UNIVERSIDADE FEDERAL DE PERNAMBUCO

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Eletrônica e Sistemas

Graduação em Engenharia Eletrônica

Ricardo Silva Machado

Otimização de Layouts como um Problema de Steiner: uma Abordagem via Programação Não Linear

Recife

2025

Ricardo Silva Machado

Otimização de Layouts como um Problema de Steiner: uma Abordagem via Programação Não Linear

Trabalho de Conclusão de Curso apresentado ao Departamento de Eletrônica e Sistemas, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Eletrônica. (Eng.)

Orientador(a): Prof. Patrícia Silva Lessa, D.Sc.

Recife
2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Machado, Ricardo Silva.
Otimização de layouts como um problema de Steiner: uma abordagem via programação não linear / Ricardo Silva Machado. - Recife, 2025.
67 p. : il., tab.

Orientador(a): Patrícia Silva Lessa
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Eletrônica - Bacharelado, 2025.
Inclui referências.

1. Eletrônica. 2. Otimização não linear. 3. Problema de Steiner. 4. Design de circuitos. 5. Grafos. I. Lessa, Patrícia Silva. (Orientação). II. Título.

620 CDD (22.ed.)

Ricardo Silva Machado

Otimização de Layouts como um Problema de Steiner: uma Abordagem via Programação Não Linear

Trabalho de Conclusão de Curso apresentado ao Departamento de Eletrônica e Sistemas, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Eletrônica. (Eng.)

Aprovado em: 31/03/2025

Banca Examinadora

Prof. Patrícia Silva Lessa, D.Sc.
Universidade Federal de Pernambuco

Prof. Gilson Jerônimo da Silva Júnior, D.Sc.
Universidade Federal de Pernambuco

Prof. Fernando Menezes Campello de Souza, Ph.D.
Universidade Federal de Pernambuco

“O homem que se dedica à ciência não pode se dividir em duas metades, separando sua fé de seu conhecimento; mesmo em suas investigações científicas, ele continua sendo um homem — não um ser puramente intelectual, mas uma pessoa com um coração, com afeições e emoções, com sentimento e vontade.”

Herman Bavinck

Resumo do Trabalho de Conclusão de Curso apresentado ao Departamento de Eletrônica e Sistemas, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Eletrônica. (Eng.)

Otimização de Layouts como um Problema de Steiner: uma Abordagem via Programação Não Linear

Ricardo Silva Machado

Este trabalho propõe uma abordagem para a otimização de *layouts* de circuitos eletrônicos, abordando o problema como uma variante do Problema da Árvore de Steiner. Ao representar circuitos como grafos, onde os terminais correspondem aos pontos de conexão e os pontos adicionais (pontos de Steiner) são introduzidos para reduzir o comprimento total das interconexões, a metodologia integra técnicas de programação não linear para atender às restrições reais de circuitos complexos. O trabalho abrange a fundamentação teórica dos métodos de otimização, a modelagem matemática e a implementação de um algoritmo que, aliado à análise estatística dos resultados, demonstra a rapidez da abordagem na redução de comprimentos de fios. Os experimentos indicam que a utilização de pontos de Steiner, em conjunto com a programação não linear, possibilita soluções adaptáveis às matrizes de incidência dos grafos em circuitos planares.

Palavras-chave: Grafos; otimização de *layouts*; problema de Steiner; programação não linear.

Abstract of Course Conclusion Work, presented to Departament of Eletronic and Systems, as a partial fulfillment of the requirements for the degree of Bachelor of Electronic Engineering. (Eng.)

Layouts Optimization as a Steiner Problem: A Nonlinear Programming Approach

Ricardo Silva Machado

This work proposes an approach to optimizing electronic circuit layouts by approaching the problem as a variant of the Steiner Tree Problem. By representing circuits as graphs, where terminals correspond to connection points and additional points (Steiner points) are introduced to reduce the total length of interconnections, the methodology integrates nonlinear programming techniques to address the real constraints of complex circuits. The work covers the theoretical foundation of optimization methods, mathematical modeling, and the implementation of an algorithm that, combined with statistical analysis of the results, demonstrates the efficiency of the approach in reducing wire lengths. The experiments indicate that the use of Steiner points, along with nonlinear programming, enables adaptable solutions to the incidence matrices of graphs in planar circuits.

Keywords: Graphs; layout optimization; nonlinear programming; Steiner problem.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Otimizar *layouts* de circuitos é naturalmente um desafio essencial na engenharia eletrônica. À medida que os circuitos se tornam mais complexos e miniaturizados, as técnicas convencionais de *design* enfrentam limitações, gerando a necessidade de abordagens mais sofisticadas e eficientes. Neste contexto, o uso do Problema de Steiner como modelo para o *layout* de circuitos, combinado com métodos de programação não linear, surge como uma solução promissora. Esta abordagem permite a minimização do comprimento total das conexões entre componentes. A utilização de métodos de otimização não linear para resolver o problema proporciona não apenas caminhos mais curtos, mas também soluções mais rápidas e adaptadas às complexidades das configurações reais dos circuitos, mostrando-se uma técnica altamente competitiva e vantajosa.

Embora o *design* de circuitos tenha evoluído significativamente nas últimas décadas, os métodos convencionais de otimização são por vezes obscuros e não se tem acesso a seus algoritmos de esquematização. Neste cenário, como é possível desenvolver um modelo de *layout* com meios matemáticos e computacionais acessíveis que não só minimize o comprimento das conexões, mas também se adapte às restrições reais, eventualmente difíceis de se matematizar?

1.1 Justificativa

No *design* de circuitos eletrônicos, especialmente nas etapas de prototipagem, os engenheiros enfrentam desafios significativos ao conectar componentes e roteamento de fios de forma eficiente. Ao usar matrizes de contato, por exemplo, o espaço limitado e o *layout* fixo introduzem restrições que podem levar a conexões superlotadas, caminhos de fiação ineficientes, maus contatos e, quando a coisa se estende, um amontoado de placas se conectando, obviamente com maior probabilidade de erros, como interferência, mau contato e curtos-circuitos. Além disso, conforme a complexidade do circuito aumenta, a necessidade de minimizar o comprimento das conexões para reduzir a resistência e a degradação do sinal torna-se crucial. Otimizar *layouts* para criar os caminhos mais curtos e eficientes entre os componentes é, portanto, essencial para melhorar o desempenho e a confiabilidade dos circuitos.

1.2 Objetivo Geral

Desenvolver uma abordagem otimizada para o *design* de *layouts* de circuitos eletrônicos, tratando-o como um Problema de Steiner e resolvendo-o por meio de técnicas de programação não linear. Para alcançar este objetivo, busca-se explorar o Problema de Steiner no contexto de circuitos eletrônicos e propor um modelo que permita representar esses circuitos como grafos, de forma que a otimização das conexões seja alcançada com o mínimo comprimento total.

1.3 Objetivos específicos

- Apresentar o Problema de Steiner;
- Discutir otimização matemática linear e não linear;
- Modelar circuitos eletrônicos como grafos;

- A partir da matriz de incidência dos grafos, modelar o problema de programação não linear;
- Elaborar um algoritmo acessível para solucionar o problema; e
- Analisar estatisticamente os resultados e aprimorar algoritmo.

1.4 Organização do TCC

O conteúdo deste TCC está dividido em sete capítulos. As referências encontram-se nas páginas finais. A seguir, um resumo dos capítulos seguintes do TCC.

Capítulo 2. Resumo histórico da influência, vida, influenciados e obra de Jakob Steiner.

Capítulo 3. Introdução aos grafos; apresentação do Problema de Steiner; aplicações; e o Problema de Steiner em grafos.

Capítulo 4. Origem histórica da Programação Linear; formulação em problemas genéricos; o Método Simplex.

Capítulo 5. Formulações de problemas de programação matemática em produção, armazenamento, investimento, alocação de recursos, diligência e sistemas de potência; introdução e exemplo de programação não linear.

Capítulo 6. Proposta de abordagem ao *layout* de circuitos via grafos; formulação de objetivos e restrições do problema de *layout* como programação não linear; exemplo de circuito proposto com matriz de incidência.

Capítulo 7. Elaboração do algoritmo para solução do problema; adaptação para estatística descritiva e ajuste de distribuição; ajuste em parâmetros para aprimorar resultados.

Capítulo 2

Jakob Steiner

Johann Heinrich Pestalozzi destacou-se na educação por propor uma abordagem integral, na qual os elementos essenciais do desenvolvimento humano são trabalhados desde a infância. Sua ênfase na formação do caráter, respeitando as diferenças individuais e estimulando a proatividade, influenciou práticas pedagógicas em todo o mundo – inclusive na criação da educação física – e impactou figuras como Albert Einstein. Contudo, dentre seus poucos pupilos diretos figura Jakob Steiner.

Jakob Steiner (Utzenstorf, 18 de março de 1796 – Bern, 1º de abril de 1863) teve uma trajetória singular, tendo sido analfabeto até os 14 anos, convivido com reumatismo e preparado suas palestras com escasso tempo de antecedência (Begehr e Lenz, 1998). Sua contribuição à geometria destaca-se pela revisão dos fundamentos e pela ampliação dos limites do conhecimento geométrico. Crítico da geometria analítica, Steiner enalteceu os métodos sintéticos — que se distinguem por privilegiar demonstrações puramente geométricas, baseadas em raciocínios dedutivos, sem o recurso direto a coordenadas, equações ou álgebra — evidenciados em obras como a fórmula para a partição do espaço por planos e os estudos sobre cadeias de círculos tangentes.

Na geometria projetiva, suas contribuições foram revolucionárias. Em *Systematische Entwickelung der Abhängigkeit geometrischer Gestalten von einander* (Desenvolvimento sistemático da dependência das figuras geométricas entre si), ele fun-

damentou a geometria sintética moderna, introduzindo o conceito de dualidade e explorando transformações por projeções. A abordagem das seções cônicas, conhecida como cônica de Steiner, e a demonstração, em *Die geometrischen Constructionen ausgeführt mittels der geraden Linie und eines festen Kreises*, (As construções geométricas realizadas com o auxílio da linha reta e de um círculo fixo) de que problemas de segunda ordem podem ser resolvidos apenas com régua, são marcos de seu legado.

Além disso, Steiner avançou em estudos algébricos e combinatórios. Em *Allgemeine Eigenschaften algebraischer Curven* (Propriedades gerais das curvas algébricas), abordou propriedades de curvas e superfícies algébricas e, por meio do cálculo de variações, ampliou os horizontes do cálculo. Sua contribuição para a combinatória, especialmente com o desenvolvimento dos sistemas de Steiner, consolidou seu impacto na matemática, conforme evidenciado pela publicação póstuma de seus primeiros manuscritos.

Destacam-se três áreas de sua contribuição: o desenvolvimento da geometria sintética (incluindo a cônica de Steiner), a investigação de curvas e superfícies algébricas e as inovações na combinatória com os sistemas de Steiner.

2.1 Outras obras

A seguir, enumera-se panoramicamente diferente contribuições científicas por parte de Jakob Steiner.

2.1.1 Construções de Steiner

As construções geométricas de Steiner solucionam problemas clássicos por meio de métodos puramente sintéticos. Por exemplo, o **Teorema Steiner-Lehmus** estabelece que, em um triângulo, se dois bissetores internos possuem comprimentos iguais, então o triângulo é isósceles. Em um triângulo ABC com bissetores l_1 e l_2 tais que $l_1 = l_2$, conclui-se que $AB = AC$.

2.1.2 Geometria Projetiva

Na geometria projetiva, Steiner desenvolveu teoremas fundamentais. O **Teorema de Steiner** afirma que, dada uma cônica $\mathcal{C}$ e um ponto P externo a ela, há uma correspondência bijetiva entre as retas que passam por P e os pontos de tangência em $\mathcal{C}$. Esse resultado fundamenta o princípio da dualidade e amplia a compreensão das seções cônicas.

