UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E COMPUTAÇÃO CIENTÍFICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Matheus Queiroz Mota

Otimização Multiobjetivo: teoria e aplicações

Matheus Queiroz Mota

Otimização Multiobjetivo: teoria e aplicações*

Monografia apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado II, sob a orientação da Profa. Kelly Cristina Poldi.

*Este trabalho foi financiado por PIBIC/CNPq-UNICAMP, cota 2023-2024.

Resumo

A Otimização Multiobjetivo é uma das principais áreas de pesquisa no contexto de Otimização. Um Problema de Otimização Multiobjetivo consiste em otimizar (maximizar ou minimizar) mais de uma função matemática, denominada função objetivo, sujeita a um conjunto de restrições determinadas pelo tipo de problema abordado. Pelo fato de lidarmos com múltiplas funções objetivo, que geralmente são conflitantes – ou seja, otimizar uma função pode prejudicar o valor da outra – não obtemos uma única solução ótima como ocorre em problemas com uma única função objetivo. Neste caso, obtemos um conjunto de soluções denominadas soluções de Pareto. Para resolver Problemas de Otimização Multiobjetivo são utilizados métodos específicos. Deste modo, visando compreender a teoria de problemas multiobjetivo e como resolvê-los, este trabalho busca sintetizar os principais conceitos desta área e estudar os principais métodos exatos: soma ponderada e ε-restrito e suas implementações computacionais. No fim deste trabalho, apresentamos alguns exemplos de aplicações práticas da Otimização Multiobjetivo.

Abstract

Multi-objective optimization is one of the main areas of research in optimization. A multi-objective optimization problem involves optimizing (maximizing or minimizing) more than one mathematical function, known as an objective function, subject to a set of constraints determined by the type of problem being addressed. Due to the fact that we have to deal with multiple objective functions that are generally conflicting – that is, optimizing one function can worsen the value of the other – we do not obtain a single optimal solution as we do in problems with a single objective function. In this case, we obtain a set of solutions known as Pareto solutions. To solve multi-objective optimization problems specific methods are required. Thus, aiming to understand the theory of multi-objective problems and how to solve them, this work intends to synthesize the main concepts of this area and study the two main exact methods: the weighted sum and the ε-constraint, and their computational implementations. At the end, we present some examples of practical applications of Multi-objective Optimization.

Conteúdo

1 Introdução

A Programação Multiobjetivo ou Otimização Multiobjetivo lida com problemas de otimização matemática que possuem duas ou mais funções objetivas para serem otimizadas simultaneamente tais que esses objetivos geralmente são conflitantes, ou seja, ao melhorarmos um deles causamos piora no outro.

Para exemplificar, considere uma empresa de transporte que busca maximizar o lucro de toda a sua frota. Para resolver esse problema, podemos utilizar técnicas de programação linear inteira mista e, com base na solução, obter o maior lucro possível para a empresa. No entanto, suponha que a empresa também deseja minimizar a emissão de dióxido do carbono ou gás carbônico (CO2) de sua frota. Isso caracteriza um problema bi-objetivo, onde a função lucro deve ser maximizada e a função emissão de CO2 deve ser minimizada.

Neste caso, temos um conflito entre as funções objetivas. Para maximizar o lucro da frota de transporte, teríamos que transportar mais produtos, o que resultaria em mais veículos circulando e, consequentemente, um aumento nas emissões de CO2. Deste modo, como será apresentado posteriormente, nos problemas multiobjetivo não temos apenas uma única solução, mas sim um conjunto de soluções denominadas soluções de Pareto Chankong and Haimes [1983].

Deste modo, este trabalho busca introduzir os conceitos básicos de um Problema de Otimização Multiobjetivo (POM), bem como interpretar seus resultados e como tratar tal problema computacionalmente.

Este texto está organizado da seguinte forma. O Capítulo [2] apresenta uma breve revisão dos conceitos da otimização matemática e uma introdução à teoria básica de otimização linear multiobjetivo. O Capítulo [3] apresenta dois métodos exatos clássicos da literatura para resolução de Problemas de Otimização Multiobjetivo. No Capítulo [4] são apresentados cinco problemas bi-objetivo, de modo a mostrar a fronteira de Pareto gerada pelas combinações de funções objetivas: maximização-minimização, minimização-maximização, minimização-minimização e maximização-maximização. Por fim, no Capítulo [5] é apresentado os códigos de cada um dos métodos estudados neste trabalho, bem como outros dois algoritmos para variar os parâmetros de cada método.

2 Aspectos Teóricos da Otimização Multiobjetivo

Antes de adentrar nos conceitos da Otimização Multiobjetivo, vamos introduzir brevemente os principais conceitos de Programação Linear (PL). Vale ressaltar que nesta seção é feita uma introdução dos conceitos de modelagem de um PL, caso deseje se aprofundar nos conceitos que envolvem os métodos exatos para obtenção de soluções, tais como o método Simplex ou métodos de Pontos Interiores, é recomendado a leitura da referência Arenales et al. [2006].

2.1 Conceitos de Programação Linear

Uma das principais áreas de pesquisa na matemática aplicada é a programação matemática. O objeto de estudo dessa área é a otimização de uma função, denominada função objetivo, que pode ser maximizada ou minimizada, de modo que tal função é sujeita a restrições representadas por desigualdades. Métodos de resolução de problemas de Programação Linear visam determinar soluções ótimas para os problemas que envolvam restrições lineares e uma função objetivo linear. Suas aplicações vão desde problemas de indústria até mesmo problemas relacionados a situações cotidianas, como o exemplo da dieta. Além disso, os problemas lineares podem ser resolvidos de forma eficiente por meio de métodos exatos como o Método Simplex e de Pontos Interiores.

2.1.1 Modelagem de um Problema de Programação Linear (PL)

De forma geral, podemos representar matematicamente um modelo de otimização como:

$$\min(\text{ou}\max) f(x)$$
$$\text{sujeito a: } x \in X$$

- x**:** vetor das variáveis de decisão; $x = (x_1, x_2, \cdots, x_n)^t$.
- X**:** conjunto de soluções factíveis para o problema.
- $f(x)$**:** função objetivo a ser otimizada.

Exemplo 2.1.1. *(Produção de Brinquedos)*

Pedro é proprietário de uma empresa que comercializa dois tipos de brinquedos de pelúcia: tartarugas e pandas. No processo de fabricação, são necessárias duas horas de costura e uma hora de tingimento para cada unidade de tartaruga, enquanto para os pandas são necessárias três horas de costura e uma hora de tingimento. Devido às restrições de tempo, Pedro tem apenas 220 horas disponíveis para tingimento e 240 horas para costura a cada mês.

Embora a demanda pelas tartarugas seja ilimitada, a demanda pelos pandas é de no máximo 80 unidades por mês. Dado que o custo de produção de cada tartaruga é de R$20,00 e é vendida por R$35,00, e que o panda custa R$30,00 e é vendido por R$50,00, Pedro está buscando maximizar o lucro de sua empresa.

Deste modo, Pedro elaborara um problema de programação linear para que ele possa obter o lucro máximo nas vendas de tartarugas e pandas.

Resolução:

Variáveis de decisão: podemos definir como x_1 a quantidade de Tartarugas fabricadas e x_2 a quantidade de Pandas fabricados.

