CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA CELSO SUCKOW DA FONSECA
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO DE PESQUISA
COORDENADORIA DE PESQUISA E ESTUDOS TECNOLÓGICOS

RELATÓRIO FINAL DE INICIAÇÃO CIENTÍFICA

A PROGRAMAÇÃO INTEIRA E SUA APLICAÇÃO AOS PROBLEMAS DO CAIXEIRO VIAJANTE E DA DESIGNAÇÃO

Edital PIBIC 2024

Aluno:

Caio Passos Torkst Ferreira

Ciência da Computação – 6º Período

Aluno: Com bolsa

Professor Orientador:

Helder Manoel Venceslau, D. Sc.

Rio de Janeiro, RJ - Brasil

Setembro / 2025

RESUMO

Este trabalho investiga a aplicação da Programação Linear Inteira (PLI) a dois problemas clássicos da otimização combinatória: o Problema do Caixeiro Viajante (TSP) e o Problema da Designação (PD). Foram implementadas formulações matemáticas, solvers comerciais e *open-source*, além de heurísticas construtivas e de melhoria, com experimentos conduzidos em Python e AMPL sobre instâncias reconhecidas da TSPLIB e OR-Library, seguindo um protocolo reprodutível. Além de evidências empíricas, o estudo destaca limitações do escopo, de natureza computacional e experimental, não refletindo diretamente as condições de uma modelagem aplicada.

Os resultados evidenciaram a relevância da PLI como ferramenta de modelagem, mas também suas limitações práticas diante do crescimento combinatório, que inviabiliza instâncias de maior porte em problemas difíceis. Nesses casos, heurísticas e solvers especializados, como LKH e Concorde, para o TSP, mostraram-se mais eficazes. No PD, por sua estrutura (unimodular), tanto os solvers genéricos quanto algoritmos específicos apresentaram bom desempenho, com destaque para o Algoritmo Húngaro, que se mostrou consideravelmente mais eficiente. Observou-se ainda a importância da qualidade da solução inicial em heurísticas construtivas, bem como a necessidade de mecanismos de perturbação para escapar de ótimos locais. Nesse sentido, os resultados reforçam a importância de alinhar fundamentos teóricos, computabilidade e requisitos práticos na escolha metodológica de modelagem e solução dos problemas.

SUMÁRIO

1. INTRODUÇÃO

A otimização combinatória reúne alguns dos problemas mais relevantes da pesquisa operacional e da ciência da computação, com aplicações em logística, transporte, manufatura, planejamento de produção e alocação de recursos. Nesse contexto, a Programação Linear Inteira (PLI) destaca-se como um arcabouço versátil para modelar e resolver problemas discretos, pois permite representar variáveis inteiras ou binárias que capturam restrições lógicas e objetivos complexos, frequentemente presentes em cenários reais.

Entre os problemas clássicos que servem como benchmarks teóricos e casos aplicados de grande impacto estão o Problema do Caixeiro Viajante (TSP, *Travelling Salesman Problem*) e o Problema da Designação (PD, *Assignment Problem*). O TSP descreve situações em que é necessário visitar um conjunto de localidades com o menor custo total, retornando ao ponto de partida — um modelo que aparece desde o roteamento de entregas até o planejamento de circuitos integrados (*VLSI*). Já o PD surge em contextos de alocação de agentes a tarefas, distribuição de recursos ou pareamento de elementos (pessoas, equipes, máquinas), sempre considerando um custo associado à atribuição.

Diante da relevância prática e teórica desses problemas, este projeto de Iniciação Científica investiga como a PLI pode ser empregada para modelar e resolver o TSP e o PD, explorando fundamentos teóricos, formulações matemáticas e estratégias computacionais. Foram desenvolvidas implementações em Python e empregadas ferramentas de resolução (*solvers* comerciais e open-source) para comparar o desempenho de abordagens exatas e heurísticas em métricas de tempo de execução, qualidade de solução (*gap*) e robustez.

Por fim, esta seção apresenta o referencial teórico que sustenta as escolhas metodológicas do trabalho. Primeiramente, introduz-se a Programação Linear Inteira (1.1), seguida da descrição dos problemas da Designação (1.2) e do Caixeiro Viajante (1.3). Em seguida, discutem-se aspectos de complexidade computacional (1.4), os métodos exatos para PLI (1.5) e os principais métodos heurísticos (1.6), incluindo heurísticas de construção (1.6.1), heurísticas de melhoria (1.6.1.1) e metaheurísticas (1.6.2).

1.1. Programação Linear Inteira

A Programação Linear Inteira (PLI) é um ramo da Programação Linear (PL) dedicado a problemas em que as variáveis de decisão devem assumir valores inteiros — frequentemente binários em aplicações combinatórias. Essa característica é essencial em situações práticas em que soluções fracionárias não têm interpretação válida, como na designação de tarefas, ativação de recursos ou escolha de rotas em redes de transporte.

Um modelo típico de PLI pode ser formulado como:

$$\text{Minimizar: } c^T x$$

$$\text{Sujeito a: } Ax \leq b$$

$$x \in Z_+$$

em que c é o vetor de coeficientes da função objetivo, x o vetor de variáveis de decisão, A a matriz de coeficientes e b o vetor de limites das restrições. Apesar da aparência simples, a exigência de integralidade torna os problemas de PLI substancialmente mais difíceis que sua versão contínua (o relaxamento linear), pois o espaço de soluções passa a ser discreto e, em geral, muito mais complexo.

O relaxamento linear, entretanto, desempenha papel fundamental: seu valor ótimo fornece sempre um limite inferior em problemas de minimização, enquanto qualquer solução inteira viável gera um limite superior. A diferença entre esses limites — o *gap* de integralidade — é uma medida direta da dificuldade do problema. Propriedades estruturais, como a unimodularidade da matriz de restrições, podem reduzir ou até eliminar esse *gap*, permitindo que certos problemas sejam resolvidos eficientemente apenas com técnicas de programação linear contínua.