2.1.3 Superfícies de Steiner e Sistema de Círculos

Steiner também investigou superfícies especiais e sistemas de círculos. Seu estudo sobre cadeias de círculos tangentes a três círculos preestabelecidos – as chamadas *Cadeias de Steiner* – e a análise de superfícies, como a superfície romana, contribuíram significativamente para o avanço da geometria diferencial (ou riemanniana).

2.1.4 Influência na Geometria Moderna

A adoção dos métodos sintéticos por Steiner impactou a geometria moderna, favorecendo o desenvolvimento de novas construções e a integração entre os métodos sintéticos e analíticos. Seus teoremas e metodologias permanecem essenciais, proporcionando soluções elegantes para problemas clássicos e incentivando a unificação dos conceitos geométricos. Assim, o legado de Steiner como pioneiro na geometria sintética integra, de forma indissociável, o panorama da matemática contemporânea.

2.2 O Problema da Árvore de Steiner

Entre os diversos tópicos abordados por Steiner, destaca-se o problema da árvore de Steiner, que engloba várias questões de otimização combinatória. Sua abordagem consiste em encontrar interconexões ótimas entre múltiplos elementos, conforme uma função objetivo predefinida. Esse problema manifesta-se em diversas configurações, originando variações como a árvore de Steiner em grafos, a árvore de Steiner euclidiana (Soothill, 2010) e a árvore de Steiner retilinear mínima, cada qual com desafios

e *insights* específicos.

Historicamente, o problema da árvore de Steiner relaciona-se intimamente a dois problemas clássicos de otimização: o do caminho mais curto e o da árvore de expansão mínima. Por exemplo, a versão em grafos pode ser reduzida ao problema do caminho mais curto quando há dois terminais ou tornar-se equivalente ao da árvore de expansão mínima quando todos os vértices são terminais. Diferentemente destes, que são solucionáveis em tempo polinomial, a variante de decisão da árvore de Steiner é NP-completa, posicionando-o entre os 21 problemas NP-completos identificados por Karp.

Apesar da complexidade, o problema da árvore de Steiner tem ampla aplicação prática, como em *layouts* de circuitos e design de redes, o que gerou diversas variações e contribuiu ricamente para a matemática. Embora a maioria de suas versões seja NP-difícil, alguns casos restritos podem ser resolvidos em tempo polinomial, e soluções eficientes têm sido desenvolvidas para variantes em grafos e contextos retilineares. No capítulo seguinte, o problema é analisado de forma mais aprofundada.

Capítulo 3

Problema da árvore de Steiner

Grosseiramente, o problema de Steiner é definido em um "mapa" de pontos interligados. Em muitos casos, busca-se otimizar rotas entre dois pontos (por exemplo, determinar as cidades no trajeto mais curto de Pombal a Catolé do Rocha) ou identificar a posição ideal para inserir um novo ponto (como a localização de uma fábrica). Assim, a abordagem do problema ocorre tanto em representações euclidianas quanto via grafos. A seguir, apresenta-se uma introdução à teoria dos grafos.

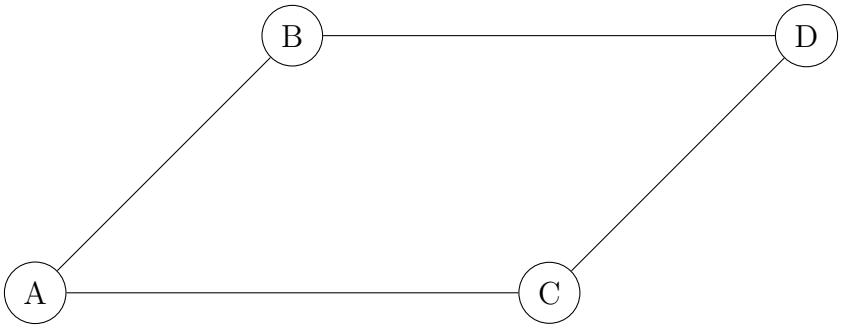
3.1 Grafos: Uma Apresentação

Um grafo (Maculan, 1987) G é uma estrutura matemática que modela relações entre objetos, sendo definido por:

- Um conjunto de vértices V.
- Um conjunto de arestas E, em que cada aresta conecta dois vértices.

Os grafos podem ser classificados como direcionados ou não, ponderados ou não, e simples ou multigrafos (quando permitem laços). A Figura 3.1 ilustra um exemplo genérico.

Figura 3.1: Exemplo de um grafo.



Fonte: o autor, via LaTeX.

3.1.1 Conceitos Básicos

1. **Grau de um Vértice:** Número de arestas incidentes; em grafos direcionados, distinguem-se grau de entrada e de saída.

2. **Caminho:** Sequência de vértices consecutivos, conectados por arestas.

3. **Ciclo:** Caminho que se inicia e termina no mesmo vértice.

4. **Grafo Conectado:** Existe um caminho entre todo par de vértices.

5. **Árvore:** Grafo conectado sem ciclos.

6. **Árvore Geradora:** Subgrafo que é uma árvore e abrange todos os vértices do grafo original.

3.1.2 Representações de Grafos

1. **Matriz de Adjacência:** Matriz quadrada em que a entrada a_{ij} indica o peso da aresta entre os vértices i e j.

2. **Lista de Adjacência:** Lista associando a cada vértice seus vizinhos.

3. **Matriz de Incidência:** Matriz simétrica (para grafos não direcionados) que indica as conexões entre os pontos.

3.1.3 Algoritmos de Grafos

Para resolver problemas como o *Steiner Tree Problem* (STP, problema da árvore de Steiner), são fundamentais:

1. **Algoritmos de Busca:** BFS (Busca em Largura) e DFS (Busca em Profundidade).
2. **Algoritmos de Caminho Mais Curto:** Dijkstra e Bellman-Ford.
3. **Algoritmos de Árvore Geradora Mínima:** Kruskal e Prim.

3.1.4 Classes de Grafos

1. **Grafos Bipartidos:** Vértices divididos em dois conjuntos disjuntos, com arestas ligando vértices de conjuntos distintos.
2. **Grafos Planos:** Podem ser desenhados no plano sem que as arestas se cruzem.
3. **Grafos Completos:** Todo par de vértices distintos é conectado por uma aresta única.

3.1.5 Aplicabilidade da Teoria dos Grafos

Na engenharia eletrônica, grafos auxiliam na análise e no projeto de circuitos, facilitando a identificação de conexões e fluxos. Em engenharia de sistemas, sua modelagem revela gargalos e otimiza processos. Na otimização e programação matemática, algoritmos baseados em grafos solucionam problemas que vão da logística à gestão de redes. Ademais, na gestão de negócios, a representação gráfica de cadeias de suprimentos e relações organizacionais orienta decisões estratégicas.

3.2 Introdução ao Problema da Árvore de Steiner

Com a representação de sistemas reais via grafos, o Problema da Árvore de Steiner (STP) torna-se mais inteligível. Formalmente, o STP consiste em encontrar a

rede interconectada mínima para um conjunto de pontos (terminais) em um grafo ponderado (Dreyfus e Wagner, 1971). Diferente do Problema da Árvore Geradora Mínima, que conecta todos os vértices, o STP permite a inclusão de vértices adicionais (pontos de Steiner) para reduzir o custo total das conexões.

Uma formulação matemática por programação linear inteira é:

$$\begin{aligned} \text{minimizar} \quad & \sum_{e \in E} w_e \, x_e, \\ \text{sujeito a} \quad & \sum_{e \in \delta(S)} x_e \geq 1, \quad \forall S \subset V,\ T \subset S,\ S \neq V, \\ & x_e \in \{0, 1\}, \quad \forall e \in E, \end{aligned}$$

em que V é o conjunto de todos os vértices (ou nós) do grafo; T é um subconjunto fixo de vértices dentro de V, chamado de terminais (são "pontos obrigatórios", que precisam estar todos ligados pela solução); S é um conjunto genérico de vértices usado para indexar as restrições de corte; w_e é o peso da aresta e; x_e indica sua inclusão na solução e $\delta(S)$ representa as arestas com uma extremidade em S.

O STP é NP-difícil, sendo sua versão de decisão NP-completa. Entre as abordagens para solucioná-lo estão:

1. **Algoritmos Exatos:** Métodos de ramificação, planos de corte e programação linear inteira.

2. **Algoritmos Heurísticos:** Técnicas que geram soluções quase ótimas, como adaptações do algoritmo de Kruskal.

3. **Algoritmos de Aproximação:** Métodos com garantias de qualidade em relação à solução ótima, por exemplo, o algoritmo de aproximação 2 para o STP métrico.

3.2.1 Aplicabilidade do STP

- Na engenharia eletrônica, o STP é crucial para o projeto de placas de circuito impresso e circuitos integrados, permitindo a redução da complexidade do

cabeamento e a otimização dos sinais.

- No projeto de redes, ele contribui para a criação de infraestruturas de comunicação eficientes e tolerantes a falhas.
- Em VLSI (*very large scale integration*), sua aplicação resulta em *layouts* compactos e de alto desempenho.
- Em planejamentos de transporte, Ademais, o problema encontra espaço otimizando a distribuição de recursos e serviços.

3.3 Conexão com o Problema da Árvore de Steiner

A teoria dos grafos oferece a base para modelar interconexões complexas e, nesse contexto, o Problema da Árvore de Steiner revela sua importância prática. É possível formalizar o problema considerando duas abordagens:

1. **Grafos Não Direcionados:** Seja $G_u = (S, E)$ um grafo conectado, em que S é o conjunto de nós (finito ou infinito) e cada aresta $a = [i, j] \in E$ (com $i \neq j$) tem comprimento $l_{i,j} > 0$. Dada uma partição $X \cup Y = S$ com $X \cap Y = \emptyset$, o objetivo é encontrar um subconjunto $Z \subseteq E$ tal que:

   (a) Todos os nós de X estejam interligados por caminhos formados por arestas de Z.

   (b) $\sum_{a \in Z} l_a$ seja minimizado.

   Dessa formulação derivam: o problema do caminho mais curto (quando $|X| = 2$) e a árvore de menor custo (quando $X = S$).

2. **Grafos Direcionados:** Seja S particionado em $\{s\}$, S_0 e S_f, em que s é a raiz (ponto para o qual não "chegam" arestas). O problema consiste em encontrar caminhos dirigidos de s a cada vértice de S_0 com custo total mínimo, considerando S_f como vértices opcionais.

A representação de sistemas por meio de grafos facilita a análise de relações e dependências entre entidades, enquanto o Problema da Árvore de Steiner ilustra como essa modelagem pode ser aplicada para otimizar redes reais. Essa sinergia entre teoria e prática reforça a importância dos grafos na resolução de problemas complexos em engenharia, tecnologia e gestão.

Capítulo 4

Programação matemática

Ao ouvir o termo "programação matemática" muitos o associam à codificação em linguagens como C, Java ou Python. Contudo, neste contexto, a programação refere-se a planejamento — uma ferramenta revolucionária que permite transformar objetivos abstratos em modelos e algoritmos, conduzindo não apenas a uma solução, mas à solução ótima para sistemas complexos.

4.1 Breve histórico da Programação Linear

A programação linear (PL)[1], por exemplo, consolidou-se como um marco na ciência da decisão (Dantzig, 2002). Seu desenvolvimento, inicialmente confidencial durante a Segunda Guerra Mundial, foi divulgado em 1947. Nesse contexto, pioneiros da ciência da decisão converteram metas abstratas em estratégias concretas por meio de modelos lineares, otimizando as refeições de soldados que estavam na guerra, tanto nutritivamente quanto o seu custo.