Função objetivo: estamos interessados em maximizar o lucro advindo da venda das pelúcias. Portanto, a função objetivo leva em conta a diferença entre o valor vendido e o valor de produção, ou seja:

$$f(x) = 15x_1 + 20x_2$$

Restrições:

- Capacidade de costura: disponibilidade máxima de 240 horas, de modo que cada Tartaruga leva 2 horas de costura, enquanto o Panda leva 3 horas. Deste modo, tal restrição pode ser modelada como:

$$2x_1 + 3x_2 \leq 240$$

- Capacidade de tingimento: disponibilidade máxima de 220 horas, de modo que cada unidade da Tartaruga e do Panda leva 1 hora de tingimento. Portanto, esta restrição é

descrita como:

$$x_1 + x_2 \leq 220$$

- Não negatividade: é evidente que Pedro não pode vender pelúcias negativas, caso contrário estaria saindo no prejuízo. Logo, devemos ter que:

$$x_1, x_2 \geq 0$$

Deste modo, temos o seguinte problema de Programação Linear:

$$\max f(x) = 15x_1 + 20x_2$$
$$\text{sujeito a: } \begin{cases} 2x_1 + 3x_2 \leq 240 \\ x_1 + x_2 \leq 220 \\ x_1 \geq 0, x_2 \geq 0 \end{cases}$$

Resolvendo tal problema por meio de algum método exato, obtemos que a solução ótima do PL é $x^* = (120, 0)^t$, de modo que o lucro máximo obtido por Pedro foi de $f(x^*) =$ R\$$1800,00$. Ou seja, Pedro irá obter o melhor lucro possível, dado as restrições, se for vender somente Tartarugas de pelúcia.

2.2 Modelagem de um POM e conceitos

Definição 2.2.1. *(Modelo de um Problema de Otimização Multiobjetivo). De forma geral, um POM com um número p de objetivos que devem ser otimizados, de forma a respeitar um conjunto de restrições $X \subseteq \mathbb{R}^n$, pode ser descrito como:*

$$\max z = (f_1(x), f_2(x), ..., f_p(x))$$
$$\text{sujeito a: } \begin{cases} x \in X \end{cases} \tag{1}$$

De modo que:

- $x = (x_1, x_2, ..., x_n)^t \in \mathbb{R}^n$ é o vetor das variáveis de decisão;
- $z_j = f_j(x), j = 1, ..., p$ é a j-ésima função objetivo a ser maximizada.

Modelo Exemplo: antes de adentrar nos conceitos da Otimização Multiobjetivo, vamos utilizar o seguinte problema para descrever os conceitos:

$$\max f_1 = 25x_1 + 20x_1$$
$$\max f_2 = x_1 + 8x_2$$
$$X: \begin{cases} x_1 + x_2 \leq 50 \\ 2x_1 + x_2 \leq 80 \\ 2x_1 + 5x_2 \leq 220 \\ x_1 \geq 0, x_2 \geq 0 \end{cases} \tag{2}$$

2.2.1 Espaço Critério

Em um problema usual de Programação Linear, cada solução factível ($x \in X$) aplicada na função objetivo retorna um valor real. No entanto, em um Problema de Otimização Multiobjetivo (POM), precisamos analisar os valores de p funções objetivo simultaneamente. Isso significa que, em vez de obtermos apenas um único valor real, obtemos um conjunto de pontos em $\mathbb{R}^p$.

Definição 2.2.2. *(Espaço Critério).O Espaço Critério* $Z = \{z = f(x) \in \mathbb{R}^p : x \in X\}$ *é o espaço formado pela aplicação de cada solução* ($x \in X$) *em um vetor* $z \in \mathbb{R}^p$. *Ou seja, para cada* $x \in X$, *temos um vetor* $z = (z_1, z_2, ..., z_p) = f(x) = (f_1(x), f_2(x), ..., f_p(x))$.

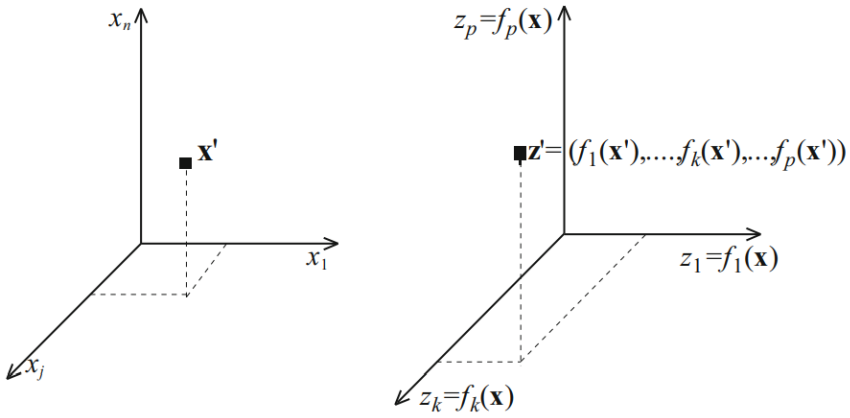


Figura 1: Imagem ilustrativa do Espaço Decisão à esquerda e do Espaço Critério à direita. Retirada de [Antunes et al. [2016]].

Exemplo 2.2.1. *(Espaço Critério do problema [2]).*

Para encontrar o Espaço Critério deste problema, devemos obter x_1 e x_2 em função de f_1 e f_2 e substituir nas desigualdades de X:

$$\begin{cases} 25x_1 + 20x_2 = f_1 \\ x_1 + 8x_2 = f_2 \end{cases} \Rightarrow \begin{cases} x_1 = \dfrac{2f_1 - 5f_2}{45} \\ x_2 = \dfrac{25f_2 - z_1}{180} \end{cases}$$

Observe que, para alguns problemas, não é uma tarefa simples calcular x_1, x_2 em função de z_1, z_2. Agora devemos substituir x_1, x_2 nas restrições de X:

$$X: \begin{cases} x_1 + x_2 \leq 50 \\ 2x_1 + x_2 \leq 80 \\ 2x_1 + 5x_2 \leq 220 \\ x_1 \geq 0, x_2 \geq 0 \end{cases} \Rightarrow Z: \begin{cases} 7f_1 + 5f_2 \leq 9000 \\ f_1 - f_2 \leq 960 \\ 11f_1 + 85f_2 \leq 39600 \\ 2f_1 - 5f_2 \geq 0 \\ 25f_2 - f_1 \geq 0 \end{cases} \tag{3}$$

Com isso, obtemos o Espaço Critério para o problema. É evidente que há muito trabalho ao substituir os valores de x_1, x_2 nas desigualdades. Todavia, um software que pode facilitar este processo é o Wolfram Mathematica, disponível para alunos da Unicamp.

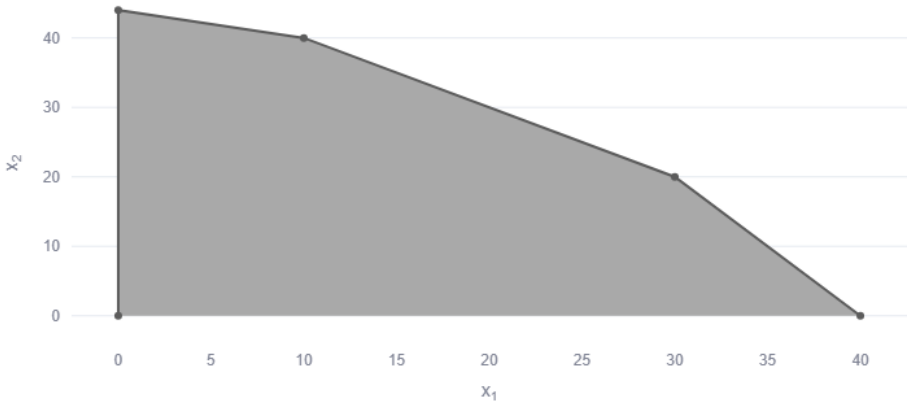


Figura 2: Espaço Decisão do problema [2].

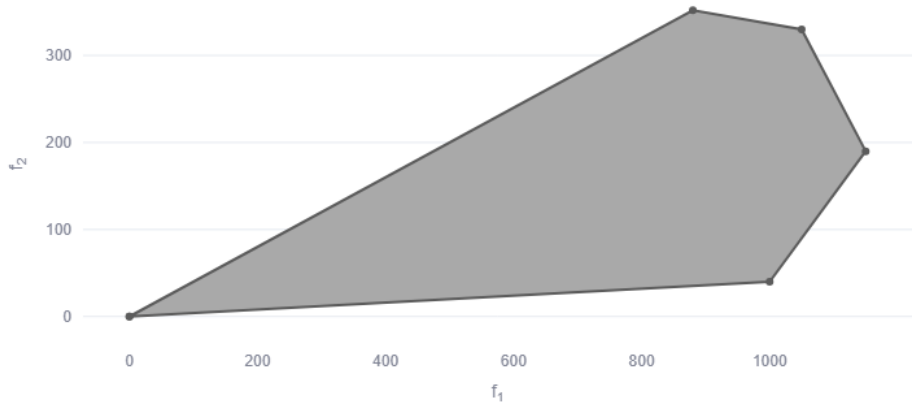


Figura 3: Espaço Critério do problema [2].