Na maioria dos casos, contudo, são necessários métodos especializados para garantir integralidade. Os principais *solvers* de PLI combinam relaxações lineares, geração de cortes (*cutting planes*) e estratégias de ramificação (*Branch-and-Bound* ou *Branch-and-Cut*). Esse conjunto de técnicas não só possibilita resolver instâncias grandes de forma exata, como também transforma a PLI em uma verdadeira linguagem de modelagem, capaz de expressar restrições lógicas e objetivos práticos de maneira direta.

Embora a PLI seja uma das abordagens mais gerais para a otimização combinatória, ela não é a única. Muitos problemas admitem formulações mais específicas, baseadas em grafos, programação dinâmica ou decomposição. Ainda assim, a PLI se destaca por oferecer uma estrutura unificada que favorece tanto a análise teórica quanto a aplicação prática: um mesmo arcabouço pode ser reutilizado em problemas de naturezas muito diferentes.

Além disso, a PLI raramente atua isoladamente. Em aplicações de larga escala, é comum combiná-la a heurísticas e metaheurísticas em esquemas híbridos (matheurísticas). Nesse arranjo, relaxações lineares fornecem limites e informações duais, enquanto heurísticas produzem soluções factíveis de boa qualidade, resultando em métodos robustos e competitivos.

Entre os diversos problemas que podem ser modelados por meio da PLI, dois se destacam por sua relevância histórica e prática: o Problema da Designação e o Problema do Caixeiro Viajante (TSP). O primeiro ilustra de forma direta a utilidade da modelagem inteira em contextos de alocação, enquanto o segundo representa um dos desafios mais emblemáticos da otimização combinatória. A seguir, apresenta-se o Problema da Designação (1.2).

1.2. Problema da Designação

O Problema da Designação (PD), também conhecido como *Assignment Problem*, consiste em determinar a melhor forma de atribuir um conjunto de n agentes a n tarefas, de modo que cada agente seja designado exatamente a uma tarefa e cada tarefa seja executada por um único agente. O objetivo é minimizar o custo total da alocação, representado por uma matriz de custos c_{ij}, em que cada elemento indica o custo de designar o agente i à tarefa j.

A formulação clássica do PD em Programação Linear Inteira (PLI) utiliza variáveis de decisão binárias:

$$x_{ij} = \begin{cases} 1, & \text{se o agente } i \text{ é atribuído à tarefa } j, \\ 0, & \text{caso contrário.} \end{cases}$$

O modelo matemático é dado por:

$$\text{Minimizar} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij}$$

Sujeito a:

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad \forall i = 1, \dots, n \qquad \text{(cada agente é alocado a única uma tarefa)},$$

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad \forall j = 1, \dots, n \qquad \text{(cada tarefa é atribuída a um único agente)},$$

$$x_{ij} \in \{0,1\}, \qquad \forall i, j = 1, \dots, n.$$

Uma propriedade notável do PD é que seu relaxamento linear (permitindo que x_{ij} assuma valores reais entre 0 e 1) já conduz a soluções inteiras, em razão da unimodularidade da matriz de restrições. Isso garante que métodos de Programação Linear contínua, como o *Simplex*, sejam suficientes para obter soluções ótimas, sem necessidade de técnicas adicionais para impor integralidade.

Na formulação clássica, chamada balanceada, o número de agentes coincide com o de tarefas, de modo que cada agente é designado a exatamente uma tarefa. Em casos de desbalanceamento — quando há mais agentes que tarefas ou vice-versa — o modelo pode ser ajustado pela introdução de elementos fictícios com custo nulo, tornando-o equivalente ao caso balanceado e mantendo a resolubilidade em tempo polinomial.

Em aplicações reais, porém, surgem extensões mais gerais, como o Problema da Designação Generalizado (PDG). Nesse caso, restrições de capacidade ou qualificação rompem a estrutura unimodular, permitindo que o relaxamento linear produza soluções fracionárias. Uma formulação típica inclui, além das restrições clássicas, a seguinte condição:

$$\sum_{j=1}^{n} a_{ij}x_{ij} \leq b_i, \quad \forall i = 1, \dots, m \quad \text{(capcacidade ou recurso limitado para cada agente)}.$$

onde a_{ij} representa o consumo de recurso quando o agente i executa a tarefa j, e b_i é a capacidade máxima do agente i. Esse acréscimo eleva a complexidade do problema, que passa a exigir métodos de Programação Linear Inteira e pertence à classe NP-Hard.

O PD ilustra, portanto, como a PLI pode oferecer formulações elegantes e eficientes, mas também como pequenas mudanças no modelo podem transformar um problema polinomial em um desafio computacional. Esse contraste abre caminho para outro clássico da otimização combinatória, de dificuldade ainda maior: o Problema do Caixeiro Viajante (TSP), apresentado na próxima subseção.

1.3. Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (TSP – *Travelling Salesman Problem*) é um dos mais emblemáticos da otimização combinatória e um marco no estudo da complexidade computacional. Enquanto o Problema da Designação possui uma formulação eficiente graças à unimodularidade de suas restrições, o TSP ilustra a face oposta: um modelo simples de enunciar, mas notoriamente difícil de resolver.

Sua formulação descreve um conjunto de cidades e as distâncias entre cada par delas, buscando-se encontrar o ciclo de menor custo que visite todas as cidades exatamente uma vez e retorne ao ponto de origem. Matematicamente, costuma-se modelá-lo como um grafo completo $G = (V, E)$, em que V representa o conjunto de cidades e E o conjunto de arestas (conexões entre as cidades), cada uma associada a um custo c_{ij}. O objetivo é encontrar um ciclo hamiltoniano — isto é, um percurso fechado que passa por cada cidade exatamente uma vez — de menor custo.