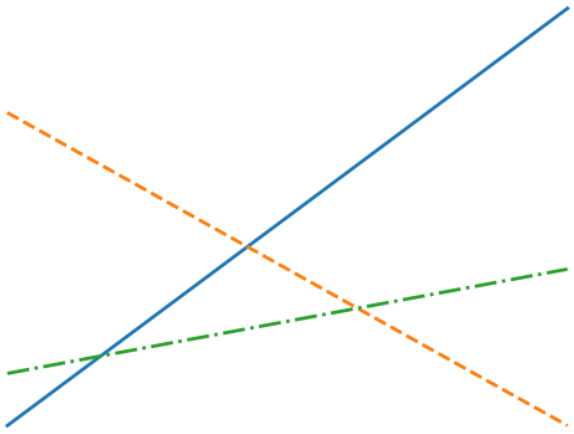
A PL evoluiu juntamente com o avanço da tecnologia computacional, permitindo a formulação e resolução de problemas reais em áreas tão diversas como a indústria, logística, economia e engenharia. Matemáticos como George Dantzig e Leonid Kantorovich foram fundamentais nesse processo, deixando um legado que perdura na aplicação prática dos métodos de otimização.

[1] Caso particular em que a função objetivo é linear.

4.2 Fundamentos da Programação Matemática

A programação matemática consiste na otimização de uma função objetivo sujeita a restrições (equações ou inequações) (Goldfarb e Todd, 1989). A Figura 4.1 ilustra um exemplo típico de restrições lineares, evidenciando como tais condições delimitam o espaço viável de soluções. A partir das curvas de restrição, forma-se um polígono interno fora do qual as soluções são inviáveis.

Figura 4.1: Exemplo de restrições lineares representadas graficamente. A área interna às retas define o conjunto de soluções viáveis.



Fonte: o autor, via *python*.

4.2.1 Exemplos Clássicos

Problema do Transporte

Considere uma empresa que deve transportar um produto de m origens para n destinos. Se a_i unidades estão disponíveis na origem i e b_j unidades são exigidas no destino j, com $\sum_{i=1}^{m} a_i = \sum_{j=1}^{n} b_j$, e o custo de transporte de i a j é c_{ij}, o problema

é formulado como:

$$\begin{aligned} \text{minimizar} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\, x_{ij}, \\ \text{sujeito a} \quad & \sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1, \ldots, m, \\ & \sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1, \ldots, n, \\ & x_{ij} \geq 0. \end{aligned}$$

Problema da Dieta

O objetivo é determinar a dieta de menor custo que satisfaça m restrições nutricionais, considerando n alimentos disponíveis. Se c_j representa o custo do alimento j, b_i a necessidade mínima do nutriente i e a_{ij} a contribuição nutricional do alimento j para o nutriente i, temos:

$$\begin{aligned} \text{minimizar} \quad & \sum_{j=1}^{n} c_j\, x_j, \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij}\, x_j \geq b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0. \end{aligned}$$

Problema da Mistura de Produtos

Uma empresa produz n produtos utilizando m recursos limitados. Seja c_j o lucro por unidade do produto j, b_i a quantidade disponível do recurso i e a_{ij} a quantidade do recurso i necessária para produzir o produto j. O problema de maximização do lucro é formulado como:

$$\begin{aligned} \text{maximizar} \quad & \sum_{j=1}^{n} c_j\, x_j, \\ \text{sujeito a} \quad & \sum_{j=1}^{n} a_{ij}\, x_j \leq b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0. \end{aligned}$$

4.3 O Método Simplex

O Método Simplex é um algoritmo utilizado para resolver problemas de programação linear. Foi desenvolvido inicialmente por George Dantzig, com importantes contribuições de Leonid Kantorovich, no período pós-Segunda Guerra Mundial. A abordagem do Simplex parte da transformação do problema original em sua forma padrão, onde as restrições lineares definem um conjunto convexo de soluções viáveis cujos vértices são candidatos à solução ótima.

A partir de um ponto viável inicial, o algoritmo realiza uma busca iterativa que consiste em avaliar no vértice em questão se os adjacentes oferecem solução melhor e passar para o vértice com melhor resposta até que se chegue em um superior aos vizinhos.

4.3.1 Formulação Matemática e Abordagem Geométrica

Considere o problema genérico de programação linear:

$$\begin{aligned} \text{minimizar} \quad & z = \sum_i c_i x_i, \\ \text{sujeito a} \quad & \sum a_{1j} x_i = b, \\ & \vdots \\ & \sum a_{ij} x_i = b, \\ & x \geq 0. \end{aligned}$$

Este pode ser representado matricialmente por:

$$\begin{aligned} \text{minimizar} \quad & z = c^T x, \\ \text{sujeito a} \quad & Ax = b, \\ & x \geq 0. \end{aligned}$$

O conjunto de soluções viáveis $P = \{x \mid Ax = b,\ x \geq 0\}$ forma um poliedro, cujos vértices são candidatos à solução ótima. Em problemas de grande escala,

a resolução gráfica torna-se inviável, justificando o uso do Método Simplex para explorar o espaço de soluções de forma iterativa.

4.3.2 Implementação Computacional

O capítulo seguinte apresenta um *script* em *Python* para solucionar problemas de programação linear. Um exemplo genérico é mostrado a seguir:

```
import numpy as np
from scipy.optimize import linprog


# Coeficientes da função objetivo (maximização)
c = np.array([coeff1, coeff2, ...])


# Coeficientes das restrições (Ax <= b)
A = np.array([[coeff_A1, coeff_A2, ...],
              [coeff_A1, coeff_A2, ...],
              ...])
b = np.array([constraint1, constraint2, ...])


# Limites para as variáveis
x_bounds = [(lower_bound0, upper_bound0),
            ...,
            (lower_boundN, upper_boundN)]


# Resolver o problema utilizando o método simplex
result = linprog(-c, A_ub=A, b_ub=b, bounds=x_bounds,
                 method='simplex')
```

Capítulo 5

Diversidade da programação matemática

Naturalmente, a programação matemática e as árvores de Steiner possuem variadas aplicabilidades, seja qual for o contexto. Exige-se, apenas, que o sistema seja adequadamente modelado. Feito isso, é possível aplica-los em qualquer cenário. Dessa forma, este capítulo dedica-se a apresentar tal abrangência.

5.1 Dois produtos genéricos

Considere uma empresa especializada em fabricar produtos genéricos x_a e x_b, que são vendidos por R$ 0,02 e R$ 0,03, respectivamente. Sabe-se que o tempo que leva para produzi-los são, na montagem, $10s$ e $20s$ cada e, na rotulação, $15s$ e $5s$ cada. O limite diário da indústria é de 30.000 minutos. Sabe-se, também, que há uma restrição técnica que $5x_b \geq 2x_a$. Naturalmente, deseja-se maximizar o lucro.

Dessa forma, enxerga-se o problema como:

$$\begin{aligned} \max \quad & 0,02x_a + 0,03x_b \\ \text{s.a.} \quad & 10x_a + 20x_b \leq 30.000 \\ & 15x_a + 5x_b \leq 30.000 \\ & x_a, x_b \geq 0. \end{aligned}$$

Então, matricialmente,

$$\begin{pmatrix} 10 & 20 \\ 15 & 5 \end{pmatrix} x \leq \begin{pmatrix} 30.000 \\ 30.000 \end{pmatrix},$$
$$\text{em que } x = \begin{pmatrix} x_a \\ x_b \end{pmatrix}$$

Sendo assim, preenche-se o *script* genérico com os coeficientes necessários. Note que os coeficientes da função objetivo estão negativos, isso se dá pois o *script* resolve o problema de minimizar, logo, utiliza-se da propriedade $\min -F = \max F$. Como resultado, obteve-se $x_a = 1.800$ e $x_b = 600$, gerando lucro máximo de 54. Via *script*, o resultado foi o mesmo. Recortando apenas as linhas que definem coeficientes, assim fica o código:

```
# Define the coefficients of the objective function to be maximized
c = np.array([-0.02, -0.03])


# Define the coefficients of the inequality constraints (Ax <= b)
A = np.array([[10, 20],
              [15,  5]])
b = np.array([30000, 30000])
```

5.2 Problema do armazenamento — modelo de Charnes, Cooper e Miller

As condições iniciais tratam da estrutura de armazém e estoque inicial. O problema envolve n períodos que referem-se ao horizonte de planejamento. Segue lista de restrições e variáveis:

- B = a capacidade fixa do armazém;
- A = estoque inicial;
- x_j = quantia a ser comprada no período j;
- y_j = quantia a ser vendida no período j;
- p_j = preço de venda por unidade no período j;
- c_j = preço de compra por unidade no período j.

Espera-se, naturalmente, que as vendas cumulativas ao início de um período não superem o estoque inicial mais o cumulativo de compras. Isso é,

$$\sum_{j=1}^{i} y_j \leq A + \sum_{j=1}^{i-1} x_j.$$

Outra restrição esperada é a de que o armazém é incapaz de suportar além de sua capacidade, podendo ser expressada por

$$\sum_{j=1}^{i} x_j \leq B - A + \sum_{j=1}^{i} y_j.$$

Tem-se, então, a função objetivo

$$max \quad \sum_{j=1}^{n} p_j y_j - \sum_{j=1}^{n} c_j x_j.$$

Toma-se como exemplo o cenário:

$$A = 100$$

$$B = 125$$

$$n = 4$$

Período (n)	Compra (c_j)	Venda (p_j)
1	25	20
2	25	35
3	25	30
4	35	25

Tabela 5.1: Preços de compra e venda por período no exemplo.

Nesse caso, somente por fins de notação, utilizar-se-á $y_j =$"x_{4+j}", para que as variáveis sejam $x_1, ..., x_8$. Então, tem-se como objetivo (vide tabela 5.1)

$$\max \quad 20x_5 + 35x_6 + 30x_7 + 25x_8 - 25(x_1 + x_2 + x_3) - 35x_4.$$

$$= \min \begin{pmatrix} 25 \\ 25 \\ 25 \\ 35 \\ -20 \\ -35 \\ -30 \\ -25 \end{pmatrix} x$$

E as restrições podem ser decompostas em

$$
\begin{aligned}
s.a. \quad & x_5 \leq A \\
& x_5 + x_6 - x_1 \leq A \\
& x_5 + x_6 + x_7 - x_1 - x_2 \leq A \\
& x_5 + x_6 + x_7 + x_8 - x_1 - x_2 - x_3 \leq A \\
& x_1 - x_5 \leq B - A \\
& x_1 + x_2 - x_5 - x_6 \leq B - A \\
& x_1 + x_2 + x_3 - x_5 - x_6 - x_7 \leq B - A \\
& x_1 + x_2 + x_3 + x_4 - x_5 - x_6 - x_7 - x_8 \leq B - A.
\end{aligned}
$$

Que podem ser vistas matricialmente como

$$
\begin{pmatrix}
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
-1 & -1 & 0 & 0 & 1 & 1 & 1 & 0 \\
-1 & -1 & -1 & 0 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 \\
1 & 1 & 1 & 0 & -1 & -1 & -1 & 0 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1
\end{pmatrix}
x \leq
\begin{pmatrix}
100 \\ 100 \\ 100 \\ 100 \\ 25 \\ 25 \\ 25 \\ 25
\end{pmatrix}.
$$

Como resultado, o lucro máximo é de $4.375,00$ (Charnes et al., 1959).

5.3 Decisões de investimento — modelo de Weingartner

Trata-se de distribuir um orçamento em diferentes investimentos oriundos de propostas (Svymbersky, 1967). Nesse modelo, os projetos não podem ser meio

aceitos ou setenta por cento recusados. Por isso, atrelada à decisão de investir ou não no projeto, está uma variável binária. Suas variáveis são

- c_{tj} = custo do projeto j no período t;
- C_j = teto orçamentário do projeto j;
- b_j = valor presente de todas as receitas e custos associados ao projeto j;
- x_j = fração do projeto j empreendida.

Tem-se como função objetivo

$$max \quad \sum_{j=1}^{n} b_j x_j.$$

Cujas restrições são

$$\sum_{j=1}^{n} c_{tj} x_j \leq C_t.$$

Além das restrições óbvias, como as não-nulidades ou $0 \leq x_j \leq N$, onde N é o número de vezes que se pode pegar o projeto.