2.2.2 Relações de Dominância

Antes de introduzirmos os conceitos de soluções eficientes para um Problema de Otimização Multiobjetivo, vamos definir a relação de dominância entre duas soluções.

Dominância entre soluções: considere duas soluções, x^* e $\bar{x}$, pertencentes ao espaço decisão X. A solução x^* domina $\bar{x}$ se as seguintes condições são satisfeitas:

1. $f_j(x^*) \geq f_j(\bar{x})$ para todo $j = 1, \ldots, p$;
2. $f_j(x^*) > f_j(\bar{x})$ para ao menos um j.

Notação: $x^* \preceq \bar{x}$.

Definição 2.2.3. *(Dominância Forte). A solução x^* domina fortemente a solução $\bar{x}$ se a seguinte condição é satisfeita:*

$$f_j(x^*) > f_j(\bar{x}), \text{ para } j = 1, ..., p.$$

Definição 2.2.4. *(Dominância Fraca) A solução x^* domina fracamente a solução $\bar{x}$ se as seguintes condições são satisfeitas:*

1. *$f_j(x^*) \geq f_j(\bar{x})$ para todo $j = 1, \ldots, p$;*
2. *$f_j(x^*) \neq f_j(\bar{x})$ para ao menos um j;*
3. *x^* não domina fortemente $\bar{x}$.*

2.2.3 Soluções Eficientes, Pontos não-dominados e Fronteira de Pareto

Um dos principais conceitos de Otimização Multiobjetivo é o de soluções eficientes, pois é por meio deste que vamos compreender quais soluções possuem mais significância para um Problema de Otimização Multiobjetivo.

Definição 2.2.5. *(Solução Eficiente). A solução x^* é dita eficiente se, e somente se, não existe nenhum x no espaço de decisão X, de modo que $x \preceq x^*$.*

Definição 2.2.6. *(Ponto não-dominado). O ponto $z(x) = (f_1(x), f_2(x), \ldots, f_p(x)) \in Z$ é dito não dominado se, e somente se, x é uma solução eficiente.*

Definição 2.2.7. *(Conjunto Eficiente). O conjunto eficiente X^* é formado por todos os elementos de X que não são dominados por outros elementos de X. Em outras palavras, $X^* = \{x^* \in X : x \not\preceq x^*, \forall x \in X\}$.*

Exemplo 2.2.2. *(Conjunto Eficiente do problema [2]).*

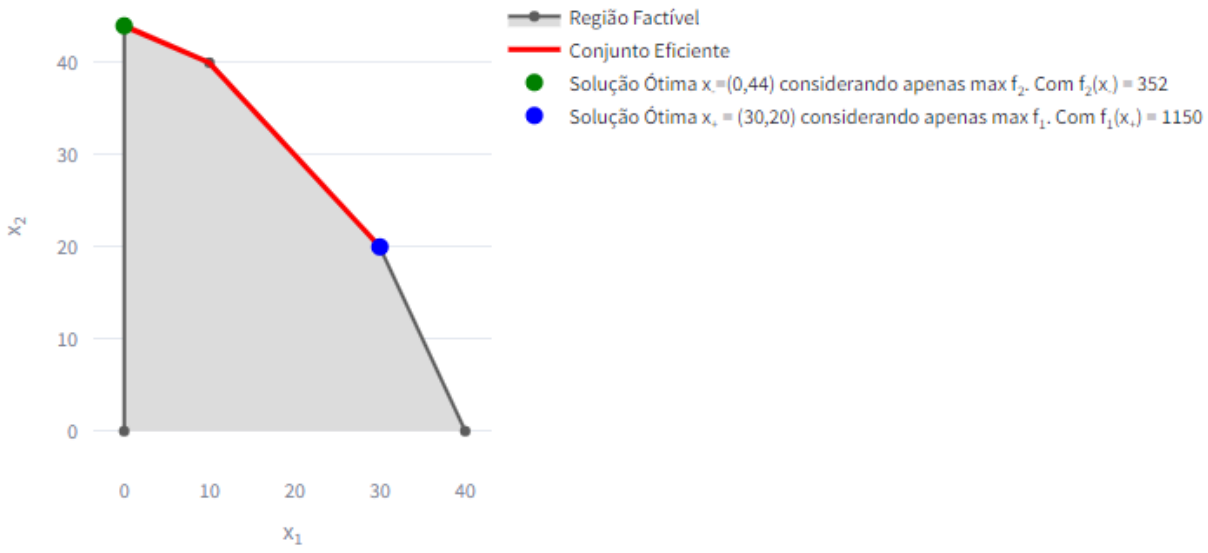


Figura 4: Conjunto Eficiente do problema [2].

Definição 2.2.8. *(Solução Fracamente Eficiente). Uma solução $\hat{x}$ é considerada fracamente eficiente se não houver nenhuma solução $x \in X$ tal que $f_k(x) > f_k(\hat{x})$ para todos os $k = 1, \ldots, p$.*

Definição 2.2.9. *(Ponto Fracamente Não-dominado). Um ponto $\hat{z}$ no espaço de critério é considerado fracamente não-dominado se sua imagem inversa for uma solução fracamente eficiente, ou seja, $\hat{z} = f(\hat{x})$.*

Exemplo 2.2.3. *(Exemplo de Relações de Dominância entre Pontos).*

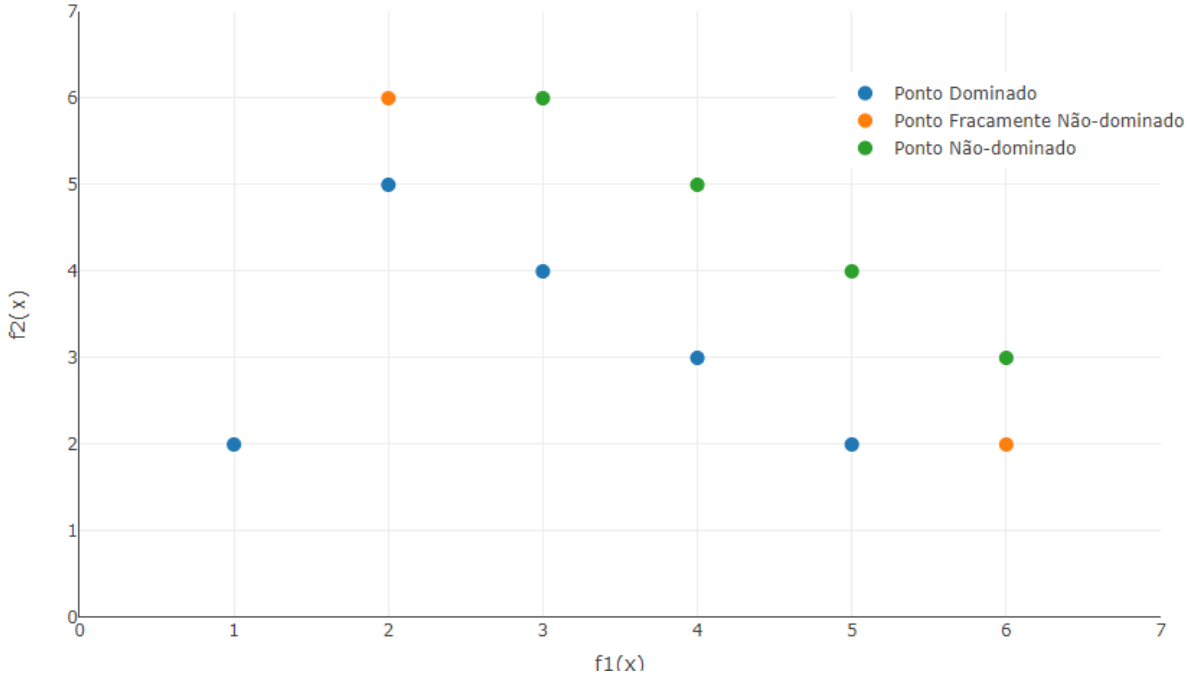


Figura 5: Gráfico com os exemplos de relação de dominância.

