Pontifícia Universidade Católica de Goiás
Programa de Mestrado em Engenharia de Produção e Sistemas

PLATAFORMA *WEB* HYBROO: AMBIENTE EXPERIMENTAL VOLTADO À HIBRIDIZAÇÃO DE ALGORITMOS DE OTIMIZAÇÃO

Edgar Marcos Ancioto Junior

2019

PLATAFORMA WEB HYBROO: AMBIENTE EXPERIMENTAL VOLTADO À HIBRIDIZAÇÃO DE ALGORITMOS DE OTIMIZAÇÃO

Edgar Marcos Ancioto Junior

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para obtenção de título de Mestre em Engenharia de Produção e Sistemas.

Orientadora: Maria José Pereira Dantas, Dra.

GOIÂNIA - GOIÁS
ABRIL 2019

A541p Ancioto Junior, Edgar Marcos
Plataforma Web Hybroo : ambiente experimental voltado à hibridização de algoritmos de otimização / Edgar Marcos Ancioto Junior.-- 2019.
98 f.: il.

Texto em português, com resumo em inglês
Dissertação (mestrado) -- Pontifícia Universidade Católica de Goiás, Goiânia, 2019
Inclui referências: f. 88-97

1. Programação não-linear. 2. World Wide Web (Sistema de recuperação da informação). I.Dantas, Maria José Pereira. II.Pontifícia Universidade Católica de Goiás - Programa de Pós-Graduação em Engenharia de Produção e Sistemas - 2019. III. Título.

CDU: Ed. 2007 -- 004.774(043)

PLATAFORMA *WEB* HYBROO: AMBIENTE EXPERIMENTAL VOLTADO À HIBRIDIZAÇÃO DE ALGORITMOS DE OTIMIZAÇÃO

Edgar Marcos Ancioto Junior

Esta Dissertação julgada adequada para obtenção do título de Mestre em Engenharia de Produção e Sistemas, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia de Produção e Sistemas da Pontifícia Universidade Católica de Goiás em abril de 2019.

Prof. Marcos Lajovic Carneiro, Dr.
Coordenador do Programa de Pós-Graduação em Engenharia de Produção e Sistemas

Banca Examinadora:

Prof.ª Maria José Pereira Dantas, Dra.
Orientadora

Prof. Flávio Henrique Teles Vieira, Dr.

Prof. Marcos Antônio Sousa, Dr.

GOIÂNIA - GOIÁS
ABRIL 2019

AGRADECIMENTOS

A presente dissertação de mestrado não poderia chegar ao final sem o apoio precioso de várias pessoas.

Em primeiro lugar agradeço a minha família, em especial minha mãe Jaqueline, que tornou esta jornada possível com seu apoio incondicional.

À minha orientadora, Profª Dra. Maria José Pereira Dantas, pela dedicação incansável ao longo da elaboração deste.

À FAPEG, por fomentar esta pesquisa por meio da concessão da bolsa de estudos.

Resumo da Dissertação apresentada ao MEPROS/ PUC Goiás como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia de Produção e Sistemas

PLATAFORMA *WEB* HYBROO: AMBIENTE EXPERIMENTAL VOLTADO À HIBRIDIZAÇÃO DE ALGORITMOS DE OTIMIZAÇÃO

Edgar Marcos Ancioto Junior

Março/2019

Orientadora: Prof.ª Maria José Pereira Dantas, Dra.

Resumo: Na otimização existem métodos aplicáveis a uma variedade de problemas de minimização. Ao definir o método a ser aplicado para um problema real, baseia-se na produção científica buscando similaridade com o problema e a eficiência de seus resultados. Surge uma lacuna no desafio de extrair informações de experimentos em trabalhos científicos que possibilitem a reprodução do estudo. Este trabalho tem como objetivo desenvolver um ambiente experimental *online* voltado à otimização, com a funcionalidade de realizar testes experimentais com algoritmos meta-heurísticos, com a possibilidade de hibridização, validando os conceitos desenvolvidos por meio da solução de problemas de *benchmark* e permitindo a comparação com outros métodos desenvolvidos na literatura. Todos os códigos são desenvolvidos na linguagem de programação Python e são de código aberto. O ambiente permitirá a configuração do algoritmo e demonstrará a resolução do problema, permitindo a compreensão da aplicação de métodos, a avaliação de hibridizações e a comparação entre todos os métodos. Para problemas de roteirização foram desenvolvidos os métodos meta-heurísticos Algoritmo Genético (GA), Recozimento Simulado (SA) e Colônia de Formigas (ACO). Para otimização não linear os métodos GA e SA foram aplicados às funções e instâncias de *benchmark* para avaliação da eficiência dos métodos. Foram avaliadas as hibridizações dos métodos utilizando a lógica de trabalho colaborativo retransmitido. Os resultados demonstraram que as técnicas híbridas tem uma capacidade superior aos métodos convencionais para resolução dos problemas de instâncias para o Problema do Caixeiro Viajante (TSP) e Problema de Roteamento de Veículos Capacitado (CVRP), sendo que a hibridização ACO+GA foi a combinação que atingiu os melhores valores de eficiência média para os problemas com valores de 97,5% para o TSP e 96,66% para o CVRP, com destaque para as execuções que envolvem os maiores problemas entre as instâncias da biblioteca A-VRP. Um ambiente foi disponibilizado na *web,* http://hybroo.hopto.org:5000/ para acesso dos métodos e execução dos testes, permitindo a conferência dos dados desta pesquisa.

Palavras-chave: Ambiente *Web* experimental; CVRP; TSP; Hibridização de meta-heurísticas; Otimização não linear.

Abstract of the Dissertation presented to MEPROS/ PUC Goiás as part of the requirements necessary to obtain a Master's degree in Production and Systems Engineering

HYBROO WEB PLATFORM: EXPERIMENTAL ENVIRONMENT TO HYBRIDIZATION OF OPTIMIZATION ALGORITHMS

Edgar Marcos Ancioto Junior

March/2019

Advisor: Prof.ª Maria José Pereira Dantas, Dra.

Abstract: In optimization there are methods applicable to a variety of minimization problems. When defining the method to be applied to a real problem, it is based on the scientific production seeking similarity with the problem and the efficiency of its results. There is a gap in the challenge of extracting information from experiments in scientific works that make possible the reproduction of the study. This work aims to develop an online experimental environment focused on optimization, with the functionality of performing experimental tests with metaheuristic algorithms, with the possibility of hybridization, validating the concepts developed through the solution of benchmark problems and allowing the comparison with other methods developed in the literature. All codes are developed in the Python programming language and are open source. The environment will allow the configuration of the algorithm and will demonstrate the resolution of the problem, allowing the understanding of the application of methods, the evaluation of hybridizations and the comparison between all the methods. For meta-heuristic methods, Genetic Algorithm (GA), Simulated Annealing (SA) and Ant Colony (ACO) were developed for routing problems. For non-linear optimization GA and SA methods were applied to the functions and benchmark instances to evaluate the efficiency of the methods. Hybridizations of the methods were evaluated using retransmitted collaborative work logic. The results demonstrated that the hybrid techniques have a superior capability to the conventional methods to solve the problems of instances for the Traveling Salesman Problem (TSP) and Vehicle Routing Problem (CVRP). The ACO + GA hybridization was the combination which reached the best average efficiency values for the problems with values of 97.5% for the TSP and 96.66% for the CVRP, highlighting the executions involving the biggest problems among the selected ones. An environment was made available on the web, http://hybroo.hopto.org:5000/ to access the methods and the execution of the tests, allowing the data conference of this research.

Key words: Experimental Web environment; CVRP; TSP; Hybridization of metaheuristics; Nonlinear optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ABC	*Algorithm Bee Colony* – Algoritmo Colônia de Abelhas
ACO	*Ant Colony Optimization* – Algoritmo Colônia de Formigas
A-VRP	Biblioteca de Instâncias do CVRP
BA	*Bat-inspired Algorithm* - Algoritmo Inspirado em Morcegos
CSS	*Cascading Style Sheets* - Folhas de Estilo em Cascata
CVRP	*Capacitated Vehicle Routing Problem* – Problema de Roteamento de Veículos Capacitado
EOMSA	*Elite Opposition-Based MSA* – MSA Baseado em Oposição de Elite
FA	*Firefly Algorithm* – Algoritmo de Vaga-lumes
GA	*Genetic Algorithm* – Algoritmo Genético
GWO	*Gray Wolf Optimization* – Algoritmo do Lobo Cinzento
HHA	*Hybrid Heuristic Algorithm* – Abordagem Heurística Híbrida
HTML	*Hypertext Markup Language* - Linguagem de Marcação de Hipertexto
LKH	*Lin Kernighan Heuristic* – Heurística de Lin Kernighan
LMFO	*Lévy-Flight Moth-Flame Algorithm* - Algoritmo Moth-Flame Lévy-Flight
MSA	*Moth Swarm Algorithm* - Algoritmo Enxame de Mariposas
PL	Programação Linear
PSO	*Particle Swarm Optimization* – Algortimo Enxame de Partículas
PSO-GSA	*Particle Swarm Optimization and Gravitational Search Algorithm* – Algoritmo Enxame de Partículas e de Busca Gravitacional
SA	*Simulated Annealing* – Recozimento Simulado
TSP	*Travelling Salesman Problem* – Problema do Caixeiro Viajante
VRP	*Vehicle Routing Problem* – Problema de Roteamento de Veículos

SUMÁRIO

1. INTRODUÇÃO

A otimização é uma área de estudo baseada na busca de uma solução ótima (melhor possível) para determinado problema (REGO *et al.*, 2011), sendo fundamental nos processos de tomada de decisão sistemática que surgem em áreas das ciências (Econômicas, Física, Química, Biologia) e Engenharias (SAMUCO, 2014).

A solução de um problema de otimização busca encontrar a melhor combinação de valores para um conjunto de variáveis de decisão que minimize (maximize) a função objetivo deste problema (GOUVÊA, 2016). Essa função é representada por uma ou mais variáveis entre pontos que pertencem a uma determinada região do espaço de busca e com um conjunto de restrições.

Problemas com poucas variáveis e restrições podem ser resolvidos manualmente em tempo hábil, porém, quando a complexidade aumenta, é necessário utilizar um *software* com um algoritmo especializado (LUENBERGER; YE, 2008), também conhecido como solucionador computacional.

1.1. Problema de pesquisa

Na área de otimização existem diversos métodos determinísticos e heurísticos que podem ser aplicados a uma variedade de tipos de problemas. Assim, há uma dificuldade na escolha do método (POLLARIS *et al.*, 2015). Para definir o método a ser aplicado para um problema real, pesquisadores baseiam-se na literatura buscando problemas já solucionados, tendo em vista a similaridade com o problema tratado e a eficiência de seus resultados. O desafio dessa busca está no fato de que nem sempre a produção científica provê os passos seguidos em seus experimentos nem o código fonte dos algoritmos, dificultando a compreensão do método executado para a resolução do problema, impossibilitando a reprodução do mesmo (TIAN *et al.*, 2017).

Observa-se uma lacuna para um ambiente experimental que permita a realização de testes com métodos de otimização existentes, usando funções e instâncias de *benchmark*, reunidos em um único local, gerando a possibilidade do acompanhamento de cada método e avaliação de sua eficiência para cada tipo de problema.

1.2. Objetivo geral

Desenvolver uma plataforma de algoritmos meta-heurísticos, com a possibilidade de hibridizá-los em alto nível, avaliando tal necessidade, para resolução de problemas. Desenvolver os algoritmos: Algoritmo Genético (HOLLAND, 1975), Algoritmo Colônia de Formigas (DORIGO; STÜTZLE, 2004) e Recozimento Simulado (KIRKPATRICK; GELATT; VECCHI, 1983). Os problemas abordados são Otimização de Funções não lineares e de Roteirização, este segundo sendo divido em Problema do Caixeiro Viajante (TSP) e o Problema de Roteamento de Veículos Capacitado (CVRP). A plataforma tem a função de ambiente experimental, para a realização de testes com os algoritmos, demonstrando a resolução do problema para cada algoritmo, a hibridização e sua eficiência para cada caso. A plataforma é disponibilizada *online*, agregando acessibilidade à ferramenta; com código fonte aberto, permitindo que o usuário desenvolva seus algoritmos baseados nos existentes; e, mostrando a evolução dos algoritmos na resolução do problema, possibilitando análise de eficiência de cada algoritmo.

1.3. Objetivos específicos

- Criar os bancos de dados para armazenamento de problemas (funções e de instâncias de *benchmark);*
- Implementar os algoritmos meta-heurísticos para resolução de problemas;
- Implementar as classes de controle de execução dos métodos;
- Implementar o *framework* do ambiente *web*;
- Demonstrar a hibridização dos métodos utilizando o Problema de Funções não lineares, o Problema do Caixeiro Viajante e o Problema de Roteamento de Veículos.

1.4. Justificativa

Este trabalho busca contribuir com a criação de um ambiente experimental disponibilizado em plataforma *web*, que possibilite a resolução de problemas por métodos meta-heurísticos, e visando a verticalização do conhecimento tratará da hibridização de métodos voltados para resolução de problemas complexos.

A plataforma proposta será disponibilizada via *web* pela motivação dos trabalhos encontrados, em sua maioria, serem disponibilizados para instalação *Desktop*, tendo como desvantagem não serem multiplataforma, dificultando a

utilização. As plataformas são desenvolvidas em Matlab (TIAN *et al.*, 2017), Java (EVANGELISTA; MAIA; ROCHA, 2009), C++ (LIEFOOGHE; JOURDAN; LEGRAND, 2007) (SHEN; ZHENG; LI, 2015) e exigem a instalação de pacotes e máquinas virtuais para o funcionamento. Em contraponto, existem em minoria, plataformas *online* sem a necessidade de instalações ou requisitos do sistema, utilizando da portabilidade e indiferença de plataformas (MARTINS, 2018).

O ambiente por meio da implementação de métodos híbridos terá como escopo problemas de funções e de roteamento, que são casos particulares dentro do assunto de otimização combinatória. O planejamento de rotas é, atualmente, um dos campos mais estudados na inteligência artificial (OSABA *et al.*, 2017). Tem como motivação a tendência de mudar dos algoritmos baseados em um único paradigma para métodos híbridos baseados em vários princípios (LAPORTE, 2009).

1.5. Organização do trabalho

Esta dissertação está organizada da seguinte forma: no Capítulo 2 é realizada uma revisão bibliográfica sobre os assuntos tratados, como otimização, métodos, plataformas e problemas; no Capítulo 3 é descrita a metodologia adotada para o desenvolvimento deste, divida em etapas e suas implementações; no Capítulo 4 são expostas especificidades referentes à codificação dos métodos a adaptações necessárias; são exibidos os resultados dos desenvolvimentos feitos no decorrer da pesquisa, além das análises de configurações de parâmetros realizadas para os algoritmos; no Capítulo 5 estão os resultados dos experimentos e comparações feitas com a literatura; e, por fim, no Capítulo 7 são apresentadas as considerações finais do trabalho.

2. REFERENCIAL TEÓRICO

2.1. Problema geral de otimização

Muitos problemas da vida real surgem de tarefas como agendamento de serviços, planejamento de produção, gerenciamento de logística, roteamento de Internet e de veículos, entre inúmeras áreas relacionadas à otimização. Tais problemas tem facilidade de gerar afirmações, porém, uma grande dificuldade em sua resolução (DUMITRESCU; STUTZLE, 2003). Tal dificuldade, unida à importância prática, levou o estudo da otimização a uma grande variedade de soluções.

O objetivo de um problema de otimização é encontrar a melhor combinação de valores para um conjunto de variáveis de decisão que minimize (maximize) a função objetivo deste problema (GOUVÊA, 2016). Essa função é representada por uma ou mais variáveis, entre pontos que pertencem a uma determinada região do espaço, geralmente possuindo um conjunto de restrições. Na formulação do problema estão presentes os seguintes conceitos (SILVA, 2001):

- **Variáveis de decisão**: São os parâmetros que podem ser alterados durante o processo de otimização. Essas variáveis podem ser contínuas ou discretas.
- **Restrições**: São funções de igualdade ou desigualdade sobre as variáveis de decisão. Elas descrevem quais as limitações impostas para a solução otimizada.
- **Domínio de busca**: É a região onde as possíveis soluções são localizáveis, a parte do domínio que respeita todas as restrições é denominada domínio viável.
- **Função objetivo**: É a função das variáveis utilizadas para o problema. Esta função será usada como medida de eficiência do processo de otimização.

O problema geral de otimização pode ser formulado como:

$$\begin{aligned} & Minimizar\ f(x) \\ sujeito\ a: \quad & h_i(x) = 0, \qquad i = 1, \dots, m_h \\ & g_j(x) \leq 0, \qquad j = 1, \dots, m_g \\ & x \in \mathbb{R}^n \end{aligned} \tag{1}$$

Na formulação de (LUENBERGER; YE, 2008), x é um vetor n-dimensional, f, g e h são funções contínuas, geralmente diferenciáveis. O conjunto $\mathbb{R}^n$ é um

subconjunto do espaço n-dimensional. A função f é a função objetivo do problema, g e h são funções pertencentes a equações, desigualdades e restrições.

Para definir um minimizador, pode se considerar uma função $f: \mathbb{R}^{\eta} \longrightarrow \mathbb{R}\, e\, x^{*} \in S \subset \mathbb{R}^{\eta}$. Dizemos que x^{*} é um minimizador local de f em S se existe $\delta > 0$ tal que $f(x^{*}) \leq f(x)$, para todo $x \in \mathbb{B}(x^{*}, \delta) \cap S$. Caso $f(x^{*}) \leq f(x)$, para todo $x \in S, x^{*}$ é chamado de minimizador global de f em S.

Quando as desigualdades forem escritas para $x^{*} \neq x$, dizemos que x^{*} é um minimizador que pode ser local ou global estrito de f em S. O objetivo é encontrar os minimizadores, porém, para isso é preciso conhecer algumas condições sobre o problema para garantir que de fato eles existem.

Os problemas de otimização podem ser classificados conforme a natureza da função objetivo e das restrições (lineares ou não), a quantidade de variáveis (pequeno ou grande), a suavidade das funções (diferenciáveis ou não), entre outras (GOUVÊA, 2016). A mais importante diferença entre os problemas está entre os que têm restrição e os que não têm. Problemas sem restrição, utilizados em muitas aplicações práticas, surgem como reformulação de problemas restritos, onde as restrições são substituídas por termos de penalização na função objetivo, na tentativa de diminuir a violação das restrições.

Um passo importante no processo da otimização é fazer a correta classificação dos problemas a serem otimizados, uma vez que os diversos algoritmos existentes são adaptados para cada tipo de problema.

2.2. Métodos Clássicos de otimização

Os métodos clássicos de otimização são úteis na determinação de funções contínuas e diferenciáveis, são analíticos e se baseiam no cálculo diferencial para encontrar o ponto ótimo (SAMUCO, 2014).

Os métodos de otimização baseados nos algoritmos determinísticos geram uma sequência determinística de possíveis soluções, requerendo, na maioria das vezes, o uso de pelo menos a primeira derivada da função objetivo em relação às variáveis de projeto (SAMUCO, 2014).

Os métodos determinísticos geralmente apresentam teoremas que lhes garantem a convergência para uma solução que não é necessariamente a ótima. Como nesses métodos a solução encontrada é extremamente dependente do ponto de partida fornecido, pode-se convergir para um ótimo local, por isso não possuem bom desempenho em otimizar funções multimodais, isto é, funções que possuem vários ótimos locais (SILVA, 2001).

De acordo com as características da função objetivo e das restrições, classificam-se os problemas de otimização nas seguintes subáreas da Programação Matemática:

- Programação Linear: quando a função objetivo e as restrições são funções lineares das variáveis.
- Programação Não-Linear: quando a função objetivo ou pelo menos uma das restrições é função não linear das variáveis.

Dependendo do tipo de problema, linear ou não linear, com ou sem restrições, aplica-se diferentes métodos para resolvê-los. Inúmeros são os métodos, bem como as classificações realizadas pelos diversos autores da literatura. A seguir é apresentada uma possível classificação geral dos métodos clássicos existentes (NEVES, 1997):

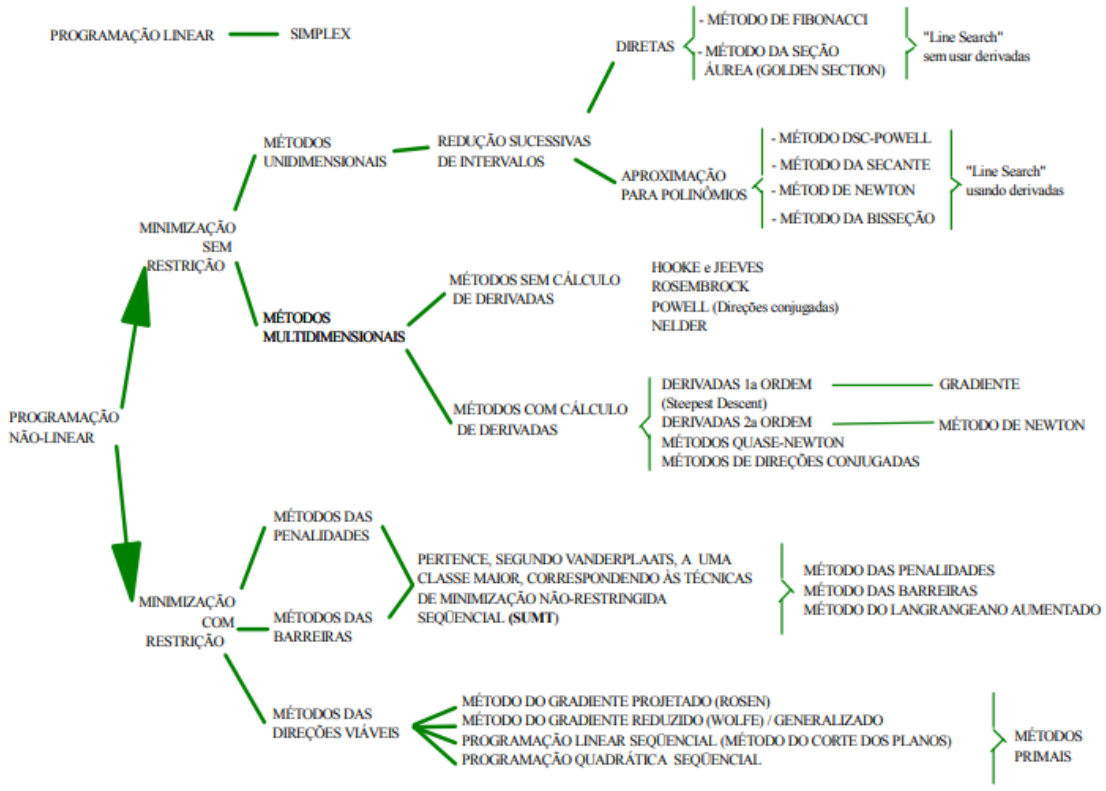

Figura 1 - Classificação Geral dos Métodos de Programação Matemática
Fonte: (NEVES, 1997)

2.3. Programação linear

Segundo (LUENBERGER; YE, 2008), a Programação Linear (PL) é um problema de otimização com uma função objetivo linear no qual as restrições são igualdades e desigualdades lineares. A exatidão dessas restrições podem distinguir os problemas, no entanto qualquer problema de PL pode ser representado por:

$$Otimizar \sum_{j=1}^{n} c_j x_j$$

$$sujeito\ a: \begin{array}{ll} \sum_{j=1}^{n} a_{ij} x_j \geq d_i, & i = 1, \dots, p \\ \sum_{j=1}^{n} a_{ij} x_j = d_i, & i = p+1, \dots, m \\ x_j \geq 0, & j = 1, \dots, q \\ x_j \in \mathbb{R} & j = q+1, \dots, n \end{array} \quad (2)$$

Onde m é o conjunto das restrições do problema, n é o número de variáveis, $A = \{a_{ij}\}$ é a matriz de restrições, x é o vetor de coluna de n componentes, c é o vetor de linha de n componentes e d é o vetor de coluna de m componentes.

A PL define-se em aplicar um conjunto de desigualdades (ou equações lineares) a variáveis, determinando valores não negativos, satisfazendo todas as restrições estabelecidas e minimizando alguma função linear das variáveis. Modelos significativos de problemas de alocação e fenômenos econômicos provaram a eficiência do estudo da PL (LUENBERGER; YE, 2008). A literatura em contínua expansão das aplicações demonstra a importância da PL como estrutura geral para a formulação de problemas (ALMEIDA *et al.*, 2018).

Existem quatro tipos de programação linear: a Programação contínua, quando o resultado das variáveis do modelo são valores reais. Esse tipo é o mais utilizado na solução de problemas práticos. Programação estruturada, quando um modelo de Programação Linear aplicado a um único processo de produção se reproduz a outros processos. Programação inteira, quando o problema admite apenas soluções inteiras. Programação inteira mista, quando o problema admite tanto soluções inteiras quanto soluções reais (VASCONCELOS, 2013).

O método mais conhecido e largamente usado para a solução de problemas de PL é o Método Simplex, que é um método iterativo e determina a solução de modo algébrico (RAIDL; PUCHINGER, 2008). Este método envolve operações com matrizes e consiste de um vetor inicial, escolhido de modo conveniente e de uma sequência de iterações a partir deste vetor inicial. A sequência dos resultados deve convergir para a solução ótima. O método tem como importante propriedade o fato de garantir que a solução ótima se encontra no contorno do domínio viável (RAIDL, 2015).

2.4. Programação não linear

Os modelos de programação não linear são diferenciados dos modelos lineares por apresentarem uma função objetivo não linear, ou pelo menos uma das restrições caracterizada por uma função não linear (LUENBERGER; YE, 2008). Fenômenos físicos ou econômicos são representados melhor por modelos não lineares.

O principal conceito envolvido em programação não linear é o de taxa de variação, onde o cálculo é feito a partir da diferenciação. A dificuldade da programação não linear encontra-se na obtenção da solução ótima global, já que existem várias soluções locais (SILVA, 2001). Os métodos se dividem em com restrições ou sem restrições, e unidimensional ou multidimensional, onde para cada caso há uma enormidade de métodos aplicáveis.

2.4.1. Otimização com restrições

Seja a formulação geral de um problema de programação não linear com restrições:

$$\begin{array}{lll} & Minimizar \quad f(x) & \\ sujeito\ a & g_i(x) \geq 0, \quad i = 1, \dots, m & \text{(restrições de desigualdade)} \\ & h_j(x) = 0, \quad j = 1, \dots, l & \text{(restrições de igualdade)} \end{array} \qquad (3)$$

Segundo (VANDERPLAATS, 1999) os métodos de otimização com restrições possuem três divisões a serem consideradas. A primeira é dos métodos que possuem técnicas de minimização sequencial sem restrições. O processo utiliza um meio para transformar o problema original em um problema sem restrições, esse meio pode ser: a penalidade, o langrangeano aumentado e a dualidade. Método esse denominado indireto.

A segunda divisão é dos métodos diretos que decompõe o cálculo da direção de descida utilizando técnicas de otimização sem restrição. A primeira etapa da decomposição é feita por meio da direção resultante da programação não linear sem restrições para a minimização de $f(x)$; a segunda etapa é a adaptação da direção ao conjunto de restrições $g_i(x)$ e $h_j(x)$ (Método das direções viáveis) (NEVES, 1997).

A última divisão é dos métodos de aproximação, assim como os diretos, não transformam o problema original em um sem restrições. Sua resolução é feita aplicando aproximação das condições de otimalidade aplicadas sobre a função lagrangeana.

2.4.2. Otimização sem restrição

Algoritmos de otimização para problemas de programação não linear sem restrições possuem, geralmente, uma estrutura comum de resolução que consiste em aplicar, a equação de recorrência (VANDERPLAATS, 1999):

$$\boldsymbol{x}^k = \boldsymbol{x}^{k+1} + \alpha^k d^k \quad (4)$$

Onde $\boldsymbol{x}$ é o vetor das variáveis, α é o tamanho do passo, d é a direção de busca e o índice k representa a iteração corrente.

As principais etapas que caracterizam os algoritmos de problemas de programação não linear sem restrições são descritos como:

- Passo 1 - Escolher um ponto de partida $\boldsymbol{x}^0$;
- Passo 2 - Determinar a direção de busca d^k;
- Passo 3 - Minimizar a função objetivo $f(x)$ na direção d^k, a partir do ponto $\boldsymbol{x}^{k-1}$, para determinar o tamanho do passo α^k;
- Passo 4 - Atualizar as variáveis $\boldsymbol{x}$, por meio da Equação (4);
- Passo 5 - Voltar para o Passo 2 caso o critério de convergência não for atendido.

Os algoritmos para esses problemas diferem na estratégia utilizada para o Passo 2 do esquema anterior, na determinação de d^k. Este será o responsável por determinar as sucessivas direções de busca no espaço do problema. Os métodos existentes necessitam de informações de pelo menos o valor da função objetivo, no entanto, alguns além desse valor necessitam ainda da primeira derivada, ou até mesmo a segunda derivada em relação à função objetivo (SILVA, 2001). Métodos estes denominados de métodos de ordem zero, de primeira ordem e segunda ordem, respectivamente.

Nesse caso, o problema geral consiste em:

$$Minimizar \quad f(x), \qquad x \in \mathbb{R}^{\eta} \quad (5)$$

A maioria dos métodos de otimização que utilizam o procedimento iterativo na Equação (4) determina o tamanho do passo α^k na direção d^k por meio de uma técnica, esta que exige que d^k seja uma direção de descida, conforme a Equação (6).

$$\nabla f(\boldsymbol{x}^k) d^k < 0 \quad (6)$$

Se d^k é uma direção de descida, então deve existir um $\alpha^k > 0$ tal que:

$$f(\boldsymbol{x}^{k+1}) < f(\boldsymbol{x}^k) \quad (7)$$

Métodos baseados em gradiente são utilizados para solução desses problemas. Esses métodos, além do modo em que a direção é buscada, diferenciam também no tamanho do passo. A direção pode ser obtida pela direção contrária a do vetor gradiente da função objetivo assim como, de forma mais onerosa computacionalmente, pela matriz Hessiana da função (NEVES, 1997).

Na Tabela 1 são apresentados alguns métodos com sua classificação segundo o critério de avaliação da derivada.

