentos de busca local pode melhorar o desempenho de metaheurísticas, por exemplo do BRKGA (CHAVES *et al.*, 2018). Segundo Blum e Roli (2008) a hibridi-

Algoritmo 5 *BRKGA* com *Q-Learning* (CHAVES; LORENA, 2021)

função BRKGA-QL(run_{time})
  Inicializar valores da $Q - table$.
  Gerar população P com p vetores de chaves aleatórias.
  Aplicar *decoder* para avaliar a ordenar P pela aptidão.
  Armazenar a melhor solução sol^*.
  para run_{time} seconds **repetir**
    Selecionar os valores para cada parâmetro usando ϵ-*Greedy* na $Q - table$.
    Gerar partição elite P_e $(p \times p_e)$
    Gerar partição mutante P_m $(p \times p_e)$
    Gerar filhos P_c usando o PUX e ρ_e.
    $P \leftarrow P_e \cup P_m \cup P_c$.
    Aplicar o *decoder* nos novos cromossomos, atualizar e ordenar a população P.
    se a melhor solução foi melhorada **então**
      Armazenar a melhor solução sol^*.
      Definir recompensa $r = 1$ e atualizar $Q - Table$.
    se Busca Local **então**
      Identifique comunidades em P_e com o método LP.
      Aplique a busca local nessas comunidades e atualize sol^*.
  return sol^*
fim função

zação de métodos pode proporcionar uma flexibilidade maior para lidar com problema de grande escala do mundo real. Uma das formas mais conhecidas de metaheurística híbrida é a combinação de heurísticas de busca local dentro de algoritmos evolutivos (CHAVES *et al.*, 2016). Esta hibridização funciona de forma que os algoritmos evolutivos exploram o espaço de busca e identificam regiões promissoras, enquanto as heurísticas de busca local são capazes de encontrar, de forma rápida, ótimos locais nessas regiões. As heurísticas de busca local são aplicadas apenas em áreas promissoras, para evitar um aumento significativo no tempo de execução.

O método utilizado como busca local foi o *Random Variable Neigborhood Descent* (RVND) (PENNA *et al.*, 2013). O RVND consiste em uma variante do método VND (MLADENOVIć; HANSEN, 1997), no qual dado um conjunto de estrutura de vizinhanças, escolhe aleatoriamente uma vizinhança desse conjunto para aplicar a busca, considerando todo o espaço de solução. No FTSP, o RVND seleciona aleatoriamente a ordem heurística a ser aplicada em uma solução que representa uma comunidade na partição Elite, conforme Pseudocódigo 6. A Lista de Heurísticas (HL) contendo um número predefinido de heurísticas organizada de forma aleatória é inicializada (linha 2). Nas linhas 4-12, uma heurística $H_i \in$ HL é escolhida aleatoriamente (linha 5) e então o melhor movimento admissível é determinado (linha 6). Uma vez que uma heurística melhora a solução s, então o algoritmo refaz toda a lista HL inicial com as heurísticas (linhas 6-9). Caso contrário, H_i é removido da lista HL (linha 11).

Algoritmo 6 RVND (PENNA *et al.*, 2013)

```
1: função FUNÇÃO RVND.(s_0, h_n)
2:     Inicializar lista HL com as heurísticas em ordem aleatória (HL ← {1, ··· h_n})
3:     s ← s_0
4:     enquanto HL ≠ 0 faça
5:         Escolher uma heurística H_i ∈ HL.
6:         Encontrar a melhor solução s' de s ∈ H.
7:         se (f(s)' < f(s)) então
8:             s' ← s.
9:             Reinicializar lista HL.
10:        senão
11:            Remover H_i da lista HL.
12:        fim-se
13:    fim-enquanto
14:    Retornar s.
15: fim função
```

3.6 Implementação

Esta seção apresenta de forma detalhada a implementação dos *decoders* e as heurísticas de busca local utilizadas para resolver o FTSP. O número de funções determinísticas utilizadas no BRKGA-QL será definido, assim como os parâmetros de entrada de cada método e os pseudocódigos referentes aos algoritmos implementados.

3.6.1 *Decoders*

A função do *decoder* é fazer o mapeamento do vetor de chaves aleatórias para o espaço de soluções, avaliando a função objetivo do problema abordado (ver Figura 3.12). O *decoder* é uma função dependente do problema em resolução. Isso significa que podem ser aplicados diferentes *decoders* em um mesmo problema para geração das soluções à critério do usuário.

Neste trabalho, foram desenvolvidos cinco *decoders* para a resolução do FTSP. Todos partem de uma solução inicial, que consiste no vetor de chaves aleatórias devidamente ordenado em ordem crescente e a partir de então aplicam-se as funções. Essas são compostas por técnicas heurísticas, que permitem solucionar o TSP de forma aproximada (BODIN, 1983). Contudo, evidencia-se a seguir as duas classes utilizadas no método proposto para resolver o FTSP: heurísticas construtivas e heurísticas de refinamento.

As heurísticas construtivas utilizam procedimentos para construção de soluções factíveis (BODIN, 1983). No caso do FTSP, as heurísticas construtivas, consistem em algoritmos que geram um circuito viável vértice a vértice, e através de critérios de escolha, o conjunto é modificado a cada iteração na busca de melhores soluções. O conjunto de critérios pode

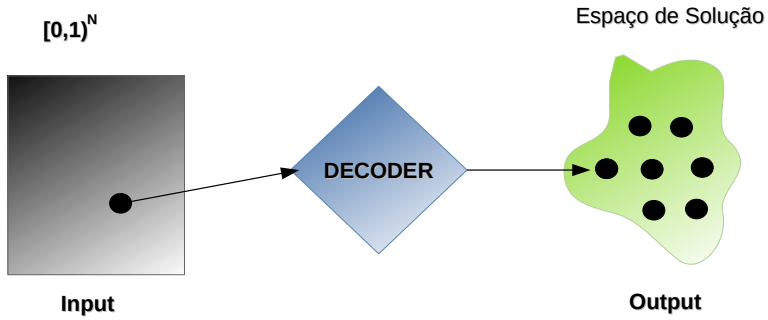

FIGURA 3.12 – Função *decoder* (adaptado de Gonçalves e Resende (2011)).

ser definidos como: inicialização, seleção e inserção. A inicialização consiste na escolha da sub-rota inicial (ponto de partida), a seleção na escolha do próximo elemento a ser inserido à solução e a inserção na definição da posição onde o novo elemento será alocado.

Como exemplo, pode-se destacar a heurística de inserção mais barata (*Cheapest Insertion*) (HASSIN; KEINAN, 2010), heurística de inserção do vizinho mais distante (*Farthest Insertion*) (CORDENONSI, 2008) e heurística de inserção do vizinho mais próximo (*Nearest Insertion*) (GOLDBARG, 2005).

A Figura 3.13 ilustra um exemplo de aplicação da heurística *Nearest Insertion*. Neste exemplo, o procedimento de construção da rota desde a seleção do primeiro nó até que todos os nós estejam inseridos na rota, passo a passo, pode ser descrito como segue:

- Inicialização: iniciar uma sub-rota com apenas um nó i, selecionado aleatoriamente; encontrar o nó j para o qual c_{ij} (distância ou custo de i a j) é mínimo e construir a sub-rota (i, j) representada por sol';
- Seleção: encontrar os nós k e j ($j \in sol'$ e $k \notin sol'$) de modo que c_{kj} seja minimizado e encontrar a aresta $\{i, j\} \in sol'$ que minimiza $c_{ik} + c_{kj} - c_{ij}$.
- Inserção: Inserir k entre i e j.

As heurísticas de refinamento/melhoria partem de uma solução inicial provavelmente obtida por heurísticas construtivas. A partir de então buscam-se melhorias através de modificações na solução (BODIN, 1983). Dentre as heurísticas de refinamento, pode-se destacar as heurísticas *2-Opt* e *3-Opt*, ambas introduzidas por Croes (1958). Estas têm seu funcionamento baseado na permutação das arestas da solução inicial de um TSP.

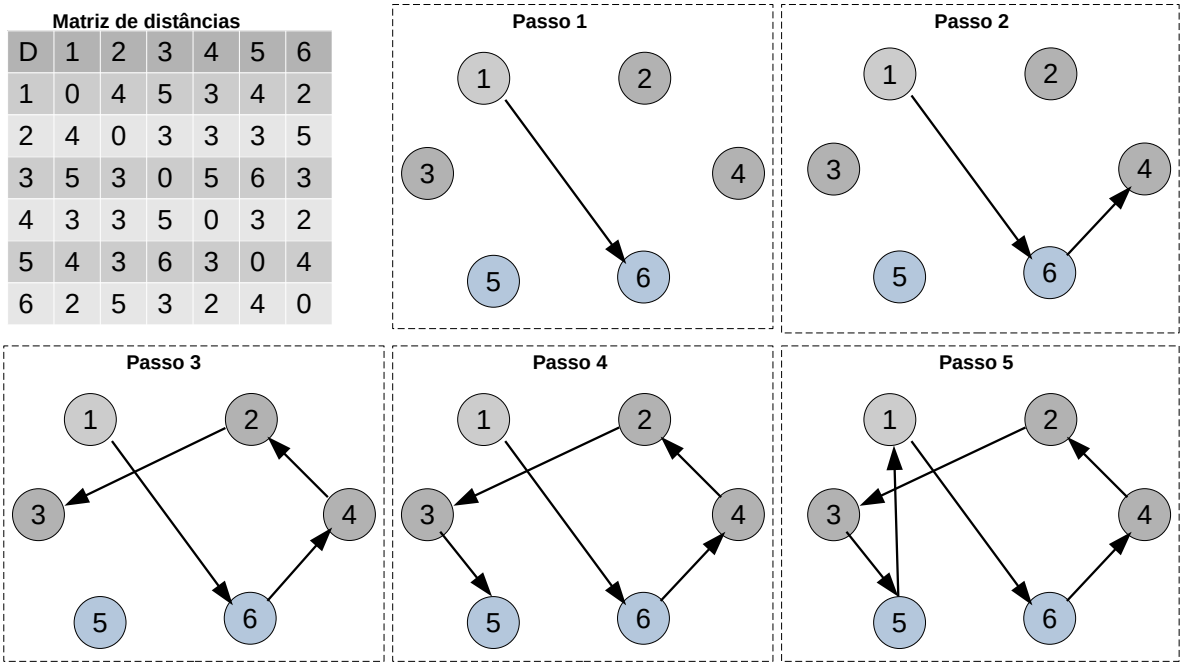

FIGURA 3.13 – Exemplo de solução com a heurística *Nearest Insertion* (adaptado de Goldbarg (2005)).

A heurística *2-Opt*, elimina dois arcos não adjacentes e reconecta as sub-rotas através de outros dois arcos, de modo a possibilitar uma solução viável com distância ou custo inferior a rota inicial. O processo é repetido para todos os pares de arestas e o circuito cuja distância total for menor é escolhido. A Figura 3.14 ilustra a aplicação da heurística *2-Opt* .

A heurística *3-Opt* é uma variante da *2-Opt*, que utiliza a permutação de três arcos (LIN, 1965). Assim como na *2-Opt*, todas as combinações de três arcos que construam uma rota viável são testados. Caso alguma das rotas obtida pelas permutações apresente melhor valor em relação à rota inicial, a troca é mantida. Um exemplo de heurística *3-Opt* é apresentada na Figura 3.15.

Os *decoders* elaborados e aplicados ao BRKGA-QL atendem as características específicas do FTSP. Para cada instância do FTSP são fornecidas as informações referentes às famílias e ao número de visitas. Os parâmetros estabelecidos para o algoritmo são: o número de nós n, o número de famílias $nfamily$, o número de visitas necessárias $nvisits$, o número de visitas para cada família i $nvisits(i)$, os membros correspondentes a cada família $membros_{family}(i)$ e a matriz de distância entre os nós $dist$.

Foram adotados dois critérios para garantir essas características: um critério de verificação, cuja função é identificar a qual família pertence cada elemento i do vetor de chaves aleatórias e um critério de contagem de visitas para cada família durante a construção da rota. Dessa forma, para cada família, atribuem-se vetores ($membros_{family}(i)$)

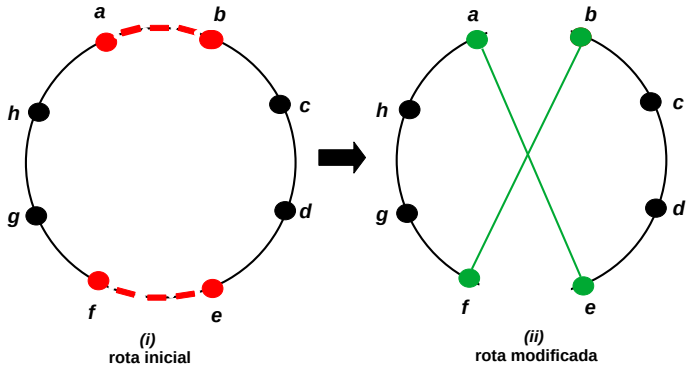

FIGURA 3.14 – Exemplo de um movimento da heurística 2-Opt (adaptado de Helsgaun (2000)).

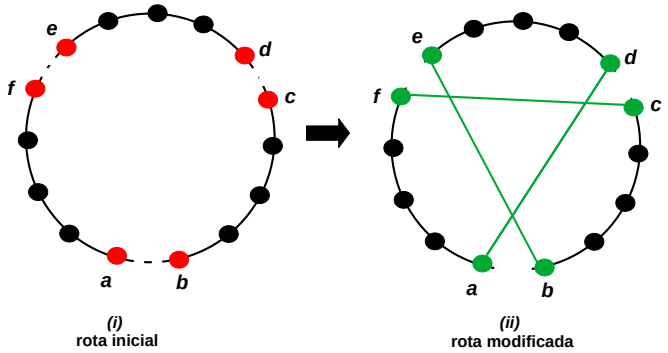

FIGURA 3.15 – Exemplo de um movimento da heurística 3-Opt (adaptado de Helsgaun (2000)).

que contêm os respectivos membros. Utilizando o exemplo dado na Seção 3.1.1.2, no qual a instância FTSP apresenta três famílias com três membros cada, têm-se os vetores: $membros_{family}(1)$ = (1,2,3), $membros_{family}(2)$ = (4,5,6) e $membros_{family}(3)$ = (7,8,9). Assim, a cada iteração do algoritmo, é realizada a assimilação do elemento i a família correspondente. Neste sentido, se o nó em verificação for o nó 6, por exemplo, sabe-se que este pertence à família 2.

Também é estabelecido um contador de visitas ($cont_{visits}$), que contabiliza o número de visitas recebidas em cada família, armazenando-as e incrementando-as a cada iteração. Trata-se de um vetor de tamanho igual a quantidade de famílias do problema (n_{family}). Ao inicializar o processo de construção da rota, todos os elementos de $cont_{visits}$ são iguais a zero. No caso do exemplo anterior, $cont_{visits}$ = (0,0,0), ou seja, ao início do processo ainda não tem nós inseridos na rota e consequentemente as famílias não receberam visitas. O contador funciona de modo que, por exemplo, caso na primeira iteração o nó i pertença à família 2 então é somado 1 ao elemento de $cont_{visits}$ correspondente, enquanto os outros permanecem zerados e como resultado ao final da primeira iteração temos: $cont_{visits}$ =

(0,1,0). Em outras palavras, a cada iteração, se i for inserido na rota, soma-se 1 na contagem do elemento correspondente em $cont_{visits}$.

Os nós (elementos) são inseridos na rota do caixeiro viajante ($rota_{sol}$) até que suas famílias estejam completas, ou seja, até que o número total de visitas preestabelecido para cada família seja atingido. Portanto, se durante o processo, um determinado elemento i pertença a uma família completa, então este é descartado bem como os nós pertencentes a mesma família que vêm na sequência. Consequentemente, a $rota_{sol}$ é construída até que todas as famílias estejam completas, significando que todas as visitas necessárias para cada família sejam realizadas.

Uma solução do BRKGA-QL (s) é composta por n+1 genes, sendo que n consiste no número de nós da instância estudada e a posição n+1 é responsável pela definição de qual *decoder* será aplicado no cromossomo. Para estabelecer um critério para escolha do *decoder* a ser aplicado, o intervalo [0,1] é igualmente dividido em k partes, em que k é o número de decodificadores disponíveis. Por exemplo, no caso de cinco *decoders* tem-se a seguinte partição: *Dec1* [0,0, 0,20], *Dec2* [0,21, 0,40], *Dec3* [0,41, 0,60], *Dec4* [0,61, 0,80] e *Dec5* [0,81, 1,0]. Portanto, se o gene da posição n+1 do vetor s for 0,45, por exemplo, aplica-se o *decoder* 3.

O depósito, representado pelo nó 0, não é considerado no cromossomo. Este é incluído na rota construída ao final da execução do algoritmo. A função objetivo calcula a distância total da rota construída, incluindo as distâncias referentes ao percurso de ida e volta ao depósito.

Neste trabalho foram implementados cinco *decoders*: Inserção simples (*Dec1*), *2-Opt* (*Dec2*), *Cheapest Insertion* (*Dec3*), *k-Farthest Insertion* (*Dec4*) e *k-Nearest Insertion* (*Dec5*). Todos os algoritmos partem da solução inicial (s) obtida pelo vetor de chaves aleatórias e em sequência faz a ordenação crescente dos genes obtendo uma sequência de pontos. A seguir apresenta-se detalhadamente estes *decoders*.

No *Dec1*, Inserção simples, partindo da sequência obtida pela ordenação do vetor de chaves aleatórias (s), a rota ($rota_{sol}$) é construída respeitando a ordem dos nós já estabelecida e o número de visitas para cada família.