Assim como no PD, o TSP pode ser formulado em Programação Linear Inteira (PLI) utilizando variáveis binárias:

$$x_{ij} = \begin{cases} 1, & \text{se a aresta (i, j) é utilizada no percurso,} \\ 0, & \text{caso contrário.} \end{cases}$$

O modelo básico pode ser escrito como:

$$\text{Minimizar} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} c_{ij} x_{ij}$$

Sujeito a:

$$\sum_{\substack{j=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall i = 1, \ldots, n \quad \text{(de cada cidade sai exatamente uma aresta),}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \qquad \forall j = 1, \ldots, n \quad \text{(em cada cidade entra exatamente uma aresta),}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i, j = 1, \ldots, n, \qquad i \neq j.$$

Embora semelhante ao modelo do PD, essa formulação não garante que o resultado seja um único ciclo que percorra todas as cidades. Podem surgir subciclos (subtours), ou seja, ciclos menores que não abrangem todas as cidades. Para eliminá-los, é necessário acrescentar restrições adicionais, cuja escolha afeta diretamente a eficiência da resolução.

Uma das formulações mais conhecidas é a de Miller–Tucker–Zemlin (MTZ), que introduz variáveis auxiliares u_i representando a ordem de visita das cidades. As restrições adicionais são:

$$u_i - u_j + nx_{ij} \leq n - 1, \qquad \forall i \neq j \quad (u_j \text{ deve vir depois de } u_i),$$
$$1 \leq u_i \leq n - 1, \qquad \forall i = 2, \dots, n.$$

garantindo que, se a aresta (i, j) pertence ao percurso, então a cidade j deve aparecer depois da cidade i na sequência, evitando a formação de subtours.

Além da versão simétrica — foco deste trabalho — existe também o Problema do Caixeiro Viajante Assimétrico (ATSP), no qual os custos de deslocamento podem diferir conforme a direção ($c_{ij} \neq c_{ji}$). Essa variante é relevante em contextos logísticos reais, mas compartilha a mesma complexidade do TSP clássico e igualmente requer restrições adicionais para eliminar subtours.

A importância do TSP vai muito além de suas aplicações diretas, como o planejamento de rotas e a logística de transportes. Ele consolidou-se como um problema central na otimização combinatória, servindo de base para o desenvolvimento e a avaliação de inúmeros métodos e algoritmos. Por isso, permanece como ferramenta fundamental tanto no ensino quanto na pesquisa em Ciência da Computação, Engenharia e Pesquisa Operacional, combinando aplicabilidade prática com um conteúdo teórico extremamente rico.

O contraste entre o PD, cuja estrutura unimodular permite soluções eficientes em tempo polinomial, e o TSP, notoriamente difícil mesmo em instâncias relativamente pequenas, evidencia a variedade de graus de dificuldade em problemas de otimização combinatória. Essa diferença é formalizada pela teoria da complexidade computacional, tema da próxima subseção (1.4).

1.4. Complexidade Computacional

A análise da complexidade computacional é fundamental para compreender o custo algorítmico envolvido na resolução dos problemas tratados neste trabalho. A estrutura dos modelos e a forma como suas restrições são impostas influenciam diretamente a viabilidade de aplicar métodos exatos ou heurísticos. Nesse sentido, a teoria da complexidade classifica os problemas em categorias que refletem a dificuldade tanto de resolver quanto de verificar soluções.

De forma geral, problemas da classe P (*polynomial time*) são aqueles que podem ser resolvidos por algoritmos cujo tempo de execução cresce polinomialmente em função do tamanho da entrada. O Problema da Designação (PD) clássico pertence a essa classe e pode ser resolvido exatamente em tempo

polinomial, por exemplo, pelo método de Kuhn–Munkres (ou Algoritmo Húngaro), cuja complexidade é $O(n^3)$.

A classe NP (*nondeterministic polynomial time*) reúne problemas para os quais uma solução candidata pode ser verificada em tempo polinomial, embora não se saiba se todos podem também ser resolvidos nesse tempo. Já a classe NP-Hard inclui todos os problemas que são pelo menos tão difíceis quanto os de NP; nem sempre pertencem a NP, pois suas soluções podem não ser verificáveis em tempo polinomial.

Nesse contexto, o Problema do Caixeiro Viajante (TSP), em sua formulação de otimização, é estritamente NP-Hard. O número de soluções possíveis cresce de forma fatorial ($n!$), tornando inviável o uso de algoritmos exatos em instâncias de grande porte sem recorrer a técnicas como relaxações, ramificações e cortes. Situação semelhante ocorre no Problema de Designação Generalizado (PDG): suas modificações em relação ao problema clássico eliminam a estrutura unimodular e elevam a complexidade, classificando-o também como NP-Hard.

De maneira geral, problemas NP-Hard representam uma fronteira prática da computação. Em suas formas mais gerais, são considerados intratáveis e exigem o uso de heurísticas, metaheurísticas ou algoritmos especializados para obter soluções de boa qualidade em tempo razoável. Assim, a análise da complexidade computacional não apenas orienta a escolha das estratégias algorítmicas adequadas, como também estabelece os limites práticos da resolução exata desses problemas.

Na próxima subseção (1.5), serão apresentados os métodos exatos mais empregados na literatura para resolver problemas de Programação Linear Inteira, com destaque para suas estratégias, vantagens e limitações diante dos diferentes graus de complexidade discutidos.

1.5. Métodos Exatos

A resolução de problemas de Programação Linear Inteira (PLI), como o Problema da Designação (PD) e o Problema do Caixeiro Viajante (TSP), pode ser realizada por diferentes classes de métodos, que se distinguem principalmente pelas garantias de otimalidade e pelo custo computacional. Em linhas gerais, tais métodos se dividem em exatos, heurísticos e metaheurísticos. Este estudo utiliza tanto abordagens exatas quanto heurísticas, cujos fundamentos e aplicações são apresentados a seguir.