Tabela 1 - Métodos, classificações e publicações científicas

Método	**Ordem**	**Trabalhos que fazem a formulação dos métodos**
Método da Seção Áurea	0	(VANDERPLAATS, 1999), (TSAI; KOLIBAL; LI, 2010), (HERNÁNDEZ; FLORES; VÁZQUEZ, 2012), (MOHAMMADI *et al.*, 2015)
Método da Sequência de Fibonacci	0	(VANDERPLAATS, 1999), (SUBASI; YILDIRIM; YILDIZ, 2004)
Método de Powell	0	(VANDERPLAATS, 1999), (AROUXÉT; ECHEBEST; PILOTTA, 2011)
Método do Gradiente	1	(HESTENES *et al.*, 1973), (VANDERPLAATS, 1999), (VENKATARAMAN, 2001), (NOCEDAL; WRIGHT, 2006), (SOFI *et al.*, 2014), (LIU; WU, 2014)
Método do Gradiente Conjugado	1	(HESTENES *et al.*, 1973), (VAN DER VORST; DEKKER, 1988), (NOCEDAL; WRIGHT, 2006) (LIU; WU, 2014), (SOFI *et al.*, 2014), (SOUZA, 2017)
Método de Newton	2	(POLYAK, 1970), (VANDERPLAATS, 1999), (VENKATARAMAN, 2001), (NOCEDAL; WRIGHT, 2006) (POLYAK, 2007)
Método de Davidon-Fletcher-Powell (DFP)	2	(KO; SAYAMA; TAKAMATSU, 1972), (HESTENES *et al.*, 1973), (BLANCO *et al.*, 1996),(NOCEDAL; WRIGHT, 2006), (LIU; SANG; ZHANG, 2016)
Método de Broyden-Fletcher-Goldfarb-Shanno (BFGS)	2	(VANDERPLAATS, 1999), (VENKATARAMAN, 2001) (NOCEDAL; WRIGHT, 2006), (IBRAHIM; MAMAT; JUNE, 2014)

Fonte: Autor

2.4.3. Otimização Combinatória

Os problemas de otimização dividem-se naturalmente em duas categorias, os que possuem as soluções com codificação de valor real e aqueles em que as soluções são codificadas com variáveis discretas. O último caso representa a classe de problemas de Otimização Combinatória (BLUM; ROLI, 2003). Um problema de Otimização Combinatória $\mathrm{P} = (S, \mathrm{f})$ pode ser definido por:

$$\mathrm{S} = \{\mathrm{s} = \{(x_1, v_1), \dots, (x_n, v_n)\} |\ v_1 \in D_1, s \qquad (8)$$

Onde $X = \{x_1, \dots, x_n\}$ é o conjunto de variáveis, $D_1, \dots, D_n$ é o domínio das variáveis, restrições sobre as variáveis, e a função objetivo f seja minimizada onde:

$$\mathrm{f}: D_1 \times \ldots \times D_n \rightarrow \mathbb{IR}^+ \quad (9)$$

Para resolver este tipo de problema é preciso encontrar uma solução $s* \in S$ com o valor mínimo da função objetivo, ou seja, $f(s*) \leq$ solução de (S, f) e o conjunto $S* \subseteq Si$. Sendo S é uma solução candidata.

Segundo (PAPADIMITRIOU; STEIGLITZ, 1998), em um problema de otimização combinatória, procura-se por um item de um conjunto finito (ou possivelmente infinitamente contável). Este item pode ser representado por um número inteiro, um subconjunto, uma permutação ou um grafo.

Problemas como de escalonamento, roteirização e agendamento são exemplos de otimização combinatória, sendo de grande importância prática e possuindo muitos algoritmos para resolvê-los. Este é o ramo de otimização mais adequado para enfrentar problemas de roteamento, já que o tratamento das variáveis de forma discreta permite uma adaptação fiel do problema (OSABA *et al.*, 2017).

Esses algoritmos podem ser classificados como completos ou aproximados, os completos possuindo a garantia de encontrar uma solução ótima para uma instância de tamanho finito em tempo limitado (PAPADIMITRIOU; STEIGLITZ, 1998). No entanto, existem problemas do tipo NP-difícil, onde o algoritmo não possui um tempo polinomial. Desta forma os métodos completos podem ter o tempo computacional exponencial no pior caso, geralmente levando tempos muito altos para um objetivo prático. É dada a preferência para estes tipos de problema para os métodos aproximados, onde a garantia de encontrar a solução ótima é sacrificada para obter-se, em tempo significativamente menor, boas soluções (BLUM; ROLI, 2003). Entre os métodos de aproximação estão os meta-heurísticos.

2.5. Métodos Meta-heurísticos

Métodos que utilizam de um processo de busca aleatório baseando em decisões probabilísticas recebem o nome de métodos probabilísticos. Esses métodos não possuem alternativas prévias para tomada de decisões, escolhendo um caminho por uma ação aleatória.

Entre os métodos probabilísticos estão os algoritmos meta-heurísticos, que vêm ganhando popularidade no meio científico devido à eficácia em encontrar soluções ótimas e na confiabilidade na solução de problemas reais (NASIR; TOKHI, 2015). Esses são inspirados em comportamentos naturais da natureza ou da biologia.

Essas meta-heurísticas incluem Algoritmos Evolutivos como Algoritmo Genético, Colônia de Formigas, Nuvem de Partículas, Recozimento Simulado, além de

inúmeros outros (TALBI, 2002). Provaram ser de grande utilidade para resolução na prática de problemas de otimização da classe NP-difícil. Em (BLUM; ROLI, 2003), os autores fazem um estudo aprofundado sobre o assunto e define uma meta-heurística como uma estrutura de algoritmos aplicados a problemas de otimização com poucas modificações para especificidades dos problemas.

Quando comparados aos métodos determinísticos, os probabilísticos diferem-se no fato de não serem influenciados pela existência de mínimos locais no problema, buscando sempre os mínimos globais. As meta-heurísticas dispensam ainda o uso de derivadas da função objetivo (BLUM; ROLI, 2003).

Os algoritmos evolucionários são as abordagens preferidas para a otimização, que utiliza de processo de seleção natural dos indivíduos, seguidos dos algoritmos de Inteligência Coletiva, que se baseiam nos comportamentos e na interação auto organizável entre os indivíduos (ERTENLICE; KALAYCI, 2018).

2.5.1. Algoritmo Genético

(HOLLAND, 1975) descreve o *Genetic Algorithm* (GA) como processo evolucionário que pode ser aplicado para solucionar uma variedade de problemas. O algoritmo, tendo como base uma população de objetos distintos, passa por transformações utilizando o princípio darwiniano de reprodução e sobrevivência dos indivíduos. Esse processo evolutivo incorpora operações genéticas como o cruzamento e mutação ao longo das gerações. Cada indivíduo da população representa uma solução possível para o problema tratado, assim a tarefa do algoritmo é encontrar uma solução boa para o problema gerando, geneticamente, a população de indivíduos.

Os parâmetros para controle do algoritmo primariamente são o tamanho da população e o número de gerações. Algumas definições são necessárias para entendimento do GA, como:

- Geração – número de iterações que o algoritmo executará simulando o processo de evolução;
- População – conjunto de cromossomos;
- Cromossomo – cadeia de informações genéticas (cadeia de genes), cada cromossomo representa uma solução viável;
- Gene – valores relativos às variáveis do problema;
- Cruzamento – operação que representa a reprodução é feita sobre cromossomos para gerar um novo indivíduo;

- Mutação – operação de variação aleatória de uma característica de um cromossomo.

Os passos probabilísticos envolvidos durante a execução do GA são:

1. Criação da população inicial;
2. Seleção de indivíduos da população para cada operação genética;
3. Seleção de pontos para troca da carga genética durante as operações.

Foi adotado o estudo do algoritmo feito por (HOLLAND, 1975) seguido por (KOZA, 1994), o fluxograma do algoritmo é exibido na Figura 2. A variável *Executar* refere-se ao número da execução, *Geração* a geração atual e *i* ao indivíduo atual da população.

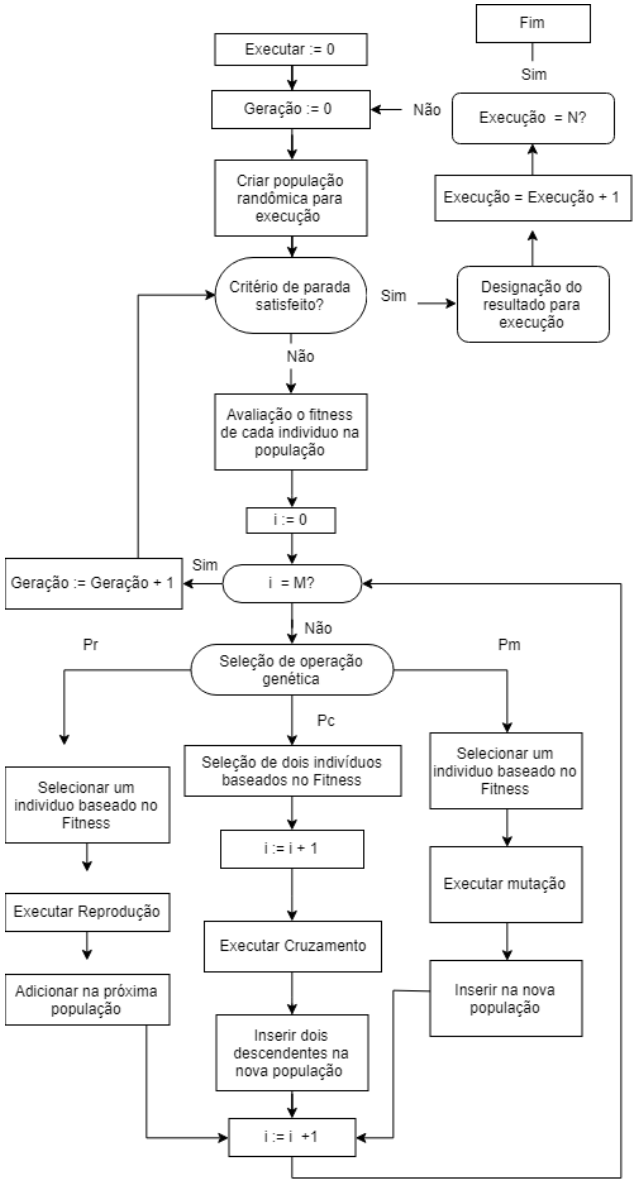

Figura 2 - Fluxograma do Algoritmo Genético convencional
Fonte: (KOZA, 1994)

A operação genética de Reprodução segue o princípio darwiniano da sobrevivência do mais apto, onde um indivíduo é selecionado probabilisticamente, em seguida copiado para a próxima geração da população. Tal seleção favorece indivíduos mais aptos, no entanto, todos tem alguma probabilidade de serem selecionados.

A operação de Cruzamento inicia com a seleção de dois indivíduos, denominados pais, baseando na carga genética destes, são gerados dois novos filhos, onde cada um possui algum material genético de cada um dos pais. Os descendentes são, geralmente, diferentes dos pais e uns dos outros, e após sua criação, são copiados para a próxima geração.

A operação de Mutação seleciona um indivíduo da população, seleciona um ponto ao longo da sequência de genes e o caractere respectivo é alterado, todos esses processos sendo realizados probabilisticamente. Este novo indivíduo é copiado para a próxima geração.

Aplicado na prática o algoritmo é eficaz na tarefa de busca de espaços complexos, altamente não lineares e multidimensionais. Um ponto positivo diferencial do algoritmo é a não necessidade de conhecimento sobre o domínio do problema ou medidas de adequação para funcionamento interno, adequando-se genericamente para diferentes problemas.

O GA tem sido amplamente aplicado na resolução de problemas de otimização, inclusive para problemas multiobjetivos, devido às vantagens do seu mecanismo de busca de soluções ótimas baseado em população (ZHANG *et al.*, 2015).

Há um grande número de variações do método GA para otimização junto de vários estudos científicos (REITER; GUTJAHR, 2012) (XIAO; KONAK, 2015) (XIAO; KONAK, 2018). As variações do GA com maior destaque atualmente são NSGA-II (*Non-dominated Sorting Genetic Algorithm*) e NSGA-III, sendo considerados os mais clássicos para problemas multiobjetivos (SHEN; ZHENG; LI, 2015) (SILVA *et al.*, 2017).

O GA possui características amplamente vantajosas, assim ocupa lugar de destaque entre as meta-heurísticas. Atualmente sua escolha se dá pela sua flexibilidade para hibridizar com outros métodos, além da facilidade de implementação para problemas multiobjetivos em otimização (ZHANG *et al.*, 2015).

2.5.2. Algoritmo Colônia de Formigas

O algoritmo colônia de formigas, também conhecido como *Ant Colony Optimization* (ACO) se baseia no comportamento de formigas ao caminhar na busca de alimentação. O principal aspecto desse comportamento é que ao caminhar as formigas liberam feromônios que marcam seu percurso, a colônia como um todo, analisa os conjuntos e traça rotas definidas pelas menores distâncias (DORIGO; BLUM, 2005).

Os estudos feitos por (DORIGO; STÜTZLE, 2004) e (DORIGO; BLUM, 2005) foram adotados neste trabalho para entendimento do algoritmo. O método tem como objetivo gerar formigas artificiais, que traçam rotas em grafos, cujos vértices são componentes de uma solução e as rotas são as soluções viáveis. O fluxograma do algoritmo é exibido na Figura 3.

A componente central do ACO é o feromônio, que consiste na formação de trilhas associadas conjunto de soluções. É usado para gerar probabilisticamente soluções para o problema, a partir da sua concentração no espaço de busca, assim o ACO faz com que as formigas espalhem uma quantidade maior de feromônio para caminhos menores, e ainda, busquem caminhar pelas regiões que contêm maior concentração do mesmo. Geralmente, o ACO aborda um problema com as seguintes etapas:

- Construir soluções usando um modelo de feromônio, ou seja, uma distribuição de probabilidade parametrizada.
- Usar as soluções para modificar os valores de feromônio de modo a se criar uma tendência.

O funcionamento do ACO é realizado de acordo com o fluxo de criação das formigas e alocação das mesmas em cada componente do espaço de busca espalhando seus feromônios. No processo de iteração as formigas se movimentam pelos demais pontos depositando seu feromônio de acordo com a parametrização estabelecida, processo de atualização de feromônio. Este processo só é definido ao final da movimentação das formigas, para que não tenha influência na iteração atual. Ao final do processo há uma operação de evaporação do feromônio para ajuste dos valores.

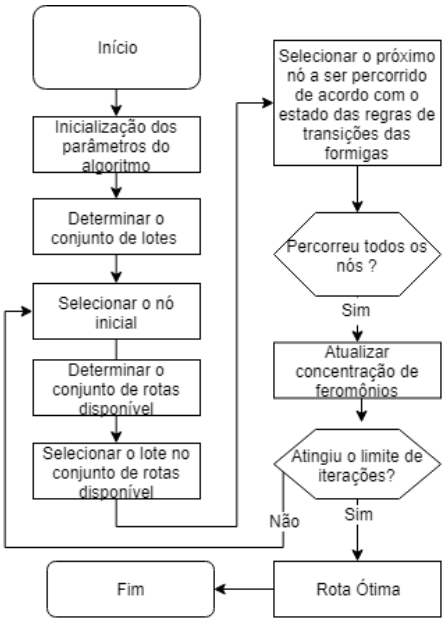

Figura 3 - Fluxograma do Algoritmo Colônia de Formigas
Fonte: Adaptado de (QIN; ZHANG; SONG, 2015)

Devido à simplicidade dos conceitos, a versatilidade e a menor quantidade de restrições, o ACO é uma alternativa promissora para problemas de ordenação sequencial, como problemas de escalonamento e roteamento (QIN; ZHANG; SONG, 2015).

Estudos identificaram que o ACO tradicional possui desvantagens de longo tempo de processamento na busca de soluções, para resolver este problema, pesquisadores propuseram alternativas como a compressão do caminho de busca, definindo os conjuntos de caminhos opcionais das formigas. (PANG *et al.*, 2014) propuseram a divisão em clusters para a instância do problema do caixeiro viajante, formando pequenas rotas para reduzir a quantidade de comparações do método, reduzindo o tempo de busca.

2.5.3. Recozimento Simulado

O algoritmo Recozimento Simulado, também conhecido como *Simulated Annealing* (SA), é uma técnica de busca local proposta por (KIRKPATRICK; GELATT; VECCHI, 1983). Este método faz analogia ao arrefecimento de elementos. No estudo da termodinâmica, o processo consiste em inicialmente fundir o material a uma alta temperatura, depois diminuir essa temperatura lentamente até que o material se solidifique e não ocorram mais mudanças.

No estudo da otimização, o método pode ser aplicado para resolver problemas gerais de otimização, em especial problemas de otimização combinatória. Computacionalmente, o *annealing* pode ser visto como um processo de determinação

estocástico da organização dos átomos de um sólido, buscando a energia mínima. Quando em alta temperatura os átomos movem-se com grande probabilidade (livremente), conforme ocorre a diminuição da temperatura, gradualmente, o movimento estabiliza em direção a uma estrutura regular (KIRKPATRICK; GELATT; VECCHI, 1983). O fluxograma do SA é exibido na Figura 4.

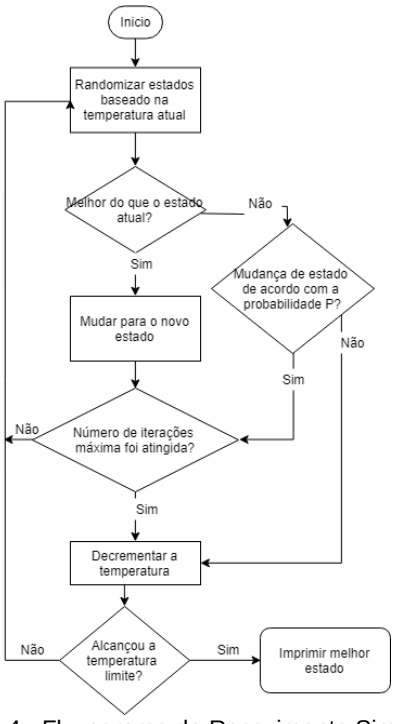

Figura 4 - Fluxograma do Recozimento Simulado
Fonte: (KIRKPATRICK; GELATT; VECCHI, 1983)

A técnica inicia a busca a partir de uma solução inicial qualquer, assumindo que exista apenas uma solução corrente (s). O procedimento consiste em gerar aleatoriamente, a cada iteração, um vizinho da solução corrente (s'). Caso ocorra melhora no valor da função objetivo da solução corrente para o vizinho gerado, o método aceita o movimento e a solução vizinha passa a ser a solução corrente, caso $\Delta < 0$ na Equação.

$$\Delta = f(s') - f(s) \tag{10}$$

Caso contrário, considera-se uma probabilidade de aceitação de um valor, que não melhore o valor da função objetivo, permitindo que o método descarte uma solução corrente por um vizinho com valor maior, ou seja, $\Delta > 0$. Essa probabilidade é dada pela Equação (11), conhecida como função de aceite, onde T é o parâmetro de temperatura do algoritmo.

$$e^{\frac{-\Delta}{T}} \tag{11}$$

A execução é iniciada com o valor T_0 elevado. Com a execução do algoritmo esse valor é reduzido gradativamente, pela Equação (12), que representa o fator de resfriamento.

$$T_k = \alpha T_{k-1} \tag{12}$$

Sendo α um valor entre 0 e 1, a função de probabilidade, inicialmente, se aproxima de um e ao final do método, quando a temperatura diminui, se aproxima do zero. Aceitando então, no decorrer da execução, cada vez menos, movimentos piores.

A técnica tem como principal vantagem, evitar a convergência precoce para um mínimo local. Conforme o tempo de execução tende ao infinito, o algoritmo converge para uma região de soluções ótimas. Quando implementado, a técnica deve adotar considerações práticas para garantir soluções boas em um tempo razoável.

(KIRKPATRICK; GELATT; VECCHI, 1983) em seu trabalho prova a eficiência do método aplicando-o no TSP. O algoritmo tem sido aplicado largamente para resolução de problemas de otimização combinatória, inclusive em problemas de roteamento de veículos.

(CHEN; CHIEN, 2011) apresentam um método baseado no uso de técnicas do Recozimento Simulado junto do Algoritmo Genético, Colônia de Formigas e Enxame de partículas para resolução de instâncias do TSP. (XIAO *et al.*, 2012) desenvolveram o SA com o conceito de troca híbrida para o problema de Taxa de Consumo de Combustível aplicado ao CVRP. (KOÇ; KARAOGLAN, 2016) desenvolveu abordagem de solução exata baseado no SA e aplicado ao VRP Verde. (TIRKOLAEE *et al.*, 2018) utiliza a hibridização do SA com o GA, sendo o primeiro para gerar soluções iniciais para o seguinte, a hibridização é aplicada ao Problema de Roteamento de Arco Capacitado e Verde. (OSABA *et al.*, 2018) desenvolve o SA evolucionário e compara com o GA tradicional e o GA baseado em Ilha, resolvendo instâncias simétricas e assimétricas do TSP.

2.6. Hibridização de meta-heurística

No contexto da otimização, diante da dificuldade em se obter a melhor solução, heurísticas são úteis para obter-se uma aproximação desta, em especial as heurísticas evolucionárias, que trabalham com algoritmos populacionais, estão entre os que encontram os resultados mais promissores. A natureza populacional é aliada a técnicas de intensificação e diversificação, tornam esses métodos aptos para proverem soluções aproximadas do ótimo (DEB, 2011). (ZHOU *et al.*, 2011) faz uma revisão detalhada do estado da arte de algoritmos evolucionários.

Nos últimos anos as pesquisas dessa área não tem foco na utilização de uma heurística específica, mas heurísticas capazes de combinar suas estratégias com outras, deixando de seguir a filosofia de heurística “pura” (RAIDL, 2006). Essa abordagem é denominada de heurística híbrida, seu objetivo está em aproveitar as boas características de um método, e evitar as características de menor qualidade, combinando estratégias para obter-se um melhor desempenho que as heurísticas originais.

A escolha de estratégias adequadas para compor um método híbrido é fundamental para atingir-se um desempenho satisfatório na resolução de problemas de otimização.

(TALBI, 2002) introduziu uma taxonomia de métodos heurísticos mono-objetivos, abordando a implementação de algoritmos e propondo uma gramática formal para esta. Seu escopo era de hibridização de heurísticas entre si, classificando diversos trabalhos da área de otimização, proporcionando uma visão geral das possibilidades de combinação de heurísticas.

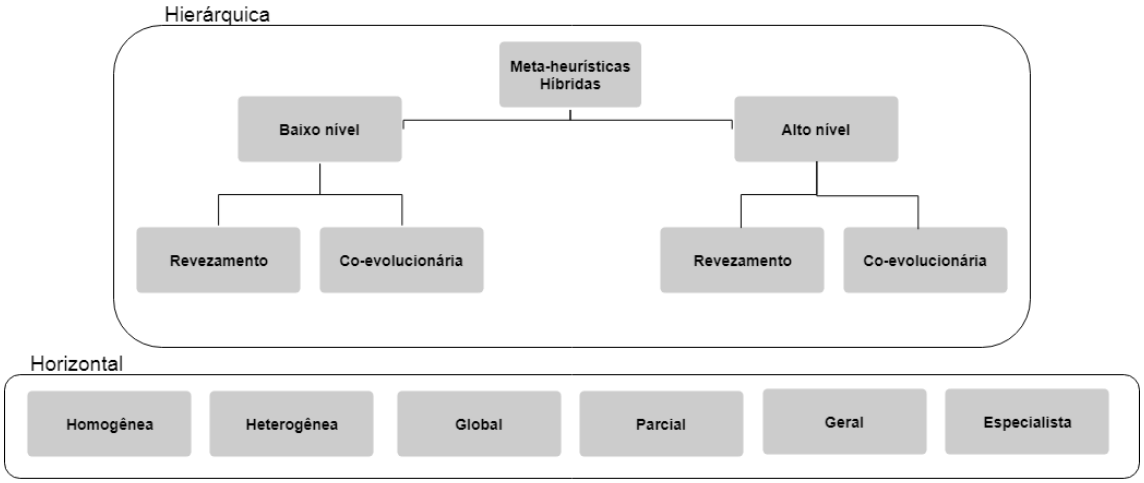

Figura 5 - Classificação das meta-heurísticas híbridas
Fonte: Adaptado de (TALBI, 2002)

As meta-heurísticas híbridas são divididas em duas classes, de alto e baixo nível. Na hibridização de baixo nível, uma estratégia específica do método original é incorporada a outro método, criando uma relação de composição. Na classe alto nível, os métodos são autônomos. Esses níveis são divididos nas classes revezamento e co-evolucionária, de acordo com a abordagem realizada. O revezamento caracteriza a aplicação de meta-heurísticas sequencialmente, onde o método gera dados para o próximo método. A co-evolucionária trabalha com o conceito de agentes cooperativos, com os métodos atuando em paralelo, onde cada agente efetua sua própria busca.

A classificação horizontal são adjetivos que caracterizam a abordagem da hibridização. Na hibridização homogênea todos os métodos utilizam da mesma meta-

heurística, independente dos parâmetros utilizados. Classifica-se como global uma hibridização onde todos os métodos exploram todo o domínio de busca. A última classificação diz respeito à resolução de problemas pelos métodos, quando todos os métodos visam o mesmo problema é classificado como geral, o oposto ocorre quando meta-heurísticas possuem formulações distintas, são ditas especialistas.

(RAIDL, 2006) estendeu a referida taxonomia, abordando propriedades adicionais que distinguem as hibridizações e proporcionam uma visão mais abrangente das mesmas.

Foram estudadas as hibridizações entre métodos meta-heurísticos e métodos exatos, como programação dinâmica, programação matemática e *branch-and-bound*. (PUCHINGER; RAIDL, 2005) adotaram duas classes de combinações, as integrativas e as colaborativas. As integrativas são as técnicas em que um algoritmo exato é embutido em uma meta-heurística (vice-versa). E as colaborativas, são técnicas onde os métodos são executados de forma independente, sequencialmente ou paralelamente. Elencaram, por meio de estudos, as vantagens e desvantagens das hibridizações existentes na literatura.

(JOURDAN; BASSEUR; TALBI, 2009) propuseram uma taxonomia de hibridização inspirada no trabalho de (TALBI, 2002), agregando ao estudo algoritmos exatos. Além da nova taxonomia, propuseram uma gramática formal, esta que proporcionou diversos algoritmos híbridos existentes para a área de otimização.

(LI, 2003) aborda em seu estudo a hibridização de meta-heurísticas evolucionárias com otimização por nuvem de partículas. (COTTA; TALBI; ALBA, 2005) e (EL-ABD; KAMEL, 2005) voltaram seus estudos para técnicas de hibridização paralela. (RAIDL; PUCHINGER, 2008) contribuíram com trabalhos de hibridização de métodos meta-heurísticos com métodos exatos, com foco nas técnicas de programação linear e inteira.

O estudo de (BLUM *et al.*, 2011) apresentou métodos híbridos de diferentes categorias, como: métodos meta-heurísticos com meta-heurísticos (heurísticos), meta-heurísticos com programação por restrições, meta-heurísticos com programação inteira, meta-heurísticos com métodos de busca em árvore subjacente e meta-heurísticos com programação dinâmica. (REITER; GUTJAHR, 2012) realizaram diferentes abordagens de hibridização com algoritmos heurísticos, uma combinação sequencial dos métodos, uma combinação interativa e uma abordagem multiobjetivo, onde antes do método heurístico iniciar, um otimizador multiobjetivo é executado e a cada iteração do método um otimizador objetivo é responsável por gerar um novo candidato.

(TANG; WANG, 2013) trabalham com a otimização por nuvem de partículas por meio da hibridização de meta-heurísticas. (KE; ZHANG; BATTITI, 2014) seguidos de (ZHANG *et al.*, 2015) realizaram a hibridização de meta-heurísticas evolucionárias com algoritmos de busca local. (RAIDL, 2015) estudou a aplicação de técnicas de decomposição em híbridos de métodos meta-heurísticas. (TALBI, 2016) trabalhou com ferramentas de otimização, utilizando a hibridização voltada para técnicas de aprendizagem de máquinas, mineração de dados, programação matemática e programação por restrições. (LIN *et al.*, 2015) desenvolvem um framework voltado à hibridização de algoritmos baseados em sistemas imunológicos artificiais.

O trabalho de (SILVA *et al.*, 2017) baseia-se na otimização por nuvem de partícula e agentes inteligentes, onde as partículas simulam agentes autônomos, que trabalham em paralelo, capazes de movimentar-se pelo espaço de busca, memorizando as melhores opções desse espaço. Este trabalho é aplicado para solução de problemas como o Caixeiro Viajante Multiobjetivo, hibridizando meta-heurísticas como GRASP, PAES, NSGA-II e Anytime-PLS. Seus resultados provam que a hibridização de tais meta-heurísticas atinge um conjunto de soluções melhores quando comparados com os conjuntos provenientes de cada método separadamente.

(RODOVALHO *et al.*, 2018) cria um procedimento híbrido para resolver um problema inverso de identificação paramétrica em mecânica de sólidos. Combina em seu estudo as meta-heurísticas: Algoritmo Genético, Colônia de Vagalumes e Evolução Diferencial. Os dois primeiros métodos trabalhando em paralelo, como agentes cooperativos, gerando resultados que após avaliação, é utilizado como entrada para o terceiro método. Tal avaliação foi feita para utilizar apenas os melhores resultados para a geração da população inicial da Evolução Diferencial. Trazem resultados que mostram que hibridizar mais de duas técnicas pode ser uma estratégia eficaz em situações onde um método tradicional, ou a hibridização de apenas dois métodos, não fornece soluções com qualidade satisfatória em custos ou tempo viável.

2.7. Plataformas para ambiente de experimentação

(PAPA *et al.*, 2017) desenvolve uma biblioteca para a implementação de métodos meta-heurísticos e otimização de funções, é composta por 15 técnicas e 112 funções de *benchmark*, além de também suportar 11 abordagens de otimização. A biblioteca é desenvolvida na linguagem C, possui código-fonte aberto e disponibiliza o código e as funções de *benchmark online*. O diferencial do trabalho está na implementação de espaços de busca baseados em hipercomplexos.

(TIAN *et al.*, 2017) desenvolve uma plataforma para otimização multiobjetivo para métodos meta-heurísticos evolutivos. Composto por 50 algoritmos evolutivos e mais de 100 problemas de teste. A plataforma é desenvolvida em Matlab, possui o código-fonte aberto e possibilita que seus resultados sejam exportados para o Excel ou Latex.

(EVANGELISTA; MAIA; ROCHA, 2009) cria uma biblioteca para implementação de algoritmos de otimização, aborda métodos meta-heurísticos evolutivos. Desenvolvido na linguagem JAVA, com código-fonte aberto, sendo aplicada nos campos de Bioinformática, a Mineração de Dados e Otimização de Redes de Computadores.

(LIEFOOGHE; JOURDAN; LEGRAND, 2007) cria uma estrutura genérica com algoritmos evolutivos multiobjetivos que incorpora técnicas para a resolução baseada em Pareto. Desenvolvido em C++, com código-fonte aberto, paradigma de orientação a objetos e o conceito bem aplicado de caixa branca, a plataforma dispõe de modelos paralelos e distribuídos, bem como procedimentos de hibridização podem ser aplicados a um algoritmo projetado dentro da plataforma.