A Figura 3.16 apresenta a construção da $rota_{sol}$ a partir do vetor de chaves aleatórias s, que é a base para construção da rota para os demais *decoders*. Neste exemplo, são consideradas três famílias, com três membros cada, sendo necessárias duas visitas para cada família. O vetor de chave aleatória contém dez elementos ($n+1$), e a chave aleatória de posição $n+1$ indica que Dec1 (Inserção Simples) deve ser aplicado (chave aleatória igual a $0,18 < 0,20$).

O primeiro passo consiste na ordenação ascendente do vetor de chaves aleatórias, obtendo uma sequência de nós. A $rota_{sol}$ é construída partindo do depósito e visitando

o primeiro elemento obtido pela ordenação do vetor de chaves aleatórias. Para cada elemento (i), realiza-se a associação à família correspondente e então a verificação da contagem de visitas recebidas em $cont_{visits}$ para a família de i. O sinal verde representa que o critério de número de visitas está sendo respeitado e portanto o elemento i pode ser inserido na rota, enquanto o sinal vermelho representa que o número de visitas dessa família já foi concluído e o nó não pode ser inserido na rota.

O nó 0, que representa o depósito, é inserido ao final da $rota_{sol}$, completando assim a rota do FTSP. Ao final do processo, a rota FTSP é composta pelos nós {0,2,4,6,1,8,9,0}. O pseudocódigo do *decoder Dec1* é apresentado no Algoritmo 7.

Algoritmo 7 *Decoder* 1: Inserção Simples

```
1: função DEC1(s, n, n_family, cont_visitas, membros_family, rota_sol, dist)
2:     Ordenar o vetor de chaves aleatórias s.
3:     Inicializar o vetor cont_visits com 0.
4:     para i=0 até i < n faça
5:         Verificar a qual família o elemento i corresponde em membros_family.
6:         Verificar quantas visitas já foram realizas para a família de i.
7:         se a família correspondente a i ainda não estiver completa então
8:             Inserir elemento i em rota_sol.
9:             Somar 1 no elemento de cont_visits correspondente a família de i.
10:        fim-se
11:    fim-para
12:    Guardar a rota_sol com os nós visitados seguido dos não visitados.
13:    Inserir elemento 0, correspondente ao depósito em rota_sol.
14:    Calcular a função objetivo e armazenar em s.fo.
15: fim função
```

O *Dec2* aplica a heurística *2-Opt* (Figura 3.14) na sequência obtida pelo *decoder* simples, considerando uma iteração completa e a estratégia *first improvement* (OCHOA *et al.*, 2010). Na estratégia *first improvement*, aceita-se um movimento assim que uma solução melhor que a solução corrente é encontrada, e então esta passa a ser a solução corrente. Possibilitando assim uma convergência melhor dos resultados. O pseudocódigo do *Dec2* está evidenciado no Algoritmo 8. Inicialmente a $Rota_{sol}$ é criada com a função DEC1 e calcula-se a função objetivo. Na sequência, aplica-se a heurística *2-Opt*. Nesta, são analisados os custos de remover duas arestas não adjacentes e inserir outras duas arestas para reconectar a rota. Se o movimento tiver um custo vantajoso frente à solução corrente então este movimento é realizado. Ao final do processo, o algoritmo armazena a rota cujo custo total for o menor. Este *decoder* foi aplicado em instâncias simétricas. Uma vez que nas instâncias simétricas, para todos os pares de nós i e j, os custos das arestas (i, j) e (j, i) são iguais e a heurística *2-Opt* precisa que esta condição seja satisfeita para calcular os custos de remoção e inserção das arestas. Contudo, essa condição não é verdadeira no caso das instâncias assimétricas. Neste caso, os custos das arestas (i, j) e (j, i) são diferentes. Portanto, aplica-se a heurística *3-Opt* considerando a possibilidade na qual o cálculo reverso não é realizado. O Algoritmo 9 representa o pseudocódigo do

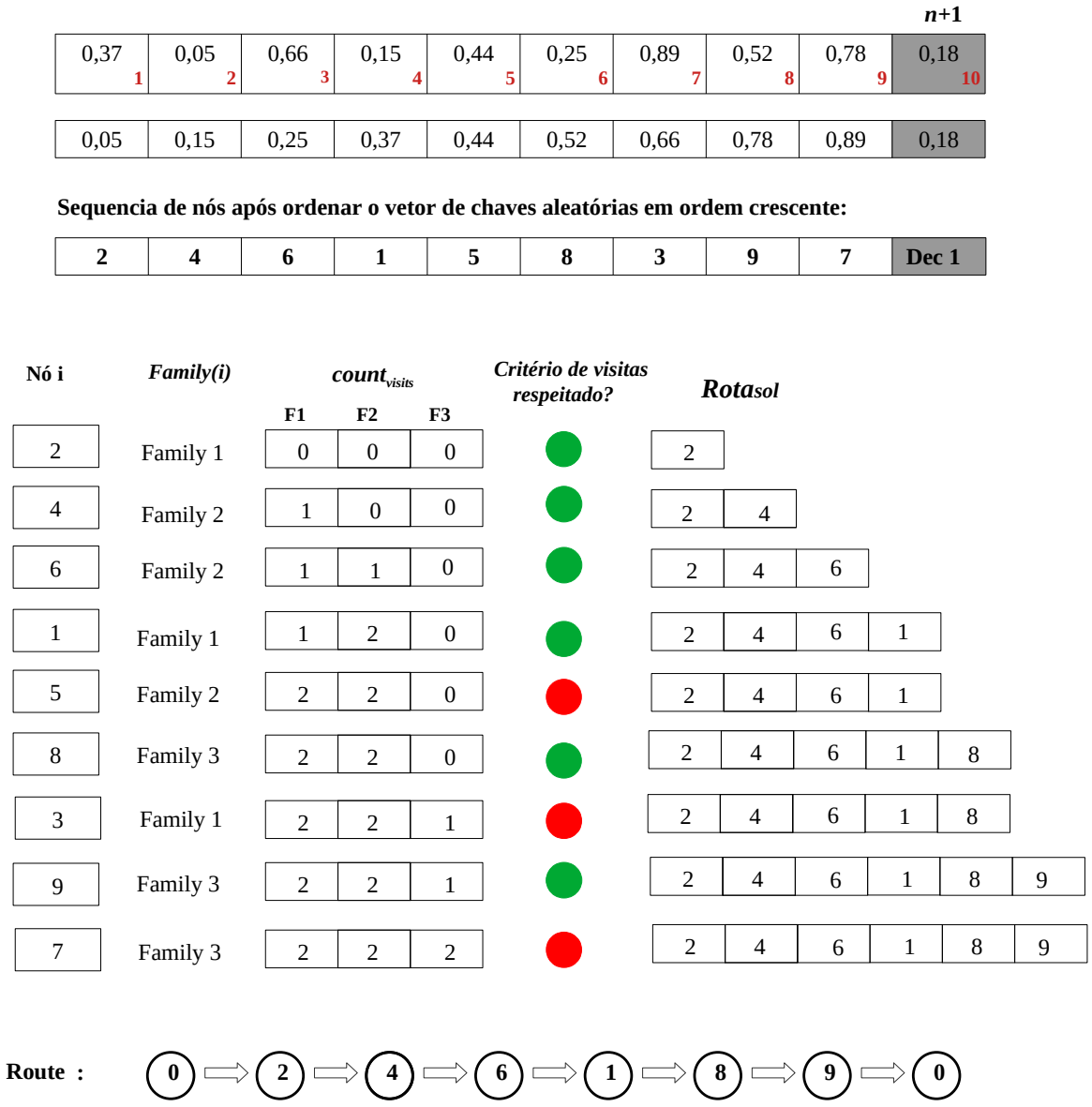

FIGURA 3.16 – Exemplo de aplicação do *decoder* Dec1. Elaborada pela autora.

Dec2 com o *3-Opt*. A diferença do algoritmo *3-Opt* consiste na permutação entre três arestas ao invés de duas como na *2-Opt*. Os procedimentos de verificação de melhoria e testagem dos pares são iguais. Ao final do processo, o algoritmo também armazena a rota de menor custo.

Algoritmo 8 *Decoder* 2: *2-Opt*

função DEC2(s, n, n_{family}, $cont_{visitas}$, $membros_{family}$, $rota_{sol}$, $dist$)
    Criar a $rota_{sol}$ com a função DEC1.
    Fazer $t \leftarrow$ tamanho($rota_{sol}$).
    Calcular a função objetivo $s.fo$.
    para i=0 até i<t **faça**
        $j \leftarrow i + 2$.
        enquanto $((j+1) mod\ t) \neq i$ **faça**
            se $dist_{i,j}$ + $dist_{i+1,j+1}$ - $dist_{i,i+1}$ - $dist_{j,j+1} < 0$ **então**
                Trocar as arestas $(i, i+1)$ e $(j, j+1)$ por (i, j) e $(i+1, j+1)$.
                Atualizar função objetivo $s.fo \leftarrow s.fo + dist_{i,j} + dist_{i+1,j+1}$ - $dist_{i,i+1}$ - $dist_{j,j+1}$.
            fim-se
            $j++$
        fim-enquanto
    fim-para
fim função

Algoritmo 9 *Decoder* 2: *3-Opt*

função DEC2(s, n, n_{family}, $cont_{visitas}$, $membros_{family}$, $rota_{sol}$, $dist$)
    Criar a $rota_{sol}$ com a função DEC1.
    Fazer $t \leftarrow$ tamanho($rota_{sol}$) (lista circular).
    Calcular a função objetivo $s.fo$.
    para i=0 até i<t **faça**
        para j= i+2 até j<t **faça**
            para k= j+2 até k<t **faça**
                Calcular: $d0 = dist_{i-1,i} + dist_{j-1,j} + dist_{k-1,k}$.
                Calcular: $d3 = dist_{i-1,j} + dist_{k-1,i} + dist_{j-1,k}$.
                se $d0 > d3$ **então**
                    Trocar a rota atual pela rota nova.
                    Atualizar função objetivo $s.fo$ += $-d0 + d3$.
                fim-se
            fim-para
        fim-para
    fim-para
fim função

O *Dec3* ordena-se o vetor de chaves aleatórias s e a partir da sequência obtida aplica-se a heurística *Cheapest Insertion*. A construção da rota inicia armazenando uma sub-rota consistindo apenas dos primeiros três nós da sequência s obtida pela ordenação do vetor de chaves aleatórias. Define-se a lista de nós candidatos, ou seja, nós não inseridos na rota. Para inserir o próximo nó da sequência na rota, o algoritmo busca uma posição cuja inserção resultará no menor custo. O algoritmo testa todas as posições e calcula o custo de inserção. Caso o algoritmo defina a posição para inserção cujo custo seja o menor já encontrado, então a posição é definida como a melhor posição (*Pbest*) e o elemento é inserido nesta. O procedimento é executado até que a rota esteja completa, ou seja, o

número de visitas de cada família é respeitado. O Algoritmo 10 apresenta o pseudocódigo do *Dec3*.

Algoritmo 10 *Decoder* 3: *Cheapest Insertion*

1: **função** DEC3(s, n, n_{family}, $cont_{visitas}$, $membros_{family}$, $rota_{sol}$, $dist$)
2: Ordenar o vetor de chaves aleatórias s
3: Construir uma rota parcial $rota_{sol}$ com os três primeiros nós da sequência obtida em s.
4: Criar a lista de candidatos s_{off} para inserção (nós não inseridos).
5: **para** i=3 até $i <$ tamanho(s_{off}) **fazer**
6: $c_{best} = infinito$
7: **para** j=0 até $j <$ tamanho($rota_{sol}$) **fazer**
8: $c_{ij} = dist_{i,j} + dist_{j+1,i}$ - $dist_{j+1,j}$.
9: **se** $c_{ij} < c_{best}$ **então**
10: Definir $c_{best} = c_{ij}$.
11: Definir $p_{best} = j$.
12: **fim-se**
13: **fim-para**
14: Inserir elemento i na posição p_{best}.
15: Atualizar a lista de candidatos s_{off}.
16: **fim-para**
17: Atualizar função objetivo $s.fo$.
18: **fim função**

No *Dec4* ordena-se o vetor de chaves aleatórias s e a partir da sequência obtida aplica-se a heurística *k-Farthest Insertion*. Considera-se $k = 3$ e apenas o primeiro nó da sequência obtida pela ordenação do vetor de chaves aleatórias s é inserido na rota. Portanto, para completar a rota, o próximo nó deve ser selecionado dentre os k primeiros nós da sequência e deve ser o mais distante da rota parcialmente construída. Em sequência o algoritmo busca a posição de menor custo para inserir o elemento na rota parcial. Caso o algoritmo defina a posição para inserção cujo custo seja o menor já encontrado, então a posição é definida como a melhor posição (*Pbest*) e o elemento é inserido nesta. O processo é repetido até que a rota esteja completa. O pseudocódigo do *Dec4* está definido no Algoritmo 11.

O *Dec5*, *k-Nearest Insertion*, é igual ao *decoder* 4, a única diferença está na linha 6 do Algoritmo 11, uma vez que nesta heurística deve-se encontrar e inserir o nó mais próximo da rota parcialmente construída ao invés de inserir o nó mais distante.

Algoritmo 11 *Decoder* 4: *k-Farthest Insertion*

função DEC4(s, n, n_{family}, $cont_{visitas}$, $membros_{family}$, $rota_{sol}$, $dist$)
    Ordenar o vetor de chaves aleatórias s.
    Construir uma rota $rota_{sol}$ com o primeiro nó da sequência obtida em s.
    Criar a lista de candidatos s_{off} para inserção (nós de s não inseridos).
    enquanto a lista s_{off} ainda possuir candidatos para verificação **fazer**
        # Encontrar o elemento i mais distante da $rota_{sol}$ entre os primeiros k nós
        $c_{farthest} = infinito$
        para k=0, k<3 **faça**
            para j=0 até $j <$ tamanho($rota_{sol}$) **faça**
                se $dist_{k,j} > c_{farthest}$ **então**
                    $c_{farthest} = dist_{k,j}$.
                    $i = k$.
                fim-se
            fim-para
        fim-para
        # Encontrar a posição mais barata para inserir o elemento i na $rota_{sol}$.
        $c_{best} = infinito$
        para j=0 até $j <$ tamanho($rota_{sol}$) **fazer**
            $c_{ij} = dist_{i,j} + dist_{(j+1),i}$ - $dist_{(j+1),j}$.
            se $c_{ij} < c_{best}$ **então**
                $c_{ij} = c_{best}$.
                $p_{best} = j$.
            fim-se
        fim-para
        Inserir elemento i na posição p_{best}.
        Atualizar a lista de candidatos s_{off}.
    fim-enquanto
    Calcular a função objetivo $s.fo$.
fim função

3.6.2 Heurísticas de Busca Local

Esta seção detalha a implementação das heurísticas de busca local para resolver o FTSP. Estas heurísticas analisam todos os vizinhos possíveis para uma solução atual utilizando a estratégia *First Improvement*. As seis heurísticas de busca local são definidas:

- ***2-Opt*** (OR, 1976): Na heurística *2-Opt* remove-se duas arestas não adjacentes, e as reconecta à rota inicial da melhor maneira possível. A heurística calcula todas as soluções que podem ser geradas a partir destes movimentos. O objetivo consiste em obter uma rota com distância ou custo inferior à rota inicial por meio da substituição de arestas de maior custo por outras de menor custo. A partir de uma solução inicial s, deve-se selecionar duas arestas não adjacentes (i,i+1) e (j,j+1) e calcular o custo para reconectar as arestas (i, j) e $(i + 1$,j+1). Os movimentos de troca somente são realizados se resultarem em uma rota com custo (ou distância) inferior à rota atual. Portanto, se o custo do movimento for vantajoso frente à rota inicial, então as arestas selecionadas são trocadas e ao fim do algoritmo é atualizada a função objetivo. A Heurística é executada enquanto houver vizinhos melhores. A Figura 3.17 **a.** ilustra um exemplo do movimento *2-Opt*.

- ***3-Opt*** (LIN, 1965): Na heurística *3-Opt* remove-se três arestas não adjacentes e as reconecta à rota inicial da melhor maneira possível, considerando o caso em que não precisa reverter a rota. Esta heurística foi usada em substituição a 2-opt nas instâncias assimétricas, por isso só consideramos este caso. A heurística *3-Opt* calcula todas as soluções que podem ser geradas a partir destes movimentos. O objetivo consiste em obter uma rota com distância ou custo inferior à rota inicial por meio da substituição de arestas de maior custo por outras de menor custo. A partir de uma solução inicial s, deve-se selecionar três arestas não adjacentes (ab, cd, ef) e calcular o custo para reconectar as arestas (ad, eb, cf). Os movimentos de troca somente são realizados se resultarem em uma rota com custo (ou distância) inferior à rota inicial. Portanto, se o custo do movimento for vantajoso frente á rota inicial, as trocas são mantidas, e ao fim do algoritmo é atualizada a função objetivo. A Heurística é executada enquanto houver vizinhos melhores. A Figura 3.17 **b.** ilustra um exemplo do movimento *3-Opt*.