Definição 2.2.10. *(Fronteira de Pareto). A fronteira de Pareto, denotada por Z^* é a imagem do conjunto eficiente X^*, ou seja, $Z^* = \{z^* \in \mathbb{R}^p : z^* = f(x^*), \forall x^* \in X^*\}$*

A fronteira de Pareto é de extrema importância nos Problemas de Otimização Multiobjetivo, pois fornece as soluções mais adequadas ao problema. No entanto, a decisão final sobre qual solução é a mais apropriada dentro dessa fronteira é uma responsabilidade exclusiva do tomador de decisões. Ele deve considerar cuidadosamente as necessidades e objetivos específicos do problema para fazer a escolha mais adequada.

Exemplo 2.2.4. *(Fronteira de Pareto do problema [2]). Por meio das definições anteriores e das inequações [3], obtemos que a fronteira de Pareto é dada por:*

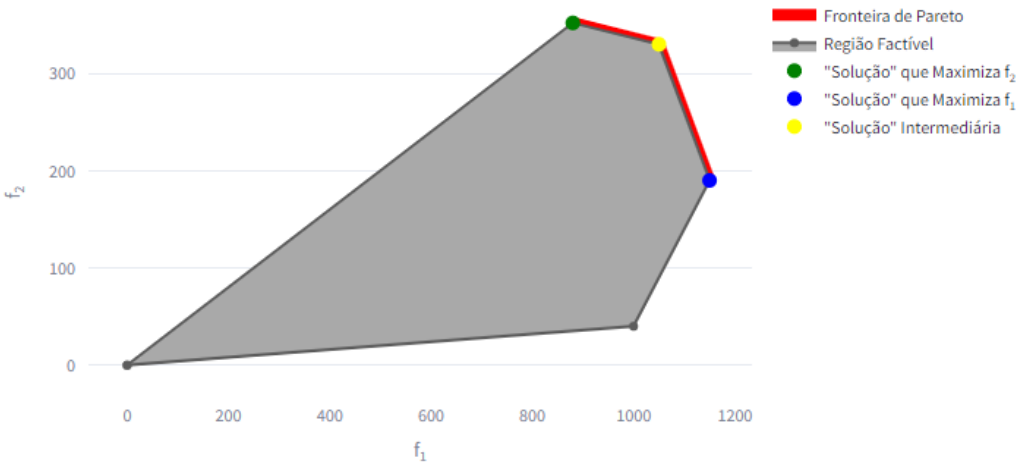


Figura 6: Fronteira de Pareto obtida para o problema [2].

3 Métodos Exatos

Na Otimização Multiobjetivo, uma das principais abordagens para obter soluções não dominadas e eficientes é por meio da técnica de escalarização. Essa técnica visa transformar um problema multiobjetivo em um problema de otimização mono-objetivo.

Neste trabalho, são apresentados dois métodos de escalarização:

1. **Método da Soma Ponderada:** nesse método, as funções objetivas são ponderadas por coeficientes específicos. A combinação linear dessas funções resulta em uma única função objetivo, que é então otimizada.

2. **Método do ε-Restrito (ou ε-Constraint):** aqui, uma das funções objetivas é otimizada, enquanto as demais são tratadas como restrições. O tomador de decisão especifica o limiar que deve ser respeitado nas restrições para cada uma das funções restantes.

3.1 Método da Soma Ponderada

Definição 3.1.1. *(Método da Soma Ponderada). Tal método se baseia em resolver um problema mono-objetivo, cuja função objetivo é uma soma ponderada das p funções originais. Com isso, para cada função f_k, associamos um peso $\lambda_k \geq 0$.*

Portanto, a solução de um POM por meio do método da Soma Ponderada pode ser obtida por meio da solução do seguinte problema:

$$\max z = \sum_{k=1}^{p} \lambda_k f_k(x)$$
$$sujeito\ a: \begin{cases} x \in X \end{cases}$$

Teorema 3.1.1. *Se $\lambda_k > 0$ para todo $k = 1, \ldots, p$, então a solução do problema ponderado é uma solução eficiente para o POM original.*

Demonstração: considere uma solução ótima, $\hat{x} \in X$, para o problema ponderado. Vamos supor que $\hat{x}$ não seja eficiente (i.e, não é solução ótima do POM), ou seja, existe uma solução $\bar{x} \in X$ de modo que $f_k(\bar{x}) \geq f_k(\hat{x})$ para todo $k = 1, \ldots, p$ e $f_i(\bar{x}) > f_i(\hat{x})$ para ao menos um $i \in \{1, \ldots, p\}$. Então,

$$\sum_{k=1}^{p} \lambda_k f_k(\bar{x}) > \sum_{k=1}^{p} \lambda_k f_k(\hat{x}),$$

mas isto contradiz o fato de $\hat{x}$ ser solução ótima do problema ponderado. Portanto, $\hat{x}$ é uma solução eficiente para o POM.

Teorema 3.1.2. *Se o problema ponderado tem uma única solução x^* com $\lambda_k \geq 0$, então x^* é eficiente.*

Demonstração: seja x^* a única solução do problema ponderado. Suponha que x^* não seja eficiente, logo, existe uma outra solução viável $\bar{x} \neq x^*$ de modo que $f_k(\bar{x}) \geq f_k(x^*)$ para todo $k = 1, \ldots, p$ e $f_i(\bar{x}) > f_i(x^*)$ para ao menos um $i \in \{1, \ldots, p\}$. Como $\lambda_k \geq 0$, temos que

$$\sum_{k=1}^{p} \lambda_k f_k(\bar{x}) \geq \sum_{k=1}^{p} \lambda_k f_k(x^*).$$

Como x^* é a solução única, podemos afirmar que

$$\sum_{k=1}^{p} \lambda_k f_k(x^*) > \sum_{k=1}^{p} \lambda_k f_k(\bar{x}),$$

mas isto contradiz a desigualdade imediatamente anterior. Logo, x^* é eficiente.

3.2 Método do ε-restrito

Definição 3.2.1. *(Método do ε-restrito). Dado um problema com p funções objetivas $(f_1(x), \ldots, f_p(x))$, selecionamos uma destas funções (digamos $f_i(x)$) para otimizar. Ademais, temos que as outras funções tornam-se parte do conjunto de restrições, de modo que para cada f_k, temos um ε_k associado de modo que $f_k(x) \geq \varepsilon_k$ $(k \neq i)$.*

Deste modo, temos que a solução de um (POM) por meio do ε-restrito é obtida por meio da solução do seguinte problema mono-objetivo:

$$\max f_i(x)$$
$$\text{sujeito a:} \begin{cases} x \in X \\ f_k(x) \geq e_k, \quad k = 1, ..., i-1, i+1, ..., p. \end{cases}$$

Teorema 3.2.1. *A solução ótima do problema do método $\varepsilon-$restrito é fracamente eficiente.*

Demonstração: análoga ao do Teorema 3.1.1.

4 Implementação Computacional e Aplicações

Nesta seção, abordamos a modelagem e a solução de Problemas de Otimização Multiobjetivo. Para resolver esses problemas, utilizamos os métodos da soma ponderada e ϵ-restrito, desenvolvendo códigos específicos para cada um. Além disso, desenvolvemos algoritmos que permitem variar os parâmetros de cada método. Para mais detalhes sobre o funcionamento de cada código, consulte o Capítulo [5].