(FARIS *et al.*, 2016) é uma plataforma que implementa algoritmos meta-heurísticos bio-inspirados. Desenvolvida em Python, de código aberto e multiplataforma, tem como objetivo facilitar o uso de meta-heurística por não especialistas. Possibilita a criação de novos algoritmos e a hibridização e analise dos atuais. Seu estudo faz a comparação com o ambiente do Matlab, concluindo que o Python mostra eficiência em termos de execução para problemas com alta dimensão e grandes números de variáveis, apontando a linguagem como a preferencial para realizar experimentos em meta-heurísticas.

(SHEN; ZHENG; LI, 2015) apresentam uma plataforma para otimização multiobjetivo, com uma biblioteca de modelos de otimização. Foi desenvolvida em C++, de código-fonte aberto, segue o paradigma de orientação a objetos, possui um banco de dados que armazena dados experimentais. Os experimentos podem ser realizados por meio de definições em um arquivo de configuração, possibilitando que sejam realizados sem a intervenção humana. Os métodos podem ser executados em paralelo e serial, e ainda pode-se ativar a opção de programação distribuída. A plataforma foi convertida em módulos na linguagem Python, para o ambiente de experimentação, concluindo que, comparado com o C++, o script Python é mais adequado para construir a plataforma experimental.

(PEREZ; JANSEN; MARTINS, 2012) cria uma plataforma para formular e resolver problemas de otimização não lineares restritos. Desenvolvido na linguagem de programação Python e integrada com as linguagens Fortran e C, por meio de uma

extensão em C++. Possui o código-fonte aberto, utilizando o paradigma de orientação a objetos, com técnicas de herança de classes e sobrecarga de operadores, onde pesquisadores com o conhecimento em desenvolvimento de softwares podem desenvolver seus próprios métodos de otimização.

(RODRIGUEZ-FDEZ *et al.*, 2015) apresenta uma plataforma de análise estatística sobre resultados obtidos por métodos de inteligência computacional. A plataforma foi desenvolvida em Python e é disponibilizada *online*, utiliza de recursos JSON e AJAX (componentes da linguagem JAVASCRIPT). Tem objetivo principal orientar o usuário em cada etapa do processo de teste, com foco na seleção do teste estatístico apropriado, priorizando a usabilidade, para apoiar usuários não especialistas. Ainda fornece seus recursos em aplicação para planilhas eletrônicas.

(EGEA *et al.*, 2014) disponibilizam uma ferramenta que implementa meta-heurísticas para problemas da bioinformática. Possui métodos para diferentes classes de problemas como: busca por dispersão para programação não linear e programação inteira mista; e busca de vizinhança variável para problemas de programação inteira. A ferramenta possui a opção de processamento paralelo, utilizando agentes cooperativos. Desenvolvido nas linguagens R e Matlab com a possibilidade de adaptação para o Python, de código-fonte aberto, permite a adição de outros métodos.

(AUDET; DANG; ORBAN, 2014) desenvolve um sistema híbrido para modelar e resolver problemas de otimização de algoritmos. Utiliza um algoritmo como entrada (script Python), sugerindo valores de parâmetros que maximizem métricas de desempenho. Trabalha com a possibilidade de processamento paralelo e torna claro o aproveitamento da sua utilização. Desenvolvido na linguagem de programação Python, com código-fonte aberto e possibilidade de combinar o sistema a outras ferramentas, para gerar modelos substitutos ou executar simulações.

(MARTINS, 2018) desenvolve uma plataforma *web* para o problema do escalonamento da produção com parâmetros inteligentes para resolução de instâncias de *benchmark*. Utiliza o Algoritmo Genético para a resolução do problema de otimização. A plataforma foi desenvolvida na linguagem Python e está disponível no endereço (http://iproductionscheduling.com/).

2.8. Problema do Caixeiro viajante

O Problema do Caixeiro Viajante (TSP) pode ser descrito como a busca do menor caminho para um vendedor que sai da sua cidade natal, visite um conjunto específico de cidades e retorne para sua origem. Dado uma matriz $D = (d_{ij})$ simétrica de n por n, onde d_{ij} representa a distância entre i e j, o problema está em organizar

os pontos, em ordem cíclica de modo que a soma de d_{ij} entre esses pontos, consecutivamente, seja a mínima. O número de combinações é dado por $\frac{1}{2}(n-1)!$, fato que ocasiona na dificuldade em escolher qual método é eficiente na tarefa de encontrar rotas mínimas quando o valor de n for grande (DANTZIG *et al.*, 1954).

Um conhecido problema de otimização da classe NP-difícil, possuindo ordem exponencial de complexidade (CUNHA; BONASSER; ABRAHÃO, 2002), consiste em traçar uma rota que visite exatamente uma vez cada vértice de um grafo ao final retornando ao ponto inicial, formando um ciclo hamiltoniano (BAE; RATHINAM, 2012).

É uma referência aplicável para várias situações da vida real que busquem soluções de menores custos. Grande parte das abordagens desenvolvidas na área de otimização foram validadas no TSP (WANG *et al.*, 2015).

O problema tem a importância indiscutível, tanto em aspecto prático, como teórico, por ter extensa aplicação prática, significativa relação com outros modelos e dificuldade de solução exata (REINELT, 1991).

Várias formulações foram criadas para o TSP, como diferentes formulações possuem propriedades distintas, o estudo torna-se de grande importância. Segue a formulação apresentada no trabalho de (DANTZIG *et al.*, 1954):

$$Minimize \sum_{j=1}^{n}\sum_{i=1}^{n} c_{ij}x_{ij} \tag{13}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j \in N \; i \neq j \tag{14}$$

$$\sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in N \; i \neq j \tag{15}$$

$$\sum_{i,j \in S_t} x_{ij} \leq |S| - 1, \quad \forall S \subset N \qquad 2 \leq |S| \leq n-2 \tag{16}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in N \tag{17}$$

Onde a variável binária x_{ij} assume valor 1 se o arco $(i,j) \in N$ for escolhido como parte da solução, S é um subgrafo de G, onde $|S|$ representa seu número de vértices.

Várias meta-heurísticas foram aplicadas para resolver o TSP, na literatura são encontradas as variações do problema e os métodos utilizados. (PACHECO; FUKASAWA, 2013) desenvolvem o algoritmo genético para o TSP. (ABELEDO *et al.*, 2013) resolveram o TSP dependente do tempo com o algoritmo *branch-cut-and-price*. (SILVA *et al.*, 2013a) desenvolvem algoritmos heurísticos para a otimização de rotas do TSP. (WANG *et al.*, 2015) aplicam o método de Colônia de Abelhas na resolução do TSP incerto e multiobjetivo. (ABREU; OLIVEIRA; LACERDA, 2015) utilizam da

técnica de mapa auto organizável com aprendizado *winner takes all* para o TSP. A literatura mostra a eficiência na utilização do TSP para minimização de custos e a importância deste conceito como origem para outros abordados atualmente.

2.9. Problema do Roteamento de veículos

O Problema do Roteamento de Veículos (VRP) consiste em traçar um conjunto de rotas em que uma frota de veículos atenda um conjunto de clientes, satisfazendo algumas restrições (BODIN, 1983). As rotas a serem traçadas, devem de alguma forma, considerar a minimização de custos como: custo total do roteamento, número de veículos a serem utilizados, tempo total gasto, entre outros (SIMAS; GÓMEZ, 2007).

O VRP foi proposto por (DANTZIG; RAMSER, 1959), como uma generalização do TSP. O VRP situa-se no controle logístico de veículos e atividades de coleta e distribuição de produtos, sua dificuldade está na construção de um conjunto viável de rotas, de modo que o caminho total percorrido seja o menor possível para cada veículo.

Serviços de entrega do mundo real podem ser formulados com as variações do VRP (SOLOMON, 1987), porém, em sua forma padrão, o problema é considerado como do tipo NP-hard, possuindo ordem exponencial de complexidade (CORDEAU *et al.*, 2002). Segundo (CORDEAU *et al.*, 2002), métodos exatos são capazes de resolver apenas instâncias do VRP com menos de vinte vértices em tempo polinomial, acima disso, os métodos exatos perdem a eficiência para este tipo de problema.

A versão clássica do VRP consiste em uma frota de veículos atendendo pedidos de clientes, partindo e retornando a um depósito central. As principais restrições do VRP são: determinar um roteamento com o menor custo possível, adotar um ponto inicial, passar por cada cliente uma única vez e retornar ao depósito (LAPORTE, 2009).

A formulação do VRP clássico é inspirada na publicada por (FISHER; JAIKUMAR, 1981). Na formulação os seguintes dados são considerados:

x_{ijk} = Variável binária que assume valor 1 quando o veículo k visita o cliente j imediatamente após o cliente i, 0 em caso contrário.

y_{ik} = Variável binária que assume o valor 1 se o cliente i é visitado pelo veículo k, 0 em caso contrário.

q_i = Demanda do cliente i.

Q_k = Capacidade do veículo k.

c_{ij} = Custo de percorrer o trecho que vai do cliente i ao j.

Considerando um grafo $G = (N, M)$ com n vértices e m arestas, o problema pode ser formulado como se segue:

$$Minimize \sum_{i,j} \left(c_{ij} \sum_{k} (x_{ijk}) \right) \tag{18}$$

Sujeito a:

$$\sum_{k} y_{ik} = 1, \qquad i = 2, \dots, n \tag{19}$$

$$\sum_{k} y_{ik} = m, \qquad i = 1 \tag{20}$$

$$\sum_{i} q_i y_{ik} \leq Q_k, \qquad k = 1, \dots, m \tag{21}$$

$$\sum_{j} x_{ijk} = \sum_{j} x_{jik} = y_{ik}, \qquad i = 1, \dots, n \quad k = 1, \dots, m \tag{22}$$

$$\sum_{i,j \in S} x_{ijk} \leq |S| - 1, \qquad \forall S \subseteq \{2, \dots, n\}, \ k = 1, \dots, m \tag{23}$$

$$y_{ik} \in \{0,1\} \qquad i = 1, \dots, n \quad k = 1, \dots, m \tag{24}$$

$$x_{ijk} \in \{0,1\} \qquad i, j = 1, \dots, n \quad k = 1, \dots, m \tag{25}$$

A restrição 19 assegura que um veículo não visite mais de uma vez um cliente, a restrição 20 garante que cada rota comece e termine no depósito, a restrição 21 delimita o uso da capacidade do veículo, a restrição 22 garante que um veículo não pare sua rota em um cliente e a restrição 23 é a eliminação de sub-rotas.

O VRP possui uma diversidade de restrições que exigem a criação de variações para o problema. O VRP Capacitado define sua frota com veículos idênticos que possuem uma capacidade de carga máxima (XIAO *et al.*, 2012), o VRP com Múltiplos Depósitos determina um conjunto de rotas para cada depósito dentro de um número limite associado ao mesmo (MONTOYA-TORRES *et al.*, 2015), o VRP Periódico armazena os pedidos dos clientes de forma que sirvam de entrada para a resolução parcial do problema (MOURGAYA; VANDERBECK, 2008), o VRP com Janelas de Tempo impõe que cada rota deve ocorrer no intervalo de tempo vinculado ao depósito (GENDREAU; TARANTILIS, 2010). O VRP com Coleta e Entrega onde o cliente pede pelos serviços de coleta e entrega simultaneamente, tornando-se um depósito (BERBEGLIA *et al.*, 2007). O VRP com Frota Heterogênea é o problema onde a frota é composta por diferentes tipos de veículos, com diferentes capacidades

(BALDACCI; BATTARRA; VIGO, 2007). O VRP Aberto desobriga o retorno do veículo para o depósito após atender o último cliente (NIU *et al.*, 2018). O VRP Dinâmico elabora as rotas de forma *online*, comunicando ao veículo qual o cliente deve ser atendido quando ficar ocioso (PILLAC *et al.*, 2013). O VRP Verde é o problema que considera aspectos do tráfego para redução de emissão de CO_2 ao meio ambiente.

Devido à importância econômica e a alta complexidade do problema, as variações do VRP estão sendo estudadas largamente pela comunidade científica (CORDEAU *et al.*, 2002) (LAPORTE, 2009) (POLLARIS *et al.*, 2015) (BRAEKERS; RAMAEKERS; VAN NIEUWENHUYSE, 2016).

Por meio de uma busca realizada na base de dados *Web of Science* e interpretada pelo *software* de análise de metadados VOSviewer, foi possível mapear quais assuntos abordam o termo "*vehicle routing problem"* nos dois últimos anos, conforme exposto na Figura 6.

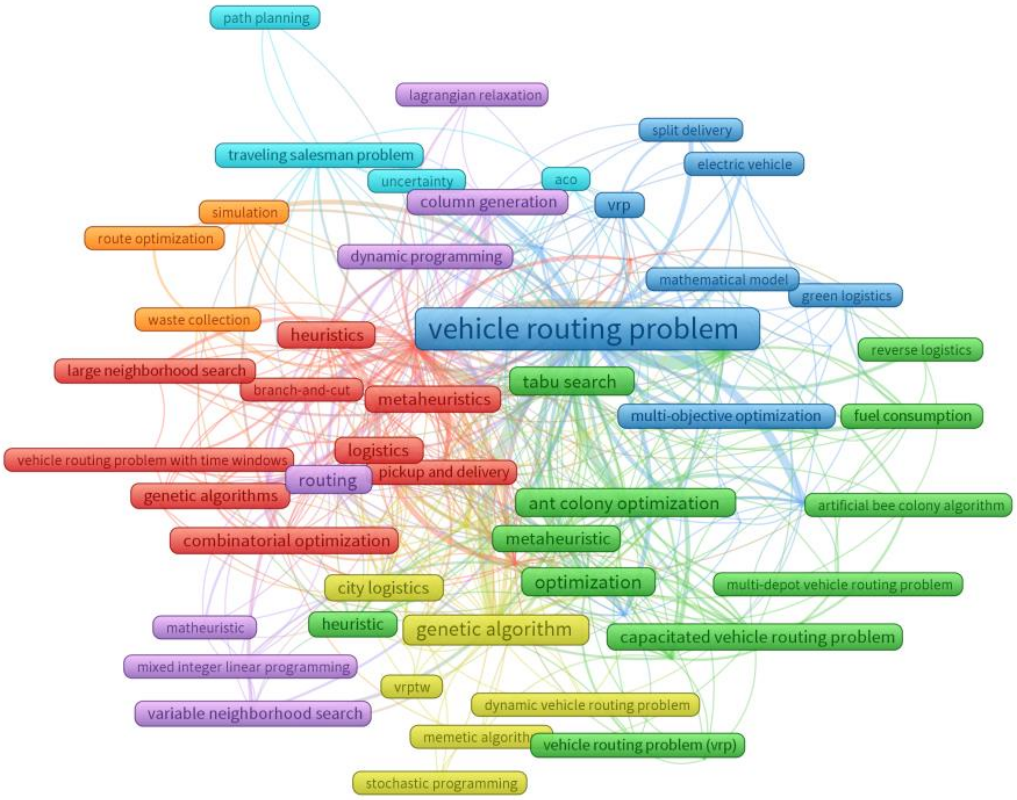

Figura 6 - Mapa de metadados da pesquisa sobre o VRP no software VOSviewer
Fonte: Autor

O estudo indica uma tendência em direções às variantes do VRP que incluem restrições e suposições da vida real, com modelos mais realistas e soluções que se aproximem das aplicáveis na prática. Considerando a modelagem de problemas reais, deve-se dar atenção aos temas de problemas VRP Aberto, VRP Dinâmico e VRP Verde (BRAEKERS; RAMAEKERS; VAN NIEUWENHUYSE, 2016).

Foi constatada a presença de meta-heurísticas e hibridizações com as técnicas de Colônia de Formigas (MESSAOUD; EL BOUZEKRI EL IDRISSI; ALAOUI, 2018), Algoritmo Genético (XIAO; KONAK, 2018), Colônia de Abelhas, Exame de Peixes (CHE *et al.*, 2018), Busca Tabu (NIU *et al.*, 2018), Programação Inteira Mista (NABIL; FAROUK; EL-KILANY, 2018) e Simulação de Crescimento de Plantas (WANG; ZHAI; GENG, 2018).

O VRP Capacitado (CVRP), selecionado como objeto de estudo deste trabalho, possui a seguinte formulação matemática (KOUBEK; LINSER, 2013):

$$Minimize \sum_{k=1}^{K} \sum_{i=1}^{N+1} \sum_{j=1}^{N+1} (d_{ij}\, x_{ij}^{k}) \tag{26}$$

De modo que:

$$\sum_{k=1}^{K} \sum_{j=2}^{N+1} x_{ij}^{k} \leq k, para\ i = 1 \tag{27}$$

$$\sum_{j=2}^{N+1} x_{ij}^{k} = \sum_{j=2}^{N+1} x_{ji}^{k}, para\ i = 1\ e\ k \in \{1,2, \dots, K\} \tag{28}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{N+1} x_{ji}^{k} = 1, para\ i \in \{2,3, \dots, N+1\} \tag{29}$$

$$\sum_{k=1}^{K} \sum_{j=1}^{N+1} x_{ji}^{k} = 1, para\ j \in \{2,3, \dots, N+1\} \tag{30}$$

$$\sum_{i=1}^{N+1} x_{ij}^{k} \sum_{j=1}^{N+1} x_{ji}^{k} = 0, para\ j\ \in \{1,2, \dots, N+1\}\ e\ k\ \in \{1,2, \dots, K\} \tag{31}$$

$$\sum_{j=2}^{N+1} q_i \left(\sum_{j=2}^{N+1} x_{ij}^{k} \right) < Q, k\ \in \{1,2, \dots, K\} \tag{32}$$

O objetivo do CVRP é minimizar a distância total percorrida (Equação (26)) pelos veículos, o problema possui a restrição de número máximo de veículos disponíveis (Equação (27)) que também representam o número de rotas a serem percorridas. Todas as rotas devem ter o depósito como origem e também como destino (Equação (28)). Todos os clientes devem ser visitados uma única vez (Equação (29) e Equação (30)). Todo veículo deve chegar e partir de cada cliente que

atender (Equação (31)). O veículo não pode carregar mais do que é imposto como capacidade Q (Equação (32)) e fazem parte de uma frota homogênea.

O problema é amplamente abordado na literatura com diferentes métodos, como: algoritmo *Branch-and-cut* (AUGERAT *et al.*, 1995) (LYSGAARD; LETCHFORD; EGLESE, 2004), algoritmo Evolução Diferencial (KOUBEK; LINSER, 2013), algoritmo Enxame de Partículas (CHEN; CHEN, 2014), Algoritmo Genético (ISWARI; ASIH, 2018), Colônia de Abelhas (MINGPRASERT; MASUCHUN, 2017). Foram aplicadas abordagens híbridas para resolução do problema, como: *Centroid-Based Heuristic* (SHIN; HAN, 2011), algoritmo *Sweep* e *Nearest Neighbor* (ABDELAZIZ; EL-GHAREEB; KSASY, 2014), algoritmo Colônia de Formigas aplicado a uma variação do CVRP (POLLARIS *et al.*, 2015), *Sweep* e Enxame de Partículas (AKHAND; JANNAT; MURASE, 2018), Algoritmo Genético e *Nearest Neighbor* (LIMA; DE ARAÚJO; SCHIMIT, 2018).

3. METODOLOGIA

3.1. Ambiente experimental de otimização

Para o desenvolvimento do ambiente de experimentação foi adotada a linguagem de programação Python. Sua escolha foi inicialmente motivada pela simplicidade oferecida pela linguagem na manipulação de funções multimodais complexas (SILVA *et al.*, 2013b), reforçada pela extensa quantidade de bibliotecas padrões, que fornecem acesso a funcionalidades do sistema operacional, como a manipulação de arquivos (ROSSUM; DRAKE, 2017a), dando as funcionalidades ao ambiente de importação e exportação de dados.

A linguagem possui, além das bibliotecas padrões, uma crescente coleção de programas e módulos individuais para pacotes e estruturas (ROSSUM; DRAKE, 2017a), como o *SymPy*, biblioteca utilizada neste ambiente para manipulação de funções e dados simbólicos; *Flask*, para criação da aplicação *web*, permitindo que a plataforma forneça seu acesso *online*, característica que torna o ambiente multiplataforma, propiciando maior usabilidade para o serviço; e a *MatPlotlib*, para manipulação de gráficos.

O Python possui manipulação de alto nível de estruturas de dados por meio de uma abordagem simples, facilitando a aplicação do paradigma de orientação a objetos, utilizado para estruturar os códigos deste ambiente. A linguagem é extensível, o que possibilita que os métodos desenvolvidos no ambiente possam ser exportados pelo usuário e adicionados como extensão (ou linguagem de comando) para o seu aplicativo, mesmo que em outras linguagens (ROSSUM; DRAKE, 2017b).

Na busca de disponibilizar acesso ao material da pesquisa, foi modelado o ambiente experimental com acesso via *web*. Foram adotadas formas de desenvolvimento para o ambiente com o objetivo de generalizar as abordagens para o acionamento dos métodos na solução de problemas. Para isso, uma padronização de código foi utilizada de modo que a inserção de novos métodos ou problemas ao ambiente seja facilitada. O ambiente possui um diagrama de classe que representa a estrutura em que a codificação foi elaborada, conforme Figura 7.

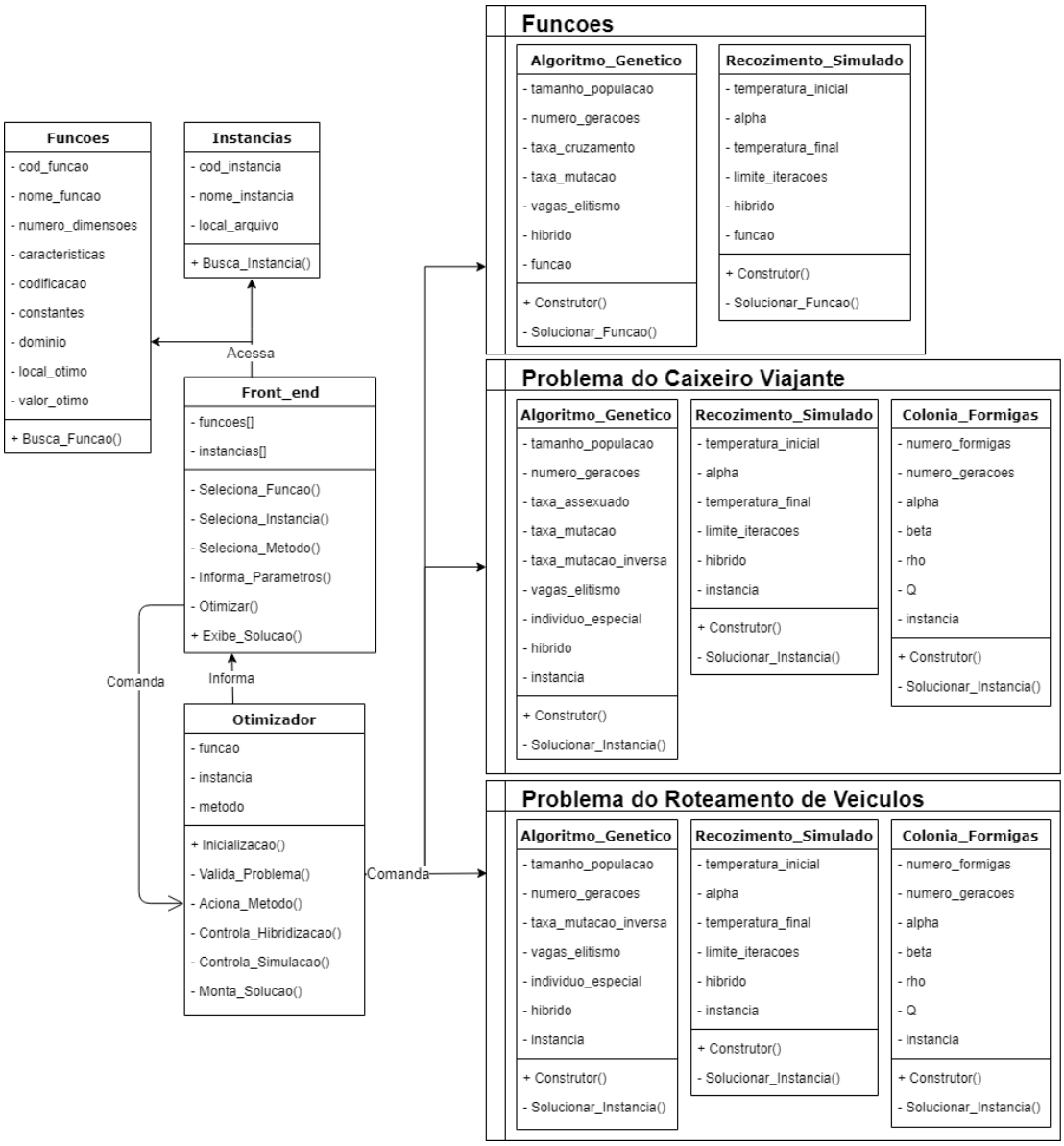

Figura 7 - Diagrama de classe do ambiente *web*
Fonte: Autor

O ambiente é organizado com os seguintes itens:

- Classe de Funções: reconhece e controla as funções de *benchmark* disponíveis no banco de funções.
- Classe de Instâncias: reconhece os arquivos de instâncias de *benchmark* disponíveis para cada tipo de problema, realiza a leitura dos dados e permite a substituição por valores informados pelo usuário.
- *Front-end*: conjunto de páginas da *web* que fornecem ao usuário o acesso ao ambiente, escolha dos métodos a serem utilizados e seleção de funções ou instâncias. É onde o usuário formulará o problema, escolhendo os parâmetros a serem utilizados. Permite que o usuário faça experimentos com os métodos, e ao final exibe os resultados obtidos.

- Otimizador: recebe do *front-end* os comandos do usuário e atende as requisições. Comanda os métodos a serem executados, passando os parâmetros e trocando as informações, nos casos de hibridização. Para o caso de simulação, controla as repetições dos métodos informando ao *front-end* a evolução do processo e os tempos aproximados de espera. Em todos os casos, gera os resultados, monta os gráficos e informa ao *front-end* para exibição ao usuário.
- Algoritmos: divididos de acordo com o direcionamento dos problemas, recebem os parâmetros do Otimizador, executam suas lógicas para solucionar o problema e, ao final, devolvem ao Otimizador seus resultados.

3.2. Criação dos Bancos de Dados dos Problemas

Para a resolução de problemas pela plataforma é necessário que o mesmo seja informado e, considerando a complexidade de informações que um problema possui, unido da necessidade da confiabilidade das informações, optou-se pelo armazenamento de diversas opções em um banco de dados.

A escolha dos mesmos foi feita levando em consideração os problemas resolvidos pela literatura, buscando diversidade e diferenças entre os problemas do mesmo tipo. Tornando possível a análise dos métodos desenvolvidos para solução de problemas de diferentes escalas, desde pequenos até muito grandes. Os problemas são divididos em duas categorias, Funções e Instâncias.

3.2.1. Banco de funções

As funções de *benchmark* são utilizadas para comparar o desempenho dos métodos, nelas já são conhecidos os parâmetros utilizados e as soluções ótimas encontradas. O ambiente possui 40 funções de *benchmark* obtidas na literatura que podem ser visualizadas na Tabela 2. Estas funções são multimodais, multidimensionais, continuas, não separáveis, diferenciáveis, convexas e não convexas, com intervalos de buscas, dimensões e escalas distintas.

Os dados das funções são armazenados em um documento de texto, com a extensão *txt*, que é lido pela plataforma nos momentos necessários. No documento existem dados necessários para a resolução de problemas, como:

- Número de dimensões – usado para os casos em que são pré-definidos, nos casos multidimensionais, possui o valor *n*, que deverá ser informado pelo usuário por meio da plataforma;

- Formulação matemática – equação a ser resolvida, informada de forma simbólica, adaptada à sintaxe de leitura da biblioteca *SymPy*, não sendo necessária a validação de sua estrutura.
- Constantes – valores auxiliares utilizados para resolução da formulação matemática, não sendo obrigatórios.
- Domínio de busca – espaço onde o método deverá buscar as soluções.

Além desses dados, ainda existem valores informativos, disponibilizados para o usuário, porém não acessados pelos métodos, como: descrição das funções, local da melhor solução dentro do espaço de busca, valor da melhor solução.

Tabela 2 - Lista de Funções adicionadas ao banco de funções

Ackley	Bohachevsky N. 2	Goldstein-Price	Qing
Ackley N. 2	Booth	Gramacy & Lee	Rastrigin
Ackley N. 3	Bukin N. 6	Himmelblau	Rosenbrock
Ackley N. 4	Cross-in-Tray	Holder-Table	Salomon
Adjiman	Deckkers-Aarts	Leon	Schaffer N. 1
Alpine N. 1	DeVilliers-Glasser 2	Levi N. 13	Schaffer N. 3
Bartels Conn	Drop-Wave	Matyas	Schaffer N. 4
Beale	Easom	McCormick	Schwefel 2.20
Bird	Egg Crate	Periodic	Sphere
Bohachevsky N. 1	Exponential	Powell Sum	Zakharov

Fonte: Autor

O uso dessas funções de *benchmark* para otimização é largamente abordado na literatura (NASIR; TOKHI, 2015), tem como vantagem a possibilidade de modificar características da função, como o número de dimensões, aumentando a complexidade da mesma (ETMINANIESFAHANI; GHANBARZADEH; MARASHI, 2018). A Tabela 3 mostra algumas das publicações que utilizaram funções de *benchmark*.

O Banco de Funções foi modelado em um arquivo, onde cada linha representa uma função, desta forma, para inserir uma nova função no ambiente, é preciso inserir uma nova linha com as informações da função.

(ARDEH, 2016) foi o objeto de estudo utilizado como referência na tarefa de levantamento dos dados das funções. Na pesquisa é disponibilizada uma diversidade de informações sobre as funções, além de gráficos e códigos de implementação do Matlab, aspectos que auxiliaram no processo de construção do banco de dados.

O ambiente exibe nas páginas de resultado gráficos da superfície e de contorno da função, para demonstrar visualmente para o usuário a complexidade da mesma. Em casos de funções multidimensionais, estes gráficos são adaptados para tridimensionais.