- ***Or-Opt*** (OR, 1976): A heurística *Or-Opt* consiste em inserir e trocar sequências (ou sub-rotas) de arestas consecutivas na rota atual, ou seja, contém todas as soluções que podem ser geradas removendo dois pontos adjacentes na posição i e $i+1$ e inserindo em outra posição da rota. A heurística *Or-opt* realiza um processo de inserção generalizada em vários estágios. Ao remover a aresta $(i, i+1)$ primeiro considera inserir entre dois elementos adjacentes j e j+1, dessa forma, para reconectar a rota é necessário a inserção de sub-rotas de três vértices. A partir desse estágio, o processo para inserir sub-rotas de dois vértices (uma aresta) é reduzido sucessivamente e por fim, são consideradas as inserções de vértices únicos, fazendo trocas sempre que um ótimo local é encontrado na vizinhança atual. O custo se inserção dos vértices i e $i+1$ em outra posição da rota é calculado por: $dist(i,j)$ + $dist(i+1,j+1)$ + $dist(i-1,i+2)$ - $dist(j,j+1)$ - $dist(i-1,i)$ - $dist(i+1,i+2)$. Os movimentos de troca somente são realizados se resultarem em uma rota com custo (ou distância) inferior à rota inicial. Portanto, se o custo do movimento for vantajoso frente á rota inicial, então as posições i e $i+1$ são removidas da rota atual e a aresta $(i, i+1)$ é inserida entre as posições j e j+1. Ao fim do algoritmo é atualizada a função objetivo. A Heurística é executada enquanto houver vizinhos melhores. A Figura 3.17 **c.** ilustra um exemplo do movimento *Or-Opt*.

- ***Node-Exchange*** (REGO; GLOVER, 2002): A heurística *Node-Exchange* contém todas as soluções que podem ser geradas trocando a posição i e a posição j entre si na rota. A partir de uma solução inicial s, a heurística *Node-Exchange* analisa para cada elemento visitado na rota, a economia obtida trocando os pontos nas posições i e j. A cada iteração, a nova distância ou custo é calculada e é verificado se houve economia no movimento. O custo da troca é calculado por: $dist(i-1,j)+dist(i,i+$

$1)+dist(j-1,i)+dist(i,j+1)-dist(i-1,i)-dist(i,i+1)-dist(j-1,i)-dist(i,j+1)$. As arestas $(i-1,j)$, $(i,i+1)$, $(j-1,i)$ e $(i,j+1)$ são as novas arestas adicionadas para reconectar a rota, e as arestas $(i-1,i)$, $(i,i+1)$, $(j-1,i)$ e $(i,j+1)$ são as arestas removidas após realizar a troca entre as posições i e j na rota. Os movimentos de troca somente são realizados se resultarem em uma rota com custo (ou distância) inferior à rota inicial. Portanto, se o custo do movimento for vantajoso frente á rota inicial, as posições i e j são trocadas de posição entre si na rota, e ao fim do algoritmo é atualizada a função objetivo. A Heurística é executada enquanto houver vizinhos melhores. A Figura 3.17 **d.** ilustra um exemplo de movimento da heurística *Node-Exchange*.

- ***Node-Insertion*** (REGO; GLOVER, 2002): A heurística *Node-Insertion* contém todas as soluções que podem ser geradas removendo o ponto da posição i da rota e inserindo-o em outra posição j da rota. A partir de uma solução inicial s, a heurística *Node-Insertion* analisa para cada elemento visitado na rota a economia obtida removendo o pontos da posição i e inserindo-o em outra posição j. A cada iteração, a nova distância ou custo é calculada e é verificado se houve economia no movimento. O custo da troca é calculado por: $dist(i,j) + dist(i,j+1) + dist(i-1,i+1)$ - $dist(j,j+1)$ - $dist(i-1,i)$ – $dist(i,i+1)$. As arestas (i,j), $(i,j+1)$ e $dist(i-1,i+1)$ são as novas arestas adicionadas para reconectar a rota, e as arestas $(j,j+1)$, $(i-1,i)$ e $(i,i+1)$ são as arestas removidas após remover a posição i na rota. Se houver economia no movimento aplicado, então a posição i é removido da rota atual e inserida na posição j da rota. Ao final do algoritmo a função objetivo é atualizada. A Heurística é executada enquanto houver vizinhos melhores. A Figura 3.17 **e.** ilustra um exemplo de movimento da heurística *Node-Insertion*.

- ***Swap-nodes***: A partir de uma solução inicial s, a heurística *Swap-nodes* analisa para cada vértice visitado na rota as soluções que podem ser geradas removendo a posição i e inserindo outro elemento j da mesma família, e que não é visitado na rota atual, na posição de inserção k mais barata da rota. Portanto, para cada elemento i faz-se uma verificação se o elemento j pertence a mesma família de i. A economia da troca entre as posições é calculada pelo custo (ou distância) de remoção dos arcos que conectam a posição i na rota e de inserção dos novos arcos para reconectar a rota após a remoção da posição i e inserção de j. Se as posições i e k forem iguais, então a economia da troca é calculada por: - $dist(i-1,i)$ - $dist(i,i+1)$ + $dist(i-1,j)$ + $dist(j,i+1)$. As arestas $(i-1,i)$ e $(i,i+1)$ correspondem as arestas removidas da rota atual e as arestas $(i-1,j)$ e $(j,i+1)$ são as novas arestas adicionadas para reconectar a rota após a troca entre os elementos i e j. Em contrapartida, se $i \neq k$ e $k \neq i+1$, então a economia da troca é calculada por: - $dist(i-1,i)$ - $dist(i,i+1)$ + $dist(i-1,i+1)$ - $dist(k-1,k)$ + $dist(k-1,j)$ + $dist(j,k)$. As arestas $(i-1,i)$,

$(i, i+1)$ e $(k-1, k)$ correspondem as arestas removidas da rota atual, as arestas $(k-1, j)$, (j, k) e $(i-1, i+1)$ são as novas arestas adicionadas para reconectar a rota após remoção da posição i e inserção do de j na posição k. Após calcular a economia obtida pela troca de posições entre os elementos, verifica-se se a economia obtida é a melhor obtida dentre todas as trocas já testadas e caso positivo, a troca é mantida. Neste caso, os valores correspondentes de i, j, k nesta troca são guardados e definidos como $melhor_i$, $melhor_j$ e $melhor_j$ pelo algoritmo. Na sequência verifica-se quais trocas serão realizadas. Se houver economia frente à rota inicial então o elemento i é removido e substituído pelo elemento j na melhor posição k. Por fim, a função objetivo é atualizada. A Heurística é executada enquanto houver vizinhos melhores. A Figura 3.17 **f.** ilustra um exemplo de movimento da heurística *Swap-nodes*.

Essas heurísticas atualizam automaticamente o valor da função objetivo de um vizinho. Portanto, aplicamos apenas o *2-Opt* nas instâncias simétricas e o *3-Opt* nas instâncias assimétricas.

Exemplos dessas heurísticas são mostrados na Figura 3.17. A rota inicial 0-1–2–3–4–5–6-7-8-9-0 apresenta um exemplo de uma rota com 9 elementos mais o depósito. Os elementos da rota inicial estão divididos em 3 famílias, com 3 membros cada, diferenciadas pelas cores. Em (a) a rota 0-1-2-7-6-5-4-3-8-9-0 é obtida aplicando *2-Opt* aos arcos 2-3 e 7-8. Em (b) a rota 0-1-5-6-7-2-3-4-8-9-0 é obtida aplicando *3-Opt* nos arcos 1-2, 4-5 e 7-8. Em (c) a rota 0-1-2-3-6-7-8-4-5-9-0 é obtida aplicando *Or-Opt*, trocando as posições 4 e 5 com 7 e 8 respectivamente. Em (d) a rota 0-1-2-6-4-5-3-7-8-9-0 é obtida aplicando *Node-Exchange* aos elementos 3 e 6. Em (e) a rota 0-1-2-3-4-6-7-8-5-9-0 é obtida aplicando *Node-Insertion* ao elemento 5 na posição 8. Em (f) a rota 0-1-2-3-4-6-10-7-8-9-0 é obtida aplicando *Swap-nodes* removendo o elemento 5 e inserindo o elemento 10, da mesma família, que não estava contido na rota inicial.

Os Pseudocódigos das heurísticas aplicas estão evidenciados pelos Algoritmos 12 a 17 no Anexo A.

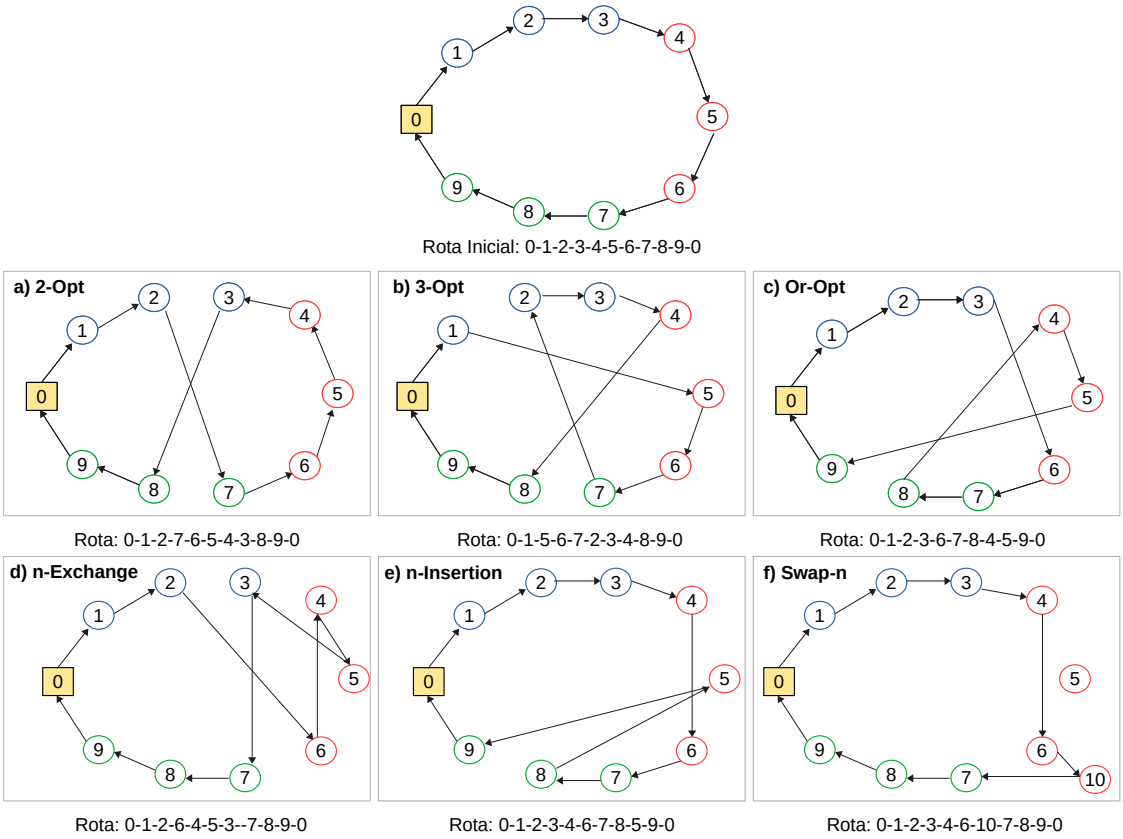

FIGURA 3.17 – Exemplos de movimentos de seis heurísticas de busca local aplicadas a uma rota inicial do FTSP. Elaborada pela autora.

4 Resultados Computacionais

Este capítulo está dividido em cinco seções que apresentam os resultados computacionais obtidos pela aplicação dos métodos propostos na resolução do FTSP, assim como, análises estatísticas, de modo a demonstrar a eficiência dos métodos. A Seção 4.1 descreve as instâncias disponíveis na literatura usadas para os experimentos computacionais. Os resultados computacionais estão projetados na Seção 4.2 em tabelas organizadas pelo critério do conjunto de instâncias bem como a forma de cálculo do desempenho para as análises estatísticas. De modo a analisar o desempenho das soluções alcançadas pelos métodos propostos, foi realizada uma comparação com os resultados obtidos pelos algoritmos encontrados na literatura, conforme 4.3. Primeiro os resultados são comparados com os resultados obtidos com o B&C híbrido proposto por (BERNARDINO; PAIAS, 2021b), que é a heurística estado-da-arte para o FTSP. Na sequência, os resultados são comparados com os resultados obtidos pelo BRKGA (MORÁN-MIRABAL *et al.*, 2014), GRASP (MORÁN-MIRABAL *et al.*, 2014), GA (BERNARDINO; PAIAS, 2021b) e ILS (BERNARDINO; PAIAS, 2021b) para as instâncias testadas por todos os métodos. A Seção 4.4 evidencia as contribuições do algoritmo Paralelo P-B&C. Por fim, a Seção 4.5 apresenta uma análise detalhada dos efeitos dos componentes do BRKGA-QL.

Os métodos propostos foram codificados usando a linguagem de programação C++. Os experimentos computacionais foram executados em uma *workstation* com dois processadores *Intel Xeon Silver 4114*, memória RAM de 96GB e sistema operacional *CentOS Linux 8.2*. O modelo de otimização do método P-B&C foi resolvido pelo solver *Gurobi 9.2* (Gurobi Optimization, LLC, 2021) em um tempo limite de 10800 segundos e com 7 *threads* dedicadas à frente B&C e 1 *thread* dedicada à frente de busca local. O critério de parada do BRKGA-QL foi definido como o número de nós (n) de cada instância, ou seja, se a instância possui 120 nós, então o critério de parada é de 120 segundos. Foram realizadas 10 execuções do BRKGA-QL para cada instância. O *framework* do BRKGA-QL encontra-se disponível em: `https://github.com/antoniochaves19/BRKGA-QL`.

4.1 Instâncias de Teste

Os experimentos foram realizados em 85 instâncias disponíveis na literatura. As instâncias estudadas estão dividas em quatro conjuntos: instâncias de *benchmark*, instâncias TSP simétricas, instâncias TSP assimétricas, instâncias UTPP assimétricas e apresentam as seguintes características:

- *Benchmark*: 21 instâncias simétricas das quais 13 possuem valor ótimo conhecido na literatura. O número de nós varia entre 14 e 1002;
- TSP simétricas: 64 instâncias das quais 49 possuem valor ótimo conhecido na literatura. O número de nós varia entre 136 e 264. Este conjunto é uma adaptação das instâncias simétricas da TSPLib sendo desenvolvido por Bernardino e Paias (2021b) para complementar as instâncias de *benchmark*;
- TSP assimétricas: compõe o conjunto 76 instâncias com número de nós do conjunto variando entre 17 e 443;
- UTPP assimétricas: 24 instâncias com número de nós variando entre 50 e 300.

Nos conjuntos assimétricos, conhecemos as soluções ótimas para todas as instâncias, exceto uma. As instâncias testadas estão disponíveis no site dedicado ao FTSP: `http://familytsp.rd.ciencias.ulisboa.pt/`.

4.2 Resultados computacionais dos métodos propostos

O desempenho dos métodos foi medido pelo *gap* (Equação 4.1):

$$gap(\%) = \frac{(UB - LB)}{LB} \times 100 \tag{4.1}$$

sendo que UB é o limite superior encontrado por P-B&C, BRKGA-QL ou B&C híbrido (BERNARDINO; PAIAS, 2021b), e LB é o limite inferior encontrado pelo P-B&C. Também será calculado o desvio percentual relativo (RPD) (Equação 4.2):

$$RPD(\%) = \frac{(UB - BKS)}{BKS} \times 100 \tag{4.2}$$

sendo que BKS é a melhor solução conhecida considerando os algoritmos encontrados na literatura (BRKGA e GRASP+evPR (MORÁN-MIRABAL *et al.*, 2014), B&C e ILS (BERNARDINO; PAIAS, 2018), GA, ILS e híbrido B&C (BERNARDINO; PAIAS, 2021b)).

As Tabelas 4.1, 4.2 e 4.3 apresentam os resultados para instâncias dos conjuntos de *benchmark*, TSP simétrico e TSP Assimétrico, respectivamente. A primeira coluna indica a instância testada e a segunda coluna, o respectivo *BKS*, as colunas UB e LB são os limites superior e inferior do P-B&C. As colunas *Best value* apresentam a melhor solução encontrada pelo algoritmo B&C híbrido (BERNARDINO; PAIAS, 2021b) e o BRKGA-QL. As colunas *gap* e *t(s)* informam os *gaps* e o tempo computacional, em segundos, informa o tempo de execução o que o algoritmo precisou para chegar na melhor solução encontrada de cada método. A coluna RPD indica o desvio percentual relativo calculado a partir da melhor solução obtida pelos métodos. A coluna $ARPD$ corresponde à média do RPD calculado em cada execução do BRKGA-QL. Os valores em negrito indicam as melhores soluções para cada instância, e valores negativos de RPD indicam as instâncias em que encontramos soluções melhores que BKS. Para melhor visualização, estão destacadas em cinza escuro as instâncias cuja otimalidade não havia sido comprovada na literatura. As instâncias destacadas em cinza claro correspondem às melhores soluções atualizadas pelos métodos propostos neste trabalho.