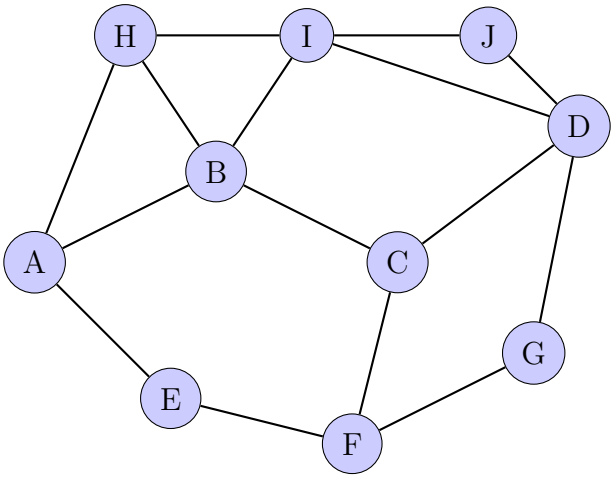
5.4 Alocação de recursos - modelo de Ijiri, Levy e Lyon

Trata-se do planejamento financeiro dentro de um orçamento (Ijiri et al., 1963). O modelo leva em consideração a planilha de balanço contábil e transações. Ao contrário do modelo de Charnes, Cooper e Miller, este não faz projeções de períodos futuros, apenas o imediatamente seguinte. Quanto às restrições, os autores estabelecem constantes-parâmetro que servem como guia para despesas específicas. Em comparação aos demais modelos, este é consideravelmente mais complexo. Apesar disso, a programação linear mostra-se útil e adaptável a cenários mil, dependendo da capacidade de modelar adequadamente o sistema.

5.5 Problema da diligência

Um caminhoneiro recebe de sua empresa o ofício de transportar determinados bens. Há um número determinado de cidades em que ele, obrigatoriamente, precisa passar e sabe-se a distância entre cada um dos municípios da região em que ele circula. Deseja-se obter a rota em que o motorista percorra as cidades necessárias acumulando a menor distância possível. Isso é, de menor custo. A Figura 5.1 representa um mapa genérico, de onde se tiraria uma rota.

Figura 5.1: Exemplo genérico de grafo representando um mapa com cidades e estradas.



Fonte: o autor, via LaTeX.

Em uma realidade como essa, é possível modelar o problema em um grafo ponderado. A decisão de ser orientado ou não, depende do contexto, isto é, há estradas que seguem apenas um sentido ou não. Em todo caso, é possível contemplar ambos, e a ideia central é submeter o grafo a um algoritmo de busca para a melhor árvore geradora do grafo que contém as cidades necessárias, dado um ponto de partida inicial. Ademais, se o contexto exige que o caminhoneiro retorne à central, o objetivo torna-se obter um ciclo otimizado.

Dessa forma, utilizando como os valores de peso dos arcos as distâncias entre municípios, percebe-se a versatilidade dos grafos em um problema de rota. Então, em um caso como esse é possível aplicar algoritmos de programação dinâmica discreta.

5.6 Sistemas de Transmissão de Potência em Redes Elétricas

Uma aplicação prática das árvores de Steiner e da programação matemática ocorre no planejamento de sistemas de transmissão de potência, em que a rede elétrica é modelada como um grafo. Nesse contexto, os pontos de Steiner passam a representar objetivos finais — isto é, a localização ótima para torres, centrais e linhas de transmissão que garantam a estabilidade e eficiência do sistema.

O desafio de expandir a rede elétrica para regiões remotas exemplifica a complexidade do problema. No caso real do Brasil, a gestão da rede envolve cerca de 3,5 milhões de variáveis e 3,2 milhões de restrições em uma formulação de programação inteira mista. Essa dimensão inviabiliza soluções triviais e exige equipes especializadas, infraestrutura computacional robusta e investimentos muitíssimo altos.

Entre as abordagens de planejamento de longo prazo, destacam-se os algoritmos heurísticos (Gonzaga et al., 1973; Gonzaga, 1973). A programação heurística busca soluções práticas e eficientes para problemas complexos, utilizando regras e métodos iterativos que, embora não garantam a solução ótima, proporcionam resultados satisfatórios em tempo adequado.

5.7 Aumentando o nível: programação não-linear

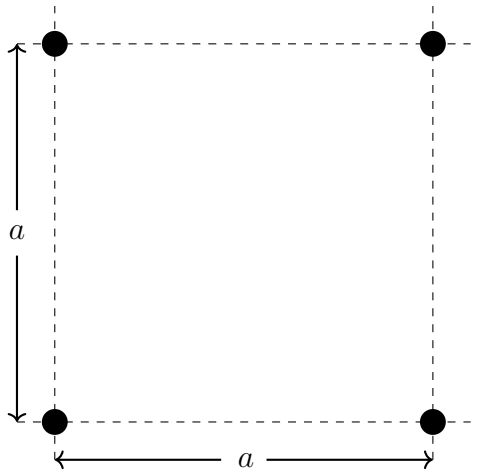
Nem tudo são flores, nem mesmo no escopo deste trabalho. Ainda que se busque modelar um problema da forma mais simples com um objetivo e restrições lineares, eventualmente a demanda pede algo mais sofisticado[1].

Um bom e clássico exemplo de problema de programação não-linear envolvendo grafos é o de se otimizar rodovias ligando cidades. Aqui, tem-se quatro municípios dispostos como pontos de um quadrado de aresta a, conforme a Figura 5.2. O objetivo é obter o trajeto de menor extensão que conecte os quatro pontos — isto é,

[1]Originalmente, era a intenção deste trabalho que o problema proposto no capítulo 6 fosse resolvido via programação linear, pelo método simplex. Entretanto, não foi possível.

cada cidade deve ter acesso às demais três. Entenda-se que pode haver cruzamentos, e que não há nenhuma outra restrição quanto à forma dos traçados.

Figura 5.2: Ilustração do exemplo das quatro cidades. Cada ponto é uma cidade, a é o comprimento da aresta do quadrado.



Fonte: o autor, via LaTeX.

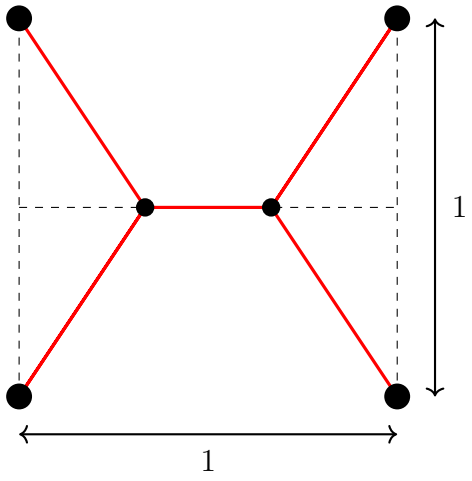
Intuitivamente percebe-se que o problema pode ser visto como um grafo. Sendo assim, há um magnífico recurso a ser aproveitado aqui: os pontos de Steiner. O conceito (neste tipo de cenário, inclusive) é bem explorado em (Bern e Graham, 1989). Trata-se de pontos "imaginários" ou extras que são adicionadas ao grafo, que podem ser úteis. Aqui, é como se a estrada, além de passar pelas quatro cidades, passarão por esses pontos de Steiner e serão mais curtas com isso.

Sabe-se de antemão que utilizar-se-ão dois pontos de Steiner. Estes serão alinhados em $y = a/2$. Busca-se, então, as coordenadas x desses pontos de Steiner, conforme ilustrado na Figura 5.3. Naturalmente, a distância entre os pontos de Steiner será menor que a, de modo que a estrada será o seguinte (a partir daqui, $a = 1$).

Chamando a distância entre os pontos de Steiner de d, é possível decompor o comprimento total estrada como sendo

$$\text{estrada} = d + 4\sqrt{\left(0 - \frac{1-d}{2}\right)^2 + (0-0,5)^2}. \tag{5.1}$$

Figura 5.3: Ilustração do exemplo das quatro cidades e sua estrada (vermelho) passando pelos dois pontos de Steiner em posições genéricas.



Fonte: o autor, via LaTeX.

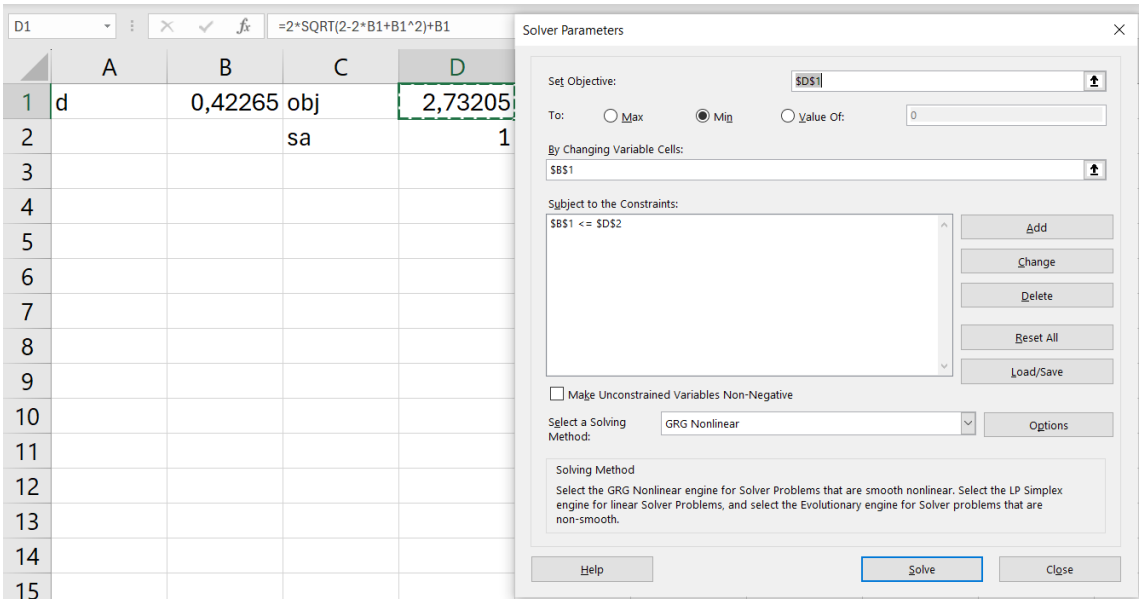
Dessa forma, tem-se formulado o problema de otimização não-linear:

$$\min \quad d + 2\sqrt{2 - 2d + d^2} \tag{5.2}$$

$$\text{s.a.} \quad d < 1. \tag{5.3}$$

Para solucionar esse problema, uma boa alternativa é o *solver* do Excel. Para isso, uma célula é separada para servir como a variável d, outra contém a função objetivo e outra com o valor 1 para servir de restrição. A solução ótima é $d = 2,732$, visto na Figura 5.4.

Figura 5.4: Excel sendo utilizado para solução do problema de otimização não-linear.



Fonte: o autor.

Capítulo 6

O problema em eletrônica

Chega-se, então, ao pontoem que se modela um problema de eletrônica. Aqui, deseja-se encontrar um método que minimize o comprimento de fios no projeto de uma suposta placa impressa. Para isso, divide-se o desafio em duas grandes etapas: a "tradução" de um circuito para um grafo; e a redação de restrições adequadas.

6.1 Circuitos eletrônicos planares em grafos

Há, em certas fontes de literatura acadêmica em eletrônica, um método de se representar circuitos em grafos, para fins de se obter equações de estado mais rapidamente. Contudo, não é esse o desejo em questão, por isso, uma inédita (até onde se sabe) abordagem será adotada aqui.

Como grafos são vistos por nós e arestas, entende-se o circuito eletrônico da seguinte forma:

- Nós: são os terminais, onde um fio se junta a um componente, ou onde vários fios se cruzam;
- Arcos: são os fios e componentes, devidamente classificados. É possível dividir o conjunto de arcos em dois fazendo-os orientados (um sentido indica que é fio e o outro, componente). Os arcos são ponderados com valores que representam

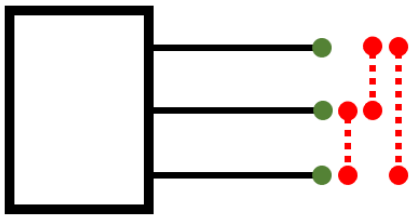
seus comprimentos.