Tabela 3 - Trabalhos envolvendo o uso de funções

TRABALHO	**PROBLEMA**
(RAIDL; PUCHINGER, 2008)	Algoritmo de otimização para Programação Linear
(EVANGELISTA; MAIA; ROCHA, 2009)	Plataforma de otimização com meta-heurísticas
(JOURDAN; BASSEUR; TALBI, 2009)	Taxonomia de métodos de otimização determinísticos e meta-heurísticos
(PEREZ; JANSEN; MARTINS, 2012)	Plataforma de otimização com métodos determinísticos em Python
(TANG; WANG, 2013)	Algoritmo híbrido multiobjetivo para otimização com meta-heurísticas
(EGEA *et al.*, 2014)	Plataforma de otimização com meta-heurísticas
(AUDET; DANG; ORBAN, 2014)	Plataforma de otimização com métodos determinísticos
(ZHANG *et al.*, 2015)	Algoritmo híbrido multiobjetivo para otimização com meta-heurísticas
(FARIS *et al.*, 2016)	Plataforma de otimização com meta-heurísticas em Python
(TIAN *et al.*, 2017)	Plataforma multiobjetivo para otimização com meta-heurísticas
(PAPA *et al.*, 2017)	Plataforma para prototipagem de técnicas de otimização com meta-heurísticas
(RODOVALHO *et al.*, 2018)	Abordagem híbrida para otimização com meta-heurísticas
(ETMINANIESFAHANI; GHANBARZADEH; MARASHI, 2018)	Algoritmo de otimização com meta-heurística

Fonte: Autor

A codificação desse tipo de problemas exige o uso de uma biblioteca que permite manipular valores simbólicos, utilizando no ambiente, a *SymPy*. Essa biblioteca permite a formulação dos problemas em forma de equações, definindo variáveis e constantes simbólicas que podem ser substituídas por valores numéricos para resolução. Para conseguir trabalhar com a biblioteca foi necessário um estudo das possíveis abordagens de trabalho com a mesma.

3.3. Banco de Instâncias

O ambiente utiliza a modelagem dos problemas de roteirização organizados em arquivos denominados instâncias, onde cada instância possui as variáveis, restrições e objetivos definidos para o mesmo (PEREZ; JANSEN; MARTINS, 2012). Este procedimento é adotado como padronização dos problemas a serem resolvidos.

A vantagem da utilização de instâncias é a quantidade de variações disponíveis na literatura junto dos testes e resultados executados pelos autores. Esta característica possibilita a avaliação dos resultados obtidos pelos métodos do ambiente, coletando os resultados dos experimentos e comparando com o material já publicado, este procedimento classifica as instâncias como de *benchmark*.

O ambiente possui a classe responsável pela leitura dos arquivos de instância. Estes arquivos têm as extensões “*tsp”* e “*vrp”*, para os problemas do caixeiro

viajante e de roteamento de veículos capacitado, respectivamente. Esta classe faz a leitura dos dados dentro da padronização definida pelo tipo de instância, seguindo os padrões definidos na literatura (REINELT, 1991).

Existe uma quantidade considerável de trabalhos envolvendo o VRP e o TSP utilizando instâncias de *benchmark*, além de *survey's* (BERBEGLIA *et al.*, 2007) (ADEWUMI; ADELEKE, 2016) e revisões (CORDEAU *et al.*, 2002)(GENDREAU; TARANTILIS, 2010)(POLLARIS *et al.*, 2015) sobre as aplicações. A Tabela 4 exibe alguns trabalhos da literatura utilizando instâncias de *benchmark* e os problemas em que foram aplicados.

Tabela 4 - Trabalhos envolvendo o uso de instâncias

TRABALHO	**PROBLEMA**
(SOLOMON, 1987)	Problema de roteamento e agendamento de veículos com janelas de tempo
(TSCHÖKE *et al.*, 1995)	Problema do caixeiro viajante
(TAM; MA, 2004)	Problema de roteamento de veículos com janelas de tempo
(BALDACCI; BATTARRA; VIGO, 2007)	Problema de roteamento de veículos com frotas heterogêneas
(MOURGAYA; VANDERBECK, 2008)	Problema de roteamento de veículos periódico
(GUNES; CORDEAU; LAPORTE, 2010)	Problema do caixeiro viajante
(REGO *et al.*, 2011)	Problema do caixeiro viajante
(ABELEDO *et al.*, 2013)	Problema do caixeiro viajante dependente do tempo
(ABREU; OLIVEIRA; LACERDA, 2015)	Problema do caixeiro viajante por mapa auto organizável
(WANG *et al.*, 2015)	Problema do caixeiro viajante multiobjetivo
(XIAO; KONAK, 2015)	Problema de roteamento de veículos verde com variação de tempo de congestionamentos
(NIU *et al.*, 2018)	Problema de roteamento de veículos verde e aberto
(XIAO; KONAK, 2018)	Problema de roteamento e agendamento de veículos verde

Fonte: Autor

Assim o ambiente disponibiliza ao usuário um banco de instâncias de *benchmark* com 270 opções retiradas da literatura, sendo 77 instâncias do TSP e 193 do CVRP. As instâncias do TSP abordam o problema de forma simétrica e foram embasadas nos trabalhos de (CHRISTOFIDES; EILON, 1969), (REINELT, 1991), (WANG *et al.*, 2015) e (OSABA *et al.*, 2018), com a maioria fazendo parte da biblioteca TSPLIB (REINELT, 1991), elas são listadas na Tabela 5.

Tabela 5 - Lista de Instâncias do TSP adicionadas ao banco de instâncias

a280	d1655	fl3795	kroD100	pr76	pr1002	rl1889	u1432
berlin52	d2103	fnl4461	kroE100	pr107	pr2392	rl5915	u1817
bier127	d15112	gil262	lin105	pr124	rat99	rl5934	u2152
brd14051	d18512	kroA100	lin318	pr136	rat195	rl11849	u2319
ch130	eil51	kroA150	linhp318	pr144	rat575	st70	usa13509
ch150	eil76	kroA200	nrw1379	pr152	rat783	ts225	vm1084
d198	eil101	kroB100	p654	pr226	rd100	tsp225	vm1748
d493	fl417	kroB150	pcb442	pr264	rd400	u159	
d657	fl1400	kroB200	pcb1173	pr299	rl1304	u574	
d1291	fl1577	kroC100	pcb3038	pr439	rl1323	u724	

Fonte: Autor

As instâncias do CVRP foram baseadas nos trabalhos de (CHRISTOFIDES; EILON, 1969), (CHRISTOFIDES; MINGOZZI; TOTH, 1979), (FISHER, 1994), (AUGERAT *et al.*, 1995) e (UCHOA *et al.*, 2016), este último faz um relacionamento entre todas as instâncias e disponibilizando-as de forma organizada e simplificada para utilização. As instâncias são exibidas na Tabela 6.

Tabela 6 - Lista de Instâncias do CVRP adicionadas ao banco de instâncias

A-n32-k5	A-n69-k9	B-n78-k10	P-n21-k2	X-n115-k10	X-n228-k23	X-n344-k43	X-n586-k159
A-n33-k5	A-n80-k10	E-n101-k14	P-n22-k2	X-n120-k6	X-n233-k16	X-n351-k40	X-n599-k92
A-n33-k6	B-n31-k5	E-n101-k8	P-n22-k8	X-n125-k30	X-n237-k14	X-n359-k29	X-n613-k62
A-n34-k5	B-n34-k5	E-n22-k4	P-n23-k8	X-n129-k18	X-n242-k48	X-n367-k17	X-n627-k43
A-n36-k5	B-n35-k5	E-n23-k3	P-n40-k5	X-n134-k13	X-n247-k50	X-n376-k94	X-n641-k35
A-n37-k5	B-n38-k6	E-n30-k3	P-n45-k5	X-n139-k10	X-n251-k28	X-n384-k52	X-n655-k131
A-n37-k6	B-n39-k5	E-n33-k4	P-n50-k10	X-n143-k7	X-n256-k16	X-n393-k38	X-n670-k130
A-n38-k5	B-n41-k6	E-n51-k5	P-n50-k7	X-n148-k46	X-n261-k13	X-n401-k29	X-n685-k75
A-n39-k5	B-n43-k6	E-n76-k10	P-n50-k8	X-n153-k22	X-n266-k58	X-n411-k19	X-n701-k44
A-n39-k6	B-n44-k7	E-n76-k14	P-n51-k10	X-n157-k13	X-n270-k35	X-n420-k130	X-n716-k35
A-n44-k6	B-n45-k5	E-n76-k7	P-n55-k10	X-n162-k11	X-n275-k28	X-n429-k61	X-n733-k159
A-n45-k6	B-n45-k6	E-n76-k8	P-n55-k15	X-n167-k10	X-n280-k17	X-n439-k37	X-n749-k98
A-n45-k7	B-n50-k7	F-n135-k7	P-n55-k7	X-n172-k51	X-n284-k15	X-n449-k29	X-n766-k71
A-n46-k7	B-n50-k8	F-n45-k4	P-n55-k8	X-n176-k26	X-n289-k60	X-n459-k26	X-n783-k48
A-n48-k7	B-n51-k7	F-n72-k4	P-n60-k10	X-n181-k23	X-n294-k50	X-n469-k138	X-n801-k40
A-n53-k7	B-n52-k7	M-n101-k10	P-n60-k15	X-n186-k15	X-n298-k31	X-n480-k70	X-n819-k171
A-n54-k7	B-n56-k7	M-n121-k7	P-n65-k10	X-n190-k8	X-n303-k21	X-n491-k59	X-n837-k142
A-n55-k9	B-n57-k7	M-n151-k12	P-n70-k10	X-n195-k51	X-n308-k13	X-n502-k39	X-n856-k95
A-n60-k9	B-n57-k9	M-n200-k16	P-n76-k4	X-n200-k36	X-n313-k71	X-n513-k21	X-n876-k59
A-n61-k9	B-n63-k10	M-n200-k17	P-n76-k5	X-n204-k19	X-n317-k53	X-n524-k153	X-n895-k37
A-n62-k8	B-n64-k9	P-n101-k4	X-n1001-k43	X-n209-k16	X-n322-k28	X-n536-k96	X-n916-k207
A-n63-k10	B-n66-k9	P-n16-k8	X-n101-k25	X-n214-k11	X-n327-k20	X-n548-k50	X-n936-k151
A-n63-k9	B-n67-k10	P-n19-k2	X-n106-k14	X-n219-k73	X-n331-k15	X-n561-k42	X-n957-k87
A-n64-k9	B-n68-k9	P-n20-k2	X-n110-k13	X-n223-k34	X-n336-k84	X-n573-k30	X-n979-k58
A-n65-k9							

Fonte: Autor

O ambiente realiza a leitura dos dados do arquivo de instância de forma genérica, assim a incorporação de novas instâncias procede com a inclusão do arquivo na pasta de instâncias. Estes arquivos possuem os dados necessários para o problema, entre eles estão: nome, tipo, comentários, dimensões, capacidade, tipo de comprimento entre as coordenadas, formato do comprimento entre as coordenadas, tipo das coordenadas dos pontos, seção de coordenadas dos pontos e seção de demanda dos pontos. Sendo que os dados de capacidade e seção de demanda são dados exclusivos para o CVRP. Um exemplo de arquivo de Instância é exibido no Anexo 1.

Para a implementação da classe de controle de instâncias é necessária a utilização das bibliotecas *Math* e *Path*, sendo esta última um componente do pacote *Os* (Sistema Operacional). A biblioteca *Math* é utilizada para a realização de cálculos matemáticos de distâncias entre as coordenadas dos pontos. A biblioteca *Path* é utilizada para leitura dos arquivos de instâncias, permitindo o acesso às pastas e aos arquivos do sistema, fazendo a leitura linha a linha.

3.4. Proposta de implementação dos métodos para resolução de funções

São propostos dois métodos para a resolução de problemas de funções, o Algoritmo Genético e o Recozimento Simulado. Eles necessitam, além dos parâmetros do próprio algoritmo, da função a ser resolvida. Todos os algoritmos desenvolvidos foram inspirados na melhor forma de resolver o problema, tendo como objetivos principais a eficiência na busca de valores ótimos e o tempo necessário para execução.

O problema de funções foi modelado com uma estrutura de valor inteiro representado por um vetor onde cada posição corresponde a um valor binário, que, quando usado em espaços de busca, gera soluções de alta qualidade (BLUM; ROLI, 2003). Os trabalhos (PETROWSKI, 1996) e (OSABA *et al.*, 2018) utilizam a mesma técnica. Desta forma os vetores binários são gerados aleatoriamente e tem a estrutura exibida na Figura 8, este é um exemplo do vetor com 16 *bits*.

1	1	1	0	0	0	1	1	1	1	1	0	0	1	0	0

Figura 8 - Representação do vetor binário
Fonte: Autor

As soluções representadas na base binária estão no intervalo binário $I_{x,g}$, definido como:

$$I_{x,g} = [a_{x,g}; b_{x,g}] = [{}^{(000\ldots000)}_{n_{gv}\ vezes}; {}^{(111\ldots111)}_{n_{gv}\ vezes}] \subset B^1 \tag{33}$$

Na Equação (33) B^1 representa o conjunto dos números binários. Os problemas abordados podem ser multidimensionais, para estes casos, o vetor binário será representado com o número de dimensões igual ao do problema. O intervalo $I_{x,g}$ necessita ser transformado para a base decimal, sendo transformado para $I_{x,N}$, contido no conjunto dos números inteiros, resultando:

$$I_{x,N} = [a_{x,N}; b_{x,N}] = [0.2^{n_{gv}} - 1] \subset N^1 \quad (34)$$

Tomando como base o exemplo da Figura 8, seu valor quando convertido para inteiro decimal, corresponde ao número 58340.

Todas as soluções geradas tem seu valor ajustado dentro do domínio de busca do problema, desobrigando técnicas de análise e verificações dos dados gerados.

Os métodos possuem o intuito de realizar a hibridização de alto nível, seguindo a lógica de trabalho em equipe colaborativo. O procedimento de hibridização ocorre executando um primeiro método sem nenhum conhecimento prévio de soluções, após sua finalização, a melhor solução obtida tem seu vetor binário copiado e passado como parâmetro para o próximo método, que continuará o processo de otimização, partindo de uma solução já elaborada.

Esta classe tem como resultados: a lista de valores obtidos a cada geração/iteração, o vetor binário e o local da melhor solução encontrada, o tempo de resolução do problema e o melhor valor encontrado.

3.4.1. Algoritmo Genético para Funções

O Algoritmo Genético teve sua implementação baseada na pesquisa de (HOLLAND, 1975), levando em conta considerações de (KOZA, 1994). Em sua estrutura são utilizados os operadores genéticos de cruzamento e mutação. O algoritmo utiliza do paradigma de orientação a objeto, onde cada cromossomo da população é considerado como um objeto e possui:

- Função de *benchmark* a ser otimizada.
- Vetor binário – contém a cadeia de *bits* para geração do número real da variável do problema. Possui o número de dimensões igual a da função escolhida.
- Quantidade de *bits* – representa o comprimento do vetor binário.
- Vetor de valores reais – contém valores para cada variável do problema, de acordo com as dimensões do problema.
- Valor da função objetivo.

A operação de cruzamento é realizada selecionando dois indivíduos de forma probabilística na população, baseando no valor da função objetivo, estes chamados de pais. Uma posição é sorteada dentro do vetor binário, onde fica demarcado o ponto de

troca. Então dois filhos são criados, um com o início do vetor binário do primeiro pai (até o ponto de troca) e com o final do segundo pai, procedimento inverso para a criação do segundo filho. Adotando um exemplo onde um par de pais sorteados, cada um com cromossomo de 16 *bits*, representados na Figura 9, sendo o primeiro destacado em negrito para reconhecimento.

Pai 1	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**	**0**
Pai 2	0	1	0	0	0	1	1	0	1	0	0	0	0	1	1	1

Figura 9 - Exemplo de um par de pais do GA
Fonte: Autor

Supondo que a posição de troca sorteada seja 10, os filhos gerados pelo processo de cruzamento são representados na Figura 10.

Filho 1	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**1**	0	0	0	1	1	1
Filho 2	0	1	0	0	0	1	1	0	1	0	**1**	**0**	**1**	**1**	**0**	**0**

Figura 10 - Exemplo de filhos gerados pelo GA
Fonte: Autor

A operação de mutação é realizada selecionando um indivíduo da população, probabilisticamente, uma posição do vetor binário é sorteada, onde ocorre uma mudança de valor. Considerando o valor atual do bit selecionado, é inserido o valor oposto a este. Adotando um exemplo um indivíduo é sorteado para a operação de mutação, representado na Figura 11.

Indivíduo	1	1	1	0	0	0	1	1	1	1	1	0	1	1	0	0

Figura 11 - Representação de um individuo do GA para mutação
Fonte: Autor

Supondo que a posição para mutação sorteada seja a 8, o novo indivíduo gerado terá a estrutura cromossômica representada na Figura 12.

Indivíduo	1	1	1	0	0	0	1	0	1	1	1	0	1	1	0	0

Figura 12 - Representação de um individuo do GA após a mutação
Fonte: Autor

O algoritmo evolui e repete as operações genéticas de cruzamento e mutação até o final das gerações, como exibido no fluxograma da Figura 13. Ao final de cada geração a melhor solução encontrada é armazenada. Ao final do algoritmo, temos como resultado uma lista com todos os melhores resultados obtidos a cada geração, o que torna possível analisar o desempenho algoritmo com o decorrer da execução, além de obter a melhor solução encontrada.

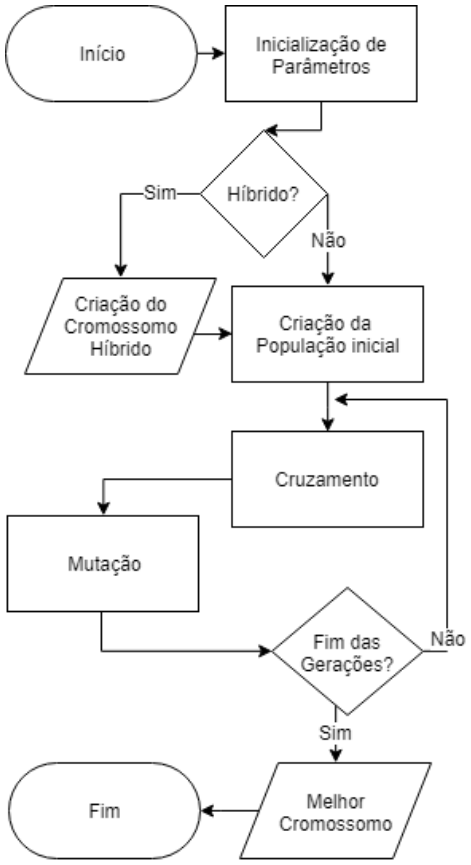

Figura 13 - Fluxograma do GA para funções
Fonte: Autor

O método possui a possibilidade de recebimento do parâmetro de indivíduo híbrido, contendo a informação do vetor binário de um método previamente executado. A partir deste é criado um cromossomo que será incorporado à população inicial, permitindo que o algoritmo inicie com um suposto bom resultado. Este procedimento caracteriza o algoritmo como híbrido, já que desta forma, é possível o recebimento de uma solução fruto do resultado de outros procedimentos.

3.4.2. Recozimento Simulado para Funções

A implementação do SA foi baseada nos estudos de (KIRKPATRICK; GELATT; VECCHI, 1983), tendo como evolução o estudo de (CHEN; CHIEN, 2011). O algoritmo baseia-se na operação com uma solução única, e com o decorrer da execução testa mudanças nessa solução para possível substituição da mesma.

As soluções são representadas como um vetor binário, assim o algoritmo gera a solução inicial probabilisticamente, assumindo esta como solução atual. Para realização do procedimento de busca de novas soluções o algoritmo utiliza a heurística de *Lin Kernighan Heuristic* (LKH) (LIN; KERNIGHAN, 1973).

O procedimento LKH é uma técnica de troca de informações de forma inversa, onde são sorteadas duas posições do vetor binário e o subconjunto existente entre as posições é sobreposto em ordem inversa. Para exemplificar a heurística, considere como solução atual a exposta na Figura 14.

1	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0

Figura 14 - Representação de uma solução do SA
Fonte: Autor

Supondo que as posições sorteadas tenham sido as 10 e 14, a Figura 15 destaca o subconjunto considerado para realização da operação, em seguida a substituição é feita pelo subconjunto inverso, exibida na Figura 16.

1	1	1	0	0	0	1	1	1	0	1	0	0	1	0	0

Figura 15 - Representação de uma solução do SA com destaque no trecho selecionado
Fonte: Autor

1	1	1	0	0	0	1	1	1	1	0	0	1	0	0	0

Figura 16 - Representação da solução após a troca inversa
Fonte: Autor

A operação de troca é realizada para encontrar um candidato à solução atual para o algoritmo, caso a solução candidata seja melhor que a solução atual, o algoritmo aceita a solução candidata como a nova solução atual. No entanto, caso a solução não seja melhor que a atual, o algoritmo realiza o cálculo de probabilidade de aceite de uma solução pior. Esse cálculo foi exibido na Equação (11) e seu resultado é totalmente dependente da temperatura do algoritmo. Quanto maior a temperatura, maior a probabilidade de aceite. A Figura 17 mostra o fluxograma do algoritmo.

A cada iteração do algoritmo, existe a probabilidade de adoção de uma nova solução e conforme o algoritmo evolui, ocorre um decremento da temperatura, diminuindo cada vez mais essa probabilidade. O algoritmo encerra quando a temperatura atinge o valor determinado como temperatura final, ou quando executa um número limite de iterações.

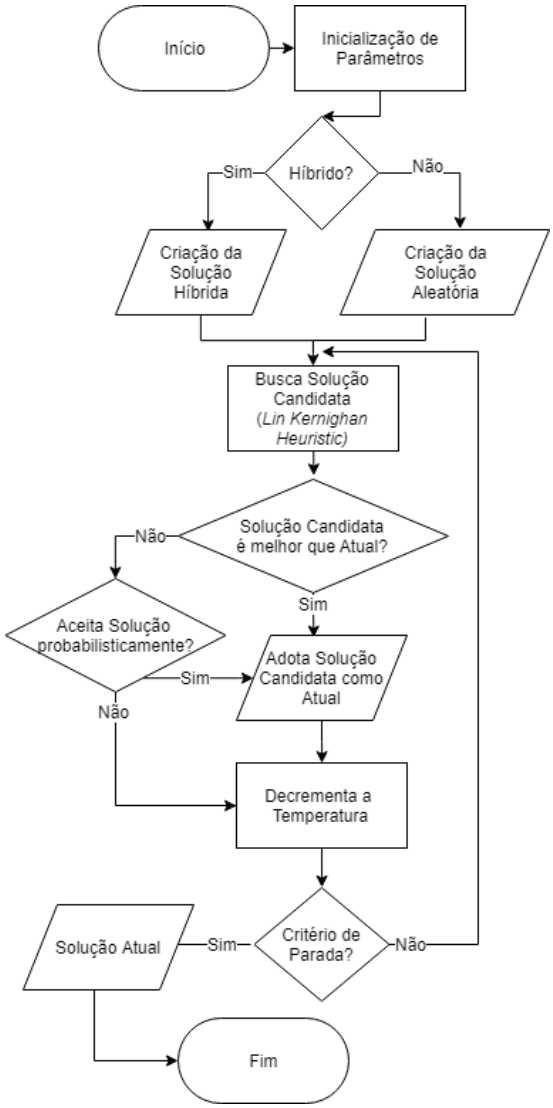

Figura 17 - Fluxograma do SA
Fonte: Autor

3.5. Implementação dos métodos para resolução de Instâncias

Para a resolução de problemas de Instâncias, são utilizados problemas do TSP e CVRP e foram implementados os métodos Algoritmo Genético, Algoritmo Colônia de Formigas e o Recozimento Simulado.

Para os problemas de instâncias a modelagem foi feita em forma de vetores com valores inteiros, onde cada posição do vetor representa um ponto do problema a ser visitado. O vetor como um todo representa a ordem em que as visitas ocorrerão (CABALLERO-MORALES; MARTINEZ-FLORES; SANCHEZ-PARTIDA, 2018). Desta forma todas as soluções existentes terão os mesmos itens para determinado problema, a única diferença é a ordem em que as soluções são organizadas. A Figura 18 representa exemplos de vetores com a ordem de um problema, considere que este problema tenha um total de cinco pontos (vértices ou clientes).

Solução 1	1	2	3	4	5
Solução 2	5	4	3	2	1
Solução 3	2	5	3	1	4
Solução 4	3	2	5	4	1

Figura 18 - Representação de soluções modeladas para os problemas de roteamento
Fonte: Autor

Uma classe foi implementada para fazer o mapeamento de arquivos de instâncias, esta classe trabalha de forma genérica para ambos os tipos, TSP ou CVRP. Esta classe faz a leitura dos dados do problema, mapeando todos os pontos existentes, armazena seus dados geográficos e calcula uma matriz com todas as distâncias entre todos os pontos. A classe de instância ainda tem a opção de calcular a distância de uma determinada rota e as limitações de capacidade sob as demandas dessa rota, este último cálculo sendo uma restrição exclusiva do problema CVRP.

Utilizando esta classe os algoritmos necessitam apenas da informação de tamanho da instância, para saber qual o tamanho do vetor de rota será utilizado. Os demais valores são manipulados apenas pela classe de instância. Assim, o papel dos algoritmos é o de gerar vetores de soluções como os exemplos da Figura 18 e enviá-los para que a classe de instâncias realize os cálculos necessários.

O ambiente utiliza esta forma de organização para que os cálculos fiquem centralizados, seguindo os conceitos de orientação à objeto e reutilização de código.

Os métodos, após sua execução geram os resultados: melhor caminho encontrado, custo do caminho, lista dos melhores resultados a cada geração/iteração, tempo de execução e, exclusivamente para os CVRP, a lista de rotas a serem traçadas.

3.5.1. Algoritmo Colônia de Formigas para Instâncias

O ACO foi desenvolvido inspirado na estrutura tradicional do método apresentado em (DORIGO; STÜTZLE, 2004). A mesma formulação do algoritmo é aplicada para ambos os tipos de problema de instâncias, já que o cálculo de distâncias e rotas é realizado pela classe de controle da instância.

O algoritmo necessita dos parâmetros: número de formigas, número de gerações, α, β, ρ e *Q*, onde α e β são fatores utilizados para determinar a influência relativa do feromônio depositado, para o cálculo de probabilidade da escolha, partindo de um ponto i para um ponto j, mostrado na Equação (35).

$$p_{ij}^{k} = \frac{[t_{ij}]^{\alpha}[n_{ij}]^{\beta}}{\sum_{l \in N_i^k}[t_{il}]^{\alpha}[n_{il}]^{\beta}}, se\ j \in n_i^k \tag{35}$$

Onde $n_{ij} = 1/d_{ij}$ é um valor heurístico que está disponível a priori, n_i^k é a vizinhança viável para a formiga k estando na cidade i.

O parâmetro ρ representa o fator de evaporação do feromônio e Q é a constante utilizada para determinar a quantidade de feromônio depositado no caminho. O fluxograma do algoritmo é mostrado na Figura 19.

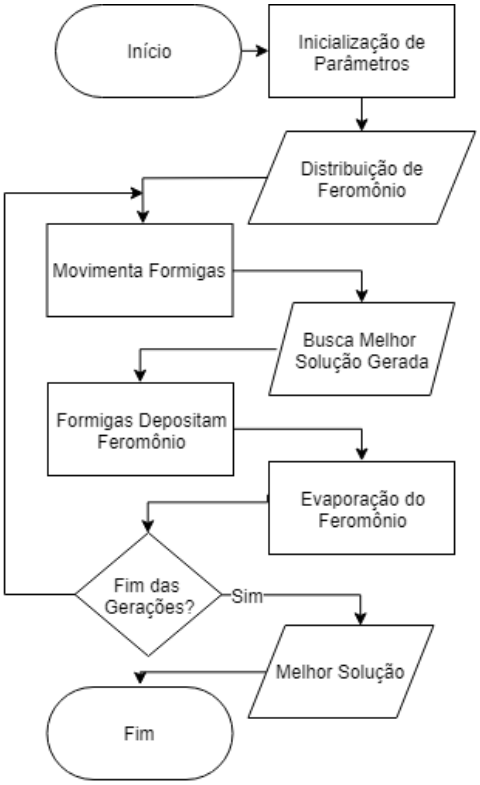

Figura 19 - Fluxograma do ACO
Fonte: Autor

Inicialmente é feita a distribuição de feromônios, iniciando em seguida o processo da evolução do algoritmo com a passagem das gerações. O método trabalha com a manipulação do objeto *Formiga*, que se movimenta pelas coordenadas do problema buscando os maiores acúmulos de feromônio. Enquanto esse movimento ocorre, as rotas traçadas são armazenadas como soluções para o problema e novas quantidades de feromônios são liberadas pelo objeto. Com o passar das gerações os feromônios depositados nos pontos passam por um processo de evaporação para controle da totalidade de feromônio existente.

Com a evolução do algoritmo, o feromônio vai estar concentrado com maior peso no melhor caminho encontrado, afunilando cada vez mais para a convergência do resultado.

A escolha probabilística da rota não garante que o valor ótimo seja encontrado, existe a possibilidade de o algoritmo encontrar no início da execução um ótimo local, e assim, alguns trajetos dessa rota podem ser reforçados pelos

feromônios levando o algoritmo a estagnação, desta forma nunca encontrando o valor ótimo para o problema (GAERTNER; CLARK, 2005).

Pelo fato de o algoritmo basear-se no movimento das formigas para geração de novas rotas, a implementação do método com a possibilidade de receber informações para hibridizar-se com outros métodos anteriormente executados não foi aplicada.

3.5.2. Recozimento Simulado para Instâncias

Assim como o ACO, a formulação do SA foi realizada da mesma forma para ambos os tipos de problemas (KIRKPATRICK; GELATT; VECCHI, 1983). A estrutura do algoritmo foi modelada da mesma forma que a versão aplicada para problemas de Função, diferenciando a forma em que as soluções são representadas, já que na formulação para funções utiliza-se de um vetor binário para representação da solução, já para instâncias utiliza-se um vetor inteiro com as posições se refere a um ponto do problema.

O algoritmo possui o objeto Solução com as seguintes informações: a ordem em que os pontos são visitados, distância percorrida, e, exclusivamente para o CVRP, divisão das rotas.

A modelagem do problema de instância é feita tendo como base um vetor que representa a ordem em que os pontos são visitados. O SA tradicionalmente utiliza uma solução inicial aleatória, sem nenhum tipo de conhecimento específico e a cada iteração do algoritmo, um vizinho é gerado e avaliado. A geração desse vizinho é feita utilizando a heurística de troca inversa de LKH. O algoritmo não utiliza dados de iterações anteriores, assim, caso uma solução seja aceita, a solução atual é substituída pela nova, sendo descartada.

3.5.3. Algoritmo Genético para Instâncias

O estudo de (HOLLAND, 1975) foi analisado para embasamento da implementação do Algoritmo Genético para instâncias, porém, a modelagem dos problemas de roteirização com tal estrutura, traria complicações. O operador de mutação foi mantido como no algoritmo de (HOLLAND, 1975).

A principal dificuldade encontrada se deu no fato de na operação de cruzamento, os itens selecionados trocarem informações entre si. Como o problema trata de um vetor onde cada posição representa um ponto a ser visitado tendo como obrigatoriedade em que os pontos sejam visitados exatamente uma vez, a operação

de cruzamento tradicional, gera a possibilidade da criação de filhos visitando mais de uma vez um mesmo ponto, sendo considerado infactível, violando a restrição dos problemas.