TABELA 4.1 – Resultados computacionais do FTSP para o conjunto de instâncias de *Benchmark*

Instância	*BKS*	P-B&C					B&C Híbrido[1]				BRKGA-QL				
		LB	*UB*	*gap*	*RPD*	*t (s)*	*Best value*	*gap*	*RPD*	*t (s)*	*Best value*	*gap*	*RPD*	*ARPD*	*t (s)*
burma14_1	13,93	13,93	**13,93**	0,00	0,00	0,09	**13,93**	0,00	0,00	0,00	**13,93**	0,00	0,00	0,00	0,03
burma14_2	25,66	25,66	**25,66**	0,00	0,00	0,03	**25,66**	0,00	0,00	0,00	**25,66**	0,00	0,00	0,00	0,04
burma14_3	11,89	11,89	**11,89**	0,00	0,00	0,02	**11,89**	0,00	0,00	0,00	**11,89**	0,00	0,00	0,00	0,03
bayg29_1	5345,86	5345,86	**5345,86**	0,00	0,00	0,36	**5345,89**	0,00	0,00	0,00	**5345,86**	0,00	0,00	0,00	0,10
bayg29_2	5791,01	5791,01	**5791,01**	0,00	0,00	0,50	**5791,01**	0,00	0,00	0,00	**5791,01**	0,00	0,00	0,00	0,08
bayg29_3	5544,33	5544,33	**5544,33**	0,00	0,00	0,18	**5544,33**	0,00	0,00	0,00	**5544,33**	0,00	0,00	0,00	0,09
att48_1	23686,00	23686,02	**23686,02**	0,00	0,00	1,17	**23686,02**	0,00	0,00	0,00	**23686,02**	0,00	0,00	0,00	0,15
att48_2	20609,09	20609,09	**20609,09**	0,00	0,00	2,67	**20609,08**	0,00	0,00	1,00	**20609,09**	0,00	0,00	0,00	0,11
att48_3	9024,50	9024,58	**9024,58**	0,00	0,00	1,64	**9024,58**	0,00	0,00	1,00	**9024,58**	0,00	0,00	0,00	0,14
bier127_1	33709,70	33709,75	**33709,75**	0,00	0,00	31,70	34100,3	1,16	1,16	8,00	**33709,75**	0,00	0,00	0,56	41,81
bier127_2	88736,40	88736,43	**88736,43**	0,00	0,00	69,19	89147,93	0,46	0,46	7,00	88761,09	0,03	0,03	0,59	26,08
bier127_3	47726,30	47726,32	**47726,32**	0,00	0,00	86,46	48374,13	1,36	1,36	6,00	47873,62	0,31	0,31	0,57	20,83
a280_1	1739,30	1692,92	**1692,92**	0,00	-2,67	4155,57	1739,30	2,74	0,00	63,00	1728,02	2,07	-0,65	-0,08	58,57
a280_2	1556,08	1543,72	**1543,72**	0,00	-0,79	10573,64	1556,08	0,80	0,00	58,00	1551,96	0,53	-0,26	-0,02	16,54
a280_3	1408,14	1408,14	**1408,14**	0,00	0,00	2436,50	1454,07	3,26	3,26	50,00	1431,40	1,65	1,65	3,09	55,40
gr666_1	1531,93	1400,60	**1484,91**	6,02	-3,07	10801,35	1531,93	9,38	0,00	906,00	1507,18	7,61	-1,62	-0,80	96,41
gr666_2	1236,21	1084,50	1227,25	13,16	-0,72	10800,77	1237,03	14,06	0,07	914,00	**1222,33**	12,71	-1,12	-0,45	80,71
gr666_3	1225,61	1045,03	**1163,27**	11,31	-5,09	10800,79	1225,61	17,28	0,00	968,00	1182,99	13,20	-3,48	-3,04	96,92
pr1002_1	132821,40	120663,48	133373,93	10,53	0,42	10817,03	134625,52	11,57	1,36	3008,00	**128772,19**	6,72	-3,05	-2,07	336,76
pr1002_2	144543,77	130536,61	147845,90	13,26	2,28	10800,83	145326,69	11,33	0,54	2537,00	**140129,05**	7,35	-3,05	-1,93	495,61
pr1002_3	126282,93	108471,18	127056,88	17,13	0,61	10801,47	126282,93	16,42	0,00	3085,00	**117138,23**	7,99	-7,24	-6,47	461,20
média				3,40	-0,43	3913,43		4,28	0,39	552,95		2,87	-0,88	-0,48	85,12

[1](BERNARDINO; PAIAS, 2021b)

Considerando as instâncias de *benchmark* (Tabela 4.1), o algoritmo P-B&C encontrou todas as soluções com valor ótimo conhecido e conseguiu provar a otimalidade das instâncias a280_1 e a280_2, das oito instâncias com valores ótimos desconhecidos. Nas outras seis instâncias, o BRKGA-QL encontrou quatro novos limites superiores melhores

(instâncias gr666_2, pr1002_1, pr1002_2 e pr1002_3), e o P-B&C, em um limite de tempo de 10.800 segundos, encontrou dois novos limites superiores melhores (instâncias gr666_1 e gr666_3). No geral, os dois métodos propostos encontraram seis novos limites superiores melhores, com um *gap* médio de 8,68%, e provaram otimalidade em 25% das instâncias com valor ótimo desconhecido.

TABELA 4.2 – Resultados Computacionais do FTSP para o conjunto de instâncias TSP Simétricas.

Instância	*BKS*	P-B&C					B&C Híbrido[1]				BRKGA-QL				
		LB	*UB*	*gap*	*RPD*	*t (s)*	*Best value*	*gap*	*RPD*	*t (s)*	*Best value*	*gap*	*RPD*	*ARPD*	*t (s)*
pr136_1	61448	61448	**61448**	0,00	0,00	118,68	61948	0,81	0,81	7,00	**61448**	0,00	0,00	0,31	8,28
pr136_2	43522	43522	**43522**	0,00	0,00	415,46	44209	1,58	1,58	7,00	**43522**	0,00	0,00	0,00	7,34
pr136_3	81481	81481	**81481**	0,00	0,00	63,67	81532	0,06	0,06	12,00	81531	0,06	0,06	0,17	44,28
pr136_4	63246	63246	**63246**	0,00	0,00	79,17	63321	0,12	0,12	6,00	**63246**	0,00	0,00	0,00	10,94
gr137_1	44263	44263	**44263**	0,00	0,00	66,62	44391	0,29	0,29	11,00	**44263**	0,00	0,00	0,00	0,33
gr137_2	36435	36435	**36435**	0,00	0,00	93,74	**36435**	0,00	0,00	7,00	**36435**	0,00	0,00	0,00	0,18
gr137_3	55919	55919	**55919**	0,00	0,00	34,12	56265	0,62	0,62	6,00	**55919**	0,00	0,00	0,00	3,63
gr137_4	46620	46620	**46620**	0,00	0,00	48,66	**46620**	0,00	0,00	7,00	**46620**	0,00	0,00	0,00	0,43
pr144_1	46376	46376	**46376**	0,00	0,00	44,86	**46376**	0,00	0,00	7,00	**46376**	0,00	0,00	0,00	0,35
pr144_2	36518	36518	**36518**	0,00	0,00	110,64	**36518**	0,00	0,00	8,00	**36518**	0,00	0,00	0,00	0,34
pr144_3	54635	54635	**54635**	0,00	0,00	124,31	54869	0,43	0,43	5,00	**54635**	0,00	0,00	0,00	2,02
pr144_4	49379	49379	**49379**	0,00	0,00	449,21	**49379**	0,00	0,00	15,00	**49379**	0,00	0,00	0,00	0,60
kroA150_1	14307	14307	**14307**	0,00	0,00	80,32	14471	1,15	1,15	15,00	14351	0,31	0,31	0,44	7,63
kroA150_2	9481	9481	**9481**	0,00	0,00	143,35	9496	0,16	0,16	12,00	9496	0,16	0,16	0,91	14,51
kroA150_3	20880	20880	**20880**	0,00	0,00	248,89	21038	0,76	0,76	11,00	**20880**	0,00	0,00	0,10	32,86
kroA150_4	13404	13404	**13404**	0,00	0,00	147,49	**13404**	0,00	0,00	10,00	**13404**	0,00	0,00	0,00	0,33
kroB150_1	14522	14522	**14522**	0,00	0,00	78,96	14661	0,96	0,96	16,00	**14522**	0,00	0,00	0,35	8,05
kroB150_2	9555	9555	**9555**	0,00	0,00	278,81	10015	4,81	4,81	12,00	**9555**	0,00	0,00	0,04	0,74
kroB150_3	19925	19925	**19925**	0,00	0,00	7,37	19995	0,35	0,35	11,00	**19925**	0,00	0,00	0,00	1,26
kroB150_4	12532	12532	**12532**	0,00	0,00	84,70	12587	0,44	0,44	9,00	**12532**	0,00	0,00	0,00	0,56
pr152_1	51806	51806	**51806**	0,00	0,00	297,67	51925	0,23	0,23	10,00	**51806**	0,00	0,00	0,16	27,47
pr152_2	45810	45810	**45810**	0,00	0,00	721,07	45902	0,20	0,20	19,00	**45810**	0,00	0,00	0,02	9,82
pr152_3	64425	64425	**64425**	0,00	0,00	591,80	64757	0,52	0,52	8,00	**64425**	0,00	0,00	0,04	25,02
pr152_4	57337	57337	**57337**	0,00	0,00	974,75	58147	1,41	1,41	30,00	57365	0,05	0,05	0,41	0,84
u159_1	29821	29821	**29821**	0,00	0,00	420,29	30035	0,72	0,72	19,00	**29821**	0,00	0,00	0,00	2,07
u159_2	23404	23401	**23401**	0,00	-0,01	781,11	26760	14,35	14,34	19,00	**23401**	0,00	-0,01	0,24	49,39
u159_3	36399	36399	**36399**	0,00	0,00	50,29	36597	0,54	0,54	10,00	**36399**	0,00	0,00	0,00	0,72
u159_4	30845	30845	**30845**	0,00	0,00	528,17	31057	0,69	0,69	37,00	30865	0,06	0,06	0,25	23,42
rat195_1	1285	1274	**1274**	0,00	-0,86	6758,69	1300	2,04	1,17	146,00	1278	0,31	-0,54	-0,36	27,48
rat195_2	912	912	**912**	0,00	0,00	345,14	921	0,99	0,99	21,00	916	0,44	0,44	0,50	6,54
rat195_3	1814	1797	**1797**	0,00	-0,94	1669,96	1849	2,89	1,93	14,00	1801	0,22	-0,72	-0,24	55,19
rat195_4	1320	1315	**1315**	0,00	-0,38	1055,73	1330	1,14	0,76	124,00	1317	0,15	-0,23	0,70	17,21
d198_1	10945	10945	**10945**	0,00	0,00	1186,03	10988	0,39	0,39	27,00	10951	0,05	0,05	0,21	3,45
d198_2	10212	10212	**10212**	0,00	0,00	1998,06	10237	0,24	0,24	35,00	10223	0,11	0,11	0,13	13,79
d198_3	13843	13843	**13843**	0,00	0,00	275,53	13967	0,90	0,90	17,00	**13843**	0,00	0,00	0,12	81,90
d198_4	12418	12418	**12418**	0,00	0,00	697,22	12429	0,09	0,09	46,00	**12418**	0,00	0,00	0,01	19,21
kroA200_1	16441	16432	**16432**	0,00	-0,05	887,20	16767	2,04	1,98	19,00	**16432**	0,00	-0,05	-0,05	2,61
kroA200_2	12416	12416	**12416**	0,00	0,00	1076,25	12973	4,49	4,49	22,00	**12416**	0,00	0,00	0,16	21,81
kroA200_3	24471	24252	**24252**	0,00	-0,89	3985,73	24609	1,47	0,56	1814,00	**24297**	0,19	-0,71	-0,52	23,33
kroA200_4	16518	16518	**16518**	0,00	0,00	269,31	16712	1,17	1,17	16,00	**16518**	0,00	0,00	0,00	32,47
kroB200_1	17527	17527	**17527**	0,00	0,00	986,88	17755	1,30	1,30	23,00	17587	0,34	0,34	0,35	5,21
kroB200_2	12881	12881	**12881**	0,00	0,00	1167,84	13221	2,64	2,64	22,00	12912	0,24	0,24	0,67	10,68
kroB200_3	24088	24047	**24047**	0,00	-0,17	738,97	24542	2,06	1,88	178,00	**24047**	0,00	-0,17	0,12	52,68

Continua na próxima página

Tabela 4.2 – *continua na próxima página*

Instância	*BKS*	P-B&C					B&C Híbrido[1]				BRKGA-QL				
		LB	*UB*	*gap*	*RPD*	*t (s)*	*Best value*	*gap*	*RPD*	*t (s)*	*Best value*	*gap*	*RPD*	*ARPD*	*t (s)*
kroB200_4	17583	17583	**17583**	0,00	0,00	1153,84	17605	0,13	0,13	15,00	17592	0,05	0,05	0,06	2,77
gr202_1	23394	23394	**23394**	0,00	0,00	620,93	23604	0,90	0,90	64,00	23508	0,49	0,49	0,91	22,16
gr202_2	14957	14957	**14957**	0,00	0,00	951,33	15018	0,41	0,41	66,00	**14957**	0,00	0,00	0,11	10,73
gr202_3	34547	34547	**34547**	0,00	0,00	342,05	34746	0,58	0,58	230,00	34562	0,04	0,04	0,86	54,47
gr202_4	28039	28025	**28025**	0,00	-0,05	1181,32	28285	0,93	0,88	39,00	**28025**	0,00	-0,05	0,16	24,28
pr226_1	52109	52109	**52109**	0,00	0,00	2534,35	52286	0,34	0,34	77,00	**52109**	0,00	0,00	0,00	0,47
pr226_2	47585	47585	**47585**	0,00	0,00	6662,67	47628	0,09	0,09	29,00	**47585**	0,00	0,00	0,00	0,46
pr226_3	66812	66812	**66812**	0,00	0,00	168,04	67286	0,71	0,71	20,00	**66812**	0,00	0,00	0,00	3,37
pr226_4	51905	51905	**51905**	0,00	0,00	2047,74	**51905**	0,00	0,00	18,00	**51905**	0,00	0,00	0,00	0,60
gr229_1	70741	70725	**70725**	0,00	-0,02	1174,76	71612	1,25	1,23	366,00	**70725**	0,00	-0,02	0,60	27,59
gr229_2	31653	31653	**31653**	0,00	0,00	1222,28	31789	0,43	0,43	40,00	**31653**	0,00	0,00	0,15	60,88
gr229_3	102841	102841	**102841**	0,00	0,00	353,71	105463	2,55	2,55	37,00	**102841**	0,00	0,00	0,60	47,62
gr229_4	46231	46231	**46231**	0,00	0,00	263,64	46522	0,63	0,63	26,00	46337	0,23	0,23	1,03	48,97
gil262_1	1526	1519	**1519**	0,00	-0,46	4193,47	1526	0,46	0,00	197,00	1525	0,39	-0,07	0,25	31,97
gil262_2	1069	1069	**1069**	0,00	0,00	3523,62	1077	0,75	0,75	75,00	1075	0,56	0,56	1,14	4,92
gil262_3	1988	1976	**1976**	0,00	-0,60	1711,25	1988	0,61	0,00	1822,00	1977	0,05	-0,55	-0,41	41,83
gil262_4	1671	1661	**1661**	0,00	-0,60	3194,70	1671	0,60	0,00	1827,00	1672	0,66	0,06	0,08	32,04
pr264_1	33904	33904	**33904**	0,00	0,00	5352,92	34039	0,40	0,40	192,00	34005	0,30	0,30	0,53	30,67
pr264_2	28748	28492	**28492**	0,00	-0,89	7576,78	28778	1,00	0,10	66,00	28505	0,05	-0,85	-0,67	63,29
pr264_3	40705	40705	**40705**	0,00	0,00	907,08	41049	0,85	0,85	30,00	**40705**	0,00	0,00	0,03	94,05
pr264_4	35153	35153	**35153**	0,00	0,00	3846,39	35325	0,49	0,49	201,00	35177	0,07	0,07	0,25	6,36
média				0,00	-0,09	1236,96		1,08	0,99	129,80		0,09	-0,01	0,17	19,79

[1](BERNARDINO; PAIAS, 2021b)

No conjunto de instâncias TSP simétricas (Tabela 4.2), o P-B&C foi capaz de provar a otimalidade em todas as instâncias, inclusive nas 15 instâncias que não havia valor ótimo conhecido na literatura. A média de tempo computacional foi de 1236,96 segundos. O BRKGA-QL encontrou a solução ótima em 39 instâncias (61%). Em comparação com o B&C híbrido (BERNARDINO; PAIAS, 2021b), BRKGA-QL encontrou soluções melhores em 55 instâncias, a mesma solução em 8 instâncias, e apenas em uma instância encontrou uma solução pior que o híbrido B&C (instância gil262_4, mas apenas por uma unidade). O *gap* médio do BRKGA-QL foi de 0,09% enquanto o *gap* médio do algoritmo B&C híbrido foi de 1,08%.

No conjunto de instâncias TSP assimétricas (Tabela 4.3), o algoritmo P-B&C proposto foi capaz de provar a otimalidade em todas essas instâncias. Neste conjunto havia apenas uma instância com valor ótimo desconhecido na literatura (rbg443_2). Para esta instância, o P-B&C encontrou a solução ótima (156) em 863 segundos. O BRKGA-QL encontrou a melhor solução (192) em 23 segundos, enquanto a melhor solução do Algoritmo Híbrido foi 350. A média de tempo computacional do algoritmo P-B&C foi 58,11 segundos enquanto do Algoritmo Híbrido foi de 496,51 segundos. O BRKGA-QL foi capaz de encontrar a solução ótima de 62 instâncias desse conjunto (79,5%) e o *gap* médio foi de 4,89%. O Algoritmo Híbrido não foi capaz de provar a otimalidade apenas para uma instância desse conjunto (rbg443_2) e apresentou *gap* médio de 3,71%.

TABELA 4.3 – Resultados Computacionais do FTSP para o conjunto de instâncias TSP Assimétricas.