Entretanto, dividir de tal forma não contempla uma fração útil dos circuitos planares, pois, enxerga-se um problema na hora de traduzir um componente com mais de dois terminais. Para isso, utilizar-se-ão nós em cada pino do componente, acrescentando mais arcos do tipo componente, fixando as distâncias par a par entre cada terminal. Analisa-se, especialmente, dois casos que, genericamente, contemplam tal problemática.

6.1.1 Transistores, potenciômetros e semelhantes

Esses são os componentes que possuem três terminais (vide Figura 6.1). Nesses casos, um componente tem três nós e três arcos fixando as distâncias entre eles.

Figura 6.1: Exemplo genérico de um componente de três terminais com os arcos representados em vermelho tracejado e os nós de verde.



Fonte: o autor, via PowerPoint.

6.1.2 Circuitos integrados

Nesses, o número N de terminais é par, com $N/2$ de um lado e $N/2$ do outro. Importa que se saiba uma relação em função de N para o número de arcos necessários (Figura 6.2). Para facilitar os cálculos, divide-se o problema em arcos "externos" e arcos "internos"[1]. Os chamados externos são os arcos que conectam terminais que estão do mesmo lado do CI, enquanto os internos conectam de um lado a outro.

[1]Tal nomenclatura é apenas didática, com fins de tornar os cálculos mais organizados, em etapas.

Para os internos, basta escolher um dos lados e — pino a pino — ligar um a todos os outros que estão do outro lado. Ao percorrer todos os terminais de um lado, todos os arcos internos estarão estabelecidos. Logo, há $N/2$ arcos internos para cada pino, percorrido $N/2$ vezes, resultando em

$$I = (N/2)^2. \tag{6.1}$$

Para os externos, inicia-se por um pino, estabelecendo seus arcos com os demais que estão do mesmo lado. Ao passar para o próximo, será necessária uma conexão a menos. No primeiro, há $N/2 - 1$ arcos e zero no último, pois seus arcos foram estabelecidas pelos anteriores. Assim, para cada lado há $\sum_{i=1}^{N/2-1} i$ arcos. Como há dois lados, a relação para os externos é

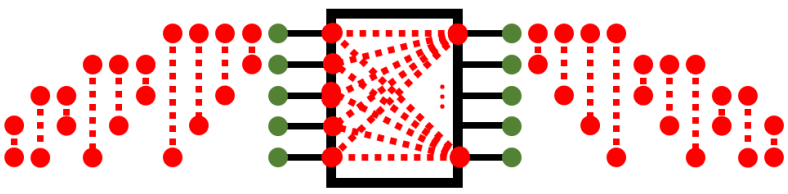
$$E = 2 \sum_{i=1}^{N/2-1} i. \tag{6.2}$$

Logo, em casos de elementos com mais de dois terminais, tendo três terminais, serão necessários representar 3 arcos de componente. Sendo circuitos integrados de N terminais, necessitar-se-á de

$$A = I + E = (N/2)^2 + 2 \sum_{i=1}^{N/2-1} i. \tag{6.3}$$

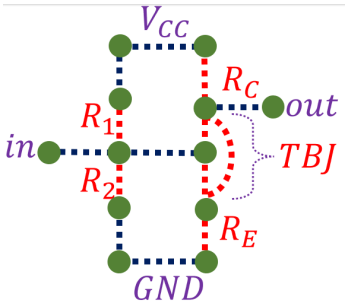
Um exemplo feito a partir de tal formulação é visto na Figura 6.3.

Figura 6.2: Exemplo genérico de um componente de dez terminais com os arcos representados em vermelho tracejado e os nós de verde. Apenas alguns dos arcos internos foram representados.



Fonte: o autor, via PowerPoint.

Figura 6.3: Exemplo genérico da representação em grafo de uma configuração emissor comum com transistor bipolar de junção. O circuito possui quatro resistores e um transistor. Arcos de fio estão em azul, arcos de componente estão em vermelho, nós estão em verde.



Fonte: o autor, via PowerPoint.

6.2 Objetivo e restrições

Cada nó terá uma coordenada cartesiana, e o posicionamento de cada um deles é o que será otimizado pelo método. Além disso, levar-se-á em consideração a matriz de incidência do grafo do circuito, M, em que

$$m_{ij} = \begin{cases} 0, & \text{se não há conexão entre os nós, ou } i = j; \\ 1, & \text{se a conexão entre } i \text{ e } j \text{ é de fio;} \\ -1, & \text{se a conexão é tipo componente.} \end{cases}, \tag{6.4}$$

$$M = [m_{ij}]. \tag{6.5}$$

Nesse caso, como o desejo é minimizar o comprimento total de fios, espera-se que a função objetivo seja algo do tipo

$$\min \quad 1/2 \sum \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad , \quad m_{ij} = 1. \tag{6.6}$$

Isso é, minimizar o somatório das distâncias euclidianas dos pontos que são ligados por fios, dividido por dois pois $d(i, j) = d(j, i)$.

Boa parte do esforço quando se trabalha com programação matemática é formalizar restrições para evitar respostas óbvias e que não refletem a realidade do algoritmo. Por isso, é necessário adotar as seguintes restrições para o problema:

- Componentes de comprimento fixo: não se deseja que o algoritmo "estique" ou contraia resistores, logo, espera-se algo como

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} = l \quad , \quad m_{ij} = -1. \tag{6.7}$$

- Sem sobreposição de pontos: caso contrário, o algoritmo poderia muito bem alocar todos os nós no mesmo local. Logo, $x_1 \neq x_2 \neq ... \neq x_n$ e $y_1 \neq ... \neq y_n$.

- Comprimento mínimo de fio: adotar-se-á um comprimento singelo mínimo de fio para conectar os componentes e evitar que simplesmente não existam fios na solução. Então,

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq f_{min} \quad , \quad m_{ij} = 1. \tag{6.8}$$

Sendo assim, percebe-se que o problema não é linear. Linearizá-lo seria uma opção, porém apenas o tornaria mais complexo. Sendo assim, tem-se em mãos um problema de programação não-linear.

6.3 O exemplo proposto

Talvez o circuito mais trabalhado no curso de eletrônica, a configuração emissor comum com um transistor bipolar de junção[2] (Figura 6.4) é o exemplo em questão para testar a formulação aqui feita, cuja matriz de incidência é vista na Tabela 6.1.

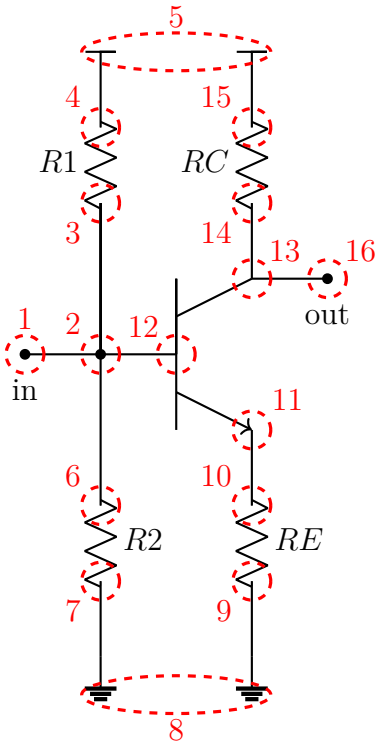
- Numera-se, então, os pontos que serão utilizados no passo-a-passo.

- Com isso feito, o próximo passo é montar a matriz de incidência de tal circuito, segundo as orientações *acima*.

- Assim, resta apenas encontrar um solucionador capaz de fornecer a solução ótima.

[2]Para simplificar o exemplo-teste, desconsidera-se capacitores de acoplamento e desacoplamento que normalmente se tem num circuito desse.

Figura 6.4: Circuito cobaia para o passo-a-passo proposto neste trabalho. Em vermelho, estão os pontos do grafo numerados.



Fonte: o autor, via LaTeX.

Tabela 6.1: Matriz de incidência do circuito em questão numa tabela. Como é simétrica, omite-se o lado espelhado para facilidade na leitura.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2		0	1	0	0	1	0	0	0	0	0	1	0	0	0	0
3			0	-1	0	0	0	0	0	0	0	0	0	0	0	0
4				0	1	0	0	0	0	0	0	0	0	0	0	0
5					0	0	0	0	0	0	0	0	0	0	1	0
6						0	-1	0	0	0	0	0	0	0	0	0
7							0	1	0	0	0	0	0	0	0	0
8								0	1	0	0	0	0	0	0	0
9									0	-1	0	0	0	0	0	0
10										0	1	0	0	0	0	0
11											0	-1	-1	0	0	0
12												0	-1	0	0	0
13													0	1	0	1
14														0	-1	0
15															0	0
16																0

Capítulo 7

Uma solução

Chega-se ao momento de aplicar um solucionador ao problema proposto. A seleção de um otimizador adequado é uma etapa crítica na resolução de problemas como este, pois impacta diretamente a convergência, precisão e eficiência computacional da solução. Problemas não lineares frequentemente apresentam paisagens complexas com múltiplos mínimos locais, pontos de sela ou regiões planas, tornando o processo de otimização altamente sensível à escolha do algoritmo. Sabendo que o problema trata-se de um *layout*, é de se imaginar que, ainda que se descubra a formatação ótima, todas as suas rotações e translações também são ótimas. Ademais, sabendo a distância total de fios ótima, cada nó pode estar em posições diferentes e a variável desejada teria o mesmo valor, i.e. há, além de um ponto, uma região ótima; e/ou vários "planaltos" diferentes. O *solver* do Excel, por exemplo, foi testado e levou dias sem apresentar solução final. O uso de algoritmo sub-ótimos, que apresentam ótimos locais, são adotados em seguida. Para isso, é utilizado o *python*, cujo algoritmo está explicado a seguir.

7.1 O algoritmo

```
import numpy as np
from scipy.optimize import minimize, differential_evolution
```

```
import matplotlib.pyplot as plt
```

- `numpy`: será utilizado para operações como o cálculo da norma euclidiana.
- `scipy.optimize`: Sub-biblioteca do *SciPy* projetada para resolver problemas de otimização. Duas funções principais são importadas:
  1. `minimize`, utilizada para minimizar uma função de um único valor; e
  2. `differential_evolution`, um algoritmo de otimização global baseado em princípios da evolução genética.

  Essas permitem a aplicação de diferentes métodos de otimização para encontrar as melhores soluções para o problema em questão.
- `matplotlib.pyplot`: Biblioteca de visualização de dados que possibilita a criação de gráficos e visualizações em Python.

```
def objective_function(coords):
    x_coords = coords[:16]
    y_coords = coords[16:]

    specific_pairs = [(0, 1), (1, 2), (1, 5), (1, 11), (3, 4),
                      (4, 14), (6, 7), (7, 8), (9, 10), (12, 13),
                      (12, 15)]
    sum_of_distances= sum(np.linalg.norm([x_coords[i1]-x_coords[i2],
                            y_coords[i1] - y_coords[i2]]) for i1, i2
                            in specific_pairs)

    return sum_of_distances
```

Aqui, define-se a função objetivo, que calcula e retorna o valor da soma das distâncias dos pontos ligados por fios, a ser minimizado.

O parâmetro `coords` representa um *array* que contém as coordenadas X e Y dos pontos. Os primeiros 16 elementos do *array* correspondem às coordenadas X e os últimos 16 elementos correspondem às coordenadas Y. As coordenadas X e Y são extraídas do *array* `coords` utilizando a notação de fatiamento. As primeiras 16 entradas são atribuídas a `x_coords`, e as 16 últimas entradas são atribuídas a `y_coords`.