Essa característica exige que o algoritmo passe por um processo de validação de cada cromossomo criado, processo este que demanda de processamento computacional, atrasando o andamento do algoritmo. Foram pesquisadas diferentes abordagens para melhor adequação ao problema, adotando em seguida o estudo de (CHATTERJEE; CARRERA; LYNCH, 1996) como fonte de referência para o desenvolvimento.

O estudo foi escolhido por trabalhar com o TSP, evitando o uso da reprodução sexuada como forma de cruzamento, desobrigando qualquer necessidade de validação do cromossomo. (CHATTERJEE; CARRERA; LYNCH, 1996) aplica o conceito de reprodução assexuada, onde a reprodução é realizada mantendo, de forma probabilística, partes de informações de um único pai selecionado, movimentando as demais informações aleatoriamente. Fazendo assim com que parte da informação do pai seja mantida, dados sejam modificados probabilisticamente e sem a necessidade da validação do indivíduo.

Para evitar a validação desses cromossomos infactíveis, a operação de cruzamento tradicional foi substituída pelo cruzamento assexuado, o MUT3 (CHATTERJEE; CARRERA; LYNCH, 1996), ele procede com a seleção de pontos no vetor de coordenadas, onde são realizadas divisões lógicas, e após isso, reorganiza probabilisticamente o vetor.

Considerada uma técnica simples para aplicação pelas seguintes razões: permite trabalhar com a troca probabilística dos dados, nunca gera indivíduos infactíveis e possibilita que o cálculo da distância para o novo indivíduo seja realizado com poucos esforços. Esta última característica sendo bastante atraente, já o método MUT3 geralmente utiliza 2, 3 ou 4 divisões lógicas, assim o operador exige que sejam feitos 2, 3 ou 4 pares de operação (adição ou subtração) para atualizar a distância total percorrida.

Neste ponto, o algoritmo ficou com dois operadores, cruzamento assexuado e mutação, ambos baseados na estrutura de um único indivíduo para geração de novos. Uma análise da literatura foi feita com a tentativa de reconhecer trabalhos que dessem embasamento para tal situação. Foram encontrados vários trabalhos que abordam a dificuldade de aplicar a operação de cruzamento para problemas de roteirização (KURODA *et al.*, 2010).

Nos estudos iniciais do GA a operação de cruzamento era considerada de extrema importância (HOLLAND, 1975), deixando a operação de mutação como

preocupação secundária, utilizada para fornecer diversidade para a população, evitando que a mesma convergisse para a mesma região. No entanto, algoritmos puramente baseados em mutações (ÜÇOLUK, 2002) (HASSANAT *et al.*, 2016) e cruzamentos assexuados (WANG *et al.*, 2015) já tiveram seus resultados comprovados.

Com o estudo foram encontradas diversas referências que utilizam o método LKH de (LIN; KERNIGHAN, 1973) como forma de mutação inversa - método já aplicado ao algoritmo Recozimento Simulado. O método é considerado como a heurística mais eficiente para o TSP (KURODA *et al.*, 2010) (COSTA *et al.*, 2018), o que motivou a implementação do mesmo.

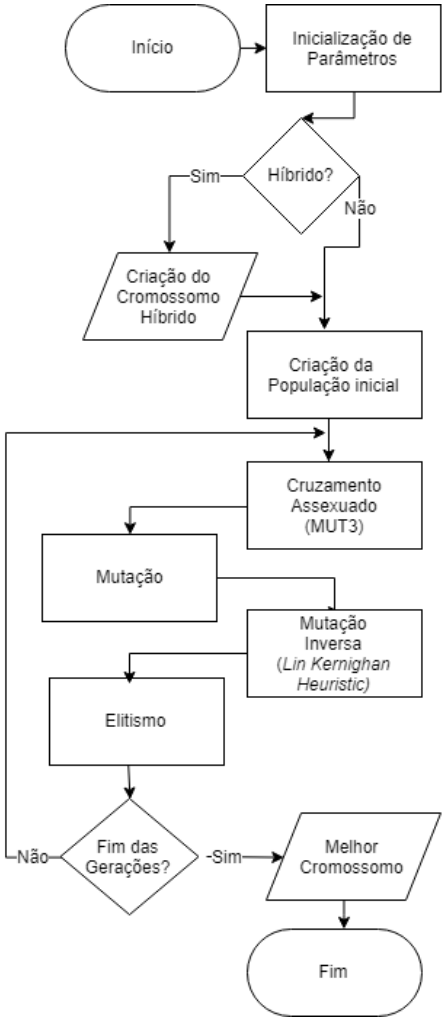

Figura 20 - Fluxograma do GA para instâncias
Fonte: Autor

A técnica de LKH aplicada ao GA ocorre com a seleção probabilística de dois pontos no vetor de coordenadas, determinando um intervalo de seleção, em seguida sobrepondo o intervalo no mesmo local, porém em ordem inversa. A aplicação da técnica demostrou grande avanço para o algoritmo. O fluxograma do algoritmo GA é

mostrado na Figura 20. O conhecimento da técnica LKH foi utilizado para aplicação no algoritmo Recozimento Simulado, no processo de busca de novas soluções, tendo eficientes resultados.

Com o conhecimento adquirido no desenvolvimento do GA para resolução de Funções, a técnica de Elitismo (DEB, 2011), foi adaptada igualmente para o problema de Instâncias, na tentativa de trazer diversidade para a população. Testes de eficiência, como a calibração de parâmetros *off-line* comprovaram resultados superiores com a aplicação da técnica de Elitismo.

3.6. Funcionamento da Plataforma

A plataforma fornece ao usuário um ambiente de teste dos algoritmos desenvolvidos, possibilitando que o mesmo selecione qual o tipo de problema quer resolver, Funções ou Instâncias, qual método ele quer utilizar para esta resolução e a informação dos parâmetros para a execução de cada método.

Existem dois modos de execução do método, o modo único, onde o problema é resolvido pelo método uma única vez, mostrando ao final da execução uma página com as informações sobre o problema, melhores valores existentes e os resultados da atual execução. O outro modo é o de simulação, onde o ambiente repete a resolução do algoritmo um determinado número de vezes, mostrando ao usuário o avanço das execuções, estimando o tempo restante para término das simulações e, ao final, exibe uma lista com todos os resultados encontrados e tempos de execução; faz uma análise estatística com os dados: média e desvio padrão dos resultados; e mostra os gráficos de dispersão dos resultados, *boxplot* dos tempos e soluções.

A abordagem feita pelo método pode ser simplificada, tendo apenas um método trabalhando na solução do problema, ou ainda híbrida, possibilitando que dois métodos trabalhem sequencialmente na busca do melhor resultado. Essa forma de hibridização é classificada como de Alto Nível, já que um método gera resultado que será usado como dado de entrada para um segundo método.

No caso dos problemas de Funções, a informação passada de um método para o outro é o vetor binário da melhor solução encontrada. Já para problemas de Instâncias, a informação passada de um algoritmo para outro é o vetor com a ordem em que os pontos são visitados. O GA cria um cromossomo com essa informação e insere na população inicial, não afetando no restante da população. O SA cria um objeto solução com esse dado e assume ele como solução inicial do algoritmo, substituindo a criação aleatória para a solução inicial.

Os resultados são distintos para os modos de Execução Única e Simulação, tendo como informação comum apenas os dados do problema. O modo de Execução Única para Funções contém o melhor valor encontrado, o tempo gasto, o ponto em que foi encontrado e um gráfico de linha com a evolução do algoritmo. No caso de hibridização, ainda contém um gráfico com a evolução serial dos métodos junto de uma análise sobre a necessidade de tal hibridização. Para Instâncias além dos dados exibidos para funções, ainda mostra um gráfico com a ligação dos pontos do melhor caminho encontrado.

O modo de Simulação para ambos os tipos de problemas, contém um relatório sobre a simulação executada. Esses dados são computados sobre os tempos de execução e os melhores valores encontrados. São eles: resumo dos cinco números, média, desvio padrão, gráficos *boxplot* dos tempos e dos custos, gráfico de dispersão dos dados, lista de dados de cada simulação, dados da melhor e da segunda melhor solução encontradas.

Assim forma-se a implementação do ambiente, disponível para acesso *online* (http://hybroo.hopto.org:5000/), possibilitando a realização de testes, além de acesso a todo o software, como código aberto por meio do repositório do GitHub (https://github.com/edgarancioto/Hybroo).

4. MÉTODOS PROPOSTOS

Em meio aos desenvolvimentos foi constatado que a biblioteca *SymPy*, utilizada para trabalhar com formulação dos problemas em forma de equações, possui limitações, como o modo de trabalhar com dados multidimensionais, representados como matrizes simbólicas, que exige um tratamento dos dados realizado em tempo de execução, processo que dispende de tempo computacional. Identificado ainda que no uso de operadores matemáticos, como “seno”, “cosseno” e “exponencial” a velocidade de processamento é inferior aos cálculos realizados pela biblioteca Math, que possui os mesmos métodos, o mesmo ocorre para o carregamento do valor da constante *PI*. Características que acarretam em atraso na resolução dos cálculos em meio à execução dos métodos.

Outra dificuldade encontrada ocorreu no uso de funções com a operação “produtório”, onde o modo de trabalho não foi adequado à operação. Assim, para tratar funções que utilizam dessa operação, seria necessário fazer adaptações no modo de leitura das funções, ou ainda, buscar outras opções de bibliotecas que tratem variáveis simbólicas.

4.1. Adaptação do Algoritmo Genético para Funções

Após o início dos testes, verificou-se que, com o decorrer das gerações, as soluções estavam tornando-se similares. Testes voltados para a análise dos indivíduos foram realizados constatando que os indivíduos estavam convergindo para o mesmo espaço, fazendo com que a população não tivesse diversidade.

(RUDOLPH, 1994) provou a convergência do GA tendo como operadores genéticos essenciais o elitismo e a mutação. O operador de elitismo combina a população antiga com a população recém-criada e escolhe manter melhores soluções da população combinada, essa técnica garante um desempenho crescente e não degradante (DEB, 2011).

Para superar a desvantagem a técnica de Elitismo foi aplicada, onde as soluções são ordenadas, escolhidas e, por meio de um número de vagas, seguem para a próxima geração. Este processo permite a diversidade da população, já que indivíduos são criados de modo probabilístico, sem perder os bons resultados já encontrados. Com a aplicação da técnica de Elitismo foi constatada uma melhoria na evolução da convergência do algoritmo no decorrer das gerações. O fluxograma do algoritmo desenvolvido pode ser visto na Figura 21.

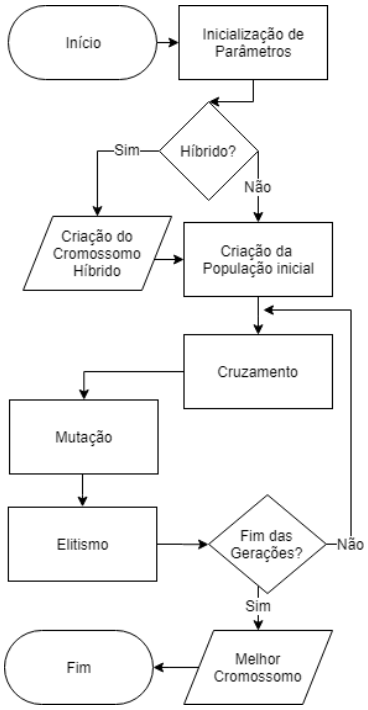

Figura 21 - Fluxograma do GA para funções após modificações
Fonte: Autor

4.2. Adaptação do Recozimento Simulado para Funções

O método foi ajustado para trabalhar com a estrutura já existente do GA. Pelo uso de conceitos de orientação a objetos aproveitou-se o objeto Cromossomo, agora intitulado Solução, aplicando a modificação: para o objeto Cromossomo a maioria dos objetos gera seu vetor binário de forma probabilística; para o objeto Solução, apenas a solução inicial será gerada desta forma. O método de geração do vetor binário aleatório também foi aproveitado. Desta forma a implementação deste método foi feita de forma rápida e eficiente pela estrutura já existente.

A execução do método exige os parâmetros a seguir: temperatura, fator de redução (α), ponto de parada para a temperatura, ponto de parada para o número de iterações; além do objeto com os dados da função a ser otimizada.

Assim como o GA, este algoritmo tem a possibilidade de receber o parâmetro de indivíduo híbrido, contendo a informação do vetor binário. Porém diferente do GA, o método não trabalha com o conceito populacional, assim a informação recebida é utilizada para a criação da solução inicial. Desta forma o algoritmo é considerado híbrido e tem a possibilidade de partir de uma solução boa.

4.3. Algoritmos Meta-heurísticos para Instâncias

Os algoritmos foram desenvolvidos para resolução de problemas do TSP e CVRP, os algoritmos ACO e SA possuem o mesmo funcionamento para ambos os problemas, modificando apenas os trechos referentes às especificidades dos problemas. O GA foi o método com maior tempo de dedicação para estudo e desenvolvimento de técnicas, assim seu desenvolvimento tratou de formas distintas cada um dos problemas.

Aplicou-se nas implementações, assim como nos métodos para funções, apenas a criação de soluções válidas, não utilizando de técnicas de validação de dados.

Os algoritmos desenvolvidos utilizam em seu código a biblioteca *Random*, utilizada como gerador de números aleatórios, escolha aleatória de itens em vetores e baralhamento de itens. O método SA utiliza também da biblioteca Math para cálculos probabilísticos na tomada de decisão.

Os métodos GA e SA foram desenvolvidos com a funcionalidade de hibridização, permitindo que a melhor solução de um algoritmo previamente executado, seja utilizada como ponto de partida. A estrutura central de dados manipulada pelos algoritmos assim como a informação do parâmetro híbrido é o vetor de coordenadas dos pontos.

4.3.1. Adaptação do Algoritmo Genético para Instâncias

O Algoritmo Genético foi desenvolvido para diferentes versões dos problemas, pelo fato de ele ter sido o primeiro no estudo e desenvolvimento dos algoritmos, as técnicas aplicadas e alterações realizadas são superiores aos demais métodos.

O Algoritmo Genético desenvolvido para o TSP foi modelado com as três possibilidades de operações, o cruzamento assexuado, a mutação e a LKH. Após testes de desempenho dos métodos, foi comprovado que a eficiência do LKH é superior aos outros dois métodos. Com esta comprovação, optou-se implementar na versão do Algoritmo Genético para o CVRP apenas o operador genético de LKH. Desta forma o GA aborda problemas de Instância de maneiras diferentes em relação aos operadores disponíveis, seu fluxograma é exibido na Figura 22.

Após estudo de variações do Algoritmo Genético aplicado a problemas de roteirização, foram encontradas técnicas de busca local para melhorar os resultados

do algoritmo (ZHANG *et al.*, 2015) (LO *et al.*, 2018), sem grandes impactos nos tempos de execução.

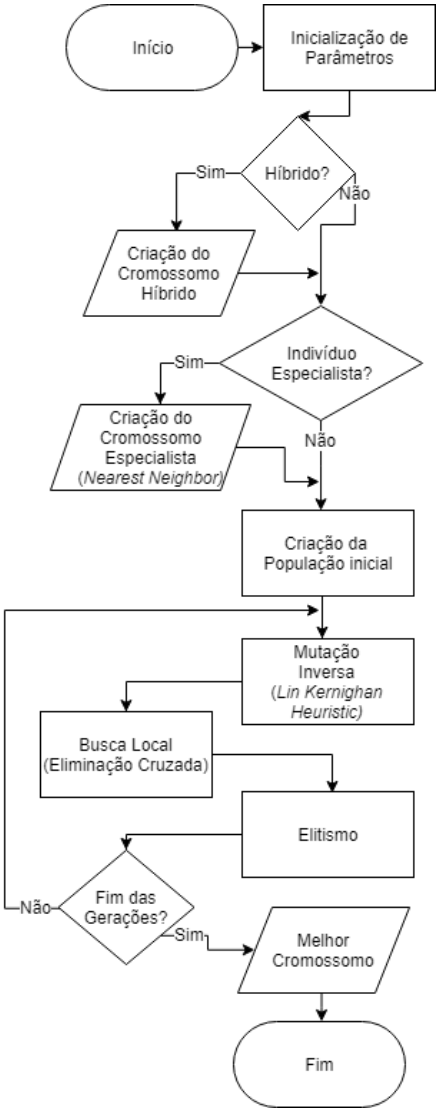

Figura 22 - Fluxograma do GA para instâncias do CVRP
Fonte: Autor

Uma técnica foi estudada e implementada para o GA, a Eliminação Cruzada (LO *et al.*, 2018). Com base conceitual na técnica 2-OPT (COSTA *et al.*, 2018), é utilizada como busca local nas soluções de problemas de roteirização. A técnica procura ativamente pela formação de cruzamentos no caminho traçado pelas soluções, tentando removê-las por meio de reordenação sequencial. Desta forma, a técnica realiza testes de troca da sequência em que duas coordenadas são visitadas, e caso minimize a distância percorrida, a troca é aceita para a solução.

Em meio ao desenvolvimento do algoritmo Recozimento Simulado para Instâncias, foi realizado o estudo de técnicas para geração de soluções iniciais baseadas em algum conhecimento específico. Foi aplicada, com êxito para o SA, a

técnica do *Nearest Neighbor* como gerador de uma solução inicial (KIRKPATRICK; GELATT; VECCHI, 1983).

O mesmo conceito foi aplicado ao GA, adaptando a técnica à modelagem do algoritmo. Assim criou-se o procedimento de geração de um indivíduo especialista, criado junto da população inicial, com a técnica do *Nearest Neighbor*. Assim o algoritmo possibilita que em meio a uma população totalmente probabilística, exista um indivíduo com conhecimento especialista.

A implementação deste procedimento faz com que bons resultados para o algoritmo sejam atingidos com um número de gerações e um tempo de execução menores.

4.3.2. Adaptação do Colônia de Formigas para Instâncias

O algoritmo necessita de uma quantidade de estruturas de dados (matrizes) superior aos demais métodos, característica que aumenta a complexidade na implementação do método. Os procedimentos realizados pelo algoritmo são dependentes entre si, que significa que os testes de validação do método são feitos, na maioria, de forma macro, analisando o algoritmo como um todo.

O algoritmo demonstrou por meio de testes a capacidade de encontrar boas soluções para os problemas de roteirização. No entanto, ele possui problemas de estagnação e convergência prematura do resultado. Além disso, o algoritmo possui a velocidade de convergência muito lenta. Esses problemas são considerados como as principais desvantagens do algoritmo e ficam mais evidentes conforme o aumento do tamanho da Instância (ALJANABY; KU-MAHAMUD; NORWAWI, 2010) (HLAING *et al.*, 2011).

Com a aplicação dos testes, ficou comprovado como relatado no estudo de (GAERTNER; CLARK, 2005), que o algoritmo, em determinadas vezes, cria buscas tendenciosas. Há casos em que soluções razoáveis são encontradas no início do algoritmo (convergência prematura), e ao decorrer, com um período sem encontrar soluções melhores, tal solução vai sendo reforçada pelo acúmulo de feromônios cada vez mais.

Como o método cria suas soluções por meio do movimento das formigas, que por sua vez movimentam-se de acordo com os acúmulos de feromônios nos pontos, a aplicação da técnica de hibridização tornou-se inviável no formato aplicado para os demais métodos. Para viabilizar a hibridização para este método seria necessário um tratamento especial dos dados, como a distribuição inicial dos feromônios de forma a favorecer uma solução prévia recebida como parâmetro. A aplicação desta técnica não

foi bem sucedida, pois o método criou uma tendência para o caminho recebido, não conseguindo atingir variações de resultados.

4.3.3. Adaptação do Recozimento Simulado para Instâncias

Após a aplicação de testes ao algoritmo SA, verificou-se uma ineficiência na convergência dos resultados. Foi realizado um estudo para reconhecimento das variações do método aplicado a problemas de roteirização e análise de efetividade de possíveis alterações. Foram identificadas técnicas aplicadas na criação da solução inicial, de forma que o algoritmo não tenha um início baseado somente em probabilidade.

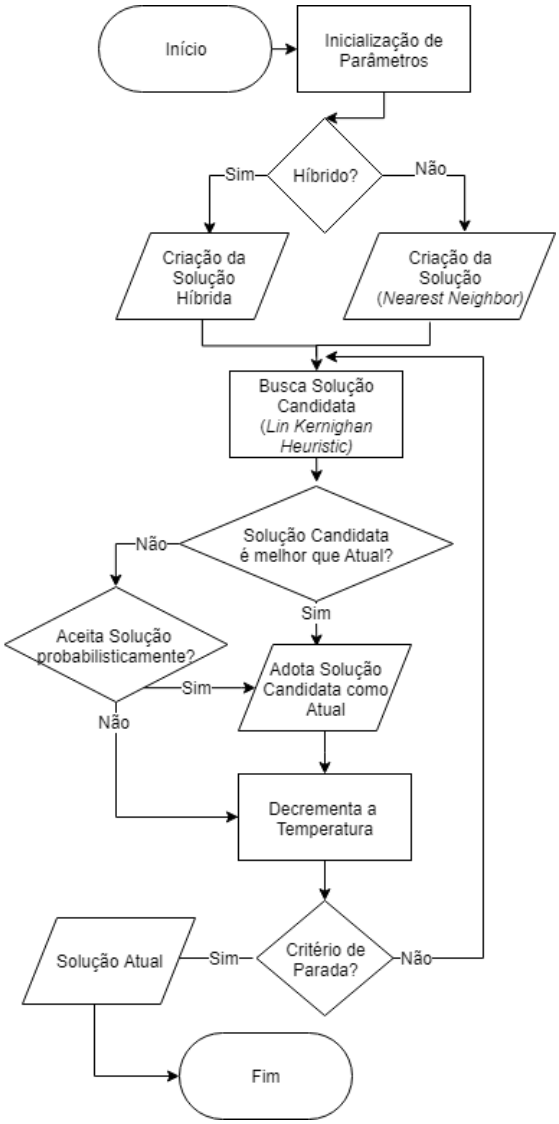

Figura 23 - Fluxograma do SA para instâncias
Fonte: Autor

O algoritmo do *Nearest Neighbor* foi encontrado em diversos trabalhos relacionados com o estudo de roteirização (FISHER; JAIKUMAR, 1981) (ADEWUMI;

ADELEKE, 2016) (OSABA *et al.*, 2017). Esse algoritmo baseia-se na escolha de um ponto aleatório e, enquanto não forem visitados todos os pontos, o ponto mais próximo do atual é visitado, elaborando uma solução inicial com um sequenciamento de distâncias mínimas. O fluxograma com o algoritmo desenvolvido é exibido na Figura 23. O algoritmo mostrou sua eficiência diversas vezes na literatura, por este motivo optou-se pelo seu desenvolvimento para a geração da solução inicial para o SA.

Assim o algoritmo saiu da geração da solução inicial probabilística e incorporou um algoritmo especialista para esta tarefa. Esta união fez com que o algoritmo partisse de uma solução muito superior a de antes, fazendo com que a evolução dos resultados fosse realizada de forma acelerada.

No decorrer dos desenvolvimentos, constatou-se a possibilidade da aplicação da técnica LKH, já aplicada anteriormente para o algoritmo na modelagem para Funções. Fazendo com que as operações de troca do algoritmo fossem realizadas pela heurística de troca inversa.

A alteração na operação de busca pela vizinhança da solução corrente (LKH), unido ao fato do algoritmo iniciar de uma solução especialista (*Nearest Neighbor*), fez com que o método tivesse um destaque na busca de boas soluções em tempos mínimos, surpreendentemente mais rápido que os demais métodos.

4.4. Desenvolvimento do Framework

O ambiente teve sua arquitetura estruturada em duas partes, o *front-end* e o *back-end*. Esta divisão foi feita para organizar o código separando as classes voltadas para a coleta de dados da execução das classes de execução em si.

No módulo *front-end* está todo o código aplicado para a comunicação com o usuário, coleta de dados, como: seleção de problemas e métodos, informações de parâmetros e modos de abordagem; e a apresentação dos resultados obtidos ao final das execuções. O módulo *front-end* é dedicado aos processos:

- Permitir ao usuário um acesso amigável ao ambiente de forma *web*;
- Recebimento e validação dos dados informados pelo usuário para escolha de problemas e métodos;
- Organização de todos os dados necessários recolhidos para acionamento do módulo *back-end*;
- Recebimento de dados resultantes das execuções pelo *back-end*;
- Exibição dos dados de resultados para o usuário.

Para desenvolvimento do ambiente *web* foram criadas além de páginas *web* (HTML), arquivos de formatação de estilo (CSS) e procedimentos de validação de

formulários (JavaScript). Todos voltados ao acesso e utilização do ambiente em uma forma amigável pelo usuário.

Esse conjunto de estruturas precisou ser integrado ao restante da codificação desenvolvida em Python, permitindo a comunicação do *front-end* com o *back-end*. A *Flask* é a biblioteca utilizada para fazer esta integração, criando possibilidade de, na codificação em Python, integrar as estruturas *web*, receber e enviar informações.

Para controle do módulo *front-end*, foi criada uma classe de controle, com o objetivo de padronizar o código nos processos de: a renderização das páginas *web*, a troca de informações com as páginas, o acionamento dos métodos do *back-end* e o recebimento dos resultados.

O módulo *back-end* possui todos os métodos meta-heurísticos desenvolvidos, além das classes de controle para acionamento dos mesmos. Foi dividido em duas partes, de acordo com os problemas, Problemas de Funções e Problemas de Instâncias. Para realizar toda a comunicação do *back-end* foi criada uma classe de controle. Esta classe possui as seguintes atribuições: validar os parâmetros recebidos pelo *front-end*, acionar a execução dos métodos para todos os tipos de problemas, controlar o processo de simulação para todos os problemas e gerar os resultados gráficos ao final das execuções.

Os Problemas de Instâncias são subdivididos em TSP e CVRP, assim o *back-end*, para acionamento de um método possui três opções: resolver um Problema de Função, resolver um Problema do TSP ou resolver um Problema do CVRP. Cada uma das três subdivisões possui uma classe para acionamento dos métodos para resolução de cada problema de acordo com os parâmetros recebidos e para retorno dos resultados encontrados.

O *back-end* durante o acionamento dos métodos é responsável por controlar o processo de hibridização, seu trabalho é o de reconhecer o método híbrido e agregar aos seus parâmetros a melhor solução encontrada por outro método.

O controle de simulação realizado pelo *back-end* incorpora o acionamento dos métodos de execução citados e a organização dos resultados obtidos de acordo com os números de repetições definidos para a simulação.

A classe *back-end* ainda tem a responsabilidade de gerar as informações para exibição pelas páginas *web* no *front-end*. Essas informações podem ser no formato de gráficos, estruturas de dados e arquivos. Os gráficos podem ser informativos sobre os problemas ou de resultados da execução, são gerados como imagens com auxílio da biblioteca *MatPlotlib*. As estruturas de dados representam os valores gerados pelos métodos como resultado da execução do problema. Os arquivos são para uso da atualização dos dados da simulação em execução, são

usados como meio de comunicação facilitada para informar o andamento do processo de simulação para o *front-end*.

4.5. Calibragem de Configurações de Parâmetros dos Métodos

Para se entender o funcionamento de cada método aplicado para cada tipo de problema foram realizadas simulações com análise dos resultados de acordo com a mudança na configuração dos parâmetros utilizados. Esta parte da pesquisa serviu para entendimento do comportamento de cada método, sobre suas especificidades perante cada problema.

Os testes de eficiência do GA para funções iniciaram com uma versão do método menos desenvolvida. Nela possuíam as operações genéticas de cruzamento e mutação, e necessitava dos seguintes parâmetros: tamanho da população, número de gerações, tamanho do cromossomo, taxa de cruzamento e de mutação. Para quantificar a eficiência dos testes foi utilizada a Equação (36), esta resulta em uma taxa de aproximação do valor encontrado em um teste em relação ao valor ótimo.

$$eficiência = \frac{valor\ ótimo}{valor\ encontrado} \tag{36}$$

Os testes iniciaram para definição do tamanho do cromossomo. Esta variável determina o tamanho do vetor binário que representa o valor de cada solução para o problema. Quanto maior o tamanho do cromossomo, maior é o número inteiro que pode ser gerado, e este quando ajustado dentro do intervalo de busca, possui uma precisão maior.

Para análise deste valor todos os demais parâmetros foram fixados, enquanto o tamanho do cromossomo era gerado aleatoriamente, analisando a variação do valor e o comportamento dos resultados. Os valores de configuração foram: população 100, número de gerações 50, cruzamento 0,5, mutação 0,1. A função utilizada para validação foi a *Easom*, que possui a melhor solução com o valor -1. Os valores para o parâmetro foram gerados em uma faixa de valores entre 10 e 130. Os testes foram repetidos 3000 vezes independentes e os resultados foram organizados em histogramas e exibidos nas Figuras 24 e 25. O primeiro exibindo todos os valores distribuídos em blocos variando de 20 em 20. O outro com os valores resumidos dentro da faixa que mais se aproximou com variação de 2 em 2.