Instância	*BKS*	P-B&C				B&C Híbrido[1]			BRKGA-QL		
		LB	*UB*	*gap*	*t(s)*	*UB*	*gap*	*t(s)*	*Best value*	*gap*	*t(s)*
br17_3_1	31	31	**31**	0,00	0,00	**31**	0,00	0,00	**31**	0,00	0,03
br17_3_2	28	28	**28**	0,00	0,00	**28**	0,00	0,00	**28**	0,00	0,02
br17_3_3	39	39	**39**	0,00	0,00	**39**	0,00	0,00	**39**	0,00	0,02
br17_3_4	36	36	**36**	0,00	0,00	**36**	0,00	0,00	**36**	0,00	0,03
ftv33_5_1	868	868	**868**	0,00	0,00	**868**	0,00	0,00	**868**	0,00	0,11
ftv33_5_2	401	401	**401**	0,00	0,00	**401**	0,00	1,00	**401**	0,00	0,07
ftv33_5_3	1286	1286	**1286**	0,00	0,00	**1286**	0,00	0,00	**1286**	0,00	0,04
ftv33_5_4	829	829	**829**	0,00	0,00	**829**	0,00	0,00	**829**	0,00	0,04
ftv35_5_1	1008	1008	**1008**	0,00	0,00	**1008**	0,00	0,00	**1008**	0,00	0,05
ftv35_5_2	530	530	**530**	0,00	1,00	**530**	0,00	1,00	**530**	0,00	0,09
ftv35_5_3	1232	1232	**1232**	0,00	0,00	**1232**	0,00	0,00	**1232**	0,00	0,08
ftv35_5_4	1008	1008	**1008**	0,00	0,00	**1008**	0,00	0,00	**1008**	0,00	0,04
ftv38_5_1	830	830	**830**	0,00	0,00	**830**	0,00	0,00	**830**	0,00	0,12
ftv38_5_2	392	392	**392**	0,00	1,00	**392**	0,00	1,00	**392**	0,00	0,12
ftv38_5_3	1449	1449	**1449**	0,00	0,00	**1449**	0,00	0,00	**1449**	0,00	0,07
ftv38_5_4	774	774	**774**	0,00	0,00	**774**	0,00	0,00	**774**	0,00	0,07
p43_6_1	5483	5483	**5483**	0,00	0,00	**5483**	0,00	0,00	**5483**	0,00	0,08
p43_6_2	5473	5473	**5473**	0,00	1,00	**5473**	0,00	1,00	**5473**	0,00	0,13
p43_6_3	5530	5530	**5530**	0,00	0,00	**5530**	0,00	0,00	**5530**	0,00	0,10
p43_6_4	5492	5492	**5492**	0,00	0,00	**5492**	0,00	0,00	**5492**	0,00	0,11
ftv44_6_1	996	996	**996**	0,00	0,00	**996**	0,00	1,00	**996**	0,00	00,12
ftv44_6_2	625	625	**625**	0,00	1,00	**625**	0,00	0,00	**625**	0,00	00,13
ftv44_6_3	1343	1343	**1343**	0,00	1,00	**1343**	0,00	0,00	**1343**	0,00	0,13
ftv44_6_4	998	998	**998**	0,00	1,00	**998**	0,00	1,00	**998**	0,00	0,059
ftv47_6_1	1179	1179	**1179**	0,00	0,00	**1179**	0,00	0,00	**1179**	0,00	0,152
ftv47_6_2	729	729	**729**	0,00	1,00	**729**	0,00	1,00	**729**	0,00	0,184
ftv47_6_3	1472	1472	**1472**	0,00	0,00	**1472**	0,00	1,00	**1472**	0,00	4,41
ftv47_6_4	1099	1099	**1099**	0,00	0,00	**1099**	0,00	0,00	**1099**	0,00	0,124
ry48p_48_6_1	10318	10318	**10318**	0,00	1,00	**10318**	0,00	1,00	**10318**	0,00	0,387
ry48p_48_6_2	6787	6787	**6787**	0,00	2,00	**6787**	0,00	1,00	**6787**	0,00	0,31
ry48p_48_6_3	12752	12752	**12752**	0,00	1,00	**12752**	0,00	1,00	**12752**	0,00	11,22
ry48p_48_6_4	10139	10139	**10139**	0,00	1,00	**10139**	0,00	1,00	**10139**	0,00	3,54
ft53_7_1	3572	3572	**3572**	0,00	1,00	**3572**	0,00	0,00	**3572**	0,00	1,81
ft53_7_2	2819	2819	**2819**	0,00	4,00	**2819**	0,00	2,00	**2819**	0,00	9,77
ft53_7_3	5972	5972	**5972**	0,00	1,00	**5972**	0,00	1,00	**5972**	0,00	9,67
ft53_7_4	4734	4734	**4734**	0,00	1,00	**4734**	0,00	0,00	**4734**	0,00	0,396
ftv55_7_1	570	570	**570**	0,00	2,00	**570**	0,00	1,00	**570**	0,00	0,127
ftv55_7_2	365	365	**365**	0,00	3,00	**365**	0,00	1,00	**365**	0,00	0,133
ftv55_7_3	1021	1021	**1021**	0,00	0,00	**1021**	0,00	1,00	**1021**	0,00	0,103
ftv55_7_4	579	579	**579**	0,00	1,00	**579**	0,00	1,00	**579**	0,00	0,15
ftv64_8_1	977	977	**977**	0,00	2,00	**977**	0,00	2,00	**977**	0,00	12,14
ftv64_8_2	660	660	**660**	0,00	5,00	**660**	0,00	4,00	**660**	0,00	0,129
ftv64_8_3	1617	1617	**1617**	0,00	0,00	**1617**	0,00	1,00	**1617**	0,00	0,546
ftv64_8_4	1515	1515	**1515**	0,00	1,00	**1515**	0,00	5,00	**1515**	0,00	0,181
ft70_8_1	21226	21226	**21226**	0,00	1,00	**21226**	0,00	1,00	**21226**	0,00	9,85
ft70_8_2	15360	15360	**15360**	0,00	2,00	**15360**	0,00	1,00	**15360**	0,00	0,503
ft70_8_3	29573	29573	**29573**	0,00	1,00	**29573**	0,00	1,00	**29603**	0,10	25,93
ft70_8_4	26817	26817	**26817**	0,00	1,00	**26817**	0,00	1,00	**26817**	0,00	17,01
ftv70_9_1	849	849	**849**	0,00	7,00	**849**	0,00	13,00	**849**	0,00	21,61
ftv70_9_2	676	676	**676**	0,00	9,00	**676**	0,00	10,00	**676**	0,00	5,387
ftv70_9_3	1342	1342	**1342**	0,00	2,00	**1342**	0,00	1,00	**1342**	0,00	16,25
ftv70_9_4	1261	1261	**1261**	0,00	1,00	**1261**	0,00	1,00	**1261**	0,00	0,376

Continua na próxima página

Tabela 4.3 – *continua na próxima página*

Instância	*BKS*	P-B&C				B&C Híbrido[1]				BRKGA-QL	
		LB	*UB*	*gap*	*t(s)*	*UB*	*gap*	*t(s)*	*Best value*	*gap*	*t(s)*
kro124p_14_1	25251	25251	**25251**	0,00	5,00	**25251**	0,00	9,00	**25251**	0,00	10,30
kro124p_14_2	12421	12421	**12421**	0,00	36,00	**12421**	0,00	38,00	**12421**	0,00	42,26
kro124p_14_3	32025	32025	**32025**	0,00	5,00	**32025**	0,00	2,00	**32025**	0,00	3,56
kro124p_14_4	18552	18552	**18552**	0,00	18,00	**18552**	0,00	40,00	**18552**	0,00	12,49
ftv170_19_1	1652	1652	**1652**	0,00	159,00	**1652**	0,00	5781,00	**1652**	0,00	20,44
ftv170_19_2	1108	1108	**1108**	0,00	901,00	**1108**	0,00	9071,00	**1108**	0,00	13,36
ftv170_19_3	2215	2215	**2215**	0,00	46,00	**2215**	0,00	19,00	**2215**	0,00	36,05
ftv170_19_4	1610	1610	**1610**	0,00	56,00	**1610**	0,00	141,00	**1610**	0,00	44,61
rbg323_34_1	337	337	**337**	0,00	20,00	**337**	0,00	36,00	350	3,86	257,68
rbg323_34_2	70	70	**70**	0,00	376,00	**70**	0,00	6684,00	99	41,43	146,76
rbg323_34_3	822	822	**822**	0,00	8,00	**822**	0,00	11,00	825	0,36	281,56
rbg323_34_4	335	335	**335**	0,00	41,00	**335**	0,00	31,00	353	5,37	244,02
rbg358_49_1	209	209	**209**	0,00	429,00	**209**	0,00	916,00	263	25,84	310,78
rbg358_49_2	50	50	**50**	0,00	278,00	**50**	0,00	328,00	80	60,00	265,43
rbg358_49_3	658	658	**658**	0,00	8,00	**658**	0,00	8,00	666	1,22	326,38
rbg358_49_4	409	409	**409**	0,00	21,00	**409**	0,00	73,00	412	0,73	325,31
rbg403_50_1	345	345	**345**	0,00	21,00	**345**	0,00	17,00	376	8,99	385,27
rbg403_50_2	32	32	**32**	0,00	534,00	**32**	0,00	1316,00	94	193,75	284,82
rbg403_50_3	1427	1427	**1427**	0,00	33,00	**1427**	0,00	134,00	1438	0,77	383,09
rbg403_50_4	486	486	**486**	0,00	17,00	**486**	0,00	12,00	499	2,67	389,57
rbg443_64_1	539	539	**539**	0,00	18,00	**539**	0,00	28,00	553	2,60	421,20
rbg443_64_2	156	156	**156**	0,00	863,00	596	282,05	10800,00	192	23,08	378,87
rbg443_64_3	1353	1353	**1353**	0,00	37,00	**1353**	0,00	42,00	1356	0,22	415,18
rbg443_64_4	729	729	**729**	0,00	429,00	**729**	0,00	2138,00	731	0,27	407,97
Média				0,00	58,11		3,71	496,51		4,89	73,17

[1] (BERNARDINO; PAIAS, 2021b)

A Tabela 4.4 descreve os resultados computacionais para o conjunto de instâncias de *benchmark* obtidos pelos métodos: BRKGA-QL, BRKGA (MORÁN-MIRABAL *et al.*, 2014), GRASP (MORÁN-MIRABAL *et al.*, 2014), GA (BERNARDINO; PAIAS, 2021b) e ILS (BERNARDINO; PAIAS, 2021b). Em comparação aos demais métodos testados para o FTSP, pode-se observar que o BRKGA-QL apresenta melhor desempenho. Visto que o BRKGA-QL encontrou resultados iguais em 10 instâncias, e melhorou em oito instâncias em relação ao BKS, enquanto os demais métodos conseguiram, em algumas instâncias, chegar aos mesmos resultados (valores em negrito), contudo não foram capazes de encontrar soluções melhores. O BRKGA-QL conseguiu melhorar os valores dos *BKS* em média 0,88% neste conjunto.

A Tabela 4.5 descreve os resultados computacionais para o conjunto de instâncias TSP simétricas cujos respectivos valores ótimos não haviam sido provados, obtidos pelos métodos: BRKGA-QL, GA (BERNARDINO; PAIAS, 2021b) e ILS (BERNARDINO; PAIAS, 2021b). Os métodos BRKGA e GRASP (ambos propostos em Morán-Mirabal *et al.* (2014)) somente foram testados para as instâncias de *benchmark*. Outro ponto a ser ressaltado, é que para o conjunto de instâncias simétricas com valor ótimo conhecido, o único método

que tem resultados disponíveis na literatura é o algoritmo híbrido de Bernardino e Paias (2021b). Para esse conjunto com 15 instâncias, o BRKGA-QL encontrou soluções melhores em 11 instâncias, a mesma solução em 3 instâncias, e apenas em uma instância encontrou uma solução pior que o *BKS*. O tempo computacional médio do BRKGA-QL foi de 30,53 segundos. O método GA não foi capaz de encontrar soluções melhores ou iguais ao *BKS* neste conjunto de instâncias, enquanto o método ILS conseguiu encontrar a mesma solução do *BKS* em duas instâncias apenas. O *RPD* médio do BRKGA-QL foi de -0,26% enquanto o *RPD* médio GA foi de 3,41% e do ILS de 2,38%.

TABELA 4.4 – Comparação entre os resultados computacionais do BRKGA-QL com os métodos BRKGA (MORÁN-MIRABAL *et al.*, 2014), GRASP (MORÁN-MIRABAL *et al.*, 2014), GA (BERNARDINO; PAIAS, 2021b) e ILS (BERNARDINO; PAIAS, 2021b) para o conjunto de instâncias *Benchmark*.

		BRKGA-QL			BRKGA[1]			GRASP[1]			GA[2]			ILS[2]		
Instância	*BKS*	*Best Value*	*RPD*	*t(s)*	*Best Value*	*RPD*	*t(s)*	*Best Value*	*RPD*	*t(s)*	*Best Value*	*RPD*	*t(s)*	*Best Value*	*RPD*	*t(s)*
burma14_1	13,93	**13,93**	0,00	0,03	**13,93**	0,00	0,01	**13,93**	0,00	0,01	**13,93**	0,00	0,00	**13,93**	0,00	0,00
burma14_2	25,66	**25,66**	0,00	0,04	**25,66**	0,00	0,01	**25,66**	0,00	0,03	**25,66**	0,00	0,00	**25,66**	0,00	0,00
burma14_3	11,89	**11,89**	0,00	0,03	**11,89**	0,00	0,01	**11,89**	0,00	0,01	**11,89**	0,00	0,00	**11,89**	0,00	0,00
bayg29_1	5345,89	**5345,86**	0,00	0,10	**5345,89**	0,00	3,40	**5345,89**	0,00	8,05	**5345,89**	0,00	3,40	**5345,86**	0,00	0,00
bayg29_2	5791,01	**5791,01**	0,00	0,08	**5791,01**	0,00	1,40	**5791,01**	0,00	75,74	**5791,01**	0,00	1,40	**5791,01**	0,00	0,00
bayg29_3	5544,33	**5544,33**	0,00	0,09	**5544,33**	0,00	1,60	**5544,33**	0,00	0,03	**5544,33**	0,00	1,60	**5544,33**	0,00	0,00
att48_1	23686	**23686,02**	0,00	0,15	**23686**	0,00	143,40	**23686**	0,00	2938,77	**23686,02**	0,00	143,40	**23686**	0,00	0,00
att48_2	20609,1	**20609,09**	0,00	0,11	**20609,1**	0,00	62,8	20635,57	0,13	7199,22	**20609,09**	0,00	62,80	**20609,1**	0,00	0,00
att48_3	9024,58	**9024,58**	0,00	0,14	**9024,58**	0,00	1,80	**9024,5**	0,00	0,05	**9024,58**	0,00	1,80	**9024,58**	0,00	0,00
bier127_1	33709,7	**33709,75**	0,00	41,81	36913,74	209,50	498,8	36800,39	9,17	3,65	36913,74	9,50	498	**33709,7**	0,00	4,00
bier127_2	88736,4	88761,09	0,03	26,08	98216,1	210,68	1413,8	97615,41	10,01	2966,56	98216,1	10,68	1413,8	**88736,4**	0,00	4
bier127_3	47726,3	47873,62	0,31	20,83	50513,1	205,84	1048,6	50715,49	6,26	11,26	50513,1	5,84	1048,6	**47726,3**	0,00	3,00
a280_1	1739,3	**1728,02**	-0,65	58,57	2126,34	222,25	14760,4	1891,16	8,73	218,28	1824,04	4,87	10,00	1769,33	1,73	45
a280_2	1556,08	**1551,96**	-0,26	16,54	1925,28	223,73	14759,2	1697,48	9,09	2700,88	1642,87	5,58	12,00	1562,14	0,39	45,00
a280_3	1408,14	1431,4	1,65	55,4	1720,23	222,16	14758,4	1597,25	13,43	6,91	1720,23	22,16	14758,4	1408,14	0,00	36,00
gr666_1	1531,93	**1507,18**	-1,62	96,41	2625,69	271,40	22271,6	1817,06	18,61	661022	1579,93	3,13	84,00	1554,86	1,50	721
gr666_2	1236,21	**1222,33**	-1,12	80,71	2275,8	284,09	22269	1443,05	16,73	4005,06	1310,33	6,00	86,00	1236,21	0,00	700,00
gr666_3	1225,61	**1182,99**	-3,48	96,92	2426,59	297,99	22269,75	1384,18	12,94	7199,99	1325,11	8,12	88	1235,48	0,81	752
pr1002_1	132821,4	**128772,19**	-3,05	336,76	421061,63	417,01	31568	163461,79	23,07	20,72	136796	2,99	237,00	**132821,4**	0,00	2037,00
pr1002_2	144543,77	**140129,05**	-3,05	495,61	421761	391,79	31566,75	182144,13	26,01	8,86	151682	4,94	257,00	**144543,77**	0,00	2013,00
pr1002_3	126282,93	**117138,23**	-7,24	461,20	284856,22	325,57	31565,50	149456,63	18,35	227,56	126307	0,02	230,00	128367,78	1,65	2037,00
média			-0,88	85,12		156,29	9950,68		8,22	32791,12		3,99	901,77		0,29	399,86

[1](MORÁN-MIRABAL *et al.*, 2014) e [2](BERNARDINO; PAIAS, 2018)

TABELA 4.5 – Comparação dos resultados computacionais dos métodos BRKGA-QL, GA e ILS (MORÁN-MIRABAL *et al.*, 2014) para o conjunto de instâncias TSP Simétricas.