Os métodos exatos têm como objetivo encontrar a solução ótima global, fornecendo garantia formal de otimalidade. São a base dos principais *solvers* de PLI e costumam iniciar pela resolução das relaxações lineares dos problemas inteiros. Para isso, o algoritmo Simplex (e sua versão dual) é amplamente empregado. Apesar de apresentar complexidade exponencial no pior caso, o Simplex é notavelmente eficiente em instâncias práticas, consolidando-se como método padrão para programação linear.

O *Branch-and-Bound* (B&B) é o principal mecanismo para tratar variáveis inteiras. Ele consiste em dividir o espaço de soluções em subproblemas menores (*ramificação*) e resolver cada relaxação linear (geralmente via Simplex). Quando a solução encontrada é inteira, ela se torna candidata à ótima; caso contrário, novas restrições são impostas e o espaço é novamente particionado. Embora eficaz na prática, o B&B tem complexidade exponencial no pior caso, pois pode exigir a exploração completa da árvore de soluções.

Os Planos de Corte, introduzidos por Gomory, complementam esse processo adicionando desigualdades (*cortes*) que eliminam soluções fracionárias sem descartar pontos inteiros viáveis. Essa técnica fortalece a formulação relaxada e, quando combinada ao *Branch-and-Bound*, origina o *Branch-and-Cut* — atualmente o método exato mais utilizado em *solvers* modernos.

Além disso, existem algoritmos baseados em programação dinâmica. O exemplo mais notável é o algoritmo de *Held-Karp* para o TSP, que encontra soluções exatas em tempo $O(n^2 2^n)$. Esse desempenho é assintoticamente melhor que a enumeração completa $O(n!)$, mas ainda exponencial, restringindo sua aplicação a instâncias de pequeno porte.

Por outro lado, o Problema da Designação clássico pode ser resolvido por algoritmos polinomiais, sem necessidade de técnicas de PLI. O Algoritmo Húngaro é o mais conhecido, com complexidade $O(n^3)$. Além de resolver o PD de forma eficiente, ele é frequentemente empregado como heurística construtiva no Problema da Designação Generalizado (PDG), fornecendo soluções iniciais de boa qualidade.

A principal vantagem dos métodos exatos é a garantia de otimalidade. Contudo, o crescimento exponencial do tempo de execução em instâncias de grande porte limita sua aplicação prática. Por esse motivo, recorre-se a *solvers* de otimização que implementam versões altamente otimizadas desses algoritmos, combinando diferentes estratégias para acelerar a busca pela solução ótima.

1.6. Métodos heurísticos e Metaheurísticos

A limitação dos métodos exatos diante de instâncias de grande porte motiva o uso de abordagens alternativas. As heurísticas surgem como técnicas que constroem ou aprimoram soluções de maneira eficiente, priorizando baixo custo computacional em detrimento da garantia de otimalidade. Essas estratégias são particularmente relevantes em problemas de grande escala, nos quais uma solução viável de boa qualidade, obtida em tempo reduzido, tem valor prático superior à busca de uma solução ótima frequentemente inatingível.

As heurísticas de construção produzem soluções iniciais a partir de decisões locais, formando uma base a ser posteriormente refinada. No caso do TSP, um exemplo clássico é o *Nearest Neighbour* (NN), que, partindo de um vértice arbitrário, escolhe sucessivamente o vizinho mais próximo ainda não visitado. Apesar de simples e de custo $O(n^2)$, gera soluções de qualidade modesta, mas úteis como ponto de partida.

Uma alternativa mais sofisticada é o algoritmo de *Christofides*, que explora a estrutura da *1-Tree* e garante uma aproximação com custo não superior a 1,5 vezes o ótimo para o TSP simétrico. Seu procedimento combina a construção de uma árvore geradora mínima, o emparelhamento perfeito mínimo entre vértices de grau ímpar e, por fim, a formação de um percurso euleriano convertido em ciclo hamiltoniano. O equilíbrio entre rigor teórico e viabilidade prática faz dessa heurística um marco no estudo do TSP.

A partir dessas soluções iniciais, heurísticas de melhoria buscam aperfeiçoar a qualidade do resultado por meio da exploração de vizinhanças. O paradigma k-opt, que remove k arestas e reconecta os segmentos de modo a reduzir o custo do ciclo, é a forma mais difundida desse tipo de abordagem. O caso 2-opt, em particular, apresenta custo $O(n^2)$ por iteração e ampla utilização prática.

Extensões como o método de *Lin-Kernighan* generalizam essa ideia ao não fixarem previamente o valor de k, permitindo a exploração adaptativa de vizinhanças mais complexas. Essa flexibilidade confere maior capacidade de escapar de mínimos locais e explica o sucesso do *solver* LKH, considerado uma das ferramentas mais eficazes para o TSP em grande escala.

Além das heurísticas tradicionais, destacam-se as metaheurísticas, que introduzem mecanismos de busca de mais alto nível, muitas vezes inspirados em processos naturais, físicos ou sociais. Entre elas, o Simulated Annealing, a Colônia de Formigas e os Algoritmos Genéticos exemplificam abordagens capazes de explorar o espaço de soluções de forma mais abrangente, incorporando aleatoriedade, memória ou aprendizado evolutivo.

Embora não forneçam garantias de otimalidade, as metaheurísticas costumam gerar soluções de alta qualidade em instâncias médias e grandes, ainda que exijam maior esforço na calibração de parâmetros. Por essa razão, constituem uma alternativa poderosa, mas não o foco principal deste estudo, que privilegia heurísticas de caráter mais direto e de implementação imediata.