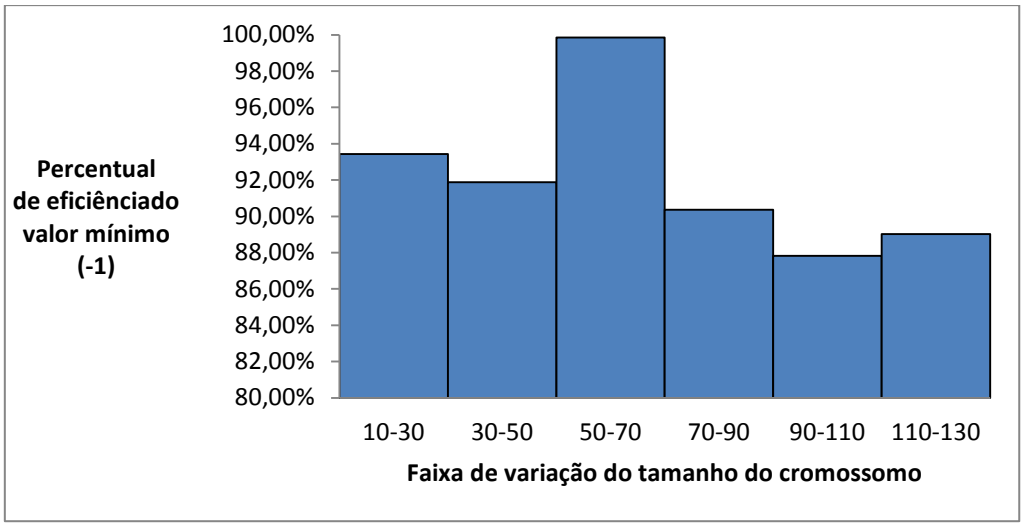

Figura 24 - Distribuição dos resultados pelas faixas de valores do tamanho do cromossomo
Fonte: Autor

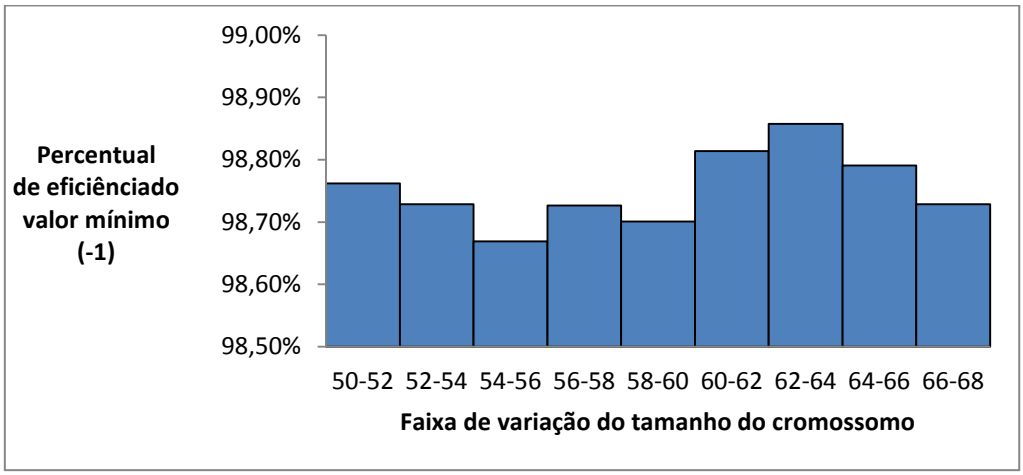

Figura 25 - Distribuição dos resultados entre 50 e 70 para o tamanho do cromossomo
Fonte: Autor

O valor que melhor se ajustou para a análise feita foi 64, assim foi definido como uma constante para os demais testes este valor para o parâmetro de tamanho do cromossomo.

Foram realizadas análises dos demais parâmetros do algoritmo genético para o problema de instância, com o objetivo de definir qual a melhor configuração de parâmetros para os operadores genéticos. Os dados foram organizados em cenários de acordo com os desempenhos obtidos com o decorrer dos testes. Foi escolhida a instância *eil51* para a realização das análises, esta instância possui o valor 426 como solução ótima.

Foram gerados 14 diferentes cenários de configuração dos parâmetros: números de gerações, tamanho da população, taxa de elitismo, taxa de reprodução assexuada, taxa de mutação simples e taxa de mutação inversa. Os valores definidos para cada cenário são mostrados na Tabela 7.

Tabela 7 - Valores definidos para as configurações de parâmetros para os cenários

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
Gerações	200	200	200	200	200	2000	2000	2000	2000	2000	1000	1000	1000	1000
População	1000	1000	1000	1000	1000	500	500	500	500	500	200	200	100	100
Elitismo	0	0,7	0	0,6	0,6	0,25	0,5	0,2	0	0,1	0,25	0,15	0,3	0,15
Assexuada	R	0	0,8	0,8	0	0	0	0	0	0	0,3	0,3	0,4	0,4
Simples	R	0,3	0	0	0	R	R	R	-	-	0,1	0,1	0,05	0,05
Inversa	R	0,8	0	0	0,8	R	R	R	R	R	0,8	0,8	0,8	0,8

Fonte: Autor

Os valores dos cenários foram escolhidos de acordo com a evolução dos testes por meio da análise estatística do “Resumo dos Cinco Números” exibidas na Tabela 8 e Figura 26.

Tabela 8 - Resumo dos Cinco Números para os cenários simulados

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
3º Quartil	746	529,5	1064	872	647	532	539	556,25	521,5	533	487	442	458	472
Máximo	1717	571	1118	920	692	703	736	728	826	998	524	452	477	524
Mediana	629	513	1042	853,5	630	493	504	514	492,5	493	474	439	451	465
Média	669,72	513,12	1039,25	851,49	627,87	511,45	520,07	531,01	511,95	525,98	475,04	438,66	452,34	466,07
Mínimo	477	462	915	755	564	448	456	440	447	436	440	426	434	442
1º Quartil	561,75	494	1019,5	832	609	474	481	488	475,25	476	464	436	446	458

Fonte: Autor

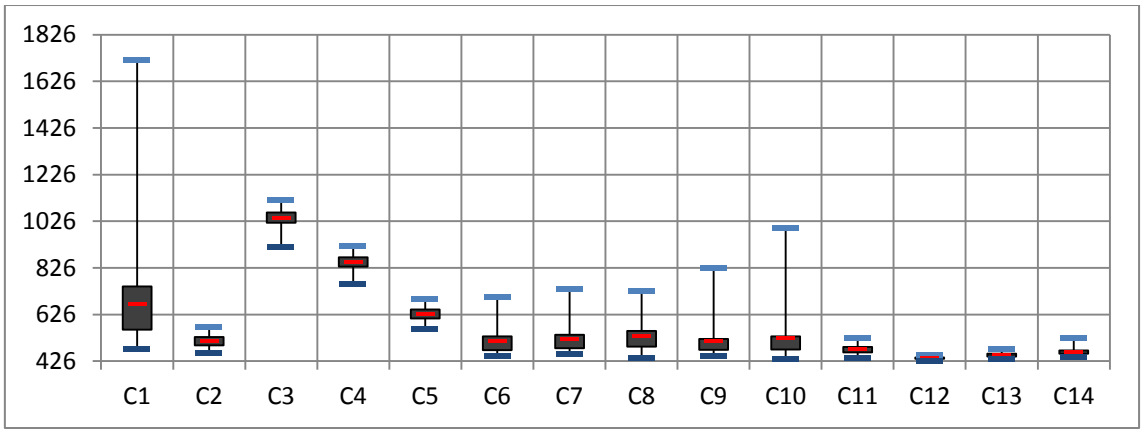

Figura 26 - *Boxplot* com os resultados simulados para os cenários
Fonte: Autor

Verificou-se após análise que, entre os cenários analisados o *C12* foi o que teve a configuração com melhor desempenho na resolução do problema. Ele foi definido usando uma população com tamanho 200, 1000 gerações, 15% de taxa de elitismo, 30% de taxa de reprodução assexuada, 10% de mutação simples e 80% de taxa de mutação inversa.

Com a análise dos testes algumas considerações podem ser feitas:

- O elitismo é uma técnica que agrega eficiência para o método. Resulta em melhores valores comparados em cenários que não a utilizam, como exemplo as comparações entre os cenários *C3* e *C4* e os cenários *C9* e *C10*.
- Ficou comprovada a superioridade da técnica *Lin Kernighan Heuristic* utilizada como forma de operador de mutação para o Algoritmo Genético.

Esta afirmação pode ser observada nos cenários *C2*, *C5*, *C11*, *C12*, *C13* e *C14*, onde o método tem taxas de aplicação superiores às outras operações genéticas.

Para determinação dos dados utilizados para o algoritmo Colônia de Formigas foram realizados testes com valores gerados aleatoriamente e aplicados a instâncias do TSP. Foi selecionada a instância *dantzig42* para validação dos testes da configuração de parâmetros, esta que possui o valor 679,2 como solução ótima.

O algoritmo utiliza os parâmetros número de formigas, número de gerações, α, β, ρ e *Q*. Para avaliação da configuração foram gerados valores aleatórios para todos os parâmetros ao mesmo tempo, executando o algoritmo para solucionar o problema e analisando as combinações de parâmetros que tiveram o melhor desempenho na solução do problema.

Os parâmetros foram gerados aleatoriamente entre as faixas descritas na Tabela 9 e as simulações foram executadas 3000 vezes independentes.

Tabela 9 - Faixas de parâmetros para simulações do ACO

	Número de Formigas	Número de Gerações	α	β	ρ	Q
Limite Inferior	10	10	0,1	1	0,1	1
Limite Superior	200	200	1	20	1	20

Fonte: Autor

Os resultados das combinações tiveram variação entre 55,9% e 99,8% de eficiência sob o problema escolhido. Foram analisados os valores que mais apareceram entre as configurações das simulações mais bem colocadas, permitindo que nova análise fosse feita diminuindo a variação das faixas de parâmetros para o método. A nova faixa foi estabelecida e mesma pode ser visualizada na Tabela 10.

Tabela 10 - Faixas de parâmetros para simulações do ACO após refinamento

	Número de Formigas	Número de Gerações	α	β	ρ	Q
Limite Inferior	70	80	0,5	5	0,2	6
Limite Superior	110	100	0,8	7	0,3	10

Fonte: Autor

As simulações novamente foram executadas para as novas configurações de parâmetros e a variação da eficiência sob o problema passou a ter o mínimo de 90,04%, confirmando melhoria da nova faixa selecionada. Analisando os novos resultados, foram encontradas coincidências de combinações em cerca de 30% dos melhores resultados das simulações. As configurações de parâmetros assumidas como boas para a resolução do problema são mostradas na Tabela 11.

Tabela 11 - Parâmetros com os melhores resultados entre os simulados

Número de Formigas	Número de Gerações	α	β	ρ	Q
80	90	0,8	6	0,3	9

Fonte: Autor

Esses dados foram considerados na definição das configurações utilizadas para o algoritmo ACO para a execução dos testes.

5. EXPERIMENTOS E ANÁLISES DE RESULTADOS

5.1. Experimentos com Funções de *Benchmark*

Nesta seção, as funções de *benchmark Ackley*, *Rastrigin*, *Rosenbrock* e *Sphere* foram aplicadas para avaliar o desempenho dos algoritmos desenvolvidos. Estas funções são amplamente utilizadas com este objetivo na literatura. Elas possuem as características: unimodal e multimodal, contínua, convexa e não convexa, diferenciável, separável e não separável. Todas as funções são multidimensionais e estão estruturadas na Tabela 12 com as informações: número de identificação, nome, formulação, dimensões (número de variáveis), domínio e valor mínimo existente.

Tabela 12 - Funções de *benchmark* utilizadas para teste

Nº	Nome	Função de *Benchmark*	*D*	Domínio	$f_{Mín}$
f_1	Ackley	$f(x) = -a\,exp\left(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(cx_i)\right) + a + e$	30	$x_i \in [-32, 32]$	0
f_2	Rastrigin	$f(x) = \sum_{i=1}^{n} [x_i^2 - 10cos(2\pi x_i + 10]$	30	$x_i \in [-5.12, 5.12]$	0
f_3	Rosenbrock	$f(x) = \sum_{x=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right]$	30	$x_i \in [-30, 30]$	0
f_4	Sphere	$f(x) = \sum_{i=1}^{n} x_i^2$	30	$x_i \in [-100, 100]$	0

Fonte: Autor

Foram executadas simulações do Algoritmo Genético na solução dessas funções para avaliar o desempenho do método na solução dos problemas, enquanto era reconhecido o nível de dificuldade dessas funções. As simulações foram executadas para cada função com 2, 5, 10 e 20 dimensões, aumentando o número de variáveis e analisando o comportamento do método de acordo com a evolução. Os parâmetros utilizados pelo GA para a resolução das funções para todas as dimensões foram: taxa de cruzamento 0,8, taxa de mutação 0,05, taxa de elitismo 0,15, população com tamanho 100 e número de gerações 200.

O valor ótimo existente para as funções selecionadas é 0, assim, método conseguiu se aproximar do valor ótimo para todos cenários de simulação, a Tabela 13 mostra os valores de mínimo, média e desvio padrão encontrados nas simulações. Cada simulação foi repetida 250 vezes.

Os resultados das simulações demonstraram que o Algoritmo Genético possui eficiência no trabalho de aproximação do ótimo global, principalmente para os

casos de problemas com poucas variáveis. Dentre as funções testadas destacam-se as funções f_3 (Rosenbrock) e f_4 (Sphere), sendo respectivamente as de maior e menor dificuldade na busca da solução ótima entre as funções testadas. Apenas nas simulações com duas dimensões o algoritmo conseguiu encontrar a solução ótima em todos os casos.

Tabela 13 – Resultados das simulações executadas para os problemas de funções

D		f_1	f_2	f_3	f_4
2	$f_{Mín}$	**0,00E+00**	**0,00E+00**	**0,00E+00**	**0,00E+00**
	$f_{Méd}$	4,44E-16	**0,00E+00**	9,41E-06	9,82E-31
	σ_f	4,94E-31	**0,00E+00**	8,82E-05	1,73E-29
5	$f_{Mín}$	3,73E-13	**0,00E+00**	2,40E-05	7,89E-31
	$f_{Méd}$	1,57E-10	2,54E-12	7,61E-01	1,51E-14
	σ_f	3,22E-10	2,69E-11	7,26E-01	1,68E-13
10	$f_{Mín}$	1,51E-06	2,88E-12	9,11E-03	2,48E-14
	$f_{Méd}$	1,44E-05	1,97E-09	5,99E+00	1,02E-11
	σ_f	1,09E-05	3,32E-09	2,04E+00	3,15E-11
20	$f_{Mín}$	3,91E-03	3,77E-04	2,09E+00	1,20E-06
	$f_{Méd}$	1,29E-02	8,36E-02	2,36E+01	9,12E-06
	σ_f	4,74E-03	2,82E-01	1,95E+01	6,14E-06

Fonte: Autor

Além do Algoritmo Genético foi desenvolvido para solução deste tipo de problema o Recozimento Simulado. Tendo como base as duas opções de meta-heurística foram realizadas as hibridizações de alto nível envolvendo os dois algoritmos. Criaram-se as combinações GA+SA e SA+GA, seguindo a lógica de hibridização de trabalho colaborativo retransmitido.

Os métodos de solução desenvolvidos GA, SA, GA+SA e SA+GA, foram testados e comparados com as seguintes meta-heurísticas da literatura: PSO-GSA (MIRJALILI; HASHIM, 2010), LMFO (LI *et al.*, 2016), BA (YANG, 2010), GWO (MIRJALILI; MIRJALILI; LEWIS, 2014), MSA (MOHAMED *et al.*, 2017), EOMSA (LUO; YANG; ZHOU, 2019). As configurações de parâmetros seguiram um padrão entre os métodos para nivelamento das execuções, utilizando sempre 50 indivíduos e máximo de 1000 iterações, permitindo comparações mais igualitárias, assim foram utilizados os seguintes valores:

- PSO-GSA e LMFO: população com tamanho 50 e número máximo de 1000 iterações.
- BA: máxima intensidade de pulso (*A*) 0,9, máxima frequência de pulso (*r*) 0,5, coeficiente de atenuação de pulso (*α*) 0,95, incremento do fator de frequência (γ) 0,05, população com tamanho 50 e número máximo de iterações 1000.
- GWO: *α* decrementado de 2 a 0 linearmente, população com tamanho 50 e número máximo de iterações 1000.

- MSA e EOMSA: população com tamanho 50, 6 desbravadores e número máximo de iterações 1000.
- GA: taxa de cruzamento 0,8, taxa de mutação 0,05, taxa de elitismo 0,15, população com tamanho 50 e número de gerações 1000.
- SA: temperatura (*T*) 10000, fator de resfriamento (α) 0,99, tolerância 1e-10 e máximo de iterações 1000.
- Para as hibridizações (GA+SA e SA+GA) os números de gerações do GA foram reduzidos para 500, assim como o número de iterações do SA.

As simulações foram executadas com esses parâmetros 50 vezes independentes para cada método, sob as mesmas condições, possibilitando uma comparação fiel com os dados da literatura. O número de dimensões atribuído às funções foi 30, utilizando-as em um nível alto de complexidade. A Tabela 14 mostra a comparação dos valores encontrados. Na tabela os valores de mínimo, máximo, média e desvio padrão das soluções são representados por $f_{Mín}$, $f_{Máx}$, $f_{Méd}$ e σ_f, respectivamente.

Tabela 14 - Comparação de resultados de funções com a literatura

		PSO-GSA	LMFO	BA	GWO	MSA	EOMSA	GA*	SA*	GA+SA*	SA+GA*
f_1	$f_{Mín}$	2,93E−10	**8,88E−16**	1,88E+01	1,15E−14	**8,88E−16**	**8,88E−16**	1,05E-01	5,35E+00	1,81E+00	6,92E-01
	$f_{Máx}$	1,89E+01	8,88E−16	2,00E+01	2,22E−14	8,88E−16	8,88E−16	2,56E+00	8,56E+00	3,42E+00	2,49E+00
	$f_{Méd}$	1,13E+01	8,88E−16	1,93E+01	1,62E−14	8,88E−16	8,88E−16	4,03E-01	6,99E+00	1,85E+00	1,45E+00
	σ_f	7,26E+00	0,00E+00	3,28E-01	2,66E−15	0,00E+00	0,00E+00	3,83E-01	6,95E-01	2,29E-01	4,52E-01
f_2	$f_{Mín}$	4,88E+01	**0,00E+00**	2,60E+02	**0,00E+00**	**0,00E+00**	**0,00E+00**	2,52E+00	4,24E+01	1,10E+01	7,75E+00
	$f_{Máx}$	2,11E+02	0,00E+00	4,72E+02	3,28E+00	0,00E+00	0,00E+00	1,43E+01	1,07E+02	2,27E+01	2,68E+01
	$f_{Méd}$	1,30E+02	0,00E+00	3,60E+02	2,18E-01	0,00E+00	0,00E+00	7,40E+00	8,02E+01	1,12E+01	1,32E+01
	σ_f	3,45E+01	0,00E+00	4,58E+01	8,30E-01	0,00E+00	0,00E+00	2,17E+00	1,33E+01	1,65E+00	3,53E+00
f_3	$f_{Mín}$	**1,42E+01**	2,64E+01	3,96E+01	2,58E+01	2,56E+01	2,46E+01	3,28E+01	1,06E+04	6,72E+02	1,84E+02
	$f_{Máx}$	1,47E+02	2,88E+01	4,89E+02	2,79E+01	2,87E+01	2,69E+01	2,74E+03	8,95E+04	7,09E+02	6,18E+02
	$f_{Méd}$	3,21E+01	2,70E+01	7,95E+01	2,68E+01	2,68E+01	2,59E+01	4,23E+02	3,04E+04	7,09E+02	3,27E+02
	σ_f	2,45E+01	3,93E-01	7,81E+01	5,51E-01	7,04E-01	4,87E-01	5,21E+02	1,44E+04	5,27E+00	9,73E+01
f_4	$f_{Mín}$	1,50E-19	1,80E-260	1,45E+04	9,35E−62	**0,00E+00**	**0,00E+00**	1,97E-01	3,32E+01	4,50E+00	9,95E-01
	$f_{Máx}$	2,00E+04	1,50E-197	2,93E+04	1,72E−57	0,00E+00	0,00E+00	1,41E+00	1,41E+02	8,39E+00	1,44E+01
	$f_{Méd}$	2,00E+03	3,00E−199	3,46E+05	1,37E−58	0,00E+00	0,00E+00	6,82E-01	6,40E+01	6,09E+00	4,28E+00
	σ_f	4,52E+03	0,00E+00	3,32E+03	3,36E−58	0,00E+00	0,00E+00	3,63E-01	1,99E+01	8,19E-01	2,50E+00

*Algoritmos desenvolvidos nesta pesquisa
Fonte: Autor

De acordo com os dados executados, os métodos desenvolvidos não atingiram o valor ótimo para nenhuma das funções executadas com 30 dimensões. As taxas de convergência e aproximação dos valores ótimos são regulares, pois em vários cenários o valor encontrado foi próximo ao valor ótimo do problema. Para o problema f_3 (*Rosenbrock*) os métodos desenvolvidos tiveram a maior dificuldade, sendo considerado o problema de maior dificuldade entre os utilizados.

Tendo em vista apenas os algoritmos desenvolvidos, o método com o melhor desempenho para estes problemas foi o GA, superando os demais em todos os cenários executados. O SA foi o método com os piores resultados encontrados, isso

se dá pela configuração dos parâmetros estabelecida, já que a limitação para nivelamento dos algoritmos, onde define o número máximo de iterações em 1000, prejudicou o desempenho do método. O método demonstrou resultados melhores com testes executados com um limite de 3000 iterações, no entanto, os dados não foram utilizados, pois tal configuração causaria um descompasso entre os parâmetros estabelecidos.

Em relação às hibridizações, o desempenho do algoritmo GA+SA foi inferior ao SA+GA, isto ocorre pelo motivo citado anteriormente, a configuração definida para o SA limita a sua evolução, impossibilitando que o mesmo atinja resultados superiores. Desta forma, quando o mesmo é executado após o GA, não há iterações suficientes para encontrar bons resultados. No caso do SA+GA, mesmo tendo resultados melhores que a outra hibridização, ainda não alcança os resultados atingidos pelo GA executando individualmente.

Esses resultados demonstram que o sucesso destes algoritmos é totalmente dependente da quantidade de iterações/gerações executadas e para realização de mudanças desses números, seriam necessários estudos sobre ajustes de configurações de parâmetros para cada variação de hibridização.

5.2. Experimentos para o Problema do Caixeiro Viajante (TSP)

Foram desenvolvidos para resolução do TSP os algoritmos GA, SA e ACO, que foram hibridizados gerando as combinações SA+GA, GA+SA, ACO+GA e ACO+SA. Os desempenhos destes algoritmos e hibridizações foram testados por meio da resolução de sete Instâncias de *benchmark* da biblioteca TSPLIB (REINELT, 1991), de diferentes portes: dantzig42, eil51, berlin52, st70, eil76, krob200 e pr299.

As Instâncias foram solucionadas pelos métodos utilizando as seguintes configurações de parâmetros:

- ACO: número de formigas 100, número de gerações 100, parâmetros de controles da importância do feromônio 0,8 e 10 (α e β), coeficiente de evaporação 0,3 (ρ), e constante de qualidade da função 1 (Q).
- GA: taxa de cruzamento 0,3, taxa de mutação inversa 0,8, taxa de mutação 0,1, taxa de elitismo 0,15, população com tamanho 50 e número de gerações 1000.
- SA: temperatura (T) 10000, fator de resfriamento (α) 0,999, tolerância 1e-10 e máximo de iterações 5000.
- Para as hibridizações envolvendo o GA os números de gerações foram reduzidos para 500.

- Para as hibridizações GA+SA e ACO+SA a temperatura (*T*), foi substituída pela função: $\log f(x_0)$.

Esta última configuração foi realizada após a constatação de que as hibridizações, quando envolvem o SA com a configuração padrão, possuem uma grande probabilidade de perda da solução recebida de um método previamente executado. Isso ocorre pela permissão de perda de soluções pelo algoritmo em momentos de altas temperaturas. Para superar esse problema foi atribuído o valor da equação $\log f(x_0)$ à temperatura inicial, reduzindo a probabilidade de perda de uma solução híbrida (ARAUJO, 2001).

As simulações foram repetidas 50 vezes independentes para cada um dos métodos e os resultados são exibidos na Tabela 15. Na tabela são expostos os dados de custo mínimo ($f_{Mín}$), custo médio ($f_{Méd}$) e desvio padrão (σ_f) encontrados.

Tabela 15 - Resultados das simulações executadas para as instâncias do TSP

Nome/ Valor Ótimo		ACO	GA	SA	SA+GA	GA+SA	ACO+GA	ACO+SA
dantzig42/ 679,20	$f_{Mín}$	689,16	**679,20**	686,88	**679,20**	**679,20**	**679,20**	**679,20**
	$f_{Méd}$	718,94	689,41	707,04	690,19	685,19	682,45	690,17
	σ_f	10,99	5,77	11,21	7,54	3,90	3,67	7,55
eil51/ 426,00	$f_{Mín}$	436,48	431,32	430,86	428,98	431,87	**426,00**	431,85
	$f_{Méd}$	458,41	436,83	448,17	436,84	438,45	431,07	442,30
	σ_f	6,25	3,09	6,23	3,79	3,02	3,62	6,27
berlin52/ 7542,00	$f_{Mín}$	7555,70	**7542,00**	**7542,00**	**7542,00**	**7542,00**	**7542,00**	**7542,00**
	$f_{Méd}$	7801,92	7716,69	8053,66	7658,96	7659,99	7544,37	7658,96
	σ_f	103,97	133,46	164,30	157,22	138,67	45,71	98,41
st70/ 675,00	$f_{Mín}$	703,47	694,76	686,77	683,09	691,25	**675,00**	701,67
	$f_{Méd}$	718,57	753,62	720,19	716,22	732,13	688,67	732,68
	σ_f	7,17	29,84	18,79	20,90	21,90	8,83	19,07
eil76/ 538,00	$f_{Mín}$	565,23	575,68	659,04	595,78	671,87	546,05	704,64
	$f_{Méd}$	570,69	608,06	763,24	622,59	757,50	557,43	762,55
	σ_f	3,72	21,33	31,25	16,79	32,41	6,27	28,81
krob200/ 29437,00	$f_{Mín}$	32099,13	35228,97	32422,85	32395,18	40331,73	32051,85	39366,21
	$f_{Méd}$	32733,43	36749,63	33579,32	33560,65	43004,56	32608,30	42682,35
	σ_f	270,79	728,80	559,44	616,17	1355,38	249,21	1283,04
pr299/ 48191,00	$f_{Mín}$	53241,22	57982,15	54599,28	54825,05	83989,27	52310,03	89263,21
	$f_{Méd}$	54750,01	61997,36	57840,54	57609,76	93861,51	54267,68	94063,95
	σ_f	614,18	2436,17	1936,60	1766,79	3337,67	608,72	2478,52

Fonte: Autor

De acordo com as simulações executadas, os métodos tiveram maior facilidade na resolução de instâncias menores, a eficiência dos métodos é comparada na Tabela 16, considerando o percentual do valor encontrado em relação ao valor ótimo.

Entre os métodos aplicados a hibridização ACO+GA é a opção com os melhores resultados na resolução das instâncias, atingindo uma eficiência média de 97,5%, com uma vantagem de 1,88% do método ACO, comprovando a vantagem da hibridização dos métodos. A combinação foi selecionada para comparações com os métodos desenvolvidos na literatura.

Tabela 16 – Comparação da eficiência* dos resultados para o TSP

	ACO	GA	SA	SA+GA	GA+SA	ACO+GA	ACO+SA
dantzig42	98,55%	**100,00%**	98,88%	**100,00%**	**100,00%**	**100,00%**	**100,00%**
eil51	97,60%	98,77%	98,87%	99,30%	98,64%	**100,00%**	98,64%
berlin52	99,82%	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**	**100,00%**
st 70	95,95%	97,16%	98,29%	98,82%	97,65%	**100,00%**	96,20%
eil76	95,18%	93,45%	81,63%	90,30%	80,07%	**98,53%**	76,35%
krob200	91,71%	83,56%	90,79%	90,87%	72,99%	**91,84%**	74,78%
pr299	90,51%	83,11%	88,26%	87,90%	57,38%	**92,13%**	53,99%
Eficiência Média	95,62%	93,72%	93,82%	95,31%	86,68%	**97,50%**	85,71%

Fonte: Autor
* Eficiência calculada com base na Equação (36).

Foram feitas comparações com dois trabalhos da literatura. O primeiro utiliza as instâncias de *benchmark dantzig42* e *st70* para validar os resultados de uma proposta de variação do método *Algorithm Bee Colony* (ABC). O método foi comparado com: *Particle Swarm Optimization* (PSO), GA, ACO, SA e ABC básico, desenvolvidos por (WANG *et al.*, 2015). A Tabela 17 expõe uma comparação dos resultados entre os métodos do autor e a hibridização ACO+GA desenvolvida neste trabalho.

As configurações de parâmetros adotadas por (WANG *et al.*, 2015) para os métodos são:

- PSO: número de indivíduos 300 e número total de iterações em 3000.
- GA: taxa de cruzamento 0,9, taxa de mutação 0,05, taxa de seleção 0,9, população com tamanho 50 e número de gerações 2000.
- ACO: número de formigas 50, número de gerações 2000, parâmetros de controles da importância do feromônio 1 e 5 (α e β) e coeficiente de evaporação 0,3 (ρ).
- SA: temperatura (T) 50000, fator de resfriamento (α) 0,95, tolerância 1e-3 e máximo de iterações 4000.
- ABC e ABC Proposto: tamanho da colônia 100, número máximo de iterações 900, taxa de cruzamento 0,75, probabilidade de reversão 0,8, colônia divida ao meio em abelhas assistentes e abelhas empregadas, apenas uma abelha batedora e limite de ineficiência 75.

Tabela 17 - 1ª Comparação de resultados do TSP com a literatura

		ACO+GA*	PSO	GA	ACO	SA	ABC	ABC Proposto
dantzig42	$f_{Mín}$	**679,20**	**679,20**	**679,20**	703,83	686,21	**679,20**	**679,20**
	$f_{Méd}$	682,45	699,87	715,83	724,68	722,85	695,38	685,40
	σ_f	3,67	13,14	22,06	10,50	22,64	12,11	5,55
st70	$f_{Mín}$	**675,00**	677,19	692,45	699,24	709,36	692,33	677,11
	$f_{Méd}$	688,67	717,73	732,06	710,39	781,54	712,09	691,63
	σ_f	8,83	22,70	21,38	4,77	33,93	15,01	8,83

*Algoritmos desenvolvidos nesta pesquisa
Fonte: Autor

A hibridização ACO+GA encontrou o valor ótimo para as duas instâncias avaliadas. Para a *dantzig42*, o método atingiu além da solução ótima, os menores valores de média e desvio padrão entre os algoritmos comparados. Já para a instância st70 o método atinge a melhor média e o segundo menor desvio padrão, sendo o único método entre os comparados a encontrar o valor ótimo da instância.

O segundo trabalho utilizado para comparação é (MAITY; ROY; MAITI, 2017), nele é feita uma proposta de hibridização entre os algoritmos ACO e GA com atualização inteligente de feromônios, denominado rACO-GA. No trabalho o método proposto foi comparado com os algoritmos GA e ACO na forma tradicional. A Tabela 18 contém os parâmetros utilizados por (MAITY; ROY; MAITI, 2017) para as configurações de seus algoritmos.

Tabela 18 - Configuração de parâmetros para os algoritmos da literatura

	ACO			GA			
Tamanho do Problema (número de vértices)	Gerações	Formigas	Controle de feromônio	Gerações	População	Cruzamento	Mutação
N ≤ 50	80	30	0,2	120	50	0,35	0,1
50 < N ≤ 100	120	50	0,2	180	100	0,3	0,15
200 < N ≤ 250	450	100	0,3	400	150	0,45	0,2
250 < N ≤ 300	400	100	0,3	500	150	0,45	0,25

Fonte: Adaptado de (MAITY; ROY; MAITI, 2017)

Os dados obtidos como referência foram organizados na Tabela 19, junto dos dados gerados pelo híbrido ACO+GA desenvolvido no presente trabalho.