		BRKGA-QL			GA[1]			ILS[1]		
Instância	BKS	*Best value*	*RPD*	*t(s)*	*Best value*	*RPD*	*t(s)*	*Best value*	*RPD*	*t(s)*
pr144_4	49379	**49379**	0,00	0,6	49763	0,78	3	**49379**	0,00	6
kroA150_3	20880	**20880**	0,00	32,86	21748	4,16	4	21459	2,77	8
pr152_3	64425	**64425**	0,00	25,02	65743	2,05	3	65339	1,42	7
rat195_1	1285	**1278**	-0,54	27,48	1321	2,80	4	1303	1,40	14
rat195_3	1814	**1801**	-0,72	55,19	1867	2,92	4	1838	1,32	10
rat195_4	1320	**1317**	-0,23	17,21	1383	4,77	4	1342	1,67	11
kroA200_1	16441	**16432**	-0,05	2,61	17176	4,47	5	17134	4,22	15
kroA200_3	24471	**24297**	-0,71	23,33	24898	1,74	2	25190	2,94	11
kroB200_3	24088	**24047**	-0,17	52,68	25250	4,82	5	25225	4,72	10
gr202_4	28039	**28025**	-0,05	24,28	28634	2,12	4	28846	2,88	13
gr229_1	70741	**70725**	-0,02	27,59	74768	5,69	7	72288	2,19	28
gil262_1	1526	**1525**	-0,07	31,97	1601	4,91	6	1567	2,69	26
gil262_3	1988	**1977**	-0,55	41,83	2085	4,88	8	2067	3,97	19
gil262_4	1671	1672	0,06	32,04	1722	3,05	7	1731	3,59	23
pr264_2	28748	**28505**	-0,85	63,29	29322	2,00	7	28748	0,00	28
média			-0,26	30,53		3,41	4,87		2,38	15,27

[1](MORÁN-MIRABAL *et al.*, 2014)

4.3 Comparação de Desempenhos dos Algoritmos

A Figura 4.1 apresenta o perfil de desempenho do algoritmo BRKGA-QL, P-B&C e B&C híbrido (BERNARDINO; PAIAS, 2021b) com os RPDs dos conjuntos *Benchmark* e TSP Simétrico. O perfil de desempenho mostra a porcentagem de problemas resolvidos por um algoritmo s baseado em um fator de desempenho $\tau \in \mathbb{R}$, que compara o algoritmo analisado com o melhor resultado obtido. O eixo y representa a probabilidade acumulada, $\phi s(\tau) = P(r_{ps} \leq \tau | 1 < s < n_s)$, considerando a taxa de desempenho r_{ps} associada ao algoritmo s estar contido no fator de desempenho τ indicado no eixo x. Assim, $\tau = 1$ (posição 0 no eixo x, $2^0 = 1$) indica a porcentagem de instâncias para as quais o algoritmo s obtém as melhores soluções. Por outro lado, $\phi s = 1$ representa o fator de desempenho τ para que o algoritmo alcance os melhores resultados. Pode-se observar que a curva do perfil de desempenho do P-B&C domina as curvas do algoritmo BRKGA-QL e B&C híbrido. O algoritmo P-B&C apresentou os melhores resultados em 96% das instâncias e atingiu 100% das instâncias com $\tau = 2^{1,1}$. O método BRKGA-QL teve o segundo melhor desempenho, apresentando os melhores resultados em 62% das instâncias e atingindo 100% das instâncias com $\tau = 2^{4,4}$. O algoritmo híbrido B&C obteve as melhores soluções em 19% das instâncias, e somente com um fator de desempenho de $\tau = 2^{7,2}$ alcançou as melhores soluções.

Para comparar a qualidade das soluções obtidas pelos algoritmos P-B&C, BRKGA-QL e B&C híbrido, foi aplicado o teste *Wilcoxon signed-rank* (WSR) (REY; NEUHÄUSER, 2011), um teste de hipótese estatística não paramétrica. O WSR indicou que houve uma

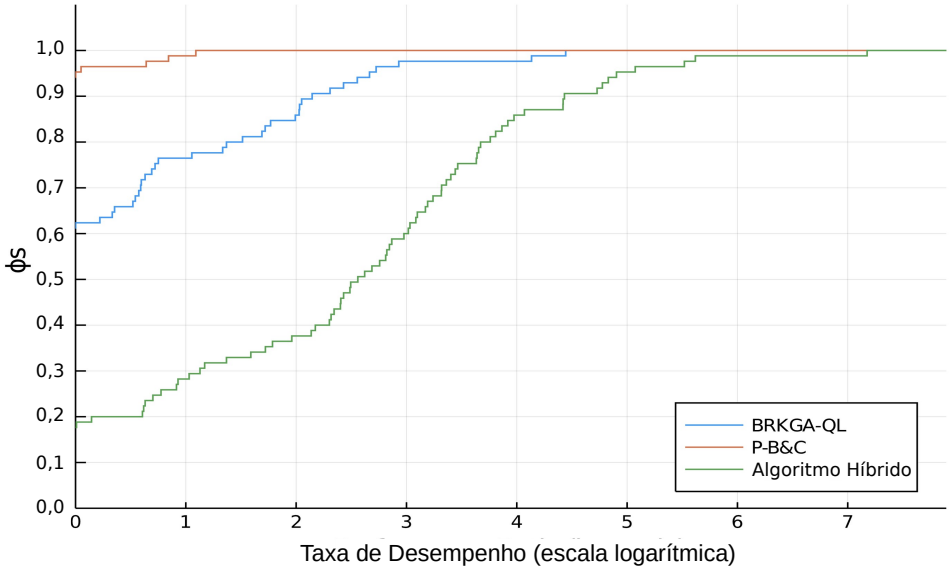

FIGURA 4.1 – *Performance profile* comparando os *gaps* obtidos pelos métodos P-B&C, BRKGA-QL e algoritmo B&C Híbrido.

diferença estatisticamente significativa entre P-B&C e BRKGA-QL (p-valor = 0,00019), P-B&C e B&C híbrido (p-valor = 2,95E-11), e entre BRKGA-QL e híbrido B&C (p-valor = 2,26E-13). Esses resultados sugerem que a técnica utilizada no P-B&C encontrou melhores soluções em termos de qualidade.

4.4 Análises do algoritmo P-B&C

Para evidenciar as contribuições da frente de busca local desenvolvida para o FTSP foi analisada a qualidade da matriz M após todas as atualizações terem sido realizadas para cada instância resolvida. Para tanto, a linha k de M correspondente ao valor da frequência máxima na diagonal foi selecionada, desempatando aleatoriamente. O valor acumulado de cada coluna (cliente) i correspondente a linha $M[k][i]$, foi dividido pelo total de atualizações realizadas ao longo da árvore B&B. Normalizando, dessa forma, todos os valores entre 0 e 1. Em seguida, dado um valor de erro máximo ε, foi calculada a porcentagem de clientes que satisfazem $d = |y_i - M[k][i]| < \varepsilon$ para todos os $i \in N$ e para todos os $\varepsilon \in [0, 1]$, onde y_i é o valor da variável de visita i em $\mathcal{S}_{Melhor}$. O valor d representa uma medida de similaridade entre y_i e $M[k][i]$ dado um valor de erro máximo (ε). Valores pequenos de d indicam uma forte semelhança entre y_i e $M[k][i]$.

Na Figura 4.2 evidencia-se o pior caso (em vermelho), o caso médio (em azul) e o melhor caso de similaridade (em verde), considerando todos os valores d de todas as instâncias simultaneamente, com ε $em[0, 1]$. O eixo x representa o valor máximo de erro e o eixo y indica a porcentagem de clientes que satisfazem $d < \varepsilon$. Pode-se ressaltar que no pior caso a similaridade é nula quando $\varepsilon = 0$, é máxima no melhor caso, e é maior que

25% em média. O valor $\varepsilon = 0$ representa um cenário estrito exigindo total igualdade entre y_i e $M[k][i]$. Adotando um cenário moderado com $\varepsilon = 0,5$, o pior caso tem similaridade próxima a 70%, e o caso médio apresenta similaridade maior que 90%. Ou seja, mais de 90% dos valores y_i em $\mathcal{S}_{Melhor}$ são semelhantes aos valores normalizados $M[k][i]$ com erro menor que ou igual a $0,5$. Assim, simplesmente arredondando os valores $M[k][i]$ para o inteiro mais próximo (0 ou 1) para todos os $i \in N$, espera-se atingir em média mais de 90% dos valores finais y_i em $\mathcal{S}_{Melhor}$. Na Figura 4.2 também estão apresentados (em linhas tracejadas) o pior e o melhor caso para todas as instâncias onde o tempo de processamento de P-B&C foi superior a 1200 segundos.

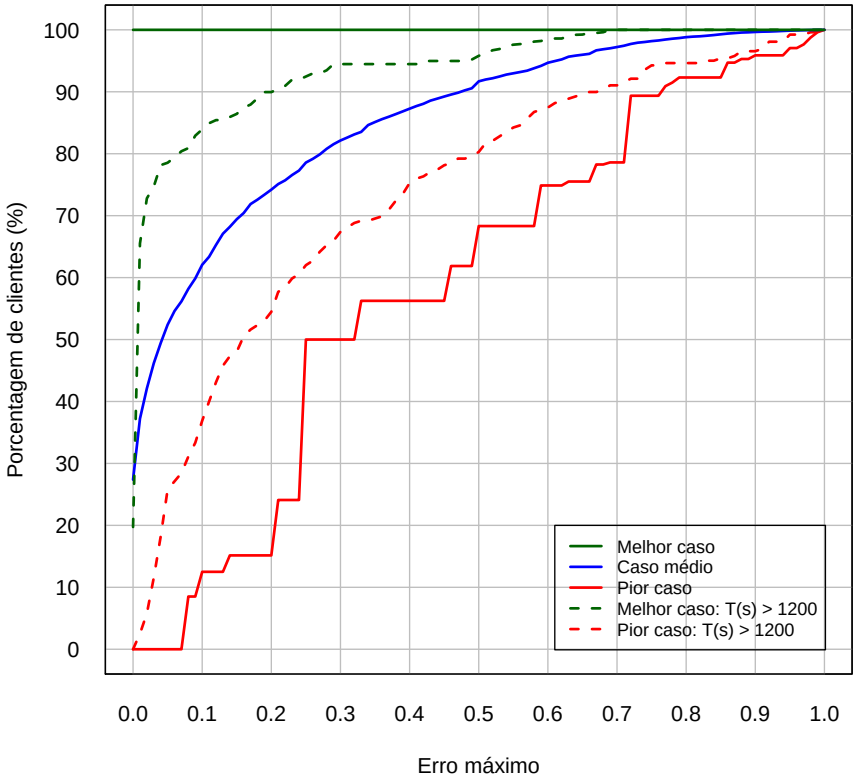

FIGURA 4.2 – Valores de similaridade entre a variável de visita y e a matriz M.

A Figura 4.3 apresenta uma descrição detalhada para a instância gr202_4. Esta instância contém 201 clientes, 30 famílias e requer 118 visitas. A otimalidade do gr202_4 foi comprovada exclusivamente pelo P-B&C com um tempo de processamento de $1181,32$ segundos e 8622 atualizações de M. Cada cliente de 1 a 201 é indicado no eixo x na Figura 4.3. Círculos azuis no eixo y com valores iguais a 0 denotam clientes não visitados ($y_i = 0$) e círculos com valores iguais a 1 representam clientes visitados ($y_i = 1$) em $\mathcal{S}_{Melhor}$. A linha vermelha exibe os valores normalizados de M para cada cliente. Observando a figura, pode-se notar que um cliente visitado está fortemente associado a grandes valores

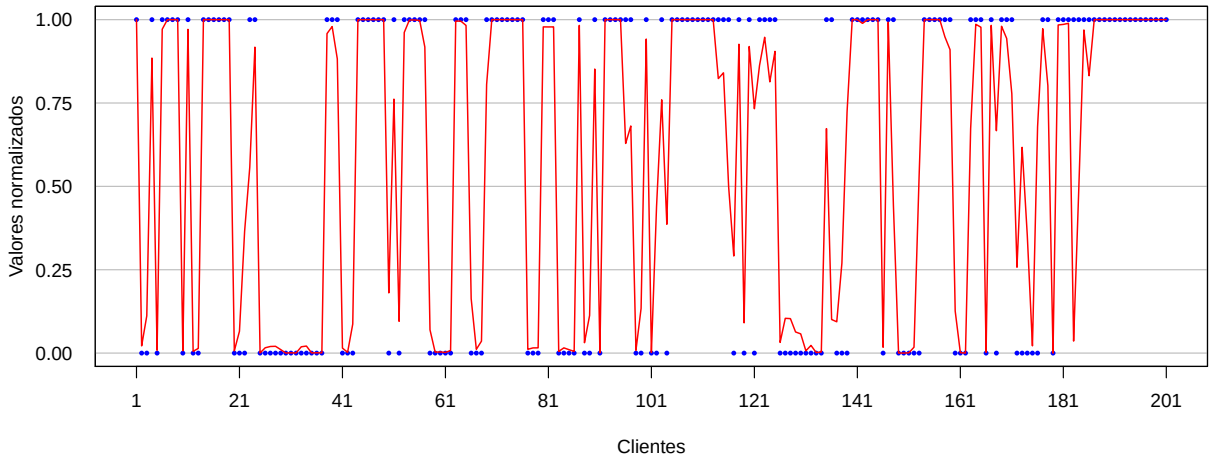

FIGURA 4.3 – Valores da Matriz M normalizados para a instância gr202_4.

em vermelho, clientes não visitados estão relacionados a pequenos valores em vermelho, e há poucos valores próximos a $0,5$. Essas características do M possibilitaram gerar bons grupos de clientes para a lista $\mathcal{T}$.

Por fim, destaca-se que a frente de busca local foi capaz de encontrar limites superiores ótimos para 93 das 185 instâncias testadas. Além disso, em 18 casos onde a otimalidade foi comprovada exclusivamente pelo P-B&C, a frente de Busca Local encontrou 11 valores ótimos. Destaca-se também que o procedimento de busca local proposto obteve soluções sub-ótimas para os 92 casos restantes, permitindo a poda antecipada de nós ativos na árvore B&B, reduzindo potencialmente o tempo de processamento necessário para resolver essas instâncias.

4.5 Efeitos dos componentes do BRKGA-QL

Para analisar os efeitos dos componentes do BRKGA-QL, foram comparados os seus resultados com as versões do BRKGA-QL sem busca local e apenas com um decodificador (Dec1). Além disso, para comparar os efeitos do módulo *Q-Learning*, o BRKGA foi calibrado através do *Bayesian Network Tuning* (BNT) (NASCIMENTO; CHAVES, 2020). O BNT retorna a melhor configuração dos parâmetros BRKGA considerando um subconjunto aleatório de instâncias e as opções listadas na Tabela 3.1. Este método também fornece uma rede de dependência estatística entre os parâmetros. A Figura 4.4 mostra a rede criada para o BRKGA. Pode-se observar que o único parâmetro independente que afeta todos os outros parâmetros é p_e, a proporção de indivíduos elite. Os parâmetros ρ_e e p podem ser alterados sem afetar os outros parâmetros e, finalmente, p_m impacta p e

é afetado por p_e. O BRKGA foi configurado com os parâmetros retornados pelo BNT e o componente *Q-Learning* foi retirado do processo evolutivo. Os parâmetros encontrados pelo BNT são:

- $p = 987$
- $p_e = 0{,}10$
- $p_m = 0{,}25$
- $\rho_e = 0{,}80$

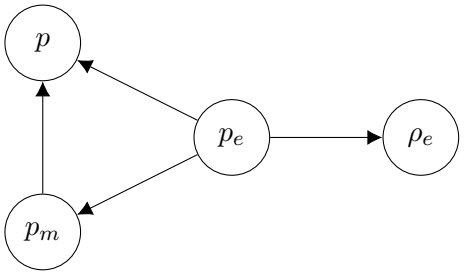

FIGURA 4.4 – Rede de dependência estatística entre os parâmetros do BRKGA.

As Tabelas 4.6 e 4.7 apresentam os resultados de diferentes versões do BRKGA-QL para analisar o impacto de cada componente do método proposto. Pode-se observar que o BRKGA-QL encontra melhores resultados do que as versões sem busca local e com um decodificador. Os resultados do BRKGA configurado com o BNT estão próximos dos resultados do BRKGA-QL. No entanto, o BRKGA-QL pode obter melhores resultados (*gaps* de 2,87% e 0,09% do BRKGA-QL *versus gaps* de 2,94% e 0,17% do BRKGA) e com significância estatística pelo WRS (*p-valor* = 0,0018). Também pode-se observar a influência do componente de busca local na convergência dos métodos. Os tempos computacionais para obter a melhor solução para métodos com busca local são 70% mais rápidos que os tempos computacionais do BRKGA-QL sem busca local.

TABELA 4.6 – Comparação dos efeito entre os componentes do BRKGA-QL.

Conjunto de Instâncias	Método	*gap*	*RPD*	*ARPD*	*t (s)*
Benchmark	BRKGA-QL	2,87	-0,88	-0,48	85,12
	Sem BL	3,75	-0,08	1,32	255,33
	Um *decoder*	3,93	0,06	0,47	79,62
	BRKGA	2,94	-0,81	-0,52	78,49
TSP Simétrica	BRKGA-QL	0,09	-0,01	0,17	19,79
	Sem BL	0,32	0,22	0,90	72,20
	Um *decoder*	0,26	0,17	0,52	19,74
	BRKGA	0,17	0,08	0,23	16,56

TABELA 4.7 – WRS – *p-values* entre as versões do BRKGA.

	Sem BL	Um *Decoder*	BRKGA
BRKGA-QL	2.95e-08	8.526e-05	0,0018

As Figuras 4.5 e 4.6 apresentam uma visão em escala log_2 dos perfis de desempenho das versões do BRKGA-QL. A tolerância dos resultados foi definida como 1%, ou seja, o tempo computacional de métodos com RPD maior que 1% foi definido como INF para mostrar que o método não satisfaz o teste de convergência para esta instância. Para este nível de precisão, observa-se na Figura 4.5 que BRKGA-QL resolve 96% das instâncias, enquanto BRKGA calibrado com BNT resolve 92% e BRKGA sem busca local e com um *decoder* resolve 85% das instâncias. O BRKGA calibrado é o método mais rápido em 64% das instâncias, BRKGA-QL é o mais rápido em 19% das instâncias. Em termos de *gap*, a Figura 4.6 mostra a porcentagem de instâncias para as quais um método obtém as melhores soluções. Pode-se observar que a curva do perfil de desempenho BRKGA-QL domina as demais versões. Uma curva dominar as outras sugere que este método teve uma maior proporção de instâncias resolvidas para qualquer fator de desempenho em comparação com os outros métodos.