2. METODOLOGIA

Neste trabalho foi adotado um protocolo experimental padronizado para permitir comparações reprodutíveis entre *solvers* exatos, *solvers* especializados e heurísticas implementadas. As modelagens de Programação Linear Inteira (PLI) foram formuladas em AMPL e resolvidas por *solvers* comerciais e *open-source* via interface AMPL, enquanto as heurísticas e rotinas de pós-processamento foram desenvolvidas em Python. Para métodos estocásticos, cada experimento foi repetido três vezes com *seeds* distintas, com coleta automatizada de logs e exportação padronizada dos resultados em formato CSV. O repositório associado contém todos os scripts, modelos AMPL e notebooks de análise, garantindo rastreabilidade e reprodução completa.

2.1. Protocolo Experimental

Os experimentos foram conduzidos em ambiente controlado, com processador AMD FX-8320E 3.20 GHz, 16 GB de RAM e sistema Ubuntu 24.04.3 LTS. As versões de software incluíram Python 3.12.3, MATLAB R2025a, AMPL 4.0, CPLEX 22.1.2, Gurobi 12.0.1, XPRESS 9.5.0, HiGHS 1.10.0, SCIP 9.0.1, CBC, LKH-2 2.0.11 e Concorde 03.12.19. Todos os *solvers* foram executados em configurações padrão, salvo indicação em contrário, e o uso de GPU foi evitado para isolar efeitos algorítmicos de diferenças de hardware.

Cada execução seguiu um fluxo padronizado: (i) pré-processamento da instância; (ii) resolução com limite de tempo (100 s para instâncias pequenas/médias e 1800 s para grandes); (iii) três repetições independentes; (iv) registro de logs; (v) exportação dos resultados para análise.

O desempenho foi avaliado a partir de métricas padronizadas. Para tempo de execução, utilizou-se a média das três rodadas por instância e, em seguida, a média e o desvio padrão por categoria de porte. Para a qualidade da solução em métodos não exatos, empregou-se o *gap* percentual em relação à melhor solução conhecida, definido por:

$$gap(\%) = 100 \times \frac{z_{alg} - z_{ref}}{z_{ref}},$$

além do número de instâncias resolvidas dentro do limite de tempo.

2.2. Instâncias

Para o Problema do Caixeiro Viajante (TSP), consideraram-se apenas instâncias simétricas da TSPLIB, organizadas por porte. (i) Pequeno porte (até 100 vértices): 28 instâncias, usadas na comparação entre *solvers* exatos; (ii) Porte médio (101 a 1000 vértices): 50 instâncias, destinadas aos testes com heurísticas. (iii) Grande porte (1001 a 10 000 vértices): 26 instâncias, reservadas à avaliação de *solvers* especializados como Concorde e LKH.

Para o Problema da Designação (PD), utilizaram-se instâncias da OR-Library. Na formulação clássica (matrizes quadradas), foram testadas oito instâncias com dimensão $n \times n$ e $n \leq 800$. Além disso, incluíram-se 20 instâncias maiores, com $n \in \{1500, 3000, 5000\}$, caracterizadas como esparsas devido à impossibilidade de certas combinações entre agentes e tarefas.

O Problema da Designação Generalizado (PDG) também foi estudado a partir de instâncias da OR-Library, mas em formato retangular. Esse conjunto reuniu 68 instâncias de menor dimensão (até 20×200) e 16 de grande porte, chegando a 4000.

2.3. Métodos Testados

Os métodos descritos nas seções anteriores fundamentam a escolha dos algoritmos avaliados. Para formulações gerais em PLI, utilizaram-se *solvers* comerciais amplamente reconhecidos — CPLEX,

Gurobi e XPRESS — e *solvers open-source* — SCIP, HiGHS e CBC. Esses sistemas incorporam, de forma otimizada, as técnicas exatas discutidas anteriormente, como relaxações lineares (Simplex ou pontos interiores) combinadas com *Branch-and-Bound* e *Branch-and-Cut*. Todos foram executados em suas configurações padrão para refletir o desempenho sem ajustes adicionais, exceto quando enunciado.

Para o PD clássico, empregou-se o Algoritmo Húngaro, implementado pela função *linear_sum_assignment* da biblioteca SciPy, que garante soluções ótimas em tempo $O(n^3)$. Para o TSP em instâncias de grande porte, incluíram-se dois *solvers* especializados: Concorde, amplamente considerado referência de desempenho exato, e LKH (*Lin-Kernighan-Helsgaun*), reconhecido como uma das heurísticas mais eficazes para TSP simétrico.

Complementarmente, foram implementadas heurísticas clássicas. No TSP, aplicaram-se heurísticas construtivas (*Nearest Neighbour* e *Christofides*) e heurísticas de melhoria (2-opt), buscando refinar as soluções iniciais. Esse conjunto de métodos — de *solvers* exatos genéricos a algoritmos especializados e heurísticas implementadas — possibilita uma avaliação comparativa abrangente em diferentes classes de instâncias.

3. RESULTADOS

Esta seção apresenta os resultados obtidos para o Problema da Designação (PD e PDG) e para o Problema do Caixeiro Viajante (TSP). A análise considera tempo de execução, qualidade das soluções (*gap* percentual) e número de instâncias resolvidas até o limite de tempo. Os resultados são discutidos por problema, tomando as tabelas como evidência direta.

3.1. Problema da Designação

O PD clássico (matrizes quadradas, n ≤ 800) foi testado em oito instâncias de referência, resolvidas tanto por solvers de Programação Linear Inteira quanto pelo Algoritmo Húngaro. O objetivo foi comparar eficiência e consistência entre métodos gerais e um algoritmo dedicado.