4.1 Problema do Transporte

Na cidade de Campinas, uma empresa possui dois armazéns (Armazéns 1 e 2) responsáveis pelo transporte de produtos alimentícios para três mercados locais (Mercados A, B e C). Um dos objetivos da empresa é minimizar os custos de transporte até os mercados. No entanto, visando emitir a quantidade miníma de CO2 por toda sua frota, a empresa estimou a produção de CO2, em média por valor unitário de produto transportado, de cada armazém

para cada cidade. Note que neste exemplo, o Decisor busca um equilíbrio entre a eficiência logística e questão ambiental.

Variáveis de decisão:

x_{1A} : quantidade de mercadorias transportadas do Armazém 1 para o Mercado A

x_{1B} : quantidade de mercadorias transportadas do Armazém 1 para o Mercado B

x_{1C} : quantidade de mercadorias transportadas do Armazém 1 para o Mercado C

x_{2A} : quantidade de mercadorias transportadas do Armazém 2 para o Mercado A

x_{2B} : quantidade de mercadorias transportadas do Armazém 2 para o Mercado B

x_{2C} : quantidade de mercadorias transportadas do Armazém 2 para o Mercado C

Funções objetivo: neste problema, a função custo é representada por $z_1(x)$ e a função que representa a quantidade de CO2 é $z_2(x)$. Os coeficientes correspondentes são dados por:

$$\min\ z_1(x) = 2x_{1A} + 4x_{1B} + 5x_{1C} + 3x_{2A} + 1x_{2B} + 2x_{2C} \quad \text{(Custo de transporte em R\$}$$

$$\min\ z_2(x) = 9x_{1A} + 4x_{1B} + x_{1C} + 2x_{2A} + 5x_{2B} + 8x_{2C} \quad \text{(Emissão de CO2)}$$

Restrições:

- Capacidade de armazenamento: cada armazém possui uma capacidade máxima de produtos que podem ser estocados, ou seja, a quantidade de mercadorias que vai do armazém para todas as cidades não deve exceder a capacidade do armazém. Os respectivos valores de capacidade são dados a seguir:

$$x_{1A} + x_{1B} + x_{1C} \leq 100 \quad \text{(Capacidade do Armazém 1)}$$

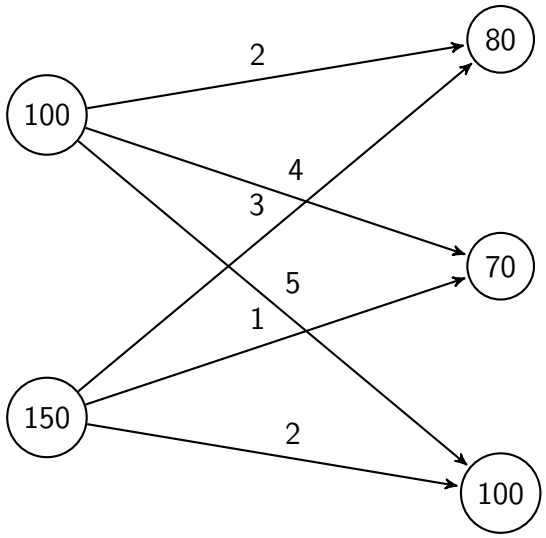
$$x_{2A} + x_{2B} + x_{2C} \leq 150 \quad \text{(Capacidade do Armazém 2)}$$

- Demanda: cada mercado possui uma demanda mínima de produtos que deve ser respeitada, ou seja, a produção de cada armazém para a cidade deve ser maior ou igual

a essa demanda mínima. Vale ressaltar que, nas restrições de demanda, estamos trabalhando com desigualdades do tipo "$\geq$", pois restrições do tipo "$=$"podem ocasionar infactibilidade. As restrições são dadas a seguir:

$$x_{1A} + x_{2A} \geq 80 \quad \text{(Demanda do Mercado A)}$$
$$x_{1B} + x_{2B} \geq 70 \quad \text{(Demanda do Mercado B)}$$
$$x_{1C} + x_{2C} \geq 100 \quad \text{(Demanda do Mercado C)}$$

Grafo do Problema de Transporte



Solução: no método do ε-restrito, variamos o valor de ε no intervalo $[-500, 500]$ em mil passos, por meio do algoritmo $percorrer\_epsilon()$. Considerando a função custo como restrição, ou seja, a função $z_1(x) \geq \varepsilon$, e aplicando o método do ε-restrito, obtivemos a mesma solução do problema ponderado, testado no algoritmo $varrer\_lambda()$ com mil passos. Os resultados obtidos são apresentados a seguir:

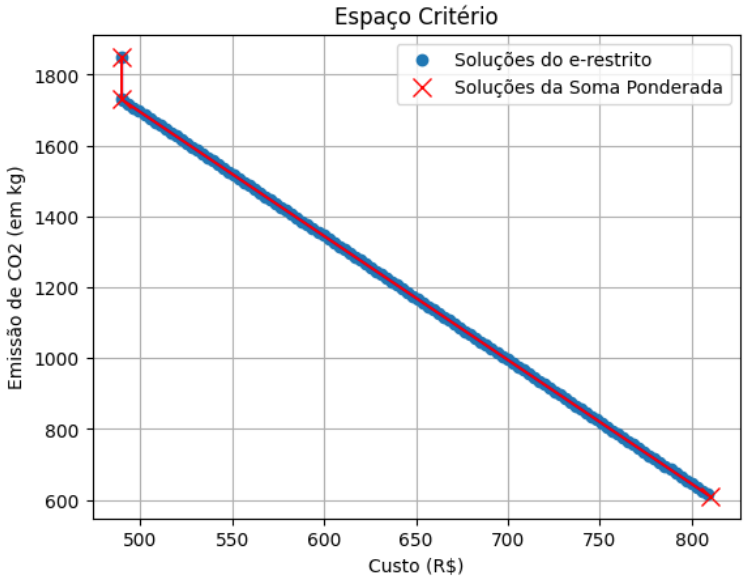


Figura 7: Gráfico da fronteira de Pareto do exemplo do transporte, para os métodos exatos.

Neste exemplo, como é possível observar na Figura [7], ao diminuir o custo de transporte, aumentamos a quantidade de CO2 emitida. No entanto, note que, no canto superior esquerdo da Figura [7], uma das soluções obtidas pelo método da soma ponderada, $s_1(x) = (490; 1850)$, possui o mesmo custo que outra solução obtida pelo mesmo método, $s_2(x) = (490; 1730)$, mas com uma emissão de CO2 superior. Conforme as Definições [2.2.8] e [2.2.9], concluímos que a solução $s_1(x)$ é fracamente eficiente (dado que este é um problema com duas minimizações).

4.2 Emissão de CO2 e Lucro em Setores de Produção

Uma empresa produz seis produtos P_1, P_2, P_3, P_4, P_5 e P_6. Cada produto gera um certo lucro e emite uma quantidade específica de CO2. O objetivo é encontrar a quantidade de cada produto a ser fabricado para maximizar o lucro e minimizar as emissões de CO2.

Variáveis de decisão:

x_1 : quantidade de P_1 a ser produzida

x_2 : quantidade de P_2 a ser produzida

x_3 : quantidade de P_3 a ser produzida

x_4 : quantidade de P_4 a ser produzida

x_5 : quantidade de P_5 a ser produzida

x_6 : quantidade de P_6 a ser produzida

Funções objetivo: neste problema, a função lucro é representada por $z_1(x)$ e a função de emissões de CO2 é representada por $z_2(x)$. As funções objetivo são dadas por:

$$\max\ z_1(x) = 15x_1 + 20x_2 + 25x_3 + 30x_4 + 22x_5 + 18x_6 \quad \text{(Lucro em R\$)}$$

$$\min\ z_2(x) = 12x_1 + 10x_2 + 18x_3 + 22x_4 + 14x_5 + 16x_6 \quad \text{(Emissão de CO2 em kg)}$$

Restrições:

- Orçamento: O custo total de produção não deve exceder o orçamento disponível:

$$50x_1 + 70x_2 + 60x_3 + 80x_4 + 65x_5 + 55x_6 \leq 15,000$$

- Horas de trabalho: o total de horas de trabalho necessárias para a produção não deve exceder as horas disponíveis:

$$2x_1 + 3x_2 + 1x_3 + 4x_4 + 2.5x_5 + 1.5x_6 \leq 2,000$$

- Demanda mínima: cada produto deve atender a uma demanda mínima de mercado:

$$x_i \geq 10 \quad \text{para } i = 1, \ldots, 6$$

Modelo Matemático Multiobjetivo:

$$\max\ z_1(x) = 15x_1 + 20x_2 + 25x_3 + 30x_4 + 22x_5 + 18x_6$$

$$\min\ z_2(x) = 12x_1 + 10x_2 + 18x_3 + 22x_4 + 14x_5 + 16x_6$$

$$\text{s.a:} \begin{cases} 50x_1 + 70x_2 + 60x_3 + 80x_4 + 65x_5 + 55x_6 \leq 15,000 \\ 2x_1 + 3x_2 + 1x_3 + 4x_4 + 2.5x_5 + 1.5x_6 \leq 2,000 \\ x_i \geq 10 \quad \text{para } i = 1, \ldots, 6 \\ x_i \geq 0 \quad \text{para } i = 1, \ldots, 6 \end{cases}$$

Solução: por meio dos métodos do $\varepsilon-$restrito e soma ponderada, obtemos a seguinte fronteira de Pareto:

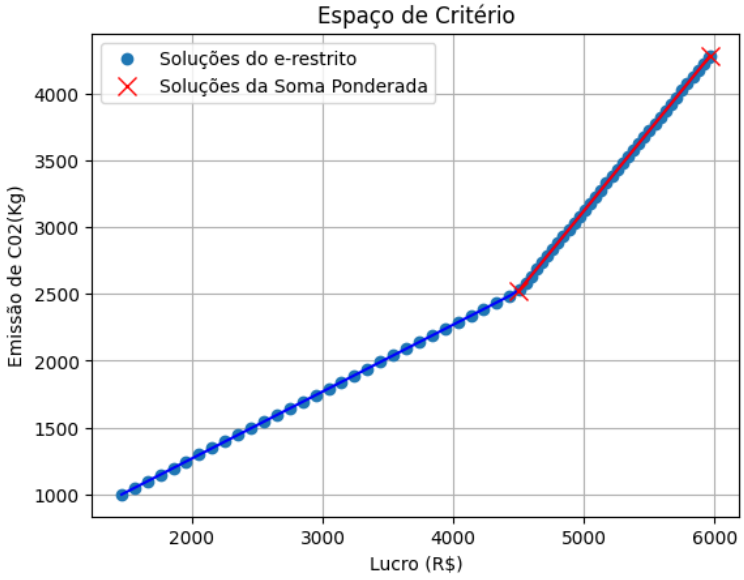


Figura 8: Gráfico da fronteira de Pareto do exemplo de lucro × CO2.

Pela Figura [8], temos que ao elevar o seu lucro, a empresa acaba por elevar a emissão de CO2 advinda de seus produtos. Tal resultado é evidente, tendo em vista que para aumentar seu lucro, tal empresa necessita produzir mais de seus produtos, que por sua vez aumentam a emissão de CO2.

4.3 Problema da Dieta

O objetivo deste modelo é obter uma dieta que minimize o custo diário de alimentos e maximize a quantidade de proteínas ingeridas. Para resolver esse problema, selecionamos uma lista de alimentos (Tabela 1), suas informações nutricionais e a dose diária recomendada de nutrientes. Os valores apresentados na tabela correspondem a 100g de cada alimento, enquanto o custo médio por 100g foi obtido a partir de preços da internet. A quantidade de calorias, sódio e gordura foi ajustada para considerar outros ingredientes envolvidos no preparo desses alimentos.

Alimento / **Nutrientes**	**Arroz** (x_1)	**Feijão** (x_2)	**Frango** (x_3)	**Coxão Mole** (x_4)	**Queijo** (x_5)	**Costela** (x_6)
Carboidrato (g)	28	14	0	0	5	0
Proteína (g)	3	5	32	32	15	14
Gordura (g)	2	2	8	12	8	13
Calorias (Kcal)	216	164	456	516	176	670
Fibra (g)	2	8	0	0	0	0
Sódio (mg)	166	167	205	209	236	228
Custo (R$)	0,8	0,9	2,7	4,0	6,5	2,8

Tabela 1: Valores nutricionais dos alimentos.

Com base nas informações, elaboramos o seguinte modelo:

$$\min\ f_1(x) = 0.8x_1 + 0.9x_2 + 2.7x_3 + 4x_4 + 6.5x_5 + 2.8x_6 \quad \text{(Custo)}$$

$$\max\ f_2(x) = 3x_1 + 5x_2 + 32x_3 + 32x_4 + 15x_5 + 14x_6 \quad \text{(Proteína)}$$

$$\text{s.a.} \begin{cases} 1800 \leq 216x_1 + 164x_2 + 456x_3 + 516x_4 + 176x_5 + 670x_6 \leq 2500 \quad \text{(Kcal)} \\ 100 \leq 28x_1 + 14x_2 + 0x_3 + 0x_4 + 5x_5 + 0x_6 \leq 150 \quad \text{(Carboidrato)} \\ 25 \leq 2x_1 + 8x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 \leq 36 \quad \text{(Fibra)} \\ 1300 \leq 166x_1 + 167x_2 + 205x_3 + 209x_4 + 236x_5 + 228x_6 \leq 2300 \quad \text{(Sódio)} \\ 22 \leq 2x_1 + 2x_2 + 8x_3 + 12x_4 + 8x_5 + 13x_6 \leq 29 \quad \text{(Gordura Saturada)} \\ x_i \geq 10 \quad \text{para } i = 1, \ldots, 6 \end{cases}$$

Ao adaptarmos tal modelo ao método da soma ponderada e ao ε-restrito, considerando a função de proteínas como uma restrição, obtivemos os seguintes resultados:

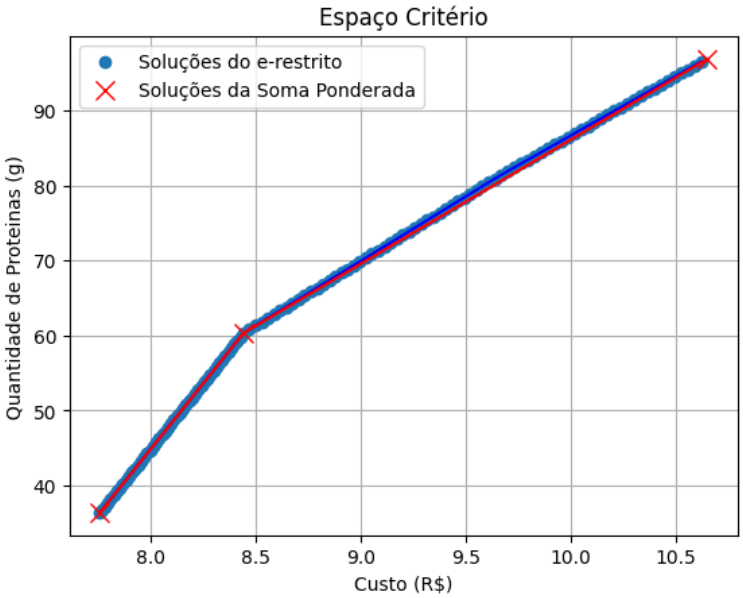


Figura 9: Gráfico da fronteira de Pareto do exemplo da dieta.

O resultado anterior nos mostra que, ao aumentar a quantidade de proteínas diárias, inevitavelmente elevamos o custo da refeição. Este resultado é corroborado pelas quantidades de alimentos selecionados pelo modelo em cada par de soluções, presentes na Tabela [2].

Na primeira solução, onde o custo é mínimo, há uma penalização devido à redução do custo, resultando na não seleção de alimentos ricos em proteínas. Foram selecionadas apenas 71g de costela, a carne com o mais baixo teor proteico, além de grandes quantidades de arroz e feijão, que possuem um custo menor. Esses alimentos, embora menos proteicos, contêm nutrientes suficientes para satisfazer as outras restrições nutricionais.