A função define uma lista de pares específicos de pontos — os ligados por fios—, denominada `specific_pairs`. Esta lista contém tuplas, em que cada tupla representa um par de índices de pontos. O cálculo da adição das distâncias é realizado utilizando uma expressão geradora dentro da função `sum`. Para cada par de índices $i1$ e $i2$ na lista `specific_pairs`, a norma euclidiana é calculada utilizando a função `np.linalg.norm`, que mede a distância entre os pontos (x_{i1}, y_{i1}) e (x_{i2}, y_{i2}). O resultado da soma das distâncias é armazenado na variável `sum_of_distances`, que é então retornada como o valor objetivo da função. Este valor será utilizado pelo algoritmo de otimização para determinar a configuração ideal dos pontos.

```
def constraint_function(coords):
    x_coords = coords[:16]
    y_coords = coords[16:]

    constraints = []

    specific_pairs_constraints = [
        (0.5, [(0, 1), (1, 2), (1, 5), (1, 11), (3, 4), (4, 14),
                (6, 7), (7, 8), (9, 10), (12, 13), (12, 15)]),
                # 0.5 limit WIRES
        (1, [(2, 3), (5, 6), (13, 14),
             (8, 9), (10, 12)]),  # 1 limit RESISTANCES
        (0.5, [(10, 11), (11, 12)])  # 0.5 limit TRANSISTORS
    ]
```

```
    for distance, pairs in specific_pairs_constraints:
        for i1, i2 in pairs:
            distance_constraint = np.linalg.norm([x_coords[i1] -
                                  x_coords[i2], y_coords[i1] -
                                  y_coords[i2]])
            constraints.append(distance_constraint - distance)

    return constraints
```

Aqui, é definida a função `constraint_function`, que avalia e retorna as restrições a serem impostas durante o processo de otimização. O parâmetro `coords` é um *array* que contém as coordenadas X e Y dos pontos, assim como na função anterior. As coordenadas X e Y são extraídas do *array* `coords`, sendo as primeiras 16 entradas atribuídas a `x_coords` e as 16 últimas a `y_coords`.

A função inicia a definição de uma lista vazia chamada `constraints`, que armazenará as restrições a serem aplicadas. Em seguida, é definida uma lista `specific_pairs_constraints` que contém as restrições a serem verificadas. Cada entrada da lista é uma tupla que contém a distância e uma lista de pares de índices de pontos que devem respeitar essa distância. As distâncias são os limiares de 0.5 para os fios, 1 para outros pares resistores e transistor, e 0.5 para cada perna de um transistor.

A função, então, itera sobre cada tupla na lista `specific_pairs_constraints`. Para cada distância e conjunto de pares, calcula-se a norma euclidiana entre os pontos correspondentes (x_{i1}, y_{i1}) e (x_{i2}, y_{i2}). A restrição para cada par de pontos é adicionada à lista `constraints` na forma de uma diferença entre a distância calculada e a distância especificada. Finalmente, a função retorna a lista `constraints`, que será utilizada pelo otimizador para assegurar que as condições de distância sejam atendidas durante a otimização.

```
def run_optimizer(method='SLSQP'):
```

```
initial_guess = np.random.rand(32)

bounds = [(-4, 4)] * 32

constraints = [{'type': 'eq', 'fun': constraint_function}]

if method == 'SLSQP':
    result = minimize(objective_function, initial_guess,
                      method='SLSQP', bounds=bounds,
                      constraints=constraints)
elif method == 'differential_evolution':
    result = differential_evolution(objective_function, bounds,
                                    constraints=[{'type':'eq','fun':
                                    constraint_function}])
else:
    raise ValueError("Unknown method: choose 'SLSQP' or
                     'differential_evolution'")

x_coords_optimal = result.x[:16]
y_coords_optimal = result.x[16:]

plt.figure(figsize=(8, 6))
plt.scatter(x_coords_optimal, y_coords_optimal, c='blue',
            label='Optimal Points')

for i, (x, y) in enumerate(zip(x_coords_optimal,
                                    y_coords_optimal),
                                    start=1):
    plt.text(x, y, str(i), fontsize=10, ha='center',
```

```
            va='bottom', color='red')

specific_pairs = [(0, 1), (1, 2), (1, 5), (1, 11), (3, 4),
                  (4, 14),(6, 7), (7, 8), (9, 10), (12, 13),
                  (12, 15)]
for i1, i2 in specific_pairs:
    plt.plot([x_coords_optimal[i1], x_coords_optimal[i2]],
             [y_coords_optimal[i1], y_coords_optimal[i2]],
             c='black')

specific_pairs_other_constraints = [(2, 3), (5, 6), (13, 14),
                                    (8, 9), (10, 12),
                                    (10, 11), (11, 12)]
for i1, i2 in specific_pairs_other_constraints:
    plt.plot([x_coords_optimal[i1], x_coords_optimal[i2]],
             [y_coords_optimal[i1], y_coords_optimal[i2]],
             c='green', linestyle='--')

plt.xlabel('X Coordinate')
plt.ylabel('Y Coordinate')
plt.title('Optimal Points')
plt.legend()
plt.grid(True)
plt.show()

print("Optimal coordinates:", result.x[:16], result.x[16:])
print("Minimum distance:", result.fun)
```

Neste segmento de código, é definida a função `run_optimizer`, que é responsável

por chamar o otimizador com base no método especificado pelo usuário. O parâmetro `method` é opcional e, por padrão, está definido como "`SLSQP`", que será o método escolhido. Tal método da Programação Quadrática em Mínimos Quadrados Sequenciais (SLSQP)[1] constrói um modelo quadrático da função lagrangeana, que combina a função objetivo e as restrições, e resolve esse modelo iterativamente. Ele gerencia as restrições por meio de técnicas como o Lagrangiano aumentado, que penaliza as violações. A função começa gerando um palpite inicial para as coordenadas. Este palpite é um vetor de 32 números aleatórios, que são gerados uniformemente entre 0 e 1. Embora o código mencione que este palpite pode ser substituído por um mais significativo, essa abordagem aleatória permite explorar uma ampla gama de soluções possíveis.

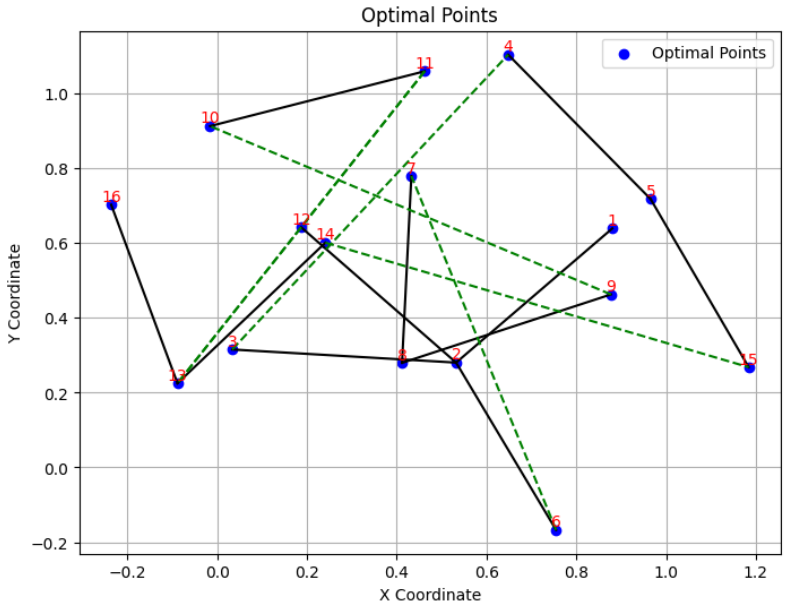
Os limites para as coordenadas são definidos como um intervalo de $[-4, 4]$ para cada uma das 32 dimensões. Isso restringe as coordenadas dos pontos otimizados a um espaço definido, o que é útil para garantir que as soluções permaneçam dentro de um intervalo delimitado, a fim de que o otimizador não decida explorar as profundezas do infinito.

As restrições são definidas como uma lista de dicionários, em que cada dicionário especifica que a função de restrição a ser utilizada é `constraint_function`, garantindo que as condições impostas anteriormente sejam atendidas durante a otimização.

Após a otimização, as coordenadas X e Y ótimas são extraídas da solução resultante. A seguir, é gerado um gráfico de dispersão (como a Figura 7.1) para visualizar os pontos ótimos. Cada ponto é marcado em azul e é rotulado com seu índice correspondente, utilizando a cor vermelha. Linhas pretas são os fios. Linhas verdes tracejadas são os componentes.

[1] Há um equivalente do método simplex para problemas de programação quadrática elaborado em (Wolfe, 1959), aproveitando a condição convexa do problema no espaço de soluções. Apesar disso, há outras propostas mais recentes com alternativas para o modelo de Wolfe, vide (Ghadle e Pawar, 2015), com exemplos resolvidos.

Figura 7.1: Um gráfico obtido ao acionar o otimizador.



Fonte: o autor, via *Python*.

7.2 A saída

Então, é hora de dizer `run_optimizer(method='SLSQP')` e lidar com as consequências. A lista de coordenadas ótimas está disposta na Tabela 7.1.

Tabela 7.1: Coordenadas (x, y) para cada ponto do grafo estabelecido após um uso do otimizador.

n	x	y
1	0.88036758	0.63951851
2	0.53237388	0.28049149
3	0.64826865	0.31558141
4	0.0336067	1.10437207
5	0.96562807	0.71800046
6	0.75501328	-0.16720451
7	0.43270768	0.77943117
8	0.41194292	0.27986251
9	0.87723114	0.46291138
10	-0.01565126	0.9132013
11	0.46232349	1.05996723
12	-0.08827256	0.64236972
13	0.18734613	0.22519539
14	0.24116435	0.60132214
15	1.18406118	0.26823721
16	-0.23596024	0.70288604

7.3 Estudando o otimizador

7.3.1 Ajustando o algoritmo

Naturalmente, para cada uso do otimizador obtém-se uma figura distinta com coordenadas distintas. E o que se faz com isso, agora? Estatística. Toma-se, então, o mesmo otimizador e aciona-lhe duas mil vezes, registrando o resultado final (i.e. a distância total ótima) a fim de se ter um bom número de observações para quaisquer inferências. Ademais, seis gráficos são extraídos para visualização. Desta forma, faz-se algumas alterações nas funções `run_optimizer`; cria-se a função `description_statistics` para calcular média, desvio padrão, variância e exibir o mínimo e o máximo; cria-se uma função destinada a plotar a grade de gráficos, `plot_results`; outra para o histograma dos resultados, `plot_histogram`; e, para estatísticas mais sofisticadas, salva-se os resultados em um `.csv`. Então, tem-se os recortes abaixo das atualizações do algoritmo.

```
def run_optimizer(method='SLSQP'):statistics
    results = []
    optimal_distances = []

    for _ in range(2000):
        initial_guess = np.random.rand(32)

        bounds = [(-4, 4)] * 32

        constraints = [{'type': 'eq', 'fun': constraint_function}]

        if method == 'SLSQP':
            result = minimize(objective_function, initial_guess,
                              method='SLSQP', bounds=bounds,
                              constraints=constraints)
```

```
        elif method == 'differential_evolution':
            result = differential_evolution(objective_function,
                                            bounds,
                                            constraints=[{'type':
                                            'eq', 'fun':
                                            constraint_function}])
        else:
            raise ValueError("Unknown method: choose 'SLSQP' or
                             'differential_evolution'")
        (result.fun)
        results.append(result.x)
        optimal_distances.append(result.fun)

    return results, optimal_distances
```

```
def description_statistics(optimal_distances):
    mean_distance = np.mean(optimal_distances)
    variance_distance = np.var(optimal_distances)
    std_dev_distance = np.std(optimal_distances)
    min_distance = np.min(optimal_distances)
    max_distance = np.max(optimal_distances)

    print(f"Mean of optimal distances: {mean_distance}")
    print(f"Variance of optimal distances: {variance_distance}")
    print(f"Standard deviation of optimal distances:
                                            {std_dev_distance}")
    print(f"Minimum optimal distance: {min_distance}")
    print(f"Maximum optimal distance: {max_distance}")
```

```
def plot_histogram(optimal_distances, bins=5):
```

```
    plt.figure(figsize=(14, 4))
    plt.hist(optimal_distances, bins=bins, color='blue',
             edgecolor='black')
    plt.title('Histogram of Optimal Distances')
    plt.xlabel('Optimal Distance')
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