Tabela 1.1 – Resultados do PD com solvers PLI (n ≤ 800, 8 instâncias)

Solver	Otimalidade
Gurobi	8/8
CPLEX	8/8
XPRESS	8/8
CBC	8/8
HiGHS	6/8
SCIP	5/8

Tabela 1.2 – Comparação do Algoritmo Húngaro e solvers PLI no PD (n ≤ 800, 8 instâncias)

Método	Tempo (s) Médio ± Desvio
Húngaro	0.01 ± 0.01
Gurobi	3.65 ± 3.11

CBC	22.30 ± 22.26

No Problema da Designação (PD), os testes confirmaram a eficiência esperada das formulações em Programação Linear Inteira: todos os *solvers* comerciais e até o CBC resolveram integralmente as instâncias, enquanto HiGHS e SCIP apresentaram desempenho um pouco inferior. O destaque, porém, foi o Algoritmo Húngaro, que alcançou tempos de execução duas a três ordens de grandeza menores, obtendo soluções ótimas praticamente instantâneas.

Esses resultados reforçam que, embora a PLI seja suficiente para resolver o PD graças à sua estrutura unimodular, o uso de *solvers* genéricos implica custos desnecessários frente a um algoritmo dedicado. Na prática, o Húngaro deve ser a escolha prioritária para instâncias puras do PD, enquanto os *solvers* mantêm relevância apenas quando o problema surge como subcomponente em modelos mais complexos.

3.2. Problema da Designação Generalizado

O PDG rompe com a estrutura unimodular do PD clássico, tornando o problema mais desafiador. Para as instâncias avaliadas ($n \times m \leq 8000$), os testes compararam diferentes *solvers* de PLI sob limite de tempo de 100 s.

Tabela 2.1 – Resultados do PDG com solvers PLI ($n \times m \leq 8000$, 36 instâncias, limite de tempo = 100s)

Solver	Otimalidade
Gurobi	29/36
CPLEX	29/36
XPRESS	29/36
HiGHS	26/36
SCIP	24/36
CBC	16/36

Tabela 2.2 – Tempos médios no PDG (36 instâncias, limite de tempo = 100s)

Método	Tempo (s) Médio ± Desvio
Gurobi	3.16 ± 4.84
HiGHS	16.44 ± 19.82

No Problema da Designação Generalizado (PDG), a perda da estrutura unimodular aumentou sensivelmente a dificuldade das instâncias. Os resultados mostraram clara separação entre *solvers* comerciais e alternativas open-source: Gurobi, CPLEX e XPRESS resolveram 29 das 36 instâncias, enquanto HiGHS e SCIP ficaram próximos, mas com menor consistência, e o CBC teve desempenho limitado. Além disso, o tempo médio de execução reforçou a vantagem dos *solvers* comerciais, especialmente do Gurobi, que apresentou menor variabilidade.

Esses achados evidenciam que, no PDG, a escolha do solver afeta diretamente a qualidade e a confiabilidade das soluções, sendo os comerciais preferíveis em aplicações críticas, ainda que soluções open-source já se mostrem competitivas em parte dos cenários.

3.3. Problema do Caixeiro Viajante com PLI

O TSP formulado em Programação Linear Inteira representa um desafio mais severo que o PD ou o PDG, já que a complexidade combinatória cresce rapidamente. Para avaliar até que ponto os *solvers* conseguem lidar com esse problema, foram testadas 28 instâncias com n ≤ 100, sob limite de 100 s.

Tabela 3.1 – Resultados do TSP com PLI (n ≤ 100, 28 instâncias, limite de tempo = 100s)

Solver	Otimalidade
Gurobi	17/28
CPLEX	15/28
XPRESS	15/28
SCIP	12/28
HiGHS	12/28
CBC	9/28

Tabela 3.2 – Tempos médios no TSP com PLI (28 instâncias, limite de tempo = 100s)

Método	Tempo (s) Médio ± Desvio
Gurobi	11.77 ± 24.99
CBC	18.85 ± 26.33

Tabela 3.3 – *Gaps* médios em instâncias não resolvidas do TSP (28 instâncias, limite de tempo = 100s)

Método	Gap (%) Médio ± Desvio
Gurobi	1.70 ± 4.11
CBC	9.47 ± 33.46

No Problema do Caixeiro Viajante (TSP), mesmo em instâncias pequenas (n ≤ 100), os *solvers* de PLI mostraram limitações claras. Embora o Gurobi tenha resolvido 17 das 28 instâncias e CPLEX/XPRESS tenham alcançado 15, nenhum solver foi capaz de resolver todas dentro do tempo limite. As alternativas open-source ficaram ainda mais atrás, com menor número de soluções ótimas e *gaps* residuais elevados. O tempo de execução apresentou grande variabilidade, refletindo a natureza irregular da dificuldade das instâncias. Nos casos não resolvidos, o Gurobi manteve *gaps* baixos, próximos de 1–2%, enquanto CBC apresentou dispersão significativa, com piores casos muito afastados do ótimo.

Esses resultados confirmam que, mesmo em porte reduzido, a formulação em PLI encontra barreiras de escalabilidade no TSP, reforçando a necessidade de heurísticas e métodos especializados para instâncias maiores.

3.4. Heurísticas ao Problema do Caixeiro Viajante

Além dos métodos exatos, avaliou-se o desempenho de heurísticas clássicas para o TSP, com o objetivo de verificar até que ponto abordagens aproximadas conseguem equilibrar qualidade da solução e tempo de execução. Foram consideradas duas estratégias: *Christofides* + 2-opt, que combina uma construção com garantia teórica seguida de refinamento local, e *Nearest Neighbor* (NN) + 2-opt, de natureza mais simples e rápida, mas sem garantias de qualidade.