Tabela 19 - 2ª Comparação de resultados do TSP com a literatura

		ACO+GA*	GA	ACO	rACO-GA
dantzig42	$f_{Mín}$	**679,20**	699,00	703,00	699,00
	$f_{Méd}$	682,45	700,07	703,51	699,27
	σ_f	3,67	0,68	0,98	0,67
eil51	$f_{Mín}$	**426,00**	**426,00**	430,00	**426,00**
	$f_{Méd}$	431,07	429,31	432,98	427,80
	σ_f	3,62	0,93	1,53	0,98
berlin52	$f_{Mín}$	**7542,00**	7623,00	7883,00	**7542,00**
	$f_{Méd}$	7544,37	7654,87	7936,35	7548,90
	σ_f	45,71	1,07	2,49	1,76
eil76	$f_{Mín}$	546,05	547,00	547,00	**538,00**
	$f_{Méd}$	557,43	545,86	567,27	539,65
	σ_f	6,27	3,65	1,93	0,67
krob200	$f_{Mín}$	32051,85	29789,00	29944,00	**29413,00**
	$f_{Méd}$	32608,30	29965,27	30887,34	29450,70
	σ_f	249,21	6,14	6,82	3,02
pr299	$f_{Mín}$	52310,03	49391,00	49765,00	**48743,00**
	$f_{Méd}$	54267,68	50831,43	52945,78	49765,60
	σ_f	608,72	8,37	10,21	5,92

*Algoritmos desenvolvidos nesta pesquisa
Fonte: Autor

Na Tabela 19 pode ser observado que o algoritmo híbrido possui eficiência para encontrar soluções ótimas para problemas com porte menor, conforme o

tamanho do problema aumenta, especialmente nos casos de 200 e 299 vértices o algoritmo não consegue atingir valores acima de 93% da solução ótima.

O algoritmo rACO-GA, usado como referência, atinge os melhores valores entre os comparados em cinco dos seis casos testados, enquanto o proposto atinge em três. O método rACO-GA é uma hibridização que tem como base os mesmos algoritmos do ACO-GA, porém com uma abordagem distinta, já que o método possui uma técnica inteligente de ajuste do parâmetro feromônio, diferencial para a eficiência do método em relação a estes problemas.

Os métodos implementados por (MAITY; ROY; MAITI, 2017) utilizam parâmetros diferentes, conforme exibido na Tabela 18. Estes parâmetros foram testados com os algoritmos desenvolvidos no presente trabalho, porém não tiveram melhoria na eficiência dos resultados e aumentaram o tempo de execução dos métodos em três vezes em alguns casos. Este aumento no tempo ocorreu pela configuração utilizada para o número de gerações do ACO.

5.3. Experimentos para o Problema de Roteamento de Veículos

Foram desenvolvidos para o VRP as mesmas opções de métodos do TSP, GA, SA e ACO, e as hibridizações SA+GA, GA+SA, ACO+GA e ACO+SA. Para testar a eficiência desses algoritmos e hibridizações foi utilizada uma biblioteca completa de Instâncias, a A-VRP (AUGERAT *et al.*, 1995). Na A-VRP o número de vértices (consumidores) varia de 32 a 80 e o número de veículos entre 5 e 10, conforme é mostrado na Tabela 20.

Tabela 20 - Descrição das instâncias do A-VRP

Instância	Total de Vértices	Número de Veículos	Capacidade Individual	Demanda Total	Solução Ótima
A-n32-k5	32	5	100	410	784
A-n33-k5	33	5	100	446	661
A-n33-k6	33	6	100	541	742
A-n34-k5	34	5	100	460	778
A-n36-k5	36	5	100	442	799
A-n37-k5	37	5	100	407	669
A-n37-k6	37	6	100	570	949
A-n38-k5	38	5	100	481	730
A-n39-k5	39	5	100	475	822
A-n39-k6	39	6	100	526	831
A-n44-k6	44	6	100	570	937
A-n45-k6	45	6	100	593	944
A-n45-k7	45	7	100	634	1146
A-n46-k7	46	7	100	603	914
A-n48-k7	48	7	100	626	1073
A-n53-k7	53	7	100	664	1010
A-n54-k7	54	7	100	669	1167
A-n55-k9	55	9	100	839	1073
A-n60-k9	60	9	100	829	1354
A-n61-k9	61	9	100	885	1034
A-n62-k8	62	8	100	733	1288
A-n63-k9	63	9	100	873	1616
A-n63-k10	63	10	100	932	1314
A-n64-k9	64	9	100	848	1401
A-n65-k9	65	9	100	877	1174
A-n69-k9	69	9	100	845	1159
A-n80-k10	80	10	100	942	1763

Fonte: Autor

As simulações foram repetidas 50 vezes independentes para cada um dos algoritmos resolvendo todas as Instâncias do A-VRP, com um total de 189 combinações, conforme mostrado na Tabela 21. A configuração dos parâmetros para a realização desses testes é a mesma aplicada ao TSP, com a diferença que o GA não possui os parâmetros de cruzamento e mutação.

Tabela 21 - Resultados das simulações executadas para as instâncias do CVRP

		ACO	GA	SA	SA+GA	GA+SA	ACO+GA	ACO+SA
A-n32-k5	$f_{Mín}$	831,85	806,48	801,37	803,31	804,57	805,19	**797,45**
784	$f_{Méd}$	849,59	865,52	851,25	849,47	848,03	820,33	837,38
	σ_f	5,63	40,16	40,23	28,00	27,19	5,45	23,17
A-n33-k5	$f_{Mín}$	692,24	668,02	**662,40**	664,79	**662,40**	668,02	666,02
661	$f_{Méd}$	699,54	695,83	686,57	684,80	691,20	680,94	690,20
	σ_f	4,29	19,04	14,00	13,83	19,45	5,53	20,93
A-n33-k6	$f_{Mín}$	768,91	744,26	**742,69**	**742,69**	**742,69**	**742,69**	743,00
742	$f_{Méd}$	787,51	778,77	764,07	762,81	766,16	755,44	768,35
	σ_f	4,78	31,52	20,26	20,86	20,73	9,88	28,15
A-n34-k5	$f_{Mín}$	814,37	**780,94**	**780,94**	**780,94**	**780,94**	**780,94**	786,44
778	$f_{Méd}$	825,71	814,76	810,76	808,68	805,25	798,11	814,93
	σ_f	3,21	20,36	20,70	17,05	17,16	7,18	21,87
A-n36-k5	$f_{Mín}$	860,05	820,75	815,57	819,65	**809,71**	814,15	**809,71**
799	$f_{Méd}$	878,20	868,20	851,19	848,05	844,98	836,34	849,36
	σ_f	7,76	27,37	33,71	18,61	18,58	11,73	19,14
A-n37-k5	$f_{Mín}$	730,02	702,61	672,75	**672,52**	**672,52**	705,69	676,21
669	$f_{Méd}$	749,59	742,54	727,20	718,08	722,67	726,49	727,53
	σ_f	7,96	26,39	38,74	27,37	32,05	7,87	34,32
A-n37-k6	$f_{Mín}$	993,81	956,81	957,03	956,81	**954,68**	958,90	973,08
949	$f_{Méd}$	1003,12	1013,57	1001,15	1005,70	1012,38	979,43	1010,78
	σ_f	3,78	28,67	24,97	22,28	25,54	8,43	24,06

Tabela 21 - Resultados das simulações executadas para as instâncias do CVRP (Continuação)

		ACO	GA	SA	SA+GA	GA+SA	ACO+GA	ACO+SA
A-n38-k5	$f_{Mín}$	760,66	745,20	734,18	**733,95**	734,18	735,22	**733,95**
730	$f_{Méd}$	779,09	793,19	776,53	773,79	784,64	751,18	778,10
	σ_f	9,69	27,77	26,45	24,73	29,09	10,29	31,35
A-n39-k5	$f_{Mín}$	863,78	848,28	**829,68**	839,28	834,74	840,46	833,57
822	$f_{Méd}$	881,93	895,61	893,61	883,21	18976,56	856,00	891,80
	σ_f	7,94	28,23	42,32	31,65	127858,04	10,35	31,82
A-n39-k6	$f_{Mín}$	865,73	846,46	**835,25**	836,56	**835,25**	844,69	**835,25**
831	$f_{Méd}$	883,28	893,28	877,44	879,34	883,73	858,42	889,14
	σ_f	9,77	32,18	30,27	26,93	38,51	7,64	49,10
A-n44-k6	$f_{Mín}$	984,92	958,24	**943,83**	956,80	960,90	960,85	949,79
937	$f_{Méd}$	993,43	1027,67	1018,09	1009,52	1013,67	972,85	1013,35
	σ_f	3,90	38,16	29,85	27,09	30,59	4,44	33,37
A-n45-k6	$f_{Mín}$	1008,44	998,42	978,04	**956,80**	980,07	983,76	968,37
944	$f_{Méd}$	1033,30	1058,91	23641,99	1009,52	1040,58	1009,66	1044,14
	σ_f	5,95	35,18	159808,26	27,09	50,97	10,15	44,69
A-n45-k7	$f_{Mín}$	1251,05	1188,57	1204,36	1195,78	**1180,37**	1193,87	1188,44
1146	$f_{Méd}$	1278,41	1252,32	1267,01	1254,03	1268,61	1225,71	1257,60
	σ_f	9,61	32,65	38,08	33,78	41,36	16,79	37,85
A-n46-k7	$f_{Mín}$	1032,82	964,37	**923,28**	926,17	923,47	965,04	951,46
914	$f_{Méd}$	1054,95	1044,06	1038,08	1039,58	1062,89	1004,49	1040,80
	σ_f	8,24	37,75	53,44	35,49	78,93	18,13	45,24
A-n48-k7	$f_{Mín}$	1161,44	1131,97	1127,86	1123,47	**1112,65**	1125,11	1166,16
1073	$f_{Méd}$	1197,87	1203,66	1194,72	1187,04	1199,32	1155,93	1179,10
	σ_f	12,54	29,31	53,41	33,42	45,02	15,08	18,30
A-n53-k7	$f_{Mín}$	1095,02	1042,70	1077,52	**1056,62**	1077,68	1059,48	1143,52
1010	$f_{Méd}$	1139,25	1130,56	1151,80	1139,59	1155,45	1106,26	1185,45
	σ_f	12,08	39,14	59,10	36,03	45,95	12,50	59,30
A-n54-k7	$f_{Mín}$	1274,00	1228,30	**1190,73**	1205,66	1215,19	1221,95	1318,20
1167	$f_{Méd}$	1303,62	1289,80	1312,92	1303,38	1313,43	1265,91	1336,83
	σ_f	7,71	32,39	48,89	38,65	42,72	17,14	26,36
A-n55-k9	$f_{Mín}$	1138,82	1103,60	1091,00	1112,65	1119,67	**1087,37**	1212,12
1073	$f_{Méd}$	1148,70	1149,72	1189,24	1154,29	1190,65	1103,96	1282,72
	σ_f	4,10	29,48	43,99	28,19	45,29	7,10	29,05
A-n60-k9	$f_{Mín}$	1460,56	1450,61	1460,06	1430,03	1452,75	**1420,93**	1518,39
1354	$f_{Méd}$	1499,25	1492,06	1560,11	1499,10	1567,69	1451,20	1541,07
	σ_f	10,82	25,45	54,13	36,52	46,99	16,03	14,09
A-n61-k9	$f_{Mín}$	1136,40	**1080,81**	1103,73	1119,09	1093,92	1093,86	1250,47
1034	$f_{Méd}$	1157,68	1143,92	1218,78	1174,85	1211,69	1125,50	1274,05
	σ_f	7,00	29,62	58,43	28,99	60,05	15,07	12,61
A-n62-k8	$f_{Mín}$	1363,02	**1345,42**	1432,81	1364,58	1379,26	1345,88	1552,11
1288	$f_{Méd}$	1369,72	1414,36	1542,00	1429,65	1543,99	1352,52	1568,01
	σ_f	5,15	33,23	76,95	36,49	66,13	4,24	3,20
A-n63-k9	$f_{Mín}$	1781,22	**1697,07**	1741,04	1712,23	1749,58	1706,07	1749,94
1616	$f_{Méd}$	1816,66	1835,34	1830,05	1815,64	1837,47	1750,49	1859,14
	σ_f	15,82	56,04	57,21	46,66	44,37	25,94	79,03
A-n63-k10	$f_{Mín}$	1410,13	1382,76	1413,04	1392,43	1368,74	**1362,88**	1402,07
1314	$f_{Méd}$	1415,91	1444,24	1505,87	1462,35	1511,44	1387,43	1518,97
	σ_f	2,86	30,80	65,81	33,25	52,42	9,46	75,52
A-n64-k9	$f_{Mín}$	1540,44	1497,06	1488,36	1474,57	1486,15	1481,19	**1472,71**
1401	$f_{Méd}$	1557,51	1567,13	1584,77	1559,93	1565,15	1507,59	1557,42
	σ_f	8,63	32,77	83,49	32,37	38,38	13,94	42,38
A-n65-k9	$f_{Mín}$	1258,42	1225,06	1268,20	1244,57	1243,72	**1211,97**	1216,17
1174	$f_{Méd}$	1273,04	1314,66	1407,72	1328,10	1403,12	1241,29	1409,92
	σ_f	6,02	50,11	89,62	37,30	77,96	11,19	88,98
A-n69-k9	$f_{Mín}$	1273,61	1239,41	1243,13	1245,54	1268,42	**1227,52**	1242,92
1159	$f_{Méd}$	1290,65	1298,72	1370,20	1311,36	1376,50	1252,81	1382,28
	σ_f	9,73	41,38	75,91	42,45	61,57	13,71	88,53
A-n80-k10	$f_{Mín}$	1897,13	1877,90	1994,95	1959,56	1970,87	**1857,31**	1987,45
1763	$f_{Méd}$	1923,52	1980,97	2101,15	2023,97	2091,36	1891,85	2097,49
	σ_f	7,27	45,44	60,02	39,82	65,87	13,09	54,66

De acordo com análise feita das simulações, os métodos desenvolvidos se aproximaram da solução ótima para a maioria das instâncias. A eficiência dos métodos é comparada na Tabela 22 com percentuais do valor encontrado em relação ao valor ótimo do problema.

Analisando a Tabela 22 fica constatado que todos os métodos tiveram uma eficiência média acima de 93%. Em apenas 12 casos algum dos métodos não conseguiu atingir 90% da eficiência de um problemas, sendo 4 destes para a instância mais robusta da biblioteca, com o maior número de vértices e de veículos necessários. A hibridização com a maior eficiência média foi a ACO+GA com 96,66%, em seguida SA+GA com 96,57% e GA+SA com 96,36%. Comprovando que os métodos híbridos superam os resultados dos métodos individuais. A hibridização ACO+SA obteve entre os métodos a melhor solução encontrada para 5 instâncias, porém sua média de eficiência a deixou em penúltimo lugar do ranking dos métodos, atingindo por sete vezes valores de eficiência inferiores a 90%.

Tabela 22 - Comparação da eficiência* dos resultados para o CVRP

	ACO	GA	SA	SA+GA	GA+SA	ACO+GA	ACO+SA
A-n32-k5	94,25%	97,21%	97,83%	97,60%	97,44%	97,37%	**98,31%**
A-n33-k5	95,49%	98,95%	**99,79%**	99,43%	**99,79%**	98,95%	99,25%
A-n33-k6	96,50%	99,70%	**99,91%**	**99,91%**	**99,91%**	**99,91%**	99,87%
A-n34-k5	95,53%	**99,62%**	**99,62%**	**99,62%**	**99,62%**	**99,62%**	98,93%
A-n36-k5	92,90%	97,35%	97,97%	97,48%	**98,68%**	98,14%	**98,68%**
A-n37-k5	91,64%	95,22%	99,44%	**99,48%**	**99,48%**	94,80%	98,93%
A-n37-k6	95,49%	99,18%	99,16%	99,18%	**99,41%**	98,97%	97,53%
A-n38-k5	95,97%	97,96%	99,43%	**99,46%**	99,43%	99,29%	**99,46%**
A-n39-k5	95,16%	96,90%	**99,07%**	97,94%	98,47%	97,80%	98,61%
A-n39-k6	95,99%	98,17%	**99,49%**	99,34%	**99,49%**	98,38%	**99,49%**
A-n44-k6	95,13%	97,78%	**99,28%**	97,93%	97,51%	97,52%	98,65%
A-n45-k6	93,61%	94,55%	96,52%	**98,66%**	96,32%	95,96%	97,48%
A-n45-k7	91,60%	96,42%	95,15%	95,84%	**97,09%**	95,99%	96,43%
A-n46-k7	88,50%	94,78%	98,99%	98,69%	**98,97%**	94,71%	96,06%
A-n48-k7	92,39%	94,79%	95,14%	95,51%	**96,44%**	95,37%	92,01%
A-n53-k7	92,24%	**96,86%**	93,73%	95,59%	93,72%	95,33%	88,32%
A-n54-k7	91,60%	95,01%	**98,01%**	96,79%	96,03%	95,50%	88,53%
A-n55-k9	94,22%	97,23%	98,35%	96,44%	95,83%	**98,68%**	88,52%
A-n60-k9	92,70%	93,34%	92,74%	94,68%	93,20%	**95,29%**	89,17%
A-n61-k9	90,99%	**95,67%**	93,68%	92,40%	94,52%	94,53%	82,69%
A-n62-k8	94,50%	**95,73%**	89,89%	94,39%	93,38%	95,70%	82,98%
A-n63-k9	90,72%	**95,22%**	92,82%	94,38%	92,37%	94,72%	92,35%
A-n63-k10	93,18%	95,03%	92,99%	94,37%	96,00%	**96,41%**	93,72%
A-n64-k9	90,95%	93,58%	94,13%	95,01%	94,27%	94,59%	**95,13%**
A-n65-k9	93,29%	95,83%	92,57%	94,33%	94,39%	**96,87%**	96,53%
A-n69-k9	91,00%	93,51%	93,23%	93,05%	91,37%	**94,42%**	93,25%
A-n80-k10	92,93%	93,88%	88,37%	89,97%	89,45%	**94,92%**	88,71%
Eficiência Média	93,28%	96,28%	96,20%	96,57%	96,39%	**96,66%**	94,43%

Fonte: Autor

* Eficiência calculada com base na Equação (36).

As hibridizações foram organizadas para realização de comparações com os algoritmos desenvolvidos na literatura. Para comparação foram buscados apenas métodos híbridos e foram selecionados: *Hybrid Heuristic Algorithm* (HHA)

(ABDELAZIZ; EL-GHAREEB; KSASY, 2014), *Centroid-based 3-phase* (SHIN; HAN, 2011), *Sweep + Cluster Adjust* (SHIN; HAN, 2011), *Sweep Nearest* (NA; JUN; KIM, 2011), *Proposed Adaptive Sweep + VTPSO* (AKHAND; JANNAT; MURASE, 2018). Os resultados utilizados para comparação estão organizados na Tabela 23.

Para as 27 instâncias avaliadas pelos métodos, em apenas uma, um dos híbridos propostos nesse trabalho não ficou com o melhor resultado entre os comparados. A instância *A-n69-k9* teve o valor mais baixo encontrado pelo método *Sweep Nearest*, com uma eficiência de 94,61%, com o algoritmo ACO+GA apenas 0,23% menos eficiente.

O método SA+GA teve a melhor eficiência para 7 instâncias das 27, o GA+SA em 10, o ACO+GA em 10 e o ACO+SA em 7, com destaque para o ACO+GA que conseguiu atingir as menores soluções para os problemas mais robustos.

Tabela 23 - Comparação de resultados do CVRP com a literatura

	SA+GA*	GA+SA*	ACO+GA*	ACO+SA*	HHA	Centroid-based 3-phase	Sweep + Cluster Adjust	Sweep Nearest	Adaptive Sweep + VTPSO
A-n32-k5	803	805	805	**797**	1012	881	872	853	882
A-n33-k5	665	**662**	668	666	847	728	788	702	698
A-n33-k6	**743**	**743**	**743**	743	919	770	829	767	751
A-n34-k5	**781**	**781**	**781**	786	933	812	852	803	785
A-n36-k5	820	**810**	814	**810**	1126	814	884	840	881
A-n37-k5	**673**	**673**	706	676	876	756	734	797	754
A-n37-k6	957	**955**	959	973	1180	1027	1050	966	1112
A-n38-k5	**734**	734	735	**734**	920	819	874	801	813
A-n39-k5	839	835	840	**834**	1147	864	971	886	877
A-n39-k6	837	**835**	845	**835**	1065	881	966	-	972
A-n44-k6	957	961	961	**950**	1356	1037	1092	1020	1056
A-n45-k6	**957**	980	984	968	1210	1040	1043	991	1073
A-n45-k7	1196	**1180**	1194	1188	1361	1288	1281	1235	1305
A-n46-k7	926	**923**	965	951	1071	992	1013	1022	975
A-n48-k7	1123	**1113**	1125	1166	1292	1145	1143	1181	1152
A-n53-k7	**1057**	1078	1059	1144	1261	1117	1116	-	1090
A-n54-k7	**1206**	1215	1222	1318	1414	1209	1320	-	1361
A-n55-k9	1113	1120	**1087**	1212	1317	1155	1192	1134	1190
A-n60-k9	1430	1453	**1421**	1518	1733	1430	1574	1446	1503
A-n61-k9	1119	1094	**1094**	1250	1285	1201	1184	1158	1164
A-n62-k8	1365	1379	**1346**	1552	1604	1470	1559	1392	1408
A-n63-k9	1712	1750	**1706**	1750	2001	1766	1823	1763	1823
A-n63-k10	1392	1369	**1363**	1402	1542	1405	1523	1475	1477
A-n64-k9	1475	1486	1481	**1473**	1821	1587	1597	1586	1598
A-n65-k9	1245	1244	**1212**	1216	1429	1276	1351	1299	1317
A-n69-k9	1246	1268	1228	1243	1333	1283	1254	**1225**	1259
A-n80-k10	1960	1971	**1857**	1987	2318	1883	2014	1896	2136

*Algoritmos desenvolvidos nesta pesquisa

Fonte: Autor

O método ACO+SA foi a hibridização com os menores valores comparando apenas entre os métodos desenvolvidos neste trabalho, no entanto, é possível observar que a eficiência dos métodos propostos superam os valores encontrados pelas hibridizações da literatura utilizados para comparação.

6. CONSIDERAÇÕES FINAIS

Neste trabalho foram descritos os problemas de *benchmark* como Funções, Instâncias do Caixeiro Viajante e do Problema de Roteamento de Veículos Capacitado, alguns mostrando difíceis soluções por algoritmos exatos, requerendo tempos inviáveis para solução.

Foram criados bancos de problemas de *benchmark* de modo que o ambiente *web* possa acessá-los e resolvê-los. Os bancos estão estruturados em arquivos de texto e arquivos de instâncias.

Foram desenvolvidas três meta-heurísticas com diferentes versões de cada, adequadas para cada problema. Neste trabalho foram aplicadas técnicas de hibridização dessas meta-heurísticas utilizando a lógica de trabalho colaborativo retransmitido, onde um método é executado e seu resultado é passado como base para execução de um segundo método.

Para os problemas de Funções foram desenvolvidos as meta-heurísticas GA, SA junto das hibridizações GA+SA e SA+GA. Os métodos tiveram suas soluções codificadas na forma binária baseados na versão tradicional dos algoritmos. Após a realização de testes iniciais verificou-se a necessidade de aplicar a técnica de Elitismo no GA.

Foram realizados experimentos utilizando quatro funções de *benchmark* com diferentes números de dimensões. Os métodos alcançaram o melhor resultado para variações até 20 dimensões. Foi definido o número de 30 dimensões para comparar com problemas da literatura. Os métodos desenvolvidos se aproximam dos valores ótimos para os problemas, porém não superam todos os métodos utilizados para comparação. Foram reconhecidas necessidades nos ajustes de parâmetros do algoritmo Recozimento Simulado, buscando melhor adaptação do método em meio as hibridizações.

Foram desenvolvidos os métodos GA, SA e ACO, além das hibridizações ACO+GA, ACO+SA, GA+SA e SA+GA para os problemas de instâncias, tanto TSP quanto CVRP. Foram realizadas adaptações nos métodos propostos para melhoria de seus desempenhos, para o GA foram implementadas as técnicas Eliminação Cruzada, LKH, e *Nearest Neighbor*, sendo que, esses dois últimos foram aplicados ao SA.

Os métodos foram executados para o TSP utilizando sete instâncias de pequeno à grande porte (42 a 299 vértices). Para comparação foram utilizados dois trabalhos da literatura, onde o primeiro aborda duas instâncias em comum e possuem seis métodos para resolver o problema, nesta comparação a hibridização utilizada foi a ACO+GA, que superou os métodos desenvolvidos na literatura.

Um segundo trabalho desenvolve três métodos, sendo um deles uma hibridização entre os métodos Algoritmo Genético e Colônia de Formigas, utilizando seis instâncias em comum para execução dos métodos. As comparações mostraram que o método ACO+GA, assim como os demais desenvolvidos neste trabalho, possuem uma dificuldade em aproximar-se dos valores ótimos para as instâncias de grande porte. Desta forma, para as instâncias de pequeno e médio porte os métodos superam ou igualam aos resultados da literatura, já para as de grande porte não atingem valores satisfatórios.

Para execução dos métodos para o CVRP foram utilizadas todas as instâncias da biblioteca A-VRP (27 instâncias; 32 a 80 vértices). Todos os métodos obtiveram uma eficiência média acima de 93% na solução dos problemas. Foram utilizadas todas as hibridizações para comparação com cinco métodos da literatura aplicados a todos os problemas da biblioteca. Em apenas um dos problemas um dos métodos da literatura obteve uma melhor eficiência em relação aos métodos desenvolvidos neste trabalho, com uma diferença da solução de apenas 0,24%. Para as demais execuções as hibridizações desenvolvidas superam os métodos da literatura.

Para controle das execuções, assim como o controle do acionamento dos métodos e configurações dos parâmetros, foi desenvolvido o *framework* do ambiente *web*. Para seu desenvolvimento foram necessários estudos sobre a implementação de páginas *web*. Esse processo agregou dificuldades à pesquisa por exigir estudo de conceitos relacionados às técnicas de desenvolvimento e estruturação das informações na *web*.

Para disponibilização do ambiente desenvolvido foi requerido um servidor para hospedagem e disponibilização de acesso *online*. Esse servidor foi cedido pela universidade, o processo de hospedagem está sendo viabilizado e aguarda-se uma melhoria dos equipamentos utilizados.

Foi investigada a abordagem de hibridização de métodos meta-heurísticos por meio da lógica de trabalho colaborativo retransmitido. Essa lógica foi aplicada na organização em série das meta-heurísticas, possibilitando que um método seja executado tendo como base um resultado obtido por outro método. As hibridizações são uma tendência na solução de problemas pela união de conceitos, agregando as melhores heurísticas de cada algoritmo em uma só.

As hibridizações possibilitaram quatro combinações de métodos SA+GA, GA+SA, ACO+GA e ACO+SA. Sendo que para o problema de Funções apenas os dois primeiros foram aplicados.

Os resultados experimentais para os problemas de instâncias de *benchmark* revelaram que a hibridização tem, em geral, uma maior eficiência quando comparadas as meta-heurísticas individuais. Apresentando expectativas condizentes com as iniciais da pesquisa.

Os algoritmos necessitam de um processo de calibragem de parâmetros mais aprofundado. Aplicando técnicas inteligentes para as configurações dos parâmetros de cada algoritmo para resolução de cada problema. São necessários ainda, estudos sobre a adaptação dessas configurações para momentos intermediários da execução do algoritmo, como modificações da configuração em tempo de execução.

As configurações dos parâmetros para hibridizações necessitam de um empenho adicional, para além de analisar a viabilidade das configurações para cada caso, fazer análises de desempenho das combinações de métodos como um todo.

Os métodos necessitam da aplicação de critérios de parada baseados na evolução do resultado. Técnicas que vão além de definir limites para iterações, como análise de estagnação dos resultados durante um determinado número de iterações faz com que o algoritmo pare. Ou ainda, um determinado número de avaliações da função objetivo faz com que o algoritmo encerre. Essas são possibilidades de medidas para que os algoritmos tenham uma evolução descartando situações tendenciosas, como para o ACO, em que as soluções são redirecionadas para soluções boas precocemente, e o algoritmo perde a capacidade de encontrar soluções melhores.

A plataforma ainda necessita de mais técnicas, proporcionando aos usuários um maior número de opções e combinações para resolver os problemas. Abordagens determinísticas foram desenvolvidas em trabalhos de iniciação científica, executados em paralelos a esta pesquisa. Estes trabalhos estão em andamento, possuem resultados parciais apresentados na JCECEC 2018 (http://sites.pucgoias.edu.br/eventos/jcecec/trabalhos-aceitos-6/) e em processo de submissão no CNMAC 2019, e ao final, terá seu conteúdo disponibilizado no ambiente *web*. Ainda existem meta-heurísticas sendo desenvolvidas dentre os planos, sendo o *Particle Swarm Optimization* (PSO) e o *Firefly Algorithm* (FA).

Trabalhos futuros envolvendo a hibridização de meta-heurísticas podem abordar diferentes técnicas de combinação dos métodos, seguindo a lógica de trabalho em equipe colaborativo, integrativo, e composição de meta-heurísticas, como na explanação (RODRIGUEZ; GARCÍA-MARTINEZ; LOZANO, 2012). Trabalhos envolvendo hibridização de algoritmos de inteligência coletiva e processamento paralelo são opções promissoras para resolução de problemas de otimização combinatória.

Sugere-se o uso de técnicas de aprendizado de máquina para a calibragem das configurações de parâmetros para os métodos. Uma das possíveis técnicas, com relevância na atualidade é a calibragem *online* das configurações.

A principal melhoria reconhecida a ser desenvolvidas para o ambiente *web* é a possibilidade de um usuário modelar seu próprio problema por meio de uma interface gráfica simplificada. Criando um módulo de resolução de problemas reais. Outra implementação que pode agregar conhecimento ao ambiente, é a análise estatística dos métodos executados pelo ambiente.

6.1. Desdobramentos do Trabalho

- O autor ministrou o curso de programação na linguagem Python;
- Coorientação de dois planos de Iniciação Científica de alunos da Ciência e Engenharia da Computação da Pontifícia Universidade Católica de Goiás, com relatórios parciais de pesquisa disponíveis em:
    - http://www.pucgoias.edu.br/prope/bolsa/relatorios/arqs2019/repar_14469.pdf
    - http://www.pucgoias.edu.br/prope/bolsa/relatorios/arqs2019/repar_14475.pdf
- Trabalho publicado em congresso (ANCIOTO JUNIOR *et al.*, 2018)

REFERÊNCIAS

ABDELAZIZ, M. M.; EL-GHAREEB, H. A.; KSASY, M. S. M. Hybrid Heuristic Algorithm for solving Capacitated Vehicle Routing problem. **INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY**, v. 12, n. 9, p. 3844–3851, 2014.