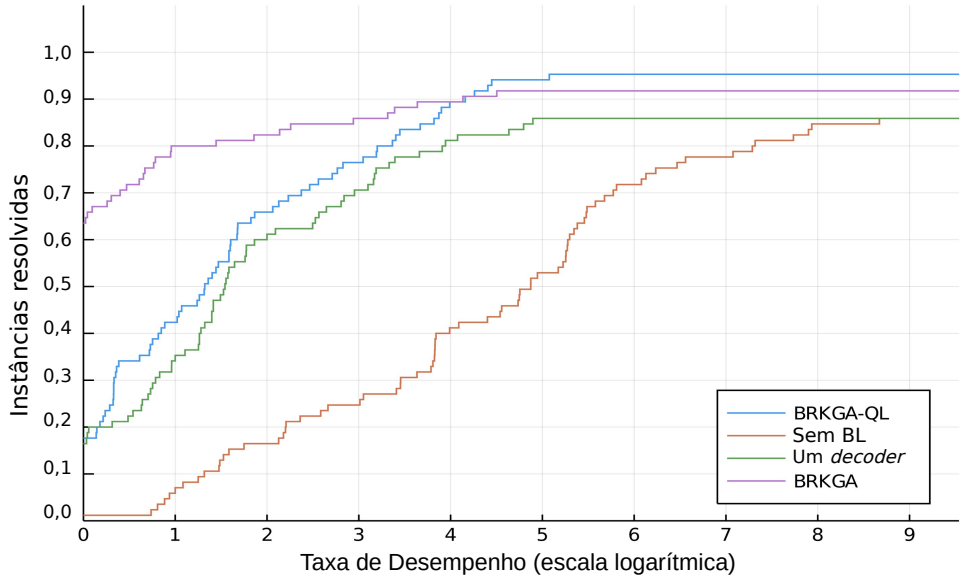

FIGURA 4.5 – *Performance Profile* comparando o tempo computacional das versões do BRKGA-QL.

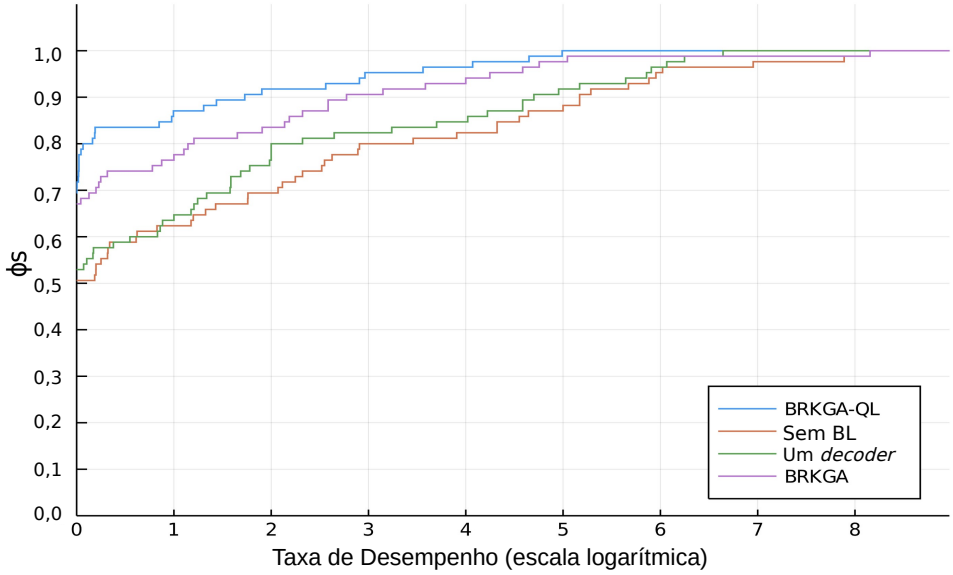

FIGURA 4.6 – *Performance profile* comparando os *gaps* obtidos pelas versões BRKGA-QL.

5 Conclusão

Este trabalho estudou o Problema do Caixeiro Viajante em Famílias (FTSP), uma variante do TSP, motivado pela criação de rotas otimizadas na coleta de pedidos de produtos similares armazenados separadamente em diferentes setores dos armazéns.

Para resolver o FTSP, foram propostos métodos exatos e heurísticos. Inicialmente, foi aplicado um algoritmo B&C paralelo (P-B&C) com um eficiente procedimento de busca local. O P-B&C foi capaz de encontrar a solução ótima em 179 das 185 instâncias de um conjunto de instâncias disponíveis na literatura. O algoritmo P-B&C provou a otimalidade em 18 de 24 (75%) das instâncias com valor ótimo desconhecido na literatura. A metaheurística BRKGA-QL, integrando BRKGA com algoritmo *Q-Learning*, foi desenvolvida para obter boas soluções em um tempo computacional competitivo. O BRKGA-QL foi capaz de encontrar as soluções ótimas ou próximas da ótima para as instâncias em que são conhecidas. Para quatro instâncias de maior escala (gr666_2, pr1002_1, pr1002_2 e pr1002_3) encontrou soluções melhores que P-B&C e a heurística estado-da-arte da literatura.

Também foi avaliado o desempenho dos componentes de nossos métodos propostos. Em P-B&C, a matriz M, que foi atualizada cumulativamente com as soluções encontradas através da árvore B&B, apresentou um desempenho satisfatório, obtendo uma similaridade em média maior que 90% em relação às melhores soluções quando um erro máximo de $0,5$ foi considerado. Os valores obtidos pela matriz M permitiram encontrar 93 limites superiores ótimos na frente de busca local, além de soluções sub-ótimas, diminuindo o tempo de processamento necessário para resolver as instâncias testadas.

O *framework* do BRKGA-QL funcionou melhor com todos os componentes: decodificador, busca local e Q-Learning. O controle online dos parâmetros permitiu um ajuste mais adequado às características das instâncias, e os decodificadores e heurísticas de busca local agregaram diversidade e intensificação ao processo de busca.

O BRKGA-QL é um bom exemplo de como podemos aplicar técnicas de inteligência artificial para obter soluções de maior qualidade em um *framework* heurístico e de forma eficiente.

5.1 Principais contribuições

As principais contribuições científicas deste trabalho são:

- O algoritmo P-B&C integrando o esquema tradicional B&C com um procedimento de busca local em um *framework* de processamento paralelizado;
- O algoritmo P-B&C com cortes sofisticados para encontrar soluções ótimas em um tempo limite;
- O método BRKGA-QL com diferentes decodificadores e procedimentos de busca local, de fácil desenvolvimento e configuração;
- Estratégias de atualização dos parâmetros QL para melhoria do BRKGA-QL;
- P-B&C conseguiu encontrar a solução ótima para 96,7% das instâncias disponíveis na literatura;
- BRKGA-QL encontrou as melhores soluções para instâncias de grande porte;

5.2 Impactos do Trabalho na Sociedade

Esta seção apresenta os impactos científicos, sociais, econômicos e ambientais deste trabalho.

O campo de Pesquisa Operacional é fundamental para o auxilio de processos de tomada de decisão através da utilização de ferramentas de otimização combinatória. Nesse contexto, a comunidade científica busca constantemente o aprimoramento das técnicas e métodos através de estudos eficientes para resolver problemas de alta complexidade, na busca de soluções com qualidade e tempo computacional aceitável. Este trabalho, no entanto, fornece uma comparação entre diferentes arranjos para a solução de um problema clássico logístico. Portanto o impacto científico desse trabalho está relacionado à base teórica que pode servir de referência para trabalhos futuros de exploração de dados e consequentemente favorecer o desenvolvimento da ciência.

Sob o viés econômico e social, o mundo alcançou o número de sete bilhões de pessoas, e consequentemente a produção e distribuição de bens de consumo e serviços aumentam em quantidades cada vez maiores. Tornando-se cada vez mais necessário, a implementação de estratégicas logísticas, de modo a proporcionar qualidade de vida à sociedade. A aplicação de métodos eficientes nos processos logísticos, é uma solução para combater custos desnecessários e integrar a sociedade ao sistema de produção e consumo estabelecidos. Portanto, o desenvolvimento de métodos de otimização eficazes são cruciais no

crescimento e desenvolvimento econômico de uma sociedade. Neste sentido, a comunidade científica tem um papel imprescindível na busca de soluções cada vez mais eficientes, através do uso da tecnologia e desenvolvimento de métodos inovadores, que sejam capazes de processar grandes quantidades de dados em segundos e oferecer soluções de qualidade e aplicáveis a um mercado altamente dinâmico.

Em relação aos impactos ambientais, o desenvolvimento de métodos adequados e eficientes na resolução de problemas logísticos, podem gerar uma redução significativa de consumo dos recursos ambientais. A otimização de rotas para transportes, por exemplo, podem levar a um significativo impacto ambiental no que tange a redução emissões de gases poluentes. A otimização de estoques e estratégias que permitam a flexibilidade nos sistemas de armazenamento logísticos, pode ter como consequência a redução de desperdícios de construção em áreas desnecessárias, como também do consumo de matéria-prima.

Referências

AIRES, C. S. F.; ALMEIDA, G.; SILVEIRA, S. O. Inteligência artificial na gestão de estoque. **Fateclog**, v. 1, p. 1–7, 2019.

AUGERAT, P.; BELENGUER, J. M.; BENAVENT, E.; CORBERÁN, A.; NADDEF, D. Separating capacity constraints in the CVRP using tabu search. **European Journal of Operational Research**, v. 106, n. 2–3, p. 546–557, 1998.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. **ORSA Journal on Computing**, v. 6, n. 2, p. 154–160, 1994. Disponível em: <https://doi.org/10.1287/ijoc.6.2.154>.

BERNARDINO, R.; PAIAS, A. Solving the family traveling salesman problem. **European Journal of Operational Research**, v. 267, n. 2, p. 453–466, 2018. ISSN 0377-2217.

BERNARDINO, R.; PAIAS, A. The family traveling salesman problem with incompatibility constraints. **Networks**, p. 1–36., 2021.

BERNARDINO, R.; PAIAS, A. Heuristic approaches for the family traveling salesman problem. **International Transactions in Operational Research**, v. 28, n. 1, p. 262–295, 2021.

BISHOP, C. M. Pattern recognition. **Machine learning**, v. 128, n. 9, 2006.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 35, n. 3, p. 268–308, set. 2003. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/937503.937505>.

BLUM, C.; ROLI, A. Hybrid metaheuristics: An introduction. In: _____. **Hybrid Metaheuristics: An Emerging Approach to Optimization**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. p. 1–30. ISBN 978-3-540-78295-7. Disponível em: <https://doi.org/10.1007/978-3-540-78295-7_1>.

BOCTOR, F. F.; LAPORTE, G.; RENAUD, J. Heuristics for the traveling purchaser problem. **Computers & Operations Research**, v. 30, n. 4, p. 491–504, 2003. ISSN 0305-0548. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0305054802000205>.

BODIN, L. Routing and scheduling of vehicles and crews, the state of the art. **Comput. Oper. Res.**, v. 10, n. 2, p. 63–211, 1983. Disponível em: <https://ci.nii.ac.jp/naid/80001882844/en/>.

CHAVES, A. A.; GONçALVES, J. F.; LORENA, L. A. N. Adaptive biased random-key genetic algorithm with local search for the capacitated centered clustering problem. **Computers & Industrial Engineering**, v. 124, n. C, p. 331–346, 2018.

CHAVES, A. A.; LORENA, L. A. N.; SENNE, E. L. F.; RESENDE, M. G. Hybrid method with cs and brkga applied to the minimization of tool switches problem. **Computers & Operations Research**, Elsevier, v. 67, p. 174–183, 2016.

CHAVES, A. A.; LORENA, L. H. N. An adaptive and near parameter-free brkga using q-learning method. In: **2021 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2021. p. 2331–2338.

CORDENONSI, A. Z. Ambientes, objetos e dialogicidade: Uma estratégia de ensino superior em heurísticas e metaheurísticas. 2008.

CROES, G. A. A method for solving traveling-salesman problems. **Operations Research**, v. 6, n. 6, p. 791–812, 1958. Disponível em: <https://doi.org/10.1287/opre.6.6.791>.

DANTZIG, G.; FULKERSON, R.; JOHNSON, S. Solution of a Large-Scale Traveling-Salesman Problem. **Journal of the Operations Research Society of America**, 1954. ISSN 0096-3984.

DORIGO, M. **Optimization, Learning and Natural Algorithms**. Tese (Doutorado) — Politecnico di Milano, Italy, 1992.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: a cooperative learning approach to the traveling salesman problem. **IEEE Transactions on evolutionary computation**, IEEE, v. 1, n. 1, p. 53–66, 1997.

ERICSSON, M.; RESENDE, M.; PARDALOS, P. A genetic algorithm for the weight setting problem in ospf routing. **Journal of Combinatorial Optimization**, v. 6, p. 299–333, 01 2002.

EVANS, B. P.; XUE, B.; ZHANG, M. An adaptive and near parameter-free evolutionary computation approach towards true automation in automl. **arXiv preprint arXiv:2001.10178**, 2020.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability; A Guide to the Theory of NP-Completeness**. USA: W. H. Freeman & Co., 1990. ISBN 0716710455.

GILMORE, R.; GOMORY, R. A linear programming approach to the cutting stock problem i. **Oper Res**, v. 9, 01 1961.

GLOVER, F. Future paths for integer programming and links to artificial intelligence. **Computers & Operations Research**, v. 13, n. 5, p. 533–549, 1986.

GOLDBARG, M. **Otimização Combinatória e Programação Linear**. [S.l.: s.n.], 2005. ISBN 85-352-1520-4.

GOLDBERG, D. E.; HOLLAND, J. H. Genetic algorithms and machine learning. Kluwer Academic Publishers-Plenum Publishers; Kluwer Academic Publishers ..., 1988.

GOLDEN, B.; NAJI-AZIMI, Z.; RAGHAVAN, S.; SALARI, M.; TOTH, P. The generalized covering salesman problem. **INFORMS Journal on Computing**, v. 24, n. 4, p. 534–553, 2012. Disponível em: <https://doi.org/10.1287/ijoc.1110.0480>.

GOMES, D. d. S. Inteligência artificial: conceitos e aplicações. **Olhar Científico. v1**, n. 2, p. 234–246, 2010.

GONÇALVES, J. F.; RESENDE, M. G. Biased random-key genetic algorithms for combinatorial optimization. **Journal of Heuristics**, 2011. ISSN 13811231.

Gurobi Optimization, LLC. **Gurobi Optimizer Reference Manual**. 2021. Disponível em: <https://www.gurobi.com>.

HARIHARAN, N.; PAAVAI, G. A brief study of deep reinforcement learning with epsilon-greedy exploration. **International Journal Of Computing and Digital System**, University of Bahrain, 2021.

HASSIN, R.; KEINAN, A. A note on greedy heuristics with regret, with application to the cheapest insertion algorithm for the traveling salesman problem. 01 2010.

HELSGAUN, K. An effective implementation of the Lin-Kernighan traveling salesman heuristic. **European Journal of Operational Research**, v. 126, n. 1, p. 106–130, October 2000. Disponível em: <https://ideas.repec.org/a/eee/ejores/v126y2000i1p106-130.html>.

HOLLAND, J. H. **Adaptation in natural and artificial systems**. [S.l.], 1975. 211 p.

JUNIOR, F. C. de L.; MELO, J. D. de; NETO, A. D. D. Using q-learning algorithm for initialization of the grasp metaheuristic and genetic algorithm. In: **2007 International Joint Conference on Neural Networks**. [S.l.: s.n.], 2007. p. 1243–1248.

JUNIOR, F. C. de L.; MELO, J. D. de; NETO, A. D. D. Using q-learning algorithm for initialization of the grasp metaheuristic and genetic algorithm. In: IEEE. **2007 International Joint Conference on Neural Networks**. [S.l.], 2007. p. 1243–1248.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. **CoRR**, cs.AI/9605103, 1996. Disponível em: <https://arxiv.org/abs/cs/9605103>.

KARIMI-MAMAGHAN, M.; MOHAMMADI, M.; MEYER, P.; KARIMI-MAMAGHAN, A. M.; TALBI, E.-G. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. **European Journal of Operational Research**, 2021. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221721003623>.

LACERDA, E. G. M. d.; CARVALHO, A. C. P. d. L. F. Introdução aos algoritmos genéticos. In: _____. [S.l.]: Ed. Universidade/UFRG/ABRH, 1999.

LAPORTE, G.; MERCURE, H.; NOBERT, Y. Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. **Discrete Applied Mathematics**, v. 18, n. 2, p. 185–197, 1987. ISSN 0166-218X. Disponível em: <https://www.sciencedirect.com/science/article/pii/0166218X87900205>.

LAWLER, E. L.; WOOD, D. E. Branch-and-bound methods: A survey. **Operations research**, INFORMS, v. 14, n. 4, p. 699–719, 1966.

LI, T.; PEI, Y. Chaotic evolution algorithms using opposition-based learning. In: **2019 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2019. p. 3292–3299.

LIN, S. Computer solutions of the traveling salesman problem. **Bell System Technical Journal**, v. 44, n. 10, p. 2245–2269, 1965. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1965.tb04146.x>.

MCCULLOCH, W.; PITTS, W. A logical calculus of ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, v. 5, p. 127–147, 1943.

MITCHELL, J. E. Branch-and-cut algorithms for combinatorial optimization problems. In: . [S.l.: s.n.], 1988.