Tabela 4.1 – Heurísticas para o TSP em instâncias pequenas (n ≤ 100, 17 resolvidas pelo Gurobi)

Método	Gap (%) Médio ± Desvio	Tempo (s) Médio ± Desvio
Christofides + 2opt	4.11 ± 2.76	0.05 ± 0.08
NN + 2opt	8.66 ± 4.78	0.01 ± 0.03
Gurobi	-	11.77 ± 24.99

Tabela 4.2 – Heurísticas para o TSP em instâncias médias (100 < n ≤ 1000, 50 instâncias)

Método	Gap (%) Médio ± Desvio	Tempo (s) Médio ± Desvio
Christofides + 2opt	4.56 ± 1.88	6.96 ± 14.54
NN + 2opt	11.04 ± 5.14	10.56 ± 26.14

As heurísticas testadas para o TSP evidenciaram um equilíbrio favorável entre tempo e qualidade. O método *Christofides* + 2-opt apresentou desempenho robusto: manteve *gaps* médios em torno de 4–5%, com baixa variabilidade, tanto em instâncias pequenas quanto médias. Já o *Nearest Neighbor* + 2-opt foi mais rápido em casos reduzidos, mas perdeu qualidade de forma consistente, alcançando *gaps* próximos de 9% nas instâncias pequenas e acima de 11% nas médias.

Em todos os cenários, os tempos de execução permaneceram muito inferiores aos de qualquer solver exato, confirmando a utilidade das heurísticas não apenas como ponto de partida, mas como alternativas reais em instâncias onde a prova de otimalidade é inviável.

3.5. Problema do Caixeiro Viajante em Grande Escala (Solvers Especializados)

Para instâncias do TSP de porte elevado, foram comparados dois *solvers* especializados amplamente reconhecidos: o LKH, heurístico baseado em *Lin-Kernighan* altamente otimizado, e o Concorde, considerado referência na resolução exata do problema. O objetivo foi avaliar como se comportam em termos de tempo e qualidade da solução quando o tamanho das instâncias ultrapassa a capacidade prática da PLI.

Tabela 5.1 – Comparação entre LKH e Concorde em grande escala (1000 < n ≤ 10000, 26 instâncias)

Solver	Timelimit
LKH	0
Concorde	14

Tabela 5.2 – Comparação entre LKH e Concorde em instâncias médias (100 < n ≤ 1000, 50 instâncias)

Solver	Tempo (s) Médio ± Desvio
LKH	3.19 ± 6.05
Concorde	13.96 ± 24.99

Tabela 5.3 – Comparação entre LKH e Concorde em instâncias pequenas (n ≤ 100, 28 instâncias)

Solver	Tempo (s) Médio ± Desvio
LKH	0.04 ± 0.04
Concorde	0.13 ± 0.16

Tabela 5.4 – Instâncias não resolvidas até a otimalidade pelo LKH (lista de casos específicos)

Instância	Custo LKH	Custo Exato	Gap (%)
gr229	134616	134602	0,01
gr341	171534	171414	0,07
si535	48452	48450	0,004
fl1577	22262	22249	0,05

Nos experimentos de grande porte, a diferença entre os *solvers* especializados ficou evidente. O LKH mostrou-se claramente mais escalável, entregando soluções para todas as instâncias em tempos reduzidos e com *gaps* inferiores a 0,1% nos poucos casos em que não atingiu o ótimo. Já o Concorde, embora mantenha o status de referência exata, encontrou limites práticos: em instâncias muito grandes, várias execuções atingiram o *timelimit*, e mesmo em tamanhos médios seu tempo foi sistematicamente superior ao do LKH.

Assim, confirma-se que, enquanto o Concorde permanece indispensável para a certificação de otimalidade em casos menores, o LKH se apresenta como a alternativa mais viável para aplicações reais em larga escala, equilibrando eficiência e qualidade de forma quase indistinguível do ótimo.

4. DISCUSSÃO

Os resultados confirmam tendências já consolidadas na literatura, mas também revelam nuances relevantes para a prática. No Problema do Caixeiro Viajante (TSP), observou-se a inviabilidade de métodos exatos em instâncias médias e grandes: os *solvers* de Programação Linear Inteira (PLI) apenas resolvem instâncias pequenas antes que a explosão combinatória inviabilize o processamento. Esse achado é esperado, mas o estudo acrescenta evidências sobre a importância da qualidade da solução inicial.

Heurísticas construtivas mais elaboradas, como *Christofides*, forneceram pontos de partida melhores do que métodos simples como *Nearest Neighbor*, reduzindo o esforço de heurísticas de melhoria subsequentes. Ainda assim, a presença de ótimos locais limitou a redução do gap, o que reforça a necessidade de estratégias adicionais de diversificação, capazes de escapar de armadilhas locais — aspecto crítico em cenários reais de logística.

No Problema da Designação (PD), a estrutura unimodular garantiu que diferentes abordagens resolvessem instâncias de forma eficiente. Entretanto, o destaque sistemático do Algoritmo Húngaro

sobre os *solvers* de PLI mostra como propriedades estruturais, quando exploradas por algoritmos dedicados, geram vantagens claras. Esse contraste com o TSP é significativo: enquanto neste a complexidade combinatória rapidamente inviabiliza abordagens exatas, no PD a exploração da estrutura matemática praticamente elimina a dificuldade computacional.

Já no Problema da Designação Generalizado (PDG), o desempenho mostrou-se altamente dependente da qualidade dos *solvers*. As versões comerciais se mostraram superiores, enquanto os *open-source* foram competitivos em instâncias pequenas, mas apresentaram maior variabilidade em instâncias de maior porte. Esse resultado ressalta que não apenas o método em si, mas também a implementação e o ajuste fino de parâmetros influenciam diretamente a performance — fator que tem implicações práticas relevantes, já que equipes com diferentes restrições de recursos podem enfrentar cenários de desempenho bastante distintos.