ABELEDO, H. et al. The time dependent traveling salesman problem: Polyhedra and algorithm. **Mathematical Programming Computation**, v. 5, n. 1, p. 27–55, 2013.

ABREU, A. A. A. M. DE; OLIVEIRA, S. L. G. DE; LACERDA, W. S. Uma resolução do problema do caixeiro-viajante por mapa auto-organizável com aprendizado winner takes all. **Revista Brasileira de Computação Aplicada**, v. 7, p. 100–109, 2015.

ADEWUMI, A. O.; ADELEKE, O. J. A survey of recent advances in vehicle routing problems. **International Journal of System Assurance Engineering and Management**, p. 1–18, 2016.

AKHAND, M. A. H.; JANNAT, Z.; MURASE, K. Capacitated Vehicle Routing Problem Solving using Adaptive Sweep and Velocity Tentative PSO. **International Journal of Advanced Computer Science and Applications**, v. 8, n. 12, 2018.

ALJANABY, A.; KU-MAHAMUD, K. R.; NORWAWI, N. M. An exploration technique for the interacted multiple ant colonies optimization framework. **ISMS 2010 - UKSim/AMSS 1st International Conference on Intelligent Systems, Modelling and Simulation**, n. December 2011, p. 92–95, 2010.

ALMEIDA, G. H. M. et al. Traffic flow management in a real mobile phone network using linear optimization. **IEEE LATIN AMERICA TRANSACTIONS**, v. 16, n. 2, p. 694–700, 2018.

ANCIOTO JUNIOR, E. M. et al. O PROBLEMA DO CAIXEIRO VIAJANTE APLICADO À ROTEIRIZAÇÃO DA MANUTENÇÃO PREDIAL DO PODER JUDICIÁRIO DO ESTADO DE GOIÁS. **XXV SIMPÓSIO DE ENGENHARIA DE PRODUÇÃO**, p. 1–10, 2018.

ARAUJO, H. A. DE. **ALGORITMO SIMULATED ANNEALING: UMA NOVA ABORDAGEM**. [s.l.] Universidade Federal de Santa Catarina como, 2001.

ARDEH, M. A. **BenchmarkFcns**. Disponível em: <http://benchmarkfcns.xyz/>.

AROUXÉT, M. B.; ECHEBEST, N.; PILOTTA, E. A. Active-set strategy in Powell's method for optimization without derivatives. **Computational & Applied Mathematics**, v. 30, n. 1, p. 171–196, 2011.

AUDET, C.; DANG, K. C.; ORBAN, D. Optimization of algorithms with OPAL. **Mathematical Programming Computation**, v. 6, n. 3, p. 233–254, 2014.

AUGERAT, P. et al. Computational results with a branch and cut code for the capacitated vehicle routing problem. **Technical Report RR 949-M**, n. January, 1995.

BAE, J.; RATHINAM, S. Approximation algorithms for multiple terminal, Hamiltonian path problems. **Optimization Letters**, v. 6, n. 1, p. 69–85, 2012.

BALDACCI, R.; BATTARRA, M.; VIGO, D. Routing a Heterogeneous Fleet of Vehicles. **Operations Research/ Computer Science Interfaces Series**, v. 43, n. January, p. 1–26, 2007.

BERBEGLIA, G. et al. Rejoinder on: Static pickup and delivery problems: a classification scheme and survey. **Top**, v. 15, n. 1, p. 45–47, 2007.

BLANCO, M. et al. Application of the Davidon-Fletcher-Powell algorithm to the resolution of multicomponent mixtures using UV-vis spectrophotometry. **Analytica Chimica Acta**, v. 327, n. 2, p. 145–152, 1996.

BLUM, C. et al. Hybrid metaheuristics in combinatorial optimization: A survey. **Applied Soft Computing**, v. 11, p. 1–17, 2011.

BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. **ACM Computing Surveys**, v. 35, n. 3, p. 189–213, 2003.

BODIN, L. D. **Routing and scheduling of vehicles and crews - The state of the art**. [s.l.] Pergamon Press, 1983. v. 10

BRAEKERS, K.; RAMAEKERS, K.; VAN NIEUWENHUYSE, I. The vehicle routing problem: State of the art classification and review. **Computers and Industrial Engineering**, v. 99, p. 300–313, 2016.

CABALLERO-MORALES, S.-O.; MARTINEZ-FLORES, J.-L.; SANCHEZ-PARTIDA, D. Dynamic Reduction-Expansion Operator to Improve Performance of Genetic Algorithms for the Traveling Salesman Problem. v. 2018, 2018.

CHATTERJEE, S.; CARRERA, C.; LYNCH, L. A. Genetic algorithms and traveling salesman problems. **European Journal of Operational Research**, v. 93, n. 3, p. 490–510, 1996.

CHE, J. et al. Application of hybrid artificial fish swarm algorithm based on similar fragments in VRP. **MIPPR 2017: Remote Sensing Image Processing, Geographic Information Systems, and Other Applications**, v. 10611, n. March, p. 43, 2018.

CHEN, R. M.; CHEN, Y. A. Heuristics based particle swarm optimization for solving vehicle routing problems. **Proceedings - 2014 International Symposium on Computer, Consumer and Control, IS3C 2014**, n. 2, p. 360–363, 2014.

CHEN, S. M.; CHIEN, C. Y. Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. **Expert Systems with Applications**, v. 38, n. 12, p. 14439–14450, 2011.

CHRISTOFIDES, N.; EILON, S. An Algorithm for the Vehicle-dispatching Problem. **Journal of the Operational Research Society**, v. 20, n. 3, p. 309–318, 1969.

CHRISTOFIDES, N.; MINGOZZI, A.; TOTH, P. The vehicle routing problem. **Combinatorial Optimization**, p. 315–338, 1979.

CORDEAU, J. F. et al. A guide to vehicle routing heuristics. **Journal of the Operational Research Society**, v. 53, n. 5, p. 512–522, 2002.

COSTA, P. R. DE O. DA et al. A Genetic Algorithm for a Green Vehicle Routing Problem. **Electronic Notes in Discrete Mathematics**, v. 64, p. 65–74, 2018.

COTTA, C.; TALBI, E.-G.; ALBA, E. Parallel Hybrid Metaheuristics - A New Class of Algorithms. In: 1 (Ed.). . **A New Class of Algorithms**. 1. ed. [s.l.] 1, 2005. p. 347–370.

CUNHA, C. B.; BONASSER, U. O.; ABRAHÃO, F. T. M. Experimentos computacionais com heurísticas de melhorias para o problema do caixeiro viajante. **Anais do XVI ANPET - Congresso da Associação Nacional de Pesquisa e Ensino em**

Transportes, n. Xvi, p. 105–117, 2002.

DANTZIG, G. B. et al. Solution of a large-scale traveling-salesman problem. **Journal of the Operations Research Society of America**, p. 393–410, 1954.

DANTZIG, G. B.; RAMSER, J. H. The Truck Dispatching Problem. **Management Science**, v. 6, n. 1, p. 80–91, 1959.

DEB, K. Multi-objective Optimisation Using Evolutionary Algorithms: An Introduction. In: - (Ed.). . **Multi-objective Evolutionary Optimisation for Product Design and Manufacturing**. [s.l.] -, 2011. p. 3–34.

DORIGO, M.; BLUM, C. Ant colony optimization theory: A survey. **Theoretical Computer Science**, v. 344, n. 2–3, p. 243–278, 2005.

DORIGO, M.; STÜTZLE, T. **Ant Colony Optimization**. Cambridge, Massachusetts: The MIT Press, 2004.

DUMITRESCU, I.; STUTZLE, T. A survey of methods that combine local search and exact algorithms. **Technical Report AIDA-03-07, FG Intellektik, FB Informatik, TU Darmstadt, Germany**, n. i, 2003.

EGEA, J. A. et al. MEIGO: An open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics. **BMC Bioinformatics**, v. 15, n. 1, p. 1–12, 2014.

EL-ABD, M.; KAMEL, M. A taxonomy of cooperative search algorithms. **Hybrid Metaheuristics**, v. 3636, p. 32–41, 2005.

ERTENLICE, O.; KALAYCI, C. B. A survey of swarm intelligence for portfolio optimization: Algorithms and applications. **Swarm and Evolutionary Computation**, v. 39, n. January, p. 36–52, 2018.

ETMINANIESFAHANI, A.; GHANBARZADEH, A.; MARASHI, Z. Fibonacci indicator algorithm: A novel tool for complex optimization problems. **Engineering Applications of Artificial Intelligence**, v. 74, n. May 2017, p. 1–9, 2018.

EVANGELISTA, P.; MAIA, P.; ROCHA, M. Implementing metaheuristic optimization algorithms with JECoLi. **ISDA 2009 - 9th International Conference on Intelligent Systems Design and Applications**, p. 505–510, 2009.

FARIS, H. et al. EvoloPy: An Open-source Nature-inspired Optimization Framework in Python. **Proceedings of the 8th International Joint Conference on Computational Intelligence**, v. 1, n. Ijcci, p. 171–177, 2016.

FISHER, M. L. Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees. **Operations Research**, v. 42, n. 4, p. 626–642, 1994.

FISHER, M. L.; JAIKUMAR, R. A generalized assignment heuristic for vehicle routing. **Networks**, v. 11, n. 2, p. 109–124, 1981.

GAERTNER, D.; CLARK, K. On Optimal Parameters for Ant Colony Optimization algorithms TSP classifications. **Ic-Ai**, p. 83–89, 2005.

GENDREAU, M.; TARANTILIS, C. Solving large-scale vehicle routing problems with time windows: The state-of-the-art. **Cirrelt**, n. 4, p. 1–45, 2010.

GOUVÊA, É. J. C. **MÉTODOS CONVERGENTES DE OTIMIZAÇÃO GLOBAL**

BASEADOS NO VETOR Q-GRADIENTE. [s.l.] Instituto Nacional de Pesquisas Espaciais - INPE, 2016.

GUNES, E.; CORDEAU, J.; LAPORTE, G. The Attractive Traveling Salesman Problem. **European Journal of Operational Research**, v. 203, p. 59–69, 2010.

HASSANAT, A. B. A. et al. Enhancing Genetic Algorithms using Multi Mutations : Experimental Results on the Travelling Salesman Problem. **International Journal of Computer Science and Information Security**, v. 14, n. 7, p. 785–801, 2016.

HERNÁNDEZ, S.; FLORES, I.; VÁZQUEZ, J. A. Improved golden-Section algorithm for the multi-Item replenishment problem. **Journal of Applied Research and Technology**, v. 10, n. 3, p. 388–397, 2012.

HESTENES, M. R. et al. Computational Schemes of the Davidon-Fletcher-Powell Method in Infinite-Dimensional Space 1. **JOURNAL OF OPTIMIZATION THEORY AND APPLICATIONS**, v. 12, n. 5, p. 447–458, 1973.

HLAING, Z. et al. An Ant Colony Optimization Algorithm for Solving Travelling Salesman Problem. **International Journal of Scientific and Research Publications**, v. 16, n. 8, p. 54–59, 2011.

HOLLAND, J. H. **Adaptation in Natural and Artificial Systems**. second edi ed. Ann Arbor, MI: University of Michigan Press, 1975.

IBRAHIM, M. A. H. BIN; MAMAT, M.; JUNE, L. W. BFGS method: A new search direction. **Sains Malaysiana**, v. 43, n. 10, p. 1591–1597, 2014.

ISWARI, T.; ASIH, A. M. S. Comparing genetic algorithm and particle swarm optimization for solving capacitated vehicle routing problem. **IOP Conference Series: Materials Science and Engineering**, v. 337, n. 1, 2018.

JOURDAN, L.; BASSEUR, M.; TALBI, E. G. Hybridizing exact methods and metaheuristics: A taxonomy. **European Journal of Operational Research**, v. 199, n. 3, p. 620–629, 2009.

KE, L.; ZHANG, Q.; BATTITI, R. Hybridization of decomposition and local search for multiobjective optimization. **IEEE Transactions on Cybernetics**, v. 44, n. 10, p. 1808–1820, 2014.

KIRKPATRICK, S.; GELATT, C. D.; VECCHI, M. P. Optimization by Simulated Annealing. **Science**, v. 220, n. 4598, p. 671–680, 1983.

KO, O.; SAYAMA, H.; TAKAMATSU, T. AN EXTENSION OF THE DAVIDON-FLETCHER-POWELL METHOD TO SENSITIVITY ANALYSIS ’. **Annual Meetinf of the Soc. of Chem. Engrs**, v. 5, n. 4, p. 413–417, 1972.

KOÇ, Ç.; KARAOGLAN, I. The green vehicle routing problem: A heuristic based exact solution approach. **Applied Soft Computing Journal**, v. 39, p. 154–164, 2016.

KOUBEK, F.; LINSER, J. Differential evolution algorithm with local search for capacitated vehicle routing problem. **Neonatal Care: New Research**, n. January, 2013.

KOZA, J. R. Introduction to Genetic Algorithms. In: **Advances in Genetic Programming**. [s.l.] MIT Press Cambridge, 1994. p. 518.

KURODA, M. et al. Development of a novel crossover of hybrid genetic algorithms for

large-scale traveling salesman problems. **Artificial Life and Robotics**, v. 15, n. 4, p. 547–550, 2010.

LAPORTE, G. Fifty Years of Vehicle Routing. **Transportation Science**, v. 43, n. 4, p. 408–416, 2009.

LI, X. A Non-dominated Sorting Particle Swarm Optimizer for Multiobjective Optimization. **Genetic and Evolutionary Computation---GECCO 2003. Proceedings, Part I**, p. 37–48, 2003.

LI, Z. et al. Lévy-Flight Moth-Flame Algorithm for Function Optimization and Engineering Design Problems. **Mathematical Problems in Engineering**, v. 2016, p. 1–22, 2016.

LIEFOOGHE, A.; JOURDAN, L.; LEGRAND, T. ParadisEO-MOEO: A software framework for evolutionary multi-objective optimization. **Lecture Notes in Computer Science**, v. 4403, p. 386–400, 2007.

LIMA, S. J. DE A.; DE ARAÚJO, S. A.; SCHIMIT, P. H. T. A hybrid approach based on genetic algorithm and nearest neighbor heuristic for solving the capacitated vehicle routing problem. **Acta Scientiarum - Technology**, v. 40, p. 1–10, 2018.

LIN, Q. et al. A Hybrid Evolutionary Immune Algorithm for Multiobjective Optimization Problems. **IEEE Transactions on Evolutionary Computation**, n. c, p. 1–1, 2015.

LIN, S.; KERNIGHAN, B. W. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. **Operations Research**, v. 21, n. 2, p. 498–516, 1973.

LIU, J.; WU, X. New three-term conjugate gradient method for solving unconstrained optimization problems. **ScienceAsia**, v. 40, n. 4, p. 295–300, 2014.

LIU, Q.; SANG, R.; ZHANG, Q. FPGA-based acceleration of Davidon-Fletcher-Powell quasi-Newton optimization method. **Transactions of Tianjin University**, v. 22, n. 5, p. 381–387, 2016.

LO, K.-M. et al. A genetic algorithm with new local operators for multiple traveling salesman problems. **International Journal of Computational Intelligence Systems**, v. 11, n. 1, p. 692–705, 2018.

LUENBERGER, D. G.; YE, Y. **Linear and Nonlinear Programming**. Third ed. [s.l.] Springer, 2008.

LUO, Q.; YANG, X.; ZHOU, Y. Nature-inspired approach: An enhanced moth swarm algorithm for global optimization. **Mathematics and Computers in Simulation**, v. 159, p. 57–92, 2019.

LYSGAARD, J.; LETCHFORD, A. N.; EGLESE, R. W. A new branch-and-cut algorithm for the capacitated vehicle routing problem. **Mathematical Programming**, v. 100, n. 2, p. 423–445, 2004.

MAITY, S.; ROY, A.; MAITI, M. An intelligent hybrid algorithm for 4- dimensional TSP. **Journal of Industrial Information Integration**, v. 5, p. 39–50, 2017.

MARTINS, D. W. P. **Escalonamento Da Produção Para Sistema De Manufatura Job Shop Com Parâmetros Inteligentes**. [s.l.] Pontifícia Universidade Católica de Goiás, 2018.

MESSAOUD, E.; EL BOUZEKRI EL IDRISSI, A.; ALAOUI, A. E. The green dynamic

vehicle routing problem in sustainable transport. **2018 4th International Conference on Logistics Operations Management (GOL)**, n. November, p. 1–6, 2018.

MINGPRASERT, S.; MASUCHUN, R. Adaptive artificial bee colony algorithm for solving the capacitated vehicle routing problem. **2017 9th International Conference on Knowledge and Smart Technology: Crunching Information of Everything, KST 2017**, p. 23–27, 2017.

MIRJALILI, S.; HASHIM, S. Z. M. A new hybrid PSOGSA algorithm for function optimization. **Proceedings of ICCIA 2010 - 2010 International Conference on Computer and Information Application**, n. 1, p. 374–377, 2010.

MIRJALILI, S.; MIRJALILI, S. M.; LEWIS, A. Grey Wolf Optimizer. **Advances in Engineering Software**, v. 69, p. 46–61, 2014.

MOHAMED, A. A. A. et al. Optimal power flow using moth swarm algorithm. **Electric Power Systems Research**, v. 142, p. 190–206, 2017.

MOHAMMADI, S. et al. Comparison of golden section search method and imperialist competitive algorithm for optimization cut-off grade-case study: Mine No. 1 of Golgohar. **JME Journal of Mining & Environment**, v. 6, n. 1, p. 63–71, 2015.

MONTOYA-TORRES, J. R. et al. A literature review on the vehicle routing problem with multiple depots. **Computers and Industrial Engineering**, v. 79, p. 115–129, 2015.

MOURGAYA, M.; VANDERBECK, F. The periodic vehicle routing problem: Classification and heuristic for tactical planning. **RAIRO Operations Research**, v. 42, n. 3, p. 415–431, 2008.

NA, B.; JUN, Y.; KIM, B. I. Some extensions to the sweep algorithm. **International Journal of Advanced Manufacturing Technology**, v. 56, n. 9–12, p. 1057–1067, 2011.

NABIL, N.; FAROUK, H. A.; EL-KILANY, K. S. Green vehicle routing and scheduling problem with optimized travel speed. **IEEE International Conference on Industrial Engineering and Engineering Management**, v. 2017–Decem, p. 1057–1061, 2018.

NASIR, A. N. K.; TOKHI, M. O. Novel metaheuristic hybrid spiral-dynamic bacteria-chemotaxis algorithms for global optimisation. **Applied Soft Computing Journal**, v. 27, p. 357–375, 2015.

NEVES, F. DE A. DAS. **Programação com multi-objetivos aplicada à otimização do projeto de pontes estaiadas**. [s.l.] Universidade Federal do Rio de Janeiro, 1997.

NIU, Y. et al. Optimizing the green open vehicle routing problem with time windows by minimizing comprehensive routing cost. **Journal of Cleaner Production**, v. 171, p. 962–971, 2018.

NOCEDAL, J.; WRIGHT, S. J. **Numerical Optimization**. [s.l: s.n.]. v. 11

OSABA, E. et al. Good practice proposal for the implementation, presentation, and comparison of metaheuristics for solving routing problems. **Neurocomputing**, v. 271, n. 2017, p. 2–8, 2017.

OSABA, E. et al. A discrete water cycle algorithm for solving the symmetric and asymmetric traveling salesman problem. **Applied Soft Computing Journal**, v. 71, p. 277–290, 2018.

PACHECO, M. A.; FUKASAWA, R. Resolução do Problema do Entregador Viajante. **Inteligência Computacional Aplicada**, v. 4, p. 1–9, 2013.

PANG, C.-Y. et al. Applying Data Clustering Feature to Speed Up Ant Colony Optimization. **Abstract and Applied Analysis**, v. 2014, p. 1–8, 2014.

PAPA, J. P. et al. LibOPT: An Open-Source Platform for Fast Prototyping Soft Optimization Techniques. **Conference'17, July 2017, Washington, DC, USA**, p. 1–7, 2017.

PAPADIMITRIOU, C. H.; STEIGLITZ, K. **Combinatorial Optimization—Algorithms and Complexity**. Dover Edit ed. Mineola, New York: Dover Publications, Inc, 1998.

PEREZ, R. E.; JANSEN, P. W.; MARTINS, J. R. R. A. PyOpt: A Python-based object-oriented framework for nonlinear constrained optimization. **Structural and Multidisciplinary Optimization**, v. 45, n. 1, p. 101–118, 2012.

PETROWSKI, A. A clearing procedure as a niching method for genetic algorithms. **Proceedings of IEEE International Conference on Evolutionary Computation ICEC-96**, p. 798–803, 1996.

PILLAC, V. et al. A review of dynamic vehicle routing problems. **European Journal of Operational Research**, v. 225, n. 1, p. 1–11, 2013.

POLLARIS, H. et al. Vehicle routing problems with loading constraints: state-of-the-art and future directions. **OR Spectrum**, v. 37, n. 2, p. 297–330, 2015.

POLYAK, B. T. Iterative Methods Using Lagrange Multipliers Constraints of the Equation Type *. **October**, v. 10, n. 5, p. 1098–1106, 1970.

POLYAK, B. T. Newton's method and its use in optimization. **European Journal of Operational Research**, v. 181, n. 3, p. 1086–1096, 2007.

PUCHINGER, J.; RAIDL, G. R. Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification. **International Work-Conference on the Interplay Between Natural and Artificial Computation**, p. 41–53, 2005.

QIN, W.; ZHANG, J.; SONG, D. An improved ant colony algorithm for dynamic hybrid flow shop scheduling with uncertain processing time. **Journal of Intelligent Manufacturing**, v. 29, n. 4, p. 891–904, 2015.

RAIDL, G. R. A Unified View on Hybrid Metaheuristics. **European RTN ADONET**, p. 1–12, 2006.

RAIDL, G. R. Decomposition based hybrid metaheuristics. **European Journal of Operational Research**, v. 244, n. 1, p. 66–76, 2015.

RAIDL, G. R.; PUCHINGER, J. Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. **Hybrid Metaheuristics: An Emerging Approach to Optimization**, v. 62, n. 2008, p. 31–62, 2008.

REGO, C. et al. Traveling salesman problem heuristics : Leading methods , implementations and latest advances. **European Journal of Operational Research**, v. 211, p. 427–441, 2011.

REINELT, G. TSPLIB—A Traveling Salesman Problem Library. **ORSA Journal on Computing**, v. 3, n. 4, p. 376–384, 1991.

REITER, P.; GUTJAHR, W. J. Exact hybrid algorithms for solving a bi-objective vehicle routing problem. **Central European Journal of Operations Research**, v. 20, n. 1, p. 19–43, 2012.

RODOVALHO, L. F. F. et al. Hybrid approach of optimization applied to an inverse problem in the dynamic modeling of a three-floor structure. **Applied Soft Computing Journal**, v. 65, p. 412–427, 2018.

RODRIGUEZ-FDEZ, I. et al. STAC: A web platform for the comparison of algorithms using statistical tests. **Fuzzy Systems (FUZZ-IEEE), 2015 IEEE International Conference on**, p. 1–8, 2015.

RODRIGUEZ, F. J.; GARCÍA-MARTINEZ, C.; LOZANO, M. Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: Taxonomy, comparison, and synergy test. **IEEE Transactions on Evolutionary Computation**, v. 16, n. 6, p. 787–800, 2012.

ROSSUM, G. VAN; DRAKE, F. L. **The Python Library ReferencePython Software Foundation**. [s.l: s.n.].

ROSSUM, G. VAN; DRAKE, F. L. **Python TutorialPython Software Foundation**. [s.l: s.n.].

RUDOLPH, G. Canonical Genetic Algorithms. **IEEE transactions on Neural Networks**, v. 5, n. 1, p. 96–101, 1994.

SAMUCO, J. M. E. **Algoritmos de Otimização Contínua Univariada**. [s.l.] Universidade de Aveiro, 2014.

SHEN, R.; ZHENG, J.; LI, M. A hybrid development platform for evolutionary multi-objective optimization. **2015 IEEE Congress on Evolutionary Computation, CEC 2015 - Proceedings**, p. 1885–1892, 2015.

SHIN, K.; HAN, S. A CENTROID-BASED HEURISTIC ALGORITHM FOR THE CAPACITATED VEHICLE ROUTING PROBLEM. **Computing and Informatics**, v. 30, p. 721–732, 2011.

SILVA, E. E. DA. **Otimização de estruturas de concreto armado utilizando algoritmos genéticos**. [s.l.] Escola Politécnica da Universidade de São Paulo, 2001.

SILVA, G. A. N. DA et al. Algoritmos Heurísticos Construtivos Aplicados Ao Problema Do Caixeiro Viajante Para a Definição De Rotas Otimizadas. **Colloquium Exactarum**, v. 5, n. 2, p. 30–46, 2013a.

SILVA, I. R. M. et al. MO-MAHM: A parallel Multi-agent Architecture for Hybridization of Metaheuristics for multi-objective problems. **2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings**, p. 580–587, 2017.

SILVA, R. M. A. et al. A Python/C library for bound-constrained global optimization with continuous GRASP. **Optimization Letters**, v. 7, n. 5, p. 967–984, 2013b.

SIMAS, E. P. L.; GÓMEZ, A. T. Comparying a tabu search process. **ICINCO 2007 - International Conference on Informatics in Control, Automation and Robotics**, p. 77–84, 2007.

SOFI, A. Z. M. et al. Conjugate gradient and steepest descent approach on quasi-Newton search direction. **AIP Conference Proceedings**, v. 1605, n. March 2016, p. 1189–1194, 2014.

SOLOMON, M. M. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. **Operations Research**, v. 35, n. 2, p. 254–265, 1987.

SOUZA, M. P. DE. **Um estudo do Método de Gradientes Conjugados não Lineares**. [s.l.] Unicamp, 2017.

SUBASI, M.; YILDIRIM, N.; YILDIZ, B. An improvement on Fibonacci search method in optimization theory. **Applied Mathematics and Computation**, v. 147, n. 3, p. 893–901, 2004.

TALBI, E. G. A taxonomy of hybrid metaheuristics. **Journal of Heuristics**, 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**, v. 240, n. 1, p. 171–215, 2016.

TAM, V.; MA, K. T. Combining meta-heuristics to effectively solve the vehicle routing problems with time windows. **Artificial Intelligence Review**, v. 21, n. 2, p. 87–112, 2004.

TANG, L.; WANG, X. A hybrid multiobjective evolutionary algorithm for multiobjective optimization problems. **IEEE Transactions on Evolutionary Computation**, v. 17, n. 1, p. 20–45, 2013.

TIAN, Y. et al. Education Forum PlatEMO: A MATLAB Platform for Evolutionary Multi-Objective Optimization. **Ieee ComputatIonal IntellIgenCe magazIne**, n. november, p. 73–87, 2017.

TIRKOLAEE, E. B. et al. A Hybrid genetic algorithm for multi-trip green capacitated Arc routing problem in the scope of urban services. **Sustainability (Switzerland)**, v. 10, n. 5, 2018.

TSAI, C. H.; KOLIBAL, J.; LI, M. The golden section search algorithm for finding a good shape parameter for meshless collocation methods. **Engineering Analysis with Boundary Elements**, v. 34, n. 8, p. 738–746, 2010.

TSCHÖKE, S. et al. Solving the Traveling Salesman Problem with a Parallel Branch-and-Bound Algorithm on a 1024 Processor Network. **IEEE Journals and Magazines**, p. 182–189, 1995.

UCHOA, E. et al. New benchmark instances for the Capacitated Vehicle Routing Problem. **European Journal of Operational Research**, v. 257, n. 3, p. 845–858, 2016.

ÜÇOLUK, G. Genetic algorithm solution of the tsp avoiding special crossover and mutation. **Intelligent Automation and Soft Computing**, v. 8, n. 3, p. 265–272, 2002.

VAN DER VORST, H. A.; DEKKER, K. Conjugate gradient type methods and preconditioning. **Journal of Computational and Applied Mathematics**, v. 24, n. 1–2, p. 73–87, 1988.

VANDERPLAATS, G. N. **Numerical Techniques for engineering design**. 3º ed. New York: McGraw-Hill Book Company, 1999.

VASCONCELOS, E. S. **Contribuições dos Métodos Simplex e das Resoluções Gráficas à Aprendizagem da Algebra Linear no Ensino Médio**. [s.l.] Instituto de Matemática e Estatística, 2013.

VENKATARAMAN, P. Applied Optimization with MATLAB Programming. In: [s.l: s.n.]. p. 526.

WANG, Y.; ZHAI, W.; GENG, X. A plant growth simulation algorithm for dynamic vehicle scheduling problem with time window. **International Journal of Innovative Computing, Information and Control**, v. 14, n. 3, p. 947–958, 2018.

WANG, Z. et al. Uncertain multiobjective traveling salesman problem. **European Journal of Operational Research**, v. 241, n. 2, p. 478–489, 2015.

XIAO, Y. et al. Development of a fuel consumption optimization model for the capacitated vehicle routing problem. **Computers and Operations Research**, v. 39, n. 7, p. 1419–1431, 2012.

XIAO, Y.; KONAK, A. Green Vehicle Routing Problem with Time-Varying Traffic Congestion. **Proceedings of the 14th INFORMS Computing Society Conference**, p. 134–148, 2015.

XIAO, Y.; KONAK, A. A genetic algorithm with exact dynamic programming for the green vehicle routing & scheduling problem. **Journal of Cleaner Production**, v. 167, p. 1450–1463, 2018.

YANG, X.-S. New Metaheuristic Bat-Inspired Algorithm. **Nature inspired cooperative strategies for optimization**, p. 65–74, 2010.

ZHANG, S. et al. Hybrid Multi-Objective Genetic Algorithm for Multi-Objective Optimization Problems. **27th Chinese Control and Decision Conference**, p. 1970–1974, 2015.

ZHOU, A. et al. Multiobjective evolutionary algorithms: A survey of the state of the art. **Swarm and Evolutionary Computation**, v. 1, n. 1, p. 32–49, 2011.

ANEXO 1

NAME : P-n16-k8
COMMENT : (Augerat et al, No of trucks: 8, Optimal value: 450)
TYPE : CVRP
DIMENSION : 16
EDGE_WEIGHT_TYPE : EUC_2D
CAPACITY : 35
NODE_COORD_SECTION
1 30 40
2 37 52
3 49 49
4 52 64
5 31 62
6 52 33
7 42 41
8 52 41
9 57 58
10 62 42
11 42 57
12 27 68
13 43 67
14 58 48
15 58 27
16 37 69
DEMAND_SECTION
1 0
2 19
3 30
4 16
5 23
6 11
7 31
8 15
9 28
10 8
11 8
12 7
13 14
14 6
15 19
16 11
DEPOT_SECTION
1
-1
EOF