```
import pandas as pd


def save_distances_to_csv_pandas(optimal_distances,
                                 filename="optimal_distances.csv"):
    df = pd.DataFrame({
        "Iteration": range(1, len(optimal_distances) + 1),
        "Optimal Distance": optimal_distances
    })
    df.to_csv(filename, index=False)
    print(f"Optimal distances saved to {filename}")
```

7.3.2 Investigando a saída

Há uma série de fatos a se observar a partir dos dois mil resultados obtidos. São eles:

1. A saída da função objetivo é extremamente consistente;
2. Apesar disso, as coordenadas — e, portanto, os *layouts* — são sempre distintas;
3. Há uma distribuição identificável no histograma dos resultados;
4. Porém sua utilidade é questionável.

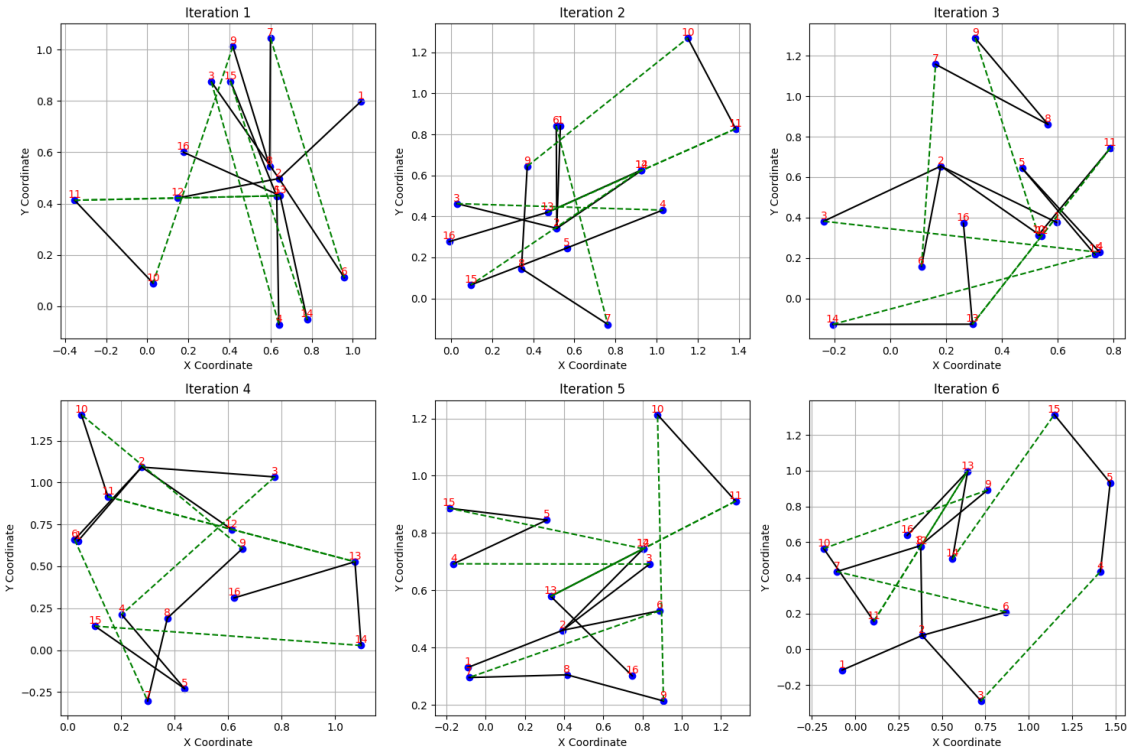
Tabela 7.2: Estatística descritiva dos resultados

	Distância ótima
Observações	2000
Média	5.500000181314856
Desvio Padrão	$8.329993826492714 \times 10^{-8}$
Variância	$6.9388797149406734 \times 10^{-15}$
Amplitude	$4.57051531 \times 10^{-7}$
Mínimo	5.500000000472287
Máximo	5.500000457523818

Fonte: o autor, via *Python*, *JASP* e LaTeX.

O primeiro dos itens é identificado na tabela com a estatística descritiva (Tabela 7.2). Se não fosse a precisão de, no mínimo, 7 casas decimais (aqui tiveram 15), não se perceberia qualquer diferença entre os dois mil resultados. O *Statistica*, *software* de estatística, recusou gerar um histograma, de tão pequena que era a variância. Os demais *software* utilizados, que não tinham tal perspicácia para identificar tal particularidade, geraram mesmo assim.

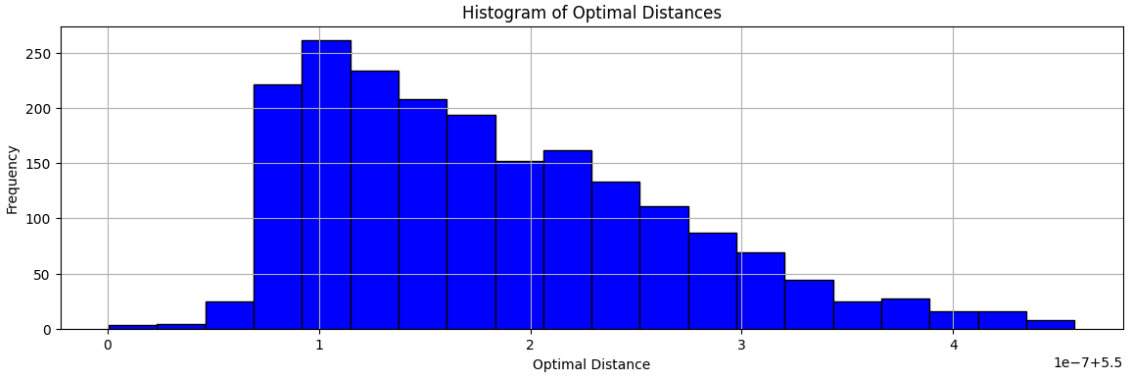
Figura 7.2: Seis gráficos dos dois mil resultados obtidos após os dois mil usos do otimizador.



Fonte: o autor, via *Python*.

O segundo item é visto na demostração de gráficos obtidos (Figura 7.2). Pelo que parece, o otimizador se concentra em condensar os pontos no intervalo determinado e respeitar o limite inferior de fios. Explicaria a consistência dos resultado. Pode ser útil; a ideia será melhor explorada a seguir.

Figura 7.3: Histograma com vinte partições dos dois mil resultados obtidos pelo otimizador.



Fonte: o autor, via *Python*.

O terceiro item é inferido a partir dos histogramas obtidos pelos programas que não se opuseram ao comando (ex.: Figura 7.3). Em outras tentativas, não se observava curva alguma. Somente quando o número de amostras subiu enormemente (para dois mil) e aumentou-se o número de partições para vinte que se pôde perceber o padrão.

A partir do *EasyFit*, programa destinado a ajuste de curvas e distribuições, percebeu-se dois bons candidatos a explicar a disposição dos dados. (1) A distribuição Gumbel-Max e (2) a distribuição Generalizada de Valores Extremos. A distribuição Gumbel-Max é uma técnica utilizada para amostrar de distribuições categóricas, ou seja, quando se tem um conjunto discreto de opções. Logo, há de ser descartada, uma vez que não se enquadra no cenário em questão, em que se tem uma variável contínua. Já a distribuição Generalizada de Valores Extremos (Provost et al., 2018) é uma família de distribuições usada para modelar o comportamento dos valores extremos (máximos ou mínimos) de um conjunto de dados. Ela unifica as distribuições de Gumbel, Fréchet e Weibull. Seus parâmetros são: localização (μ, basicamente a média dos valores), escala (σ) e forma (ξ). A Equação 7.1 descreve a

distribuição Generalizada de Valores Extremos.

$$f(x|\mu,\sigma,\xi) = \frac{1}{\sigma}\left(1+\xi\frac{x-\mu}{\sigma}\right)^{-\frac{1}{\xi}-1} e^{-\left(1+\xi\frac{x-\mu}{\sigma}\right)^{-\frac{1}{\xi}}}, \quad 1+\xi\frac{x-\mu}{\sigma} > 0 \tag{7.1}$$

O parâmetro ξ é digno de destaque, uma vez que sua variação descreve a cauda da distribuição, segundo os seguintes critérios:

- $\xi = 0$: a distribuição segue o modelo de Gumbel, que modela extremos com caudas baixas;
- $\xi > 0$: segue o modelo de Fréchet, com densidade maior nas caudas;
- $\xi < 0$: segue Weibull, com distribuição não infinita, estabelecendo limite superior.

Os valores obtidos com o ajuste da curva (Figura 7.4) são

- $\mu = 5,5$;
- $\sigma = 6,6952 \times 10^{-8}$; e
- $\xi = 7,1784 \times 10^{-5}$.

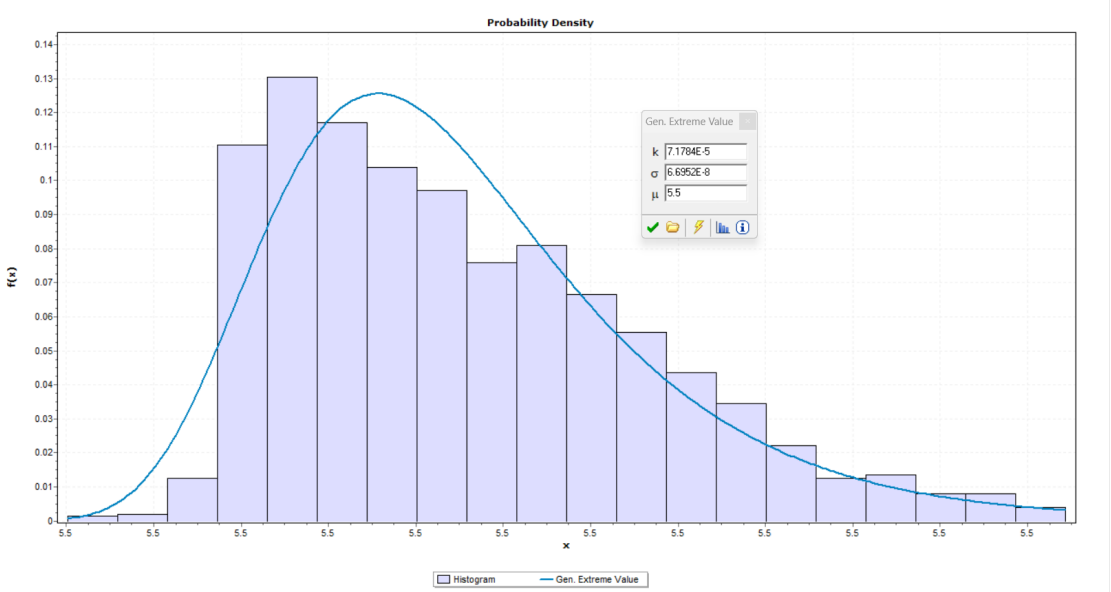
O ξ é superior a zero, mas, ao mesmo tempo, muito próximo de zero, devido à ordem de grandeza de 10^{-5}. Explica a proximidade com a distribuição de Gumbel e o teste de aderência ter sugerido Gumbel-Max como alternativa.