Com isso, os resultados mostraram que a complexidade percebida não depende apenas do tamanho da instância, mas também de sua estrutura. Em alguns casos, instâncias relativamente pequenas mostraram-se mais desafiadoras do que instâncias maiores, evidenciando que simplificações baseadas apenas na escala podem ser enganosas para gestores e planejadores. Assim, confirma-se que métodos exatos são adequados em instâncias pequenas e fornecem base teórica sólida; heurísticas e *solvers* especializados são indispensáveis em instâncias médias e grandes; e algoritmos dedicados superam abordagens generalistas quando a estrutura matemática do problema pode ser explorada.

4.1. Limitações e Trabalhos Futuros

Apesar do panorama abrangente, o estudo apresenta limitações que devem ser reconhecidas. A primeira refere-se às instâncias utilizadas, oriundas de bibliotecas clássicas (TSPLIB e OR-Library). Embora amplamente aceitas, elas não refletem integralmente a complexidade de cenários reais, que frequentemente incluem assimetrias de custos, janelas de tempo, incertezas e restrições adicionais. Trabalhos futuros devem, portanto, incorporar dados reais e variantes mais ricas, como o TSP assimétrico e problemas de roteamento de veículos.

Outra limitação diz respeito ao uso dos *solvers*. Não houve exploração aprofundada de parâmetros de configuração, embora estes possam impactar fortemente tanto o tempo quanto a qualidade da solução. Pesquisas futuras devem investigar estratégias sistemáticas de ajuste fino, comparando não apenas métodos, mas também configurações e implementações.

Também se reconhece o viés computacional da análise, centrada em aspectos algorítmicos e experimentais. Questões próprias de aplicações reais — como coleta, tratamento e validação de dados, análise de sensibilidade e impacto de variações — não foram contempladas. Esses elementos devem ser incorporados em estudos futuros para maior aderência a cenários práticos.

Além disso, a análise concentrou-se em médias e desvios, sem aplicação de testes estatísticos formais para verificar a significância das diferenças. Protocolos mais robustos, como testes não

paramétricos (ex.: Wilcoxon), múltiplas repetições em métodos estocásticos e comparações post-hoc, são caminhos promissores para aumentar a confiabilidade das conclusões.

Por fim, não foram exploradas tecnologias emergentes, como aceleração em GPU (ex.: cuOpt), abordagens quânticas ou híbridas, e metaheurísticas mais sofisticadas (GRASP, algoritmos genéticos, simulated annealing, métodos de enxame), nem a integração entre PLI e heurísticas em *matheurísticas*. Esses caminhos representam oportunidades claras para avanços metodológicos e aplicações em maior escala.

5. CONCLUSÃO

Este trabalho revisou a aplicação da Programação Linear Inteira (PLI), de heurísticas e de algoritmos especializados na resolução do Problema do Caixeiro Viajante (TSP), do Problema da Designação (PD) e do Problema da Designação Generalizado (PDG), a partir de uma análise essencialmente computacional e experimental.

Os resultados confirmaram que a adequação dos métodos varia conforme a natureza do problema e o porte das instâncias. No TSP, a explosão combinatória rapidamente inviabiliza métodos exatos, enquanto heurísticas e *solvers* especializados tornam-se indispensáveis em maior escala. No PD, a estrutura unimodular assegura soluções eficientes, com destaque para o Algoritmo Húngaro. Já no PDG, a performance dependeu fortemente da robustez dos *solvers*, com vantagem para implementações comerciais, ainda que as *open-source* tenham se mostrado competitivas em instâncias menores.

Dessa forma, a revisão mostrou que não existe um método universalmente superior: a escolha deve considerar a estrutura matemática do problema, o tamanho da instância e os objetivos da aplicação. A PLI permanece fundamental como base de modelagem e comparação teórica, mas heurísticas, algoritmos dedicados e *solvers* especializados são os recursos que garantem escalabilidade e viabilidade prática.

Em síntese, o estudo sistematiza experimentos para escolha de diferentes abordagens, destacando padrões já consolidados e apontando caminhos práticos para a aplicação em problemas de otimização combinatória.

6. REFERÊNCIAS BIBLIOGRÁFICAS

APPLEGATE, D. L., BIXBY, R. E., CHVÁTAL, V., & COOK, W. J. (2011). *The Traveling Salesman Problem - A Computational Study.* Princeton University Press. Fonte: https://www.math.uwaterloo.ca/tsp/concorde.html

BAZAARA, M. S., JARVIS, J. J., & SHERALI, H. D. (2005). *Linear Programming and Network Flows.* Wiley.

BEASLEY, J. E. (1990). *OR-Library: distributing test problems by electronic mail*. Fonte: https://people.brunel.ac.uk/~mastjjb/jeb/orlib/

CHEN, D.-S., BATSON, R. G., & Yu, D. (2010). *Applied Integer Programming - Modeling and Solutions.* Wiley.

CORMEN, T. H. (2001). *Introduction to algorithms.* MIT Press.

GOLDBARG, M., & LUNA, H. (2005). *Otimização Combinatória e Programação Linear.* Editora Campus.

HELSGAUN, K. (2000). *LKH - An Effective Implementation of the Lin-Kernighan Travelling Salesman Heuristic*. Fonte: http://webhotel4.ruc.dk/~keld/research/LKH/

MACULAN, N. F. (2006). *Otimização Linear.* Editora UnB.

REINELT, G. (1991). *TSPLIB - A Traveling Salesman Problem Library*. Fonte: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

WOLSEY, L. A. (1998). *Integer Programming.* Wiley.

Todos os códigos de implementações, resultados e experimentos feitos estão disponíveis em: https://github.com/stepsbtw/A-Programacao-Inteira-e-sua-Aplicacao-aos-Problemas-do-Caixeiro-Viajante-e-da-Designacao

7. AGRADECIMENTOS

Os autores deste trabalho agradecem ao CEFET-RJ pelo apoio a este trabalho científico.