Por outro lado, na solução em que obtivemos o maior valor de proteínas diárias, o modelo optou por reduzir a quantidade de arroz e feijão consumidos e aumentar a quantidade de frango (aproximadamente 240g), alimento com maior valor proteico entre todos os outros alimentos. Essa escolha reflete a necessidade de incluir alimentos mais caros, mas com maior conteúdo proteico, para atingir o objetivo de maximizar a ingestão de proteínas.

Ademais, a última solução da tabela é uma solução intermediária que busca equilibrar o custo e a quantidade proteica. Como podemos ver, a quantidade de frango selecionada foi de 116g, enquanto a de arroz foi de 432g.

Alimento / (Custo, Proteínas)	Arroz (x_1)	Feijão (x_2)	Frango (x_3)	Coxão Mole (x_4)	Queijo (x_5)	Costela (x_6)
(7,76; 36,40)	3,88	2,96	0,00	0,00	0,00	0,71
(10,62; 96,60)	2,30	2,55	2,40	0,00	0,00	0,00
(8,44; 60,30)	4,32	2,07	1,16	0,00	0,00	0,00

Tabela 2: Tabela com os valores das funções objetivo e soluções factíveis.

4.4 Comparação com o problema [2]

Uma maneira eficaz de comparar a eficiência dos algoritmos implementados para encontrar os pontos pertencentes à fronteira de Pareto é comparar os resultados obtidos pelos algoritmos com a fronteira de Pareto já conhecida de um problema.

Assim, utilizando o problema [2], comparamos a fronteira de Pareto obtida pelos algoritmos implementados com a solução exata, obtida por meio das inequações [3]:

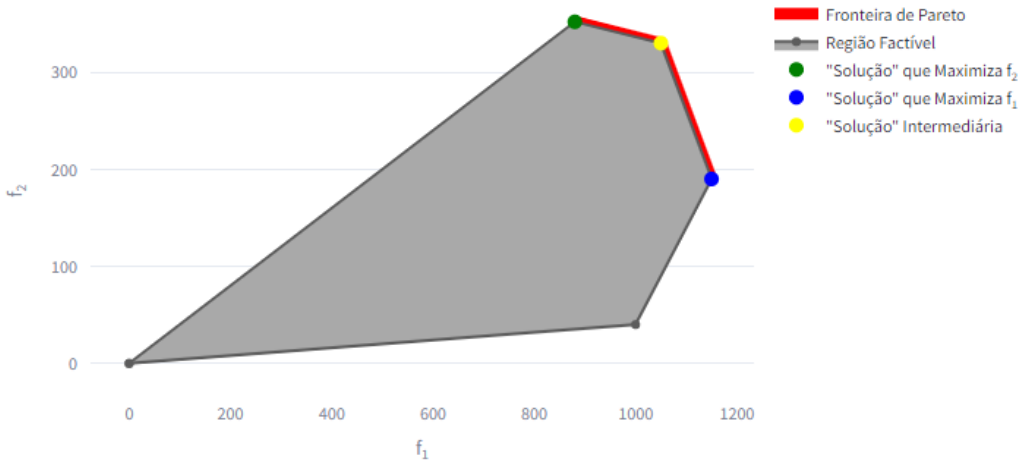


Figura 10: Espaço Critério do problema [2] obtido pelas inequações presentes em [3].

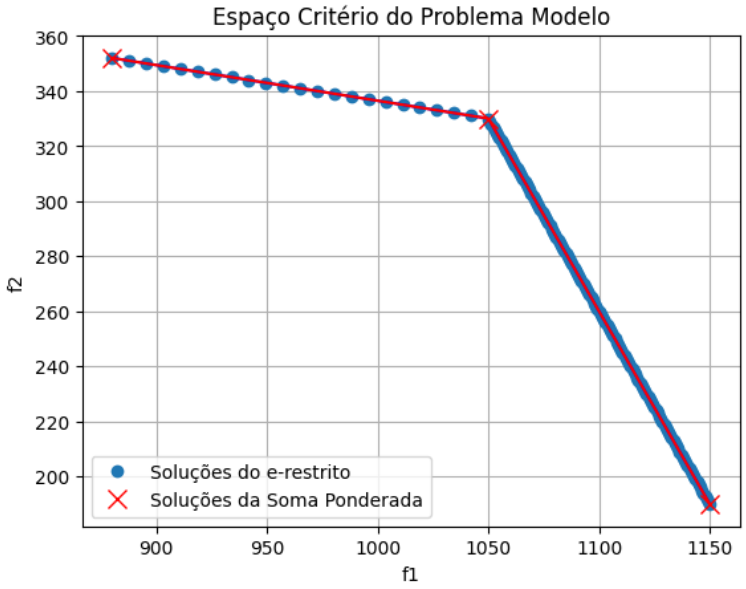


Figura 11: Soluções obtidas pelos métodos exatos.

Note que ambos os métodos exatos retornaram soluções eficientes para o problema. No entanto, o método da soma ponderada produziu apenas três soluções (que são justamente os vértices descritos na Figura [10]), enquanto o método do $\varepsilon-$restrito gerou diversas soluções. Isso ocorre porque o método do $\varepsilon-$restrito cria novos vértices para o problema original, variando os limites das funções objetivo, enquanto o método da Soma Ponderada apenas percorre os vértices da região factível do problema, limitando-se às soluções em regiões convexas da fronteira de Pareto.

5 Códigos

Os códigos a seguir foram implementados em Python, com auxílio da biblioteca PuLP (Python Optimization Tools (PyOpt)). Como é possível perceber, por meio da teoria apresentada neste trabalho e pelos algoritmos a seguir, estamos trabalhando com problemas de maximização. Caso queiramos executar problemas de minimização, devemos nos recordar que

$$\min f(x) = -\max f(-x)$$

5.1 $\varepsilon-$restrito

No código do $\varepsilon-$restrito, temos com entradas as matrizes contendo as desigualdades do tipo "$\leq$", "$\geq$", "$=$"e seus respectivos limitantes dentro dos vetores. Além disso, deve ser informado os coeficientes das funções deverão ser imposta como restrições, seus respectivos limitantes e também os coeficientes da função que deve ser maximizada. Deste modo, o modelo considerado é da forma:

$$\max z_i = f_i(x)$$
$$\text{sujeito a: } \begin{cases} Ax \leq b \\ Gx \geq g \\ Ex = e \\ f_k(x) \geq \varepsilon_k, \quad k = 1, ..., i-1, i+1, ..., p \\ x \geq 0 \end{cases}$$

```python
from pulp import LpProblem, LpMaximize, LpVariable, lpDot, LpStatus
import numpy as np
import matplotlib.pyplot as plt


def e_restrito(n=0, A=0, b=0, G=0, g=0, E=0, e=0, coeficientes_fun_pri
    =0, num_funcao_restricao=0, matriz_coeficientes_f_aux=0, epsilons=0):
     model = LpProblem(name="large-problem", sense=LpMaximize)

     # Variaveis de decisao
     x = [LpVariable(name=f"x{i}", lowBound=0) for i in range(n)]

     # Restricoes <=
     if isinstance(A, (list, np.ndarray)) and isinstance(b, (list, np.
    ndarray)):
          for i in range(len(A)):
               model += (lpDot(A[i], x) <= b[i], f"restricao_<=_{i}")

     # Restricoes >=
```

```
    if isinstance(G, (list, np.ndarray)) and isinstance(g, (list, np.ndarray)):
        for i in range(len(G)):
            model += (lpDot(G[i], x) >= g[i], f"restricao_>=_{i}")

    # Restricoes ==
    if isinstance(E, (list, np.ndarray)) and isinstance(e, (list, np.ndarray)):
        for i in range(len(E)):
            model += (lpDot(E[i], x) == e[i], f"restricao_==_{i}")

    # Funcao Objetivo Principal
    funcao_principal = lpDot(coeficientes_fun_pri, x)
    model += funcao_principal

    # Restricoes epsilon
    if isinstance(matriz_coeficientes_f_aux, (list, np.ndarray)) and isinstance(epsilons, (list, np.ndarray)):
        for i in range(num_funcao_restricao):
            model += (lpDot(matriz_coeficientes_f_aux[i], x) >= epsilons[i], f"restricao_eps_{i}")

    # Resolvendo o problema
    status = model.solve()

    # Obtencao da solucao
    sol = np.array([x[i].varValue for i in range(n)])
    coef_f_values = np.vstack((coeficientes_fun_pri, matriz_coeficientes_f_aux))

    # Calculando os valores das funcoes objetivas
    value_func_matriz = np.zeros(np.shape(coef_f_values)[0])
    for i in range(np.shape(coef_f_values)[0]):
        value_func_matriz[i] = np.dot(coef_f_values[i], sol)

    return sol, status, value_func_matriz
```

Código 1: código do método do $\varepsilon-$restrito.