O quarto item pode ser inferido tanto pela estatística descritiva (variância baixíssima), quanto pela curva da distribuição. Vide Figura 7.5.

7.4 Explorando vantagens com as peças disponíveis

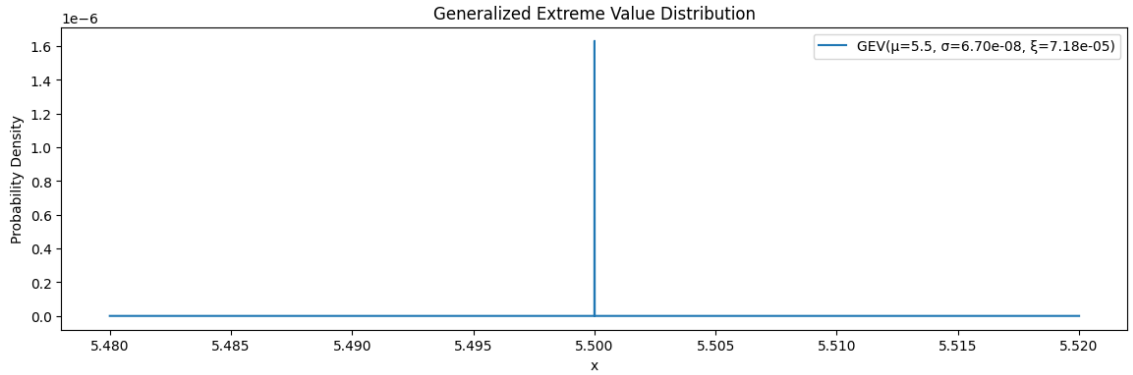
Há algumas vantagens a serem exploradas no solucionador a fim de contrapor suas fraquezas. Sua mais notável e relevante fraqueza é a mesma do modelo matemático: uma vez que os cruzamentos não foram matematizados, o otimizador não

Figura 7.4: Histograma com ajuste de distribuição.



Fonte: o autor, via *EasyFit*.

Figura 7.5: Histograma com ajuste de distribuição.



Fonte: o autor, via *Python*.

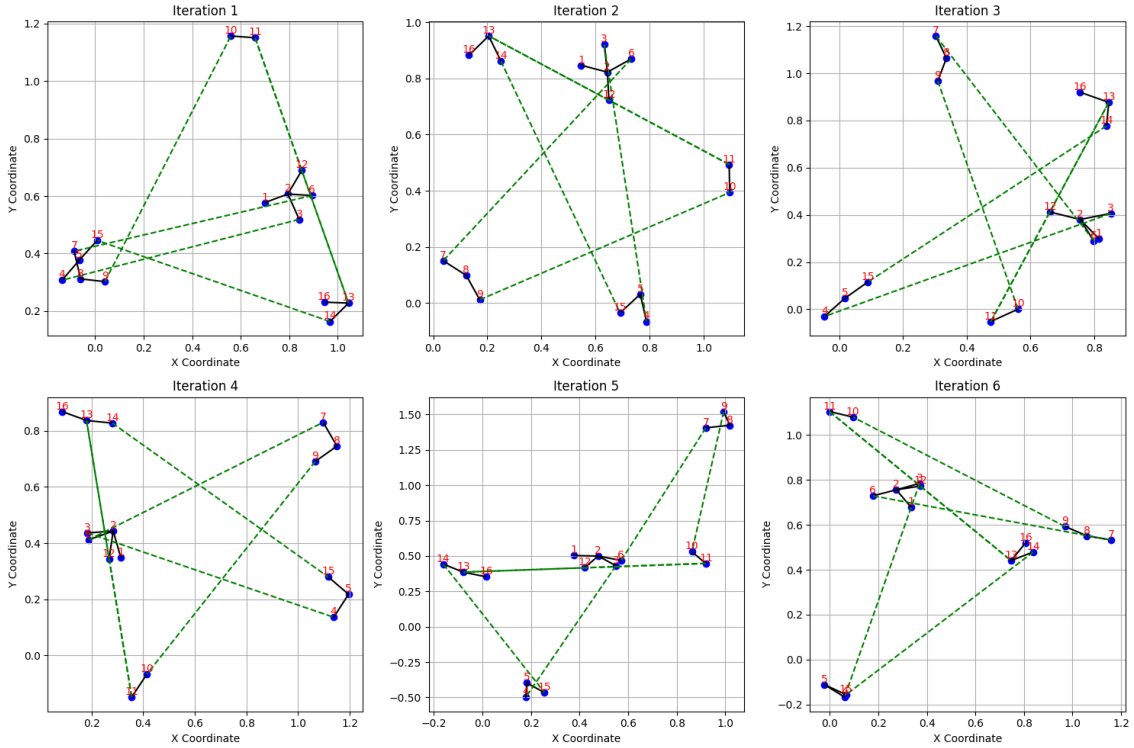
teve preocupação alguma a respeito deles. Sendo assim, praticamente todo *layout* há uma interseção de fios que, naturalmente, causariam um problema, uma vez que se espera um circuito planar. Por outro lado, aquilo que foi declarado nas restrições foi respeitosamente obedecido. Deste modo, buscar-se-á um ajuste de restrições de modo que minimize as interceptação de fios.

A ideia, em resumo, é: ser mais rigoroso nas restrições que existem e esperar um resultado consistente. Sendo rigoroso quanto ao comprimento mínimo de fio

implicaria em trechos mais curtos e menos chance de cruzamento. Sendo rigoroso quanto à coordenada máxima para x e y implicaria em menos possibilidades de *layout* que caibam no espaço.

Ao testar, verificou-se que a hipótese de fato foi útil. Nota-se, ao rodar o otimizador inúmeras vezes, que um "estilo" de *layout* foi recorrente: estes cuja trajetória de componentes (em verde) se assemelha a um ícone de estrela, que aparece nos gráficos 2, 3 e 4 da Figura 7.6. Ademais, tornou-se muito difícil enxergar cruzamentos entre fios, exceto quando alguma pluralidade de pontos há conexões entre si, onde o cruzamento não faria mal algum, sabendo que o potencial em tais pontos são iguais.

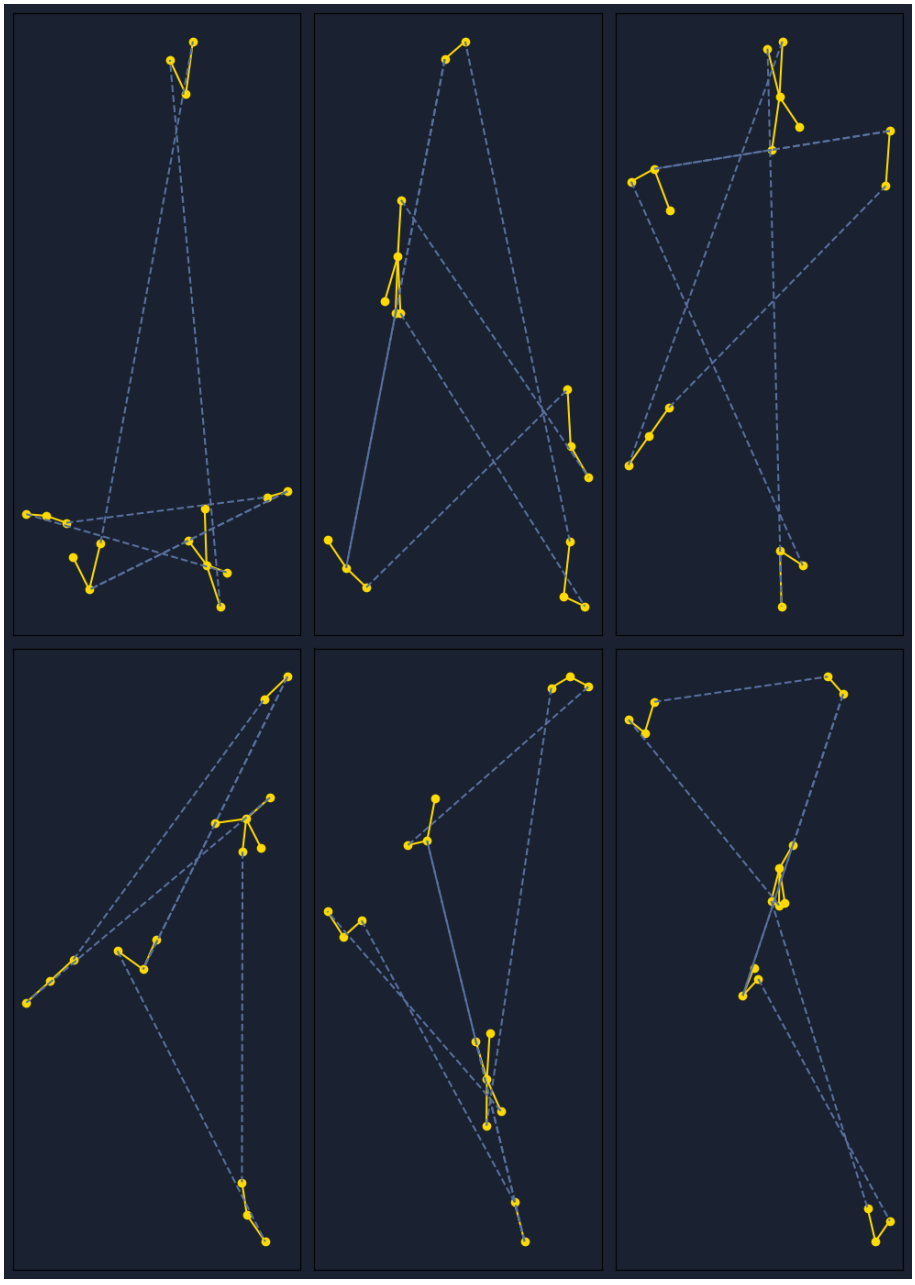
Figura 7.6: Seis *layouts* obtidos após diminuir limiar de fio e coordenadas disponíveis.



Fonte: o autor, via *Python*.

No fim, há uma estrela de componentes com pequenas estrelas de fios em seus vértices. Ao menos, o algoritmo pôde remeter à noite estrelada de Van Gogh, representado com *layouts* ótimos na Figura 7.7.

Figura 7.7: A noite estrelada dos *layouts* otimizados.



Fonte: o autor, via *Python*.

Referências

Begehr, H. e Lenz, H. (1998). Jacob steiner and synthetic geometry. In *Mathematics in Berlin*, páginas 49–54. Springer.

Bern, M. W. e Graham, R. L. (1989). The shortest-network problem. *Scientific American*, 260(1):84–89.

Charnes, A., Cooper, W. W., e Miller, M. H. (1959). Application of linear programming to financial budgeting and the costing of funds. *The Journal of Business*, 32(1):20–46.

Dantzig, G. B. (2002). Linear programming. *Operations research*, 50(1):42–47.

Dreyfus, S. E. e Wagner, R. A. (1971). The steiner problem in graphs. *Networks*, 1(3):195–207.

Ghadle, K. P. e Pawar, T. S. (2015). New approach for wolfe's modified simplex method to solve quadratic programming problems. *International Journal of Research in Engineering and Technology*, 4(1):371–376.

Goldfarb, D. e Todd, M. J. (1989). Chapter ii linear programming. *Handbooks in operations research and management science*, 1:73–170.

Gonzaga, C. (1973). *Estudo de Algoritmos de Busca em Grafos e sua Aplicaçao a Problemas de Planejamento*. PhD thesis, Tese de Doutorado, COPPE–UFRJ, Rio de Janeiro.

Gonzaga, C. C., Persiano, R. M., Carneiro, S., e Brito, S. S. (1973). Optimal planning of the expansion of a power transmission system*. *IFAC Proceedings Volumes*, 6(1):217–224. IFAC/IFORS Symposium on Systems Approaches to Developing Countries, Algiers, Algeria, 28-31 May.

Ijiri, Y., Levy, F. K., e Lyon, R. (1963). A linear programming model for budgeting and financial planning. *Journal of Accounting Research*, páginas 198–212.

Maculan, N. (1987). The steiner problem in graphs. *North-Holland Mathematics Studies*, 132:185–211.

Provost, S. B., Saboor, A., Cordeiro, G. M., e Mansoor, M. (2018). On the q-generalized extreme value distribution. *REVSTAT-Statistical Journal*, 16(1):45–70.

Soothill, G. (2010). The euclidean steiner problem. *Report, Department of Mathematical Science, Durham University, England (Undergraduate Departmental Prizes)*.

Svymbersky, P. A. (1967). Linear programming as an aid to financial decision making.

Wolfe, P. (1959). The simplex method for quadratic programming. *Econometrica: Journal of the Econometric Society*, páginas 382–398.