MLADENOVIć, N.; HANSEN, P. Variable neighborhood search. **Computers & Operations Research**, v. 24, n. 11, p. 1097–1100, 1997. ISSN 0305-0548. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0305054897000312>.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; RUSU, A. A.; VENESS, J.; BELLEMARE, M. G.; GRAVES, A.; RIEDMILLER, M.; FIDJELAND, A. K.; OSTROVSKI, G. *et al.* Human-level control through deep reinforcement learning. **nature**, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. In: **Sistemas Inteligentes Fundamentos e Aplicações**. 1. ed. Barueri-SP: Manole Ltda, 2003. p. 89–114. ISBN 85-204-168.

MORÁN-MIRABAL, L.; GONZÁLEZ-VELARDE, J.; RESENDE, M. G. Randomized heuristics for the family traveling salesperson problem. **International Transactions in Operational Research**, Wiley Online Library, v. 21, n. 1, p. 41–57, 2014.

NASCIMENTO, M. B.; CHAVES, A. A. An automatic algorithm configuration based on a bayesian network. In: **2020 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.: s.n.], 2020. p. 1–8.

OCHOA, G.; VEREL, S.; TOMASSINI, M. First-improvement vs. best-improvement local optima networks of nk landscapes. In: . [S.l.: s.n.], 2010. v. 6238, p. 104–113. ISBN 978-3-642-15843-8.

OR, I. **Traveling Salesman type combinatorial problems and their relation to the logistics of regional blood banking**. [S.l.]: Northwestern University, 1976.

PADBERG, M.; RINALDI, G. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. **SIAM Review**, v. 33, n. 1, p. 60–100, 1991.

PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. **Journal of Heuristics**, Springer, v. 19, n. 2, p. 201–232, 2013.

PRASETYO, H.; FAUZA, G.; AMER, Y.; LEE, S. Survey on applications of biased-random key genetic algorithms for solving optimization problems. In: IEEE. **Industrial Engineering and Engineering Management (IEEM), 2015 IEEE International Conference on**. [S.l.], 2015. p. 863–870.

QUEIROZ DOS SANTOS, J. P.; DE MELO, J. D.; DUARTE NETO, A. D.; ALOISE, D. Reactive search strategies using reinforcement learning, local search algorithms and variable neighborhood search. **Expert Systems with Applications**, v. 41, n. 10, p. 4939–4949, 2014. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417414000645>.

RAGHAVAN, U. N.; ALBERT, R.; KUMARA, S. Near linear time algorithm to detect community structures in large-scale networks. **Physical Review E**, APS, v. 76, n. 3, p. 036106, 2007.

REGO, C.; GLOVER, F. Local search and metaheuristics for the travelling salesman problem, g. **Gutin and AP Punnen (eds.), The Travelling Salesman Problem and its Variations**, 2002.

REINELT, G. Tsplib—a traveling salesman problem library. **ORSA Journal on Computing**, v. 3, n. 4, p. 376–384, 1991. Disponível em: <https://doi.org/10.1287/ijoc.3.4.376>.

REY, D.; NEUHÄUSER, M. Wilcoxon-signed-rank test. In: _____. **International Encyclopedia of Statistical Science**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. p. 1658–1659. ISBN 978-3-642-04898-2. Disponível em: <https://doi.org/10.1007/978-3-642-04898-2_616>.

RUSSELL, S.; NORVIG, P. **Inteligência artificial /**. 2. ed. -. ed. Rio de Janeiro :: Elsevier„ 2004.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3. ed. [S.l.]: Prentice Hall, 2010.

SAMMA, H.; MOHAMAD-SALEH, J.; SUANDI, S. A.; LAHASAN, B. Q-learning-based simulated annealing algorithm for constrained engineering design problems. **Neural Computing and Applications**, Springer, v. 32, n. 9, p. 5147–5161, 2020.

SNYDER, L. V.; SHEN, Z.-J. M. **The Traveling Salesman Problem**. [S.l.]: John Wiley & Sons, Ltd, 2019. 403-461 p.

SONG, H.; TRIGUERO, I.; ÖZCAN, E. A review on the self and dual interactions between machine learning and optimisation. **Progress in Artificial Intelligence**, Springer, v. 8, n. 2, p. 143–165, 2019.

SPEARS, W. M.; JONG, K. A. D. On the virtues of parameterized uniform crossover. **Proc. of the Fourth International Conference on Genetic Algorithms**, p. 230–236, 1991.

STüTZLE, T. **Local Search Algorithms for Combinatorial Problems Analysis, Improvements, and New Applications**. Tese (Doutorado) — Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, 1998.

SUTTON, R. S.; BARTO, A. G. **Reinforcement Learning: An Introduction**. Second. The MIT Press, 2018. Disponível em: <http://incompleteideas.net/book/the-book-2nd.html>.

TALBI, E.-G. A taxonomy of hybrid metaheuristics. **Journal of heuristics**, Springer, v. 8, n. 5, p. 541–564, 2002.

TALBI, E.-G. Combining metaheuristics with mathematical programming, constraint programming and machine learning. **Annals of Operations Research**, Springer, v. 240, n. 1, p. 171–215, 2016.

TOSO, R.; RESENDE, M. A c++application programming interface for biased random-key genetic algorithms. **Optimization Methods and Software**, v. 30, n. 1, p. 81–93, 2015.

WALTS, A. **Amazon Order Management Process: Harness Chaos for Efficiency**. 2020. `https://www.skuvault.com/blog/amazon-order-management-process/`. Accessed on January 27, 2022.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. **Machine Learning**, 1992. ISSN 0885-6125.

XUE, B.; ZHANG, M.; BROWNE, W. N.; YAO, X. A survey on evolutionary computation approaches to feature selection. **IEEE Transactions on Evolutionary Computation**, v. 20, n. 4, p. 606–626, 2016.

Anexo A - Pseudocódigos Heurísticas de Busca Local

Algoritmo 12 *Heurística de Busca Local: 2-Opt*

1: **função** HEURÍSTICA *2-Opt*(s, $dist$)
2: $t \leftarrow V$ (número de visitas do FTSP).
3: $melhorou = true$.
4: **enquanto** $melhorou$ **faça**
5: $melhorou = false$.
6: **para** i=0 até i<t **faça**
7: $j \leftarrow i + 2$.
8: **enquanto** $((j+1) mod\ t) \neq i$ **faça**
9: $foOpt = dist_{i,j}$ + $dist_{i+1,j+1}$ - $dist_{i,i+1}$ - $dist_{j,j+1}$.
10: **se** foOpt < 0 **então**
11: Trocar as arestas $(i, i+1)$ e $(j, j+1)$ por (i, j) e $(i+1, j+1)$ em s.
12: $s.fo \leftarrow s.fo + foOpt$.
13: $melhorou = true$.
14: **fim-se**
15: $j++$
16: **fim-enquanto**
17: **fim-para**
18: **fim-enquanto**
19: Retornar s.
20: **fim função**

Algoritmo 13 Heurística de Busca Local: *3-Opt*

```
1: função 3-Opt(s, rota_sol, dist).
2:     t ← V (número de visitas do FTSP).
3:     melhorou = true.
4:     enquanto melhorou faça
5:         melhorou = false.
6:         para i=1 até i<t faça
7:             para j=(i+2) até j<t faça
8:                 para k=(j+2) até k<(t+1) faça
9:                     Definir A ← i-1.
10:                    Definir B ← i.
11:                    Definir C ← j-1.
12:                    Definir D ← j.
13:                    Definir E ← k-1.
14:                    Definir F ← k.
15:                    Definir d0 ← (dist[A][B] + dist[C][D] + dist[E][F]).
16:                    Definir d1 ← (dist[A][C] + dist[B][D] + dist[E][F]).
17:                    Definir d2 ← (dist[A][B] + dist[C][E] + dist[D][F]).
18:                    Definir d3 ← (dist[A][D] + dist[E][B] + dist[C][F]).
19:                    Definir d4 ← (dist[F][B] + dist[C][D] + dist[E][A]).
20:                    se d0 > d3 então
21:                        para z = j até z <= k-1 faça
22:                            Copiar sub-rota D-E.
23:                                para z = i até z <= j-1 faça
24:                                    Copiar sub-rota B-C.
25:                                fim-para
26:                        fim-para
27:                    s.fo ← s.fo - d0 + d3.
28:                    melhorou = true.
29:                    fim-se
30:                fim-para
31:            fim-para
32:        fim-para
33:    fim-enquanto
34:    Retornar s.
35: fim função
```

Algoritmo 14 Heurística de Busca Local: *Or-Opt*

```
1: função Or-Opt(s, rota_sol, dist).
2:     t ← V (número de visitas do FTSP).
3:     melhorou = true.
4:     enquanto melhorou faça
5:         melhorou = false.
6:         para i=0 até i<m − 1 faça
7:             j ← (i + 2).
8:             enquanto j + 1 ≠ i
9:                 Definir viM = 0.
10:                se i = 0 faça
11:                    viM = (m-1).
12:                senão
13:                    viM = (i-1).
14:                    Definir foOpt = dist_{i,j} + dist_{i+1,j+1} + dist_{[viM],i+2}
15:                    - dist_{j,j+1} - dist_{[viM],i} - dist_{i+1,i+2}.
16:                    se foOpt < 0 então
17:                        se i < j + 1 então
18:                            se j = m -1 então
19:                                Inserir elemento i na última posição do vetor s.
20:                                Inserir elemento j na última posição do vetor s.
21:                            senão Adicionar elemento i+1 na posição j+1.
22:                            Adicionar elemento i na posição j+1.
23:                            fim-senão
24:                        Remover elemento i+1 do vetor s.
25:                        Remover elemento i do vetor s.
26:                        fim-se
27:                        senão se j+1 < i então
28:                            Remover elemento i+1 do vetor s.
29:                            Remover elemento i do vetor s.
30:                            Adicionar elemento i+1 na posição j+1.
31:                            Adicionar elemento i na posição j+1.
32:                        fim-senão-se
33:                        s.fo ← s.fo + foOpt.
34:                        melhorou = true.
35:                    fim-se
36:                    j + +
37:            fim-enquanto
38:        fim-para
39:    fim-enquanto
40:    Retornar s.
41: fim função
```

Algoritmo 15 Heurística de Busca Local: *Node Insertion*

função *Node-Insertion*(s, $rota_{sol}$, $dist$)
    $t \leftarrow V$ (número de visitas do FTSP).
    $melhorou = true$.
    enquanto $melhorou$ **faça**
        $melhorou = false$.
        para i=0 até i<t **faça**
            $j \leftarrow (i + 1) mod\ t$.
            enquanto $((j+1) mod\ t) \neq i$ **faça**
            Definir $viM = 0$.
                se $i = 0$ **então**
                    $viM = (t\text{-}1)$.
                se não
                    $viM = (i\text{-}1)$.
                Definir $foOpt = dist_{i,j} + dist_{i,j+1} + dist_{[viM],i+1}$ - $dist_{j,j+1}$ - $dist_{[viM],i}$ – $dist_{i,i+1}$.
                se $foOpt < 0$
                    Remover o nó i e inserir na posição j.
                    $s.fo \leftarrow s.fo + foOpt$.
                    $melhorou = true$.
                fim-se
                $j++$
            fim-enquanto
        fim-para
    fim-enquanto
    Retornar s.
fim função

Algoritmo 16 Heurística de Busca Local: *Node-Exchange*

função *Node-Exchange*(s, $rota_{sol}$, $dist$).
    $t \leftarrow V$ (número de visitas do FTSP).
    $melhorou = true$.
    enquanto $melhorou$ **faça**
        $melhorou = false$.
        para i=0 até i<$m-1$ **faça**
            $j \leftarrow (i+2)$.
            enquanto $j < m$
                se $i \neq 0$ e $j \neq m$-1 **faça**
                    Definir $viM = 0$.
                    se $i = 0$ **faça**
                        $viM = (m\text{-}1)$.
                    senão
                        $viM = (i\text{-}1)$.
                    Definir $vjP = 0$
                    se $j < m$-1 **então**
                        $vjP = (m\text{-}1)$.
                    senão
                        $vjP = 0$.
                    Definir $foOpt = dist_{[viM],j} + dist_{i,i+1} + dist_{j-1,i} + dist_{i,[vjP]}$
                    - $dist_{viM,i}$ - $dist_{i,i+1}$ - $dist_{j-1,i}$ - $dist_{i,[vjP]}$.
                    se $foOpt < 0$ **então**
                        Trocar os nós i e j na rota.
                        $s.fo \leftarrow s.fo + foOpt$.
                        $melhorou = true$.
                    fim-se
                fim-se
                $j++$
            fim-enquanto
        fim-para
    fim-enquanto
    Retornar s.
fim função

Algoritmo 17 Heurística de Busca Local: *Swap-nodes*

função *Swap-nodes*(s, $rota_{sol}$, $dist$).
    $t \leftarrow V$ (número de visitas do FTSP).
    $melhorou = true$.
    enquanto $melhorou$ **faça**
        $melhorou = false$.
        para i=0 até i<$t-1$ **faça**
            para j=t até j<n **faça**
                se $i \neq 0$ e elemento i for da mesma família que o elemento j **então**
                    Definir $viM = 0$.
                    se $i = 0$ **faça**
                        $viM = (t\text{-}1)$.
                    senão
                        $viM = (i\text{-}1)$.
                    para cada posição da rota k até k<1 **faça**
                        Definir $vkM = 0$.
                        se k=0 **então**
                            $vkM = t\text{-}1$.
                        senão $vkM = k\text{-}1$.
                        Definir uma variável *economia*.
                        se k=i **então**
                            $economia$ = - $dist_{[viM],i}$ - $dist_{i,i+1}$ + $dist_{[viM],j}$ + $dist_{j,i+1}$.
                        senão se $k \neq (i+1)$ **então**
                            $economia$ = - $dist$ - $dist_{i,i+1}$ + $dist_{[viM],i+1}$
                                - $dist_{[vkM],k}$ + $dist_{[vkM],j}$ + $dist_{j,k}$.
                        Verificar se a *economia* é a $maior_{economia}$.
                        se $economia > maior_{economia}$ **então**
                            Guardar o melhor custo: $melhor_i = i$, $melhor_j = j$ e $melhor_k = k$.
                            Definir a variável *economia*
                        fim-se
                fim-se
            fim-para
        fim-para
        se $maior_{economia} > 0$ **então**
            se $melhor_i < melhor_k$ **então**
                Substituir j por i.
                Inserir o ponto j na posição $melhor_k$.
                Remover i.
            senão se $melhor_i > melhor_k$ **então**
                Substituir j por i.
                Remover i.
                Inserir o ponto j na posição $melhor_k$.
            senão se $melhor_i = melhor_k$ **então**
                Substituir j por i.
                $s.fo \leftarrow s.fo$ - $maior_{economia}$.
                $melhorou = true$.
        fim-se
    fim-enquanto
    Retornar s.
fim função

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
DM	02 de dezembro de 2020	DCTA/ITA/DM-018/2017	100

5. TÍTULO E SUBTÍTULO:

Métodos exato e heurístico para resolução do Problema do Caixeiro Viajante em Famílias

6. AUTORA(ES):

Bárbara Lessa Vianna

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA / Universidade Federal de São Paulo – UNIFESP

8. PALAVRAS-CHAVE SUGERIDAS PELA AUTORA:

Palavra 1; Palavra 2; Palavra 3

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Palavra 1; Palavra 2; Palavra 3

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA/UNIFESP, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Pesquisa Operacional. Área de Engenharia de Produção/Pesquisa Operacional. Orientador: Prof. Dr. Antônio Augusto Chaves. Coorientador: Dr. Tiago Tiburcio da Silva Profª. Drª. Nome. Defesa em 02/12/2020. Publicada em 25/12/2020.

11. RESUMO:

No campo da Pesquisa Operacional, uma tendência é o desenvolvimento de métodos híbridos exatos e heurísticos para obter resultados de boa qualidade em problemas de otimização combinatória. Existem diversas maneiras de hibridização. Uma possibilidade de hibridizar métodos exatos é através da integração de um método exato com heurísticas de busca local. Uma versão híbrida de metaheurísticas pode ser obtida com a integração de técnicas de Aprendizado de Máquinas.

Um dos problemas mais clássicos de otimização combinatória é o Problema do Caixeiro Viajante (TSP). Nesse problema considera-se a minimização apenas dos custos operacionais envolvidos no percurso do vendedor. Contudo, o TSP pode ser adaptado para diferentes problemas que empresas logísticas enfrentam, como, por exemplo, diferentes categorias de produtos, prioridades de entregas, e localização de produtos em armazéns. Este trabalho aborda o Problema do Caixeiro Viajante em Família (FTSP, do inglês *Family Traveling Salesman Problem*), em que os clientes são agrupados em famílias que correspondem a produtos de mesma similaridade e com a demanda de visitas predefinidas. O objetivo do FTSP é determinar a rota de custo mínimo visitando apenas um subconjunto de clientes de cada família. Assim como o TSP, trata-se de um problema de otimização combinatória pertencente a classe NP-Difícil.

Para solucionar o problema proposto foram desenvolvidos dois métodos: (*i*) um *branch-and-cut* paralelo com um procedimento de busca local eficiente para obter a solução ótima, e (*ii*) uma metaheurística adaptativa que combina o método *Biased Random-key Genetic Algorithm* (BRKGA) com um algoritmo de aprendizado por reforço, *Q-Learning* (QL). Neste caso, o algoritmo *Q-Learning* é utilizado para controlar os parâmetros do BRKGA durante o processo evolutivo.

Experimentos computacionais foram realizados em um conjunto de dados de referência bem conhecido, que possui 185 instâncias. O algoritmo P-B&C desenvolvido para o FTSP prova o valor ótimo para 179 instâncias e o BRKGA-QL encontrou os melhores limites superiores para as outras quatro instâncias. Os resultados foram comparados com os melhores resultados da literatura, e ambos os métodos mostram robustez e eficiência para resolver o FTSP.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**