5.2 Soma Ponderada

No código do método da Soma Ponderada, temos como entradas as mesmas matrizes de restrições do método do $\varepsilon-$restrito. A única diferença é que devemos informar os coeficientes de todas as funções objetivos em uma única matriz, e os valores de λ que acompanha cada função (como foi explicado anteriormente, podemos otimizar a escolha dos parâmetros λ para problemas bi-objetivos por meio do Algoritmo [5.3]). Deste modo, o modelo considerado é da seguinte forma:

$$\max z = \sum_{k=1}^{p} \lambda_k f_k(x)$$

$$\text{sujeito a: } \begin{cases} Ax \leq b \\ Cx \geq d \\ Ex = e \\ x \geq 0 \end{cases}$$

```
from pulp import LpProblem, LpMaximize, LpVariable, lpDot, LpStatus
import numpy as np
import matplotlib.pyplot as plt

def soma_ponderada(n, A=0, b=0, C=0, d=0, E=0, e=0, coef_func=0,
    vetor_lambda=0):
    # Definindo o problema de maximizacao
    model = LpProblem(name="multiobjective-weighted-sum", sense=
    LpMaximize)

    # Variaveis de decisao
    x = [LpVariable(name=f"x{i}", lowBound=0) for i in range(n)]

    # Restricoes <=
    if isinstance(A, (list, np.ndarray)) and isinstance(b, (list, np.
    ndarray)):
        for i in range(len(A)):
            model += (lpDot(A[i], x) <= b[i], f"restricao_<=_{i}")

```

```
    # Restricoes >=
    if isinstance(C, (list, np.ndarray)) and isinstance(d, (list, np.
ndarray)):
        for i in range(len(C)):
            model += (lpDot(C[i], x) >= d[i], f"restricao_>=_{i}")

    # Restricoes ==
    if isinstance(E, (list, np.ndarray)) and isinstance(e, (list, np.
ndarray)):
        for i in range(len(E)):
            model += (lpDot(E[i], x) == e[i], f"restricao_==_{i}")

    # Funcao objetivo ponderada
    funcao_principal = lpDot(np.dot(np.transpose(vetor_lambda),
coef_func), x)
    model += funcao_principal

    # Resolvendo o problema
    status = model.solve()

    # Obtencao da solucao
    sol = np.array([x[i].varValue for i in range(n)])

    # Calculando os valores das funcoes objetivas
    value_func_aux = np.dot(coef_func, sol)

    return sol, status, value_func_aux
```

Código 2: código do método da Soma Ponderada.

5.3 Variação dos Parâmetros - Soma Ponderada

Para analisar as diferentes soluções do método da soma ponderada, desenvolvemos o código $varrer\_lambda()$. Esse código foi criado especificamente para problemas bi-objetivo. Inicialmente, definimos o número de passos percorridos em uma malha uniforme de zero a um. Na primeira iteração, temos $\lambda_1 = 0$, o que implica $\lambda_2 = 1$. Nas iterações subsequentes, o

valor de λ_1 é determinado pelo número de divisões da malha uniforme, com $\lambda_2 = 1 - \lambda_1$. Tal abordagem garante pesos distintos e combinações equilibradas para ambos os λ's.

```
def varrer_lambda(n_passos):
    solutions = []
    valor_f = []
    lambdas = np.linspace(0,1,n_passos)

    for lambda1 in lambdas:
        #lambda2 = np.random.uniform(0, lambda1)
        lambda2 = 1 - lambda1
        vetor_lambda = np.array([lambda1, lambda2])
        sol, status,f_values = soma_ponderada(n, A, b, C, d, E, e,
   coef_func, vetor_lambda)

        if status == 1:
            solutions.append(tuple(np.round(sol, decimals=10)))
            valor_f.append(tuple(np.round(f_values, decimals=10)))

    # Remover duplicatas
    solutions_distinct = list(set(solutions))
    f_values_distintos = list(set(valor_f))
    return solutions_distinct,f_values_distintos
```

Código 3: código que varia os λ's de problemas bi-objetivo.

5.4 Variação dos Parâmetros - $\varepsilon-$restrito

Ademais, o código $percorrer\_epsilon()$ também é específico para problemas bi-objetivo, porém possui mais parâmetros de entrada. Primeiramente, devemos informar o ε inicial do problema, o ε final e a quantidade de passo que devemos percorrer na malha uniforme entre o valor inicial e final. Diferentemente do algoritmo anterior, não temos um ε "ideal" para todos os tipos de problema. Deste modo, cabe ao tomador de decisões escolher qual seria a tolerância mínima que deve ser respeitada pelo método.

```
def percorrer_epsilon(epsilon_range=None):
    solutions = []
```

```
    valor_f = []

    for epsilon in epsilon_range:
        epsilons = np.full(num_funcao_restricao, epsilon)
        sol, status,f_val = e_restrito(n, A, b, G, g, E, e,
    coeficientes_fun_pri, num_funcao_restricao, matriz_coeficientes_f_aux
    , epsilons)

        if status == 1:  # 1 indica 'Optimal'
            solutions.append(tuple(np.round(sol, decimals=10)))
            valor_f.append(tuple(np.round(f_val, decimals=10)))
        else:
            pass

    # Remover duplicatas
    solutions_distinct = list(set(solutions))
    valor_f_distinct = list(set(valor_f))
    return solutions_distinct,valor_f_distinct
```

Código 4: código que varia o ε de problemas bi-objetivo.

6 Conclusão

Em conclusão, este trabalho apresentou uma abordagem abrangente da Otimização Multiobjetivo, explorando desde os conceitos teóricos fundamentais até a implementação computacional de métodos exatos. Inicialmente, foram detalhados os principais conceitos da Otimização Multiobjetivo, fornecendo uma base para o entendimento dos problemas de otimização que envolvem múltiplas funções objetivo. Em seguida, focamos em estudar os métodos da soma ponderada e do ε-restrito.

Além disso, implementamos os códigos correspondentes a esses métodos, permitindo a aplicação prática em problemas reais. Através de exemplos específicos, demonstramos a eficácia dos algoritmos desenvolvidos e comparamos os resultados com soluções conhecidas, validando a eficiência das abordagens propostas. Embora este trabalho aborde apenas uma pequena parte do vasto campo da Otimização Multiobjetivo, espera-se que ele tenha despertado o interesse dos leitores e inspire futuros trabalhos, dado que tal tópico ainda não é

amplamente explorado, como é o caso da graduação em Matemática Aplicada e Computacional da UNICAMP. Foram deixadas referências ao final deste trabalho para que os leitores possam explorar ainda mais este campo promissor.

Referências

C. H. Antunes, M. J. Alves, and J. N. Clímaco. *Multiobjective linear and integer programming*. Springer, Switzerland, 2016.

M. N. Arenales, V. A. Armentano, R. Morabito, and H. H. Yanasse. *Pesquisa Operacional*. Editora Campus/Elsevier, 2006.

V. Chankong and Y. Y. Haimes. *Multiobjective decision making: Theory and methodology*. Elsevier Science Publishing Co, New York, 1983.

Python Optimization Tools (PyOpt). Pulp: Lp modelling in python. URL `https://github.com/coin-or/pulp`.