Problema de Gerenciamento de Conteúdo em Plataformas de Mídia Social

Geovana Vidal Moreira[1], **Lincoln Corrêa Figueiredo Cruz**[1]

[1]Instituto de Ciências Exatas e Informática - Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

geovana.moreira@sga.pucminas.br, lcfcruz@sga.pucminas.br

Abstract. *This paper aims to conduct a comprehensive analysis of the set cover problem, employing heuristics and approximate representations through greedy and brute-force algorithms. Additionally, it seeks to contextualize the problem within the specific theme of Content Management in Social Media Platforms, recognizing the practical relevance of this approach. The structure of the work encompasses the description of the methodology employed, detailed analyses, a robust theoretical foundation, and ultimately, the classification of the problem as belonging to the Non-Deterministic Polynomial (NP) complexity class.*

Resumo. *Este trabalho visa realizar uma análise abrangente do problema da cobertura de conjuntos, explorando heurísticas e representações aproximadas por meio de algoritmos gulosos e força bruta. Adicionalmente, busca-se contextualizar o problema no tema específico de Gerenciamento de Conteúdo em Plataformas de Mídia Social, reconhecendo a relevância prática dessa abordagem. A estrutura do trabalho abrange a descrição da metodologia, análises detalhadas, um embasamento teórico sólido e a classificação do problema como pertencente à classe de complexidade Non-Deterministic Polynomial (NP).*

1. Introdução

Nas últimas décadas, as mídias sociais assumiram um papel fundamental na comunicação e interação entre indivíduos, empresas e organizações. Plataformas como Facebook, Twitter, Instagram e LinkedIn desempenham um papel crucial na promoção de produtos, serviços, compartilhamento de informações e na construção de comunidades online. A capacidade de gerenciar eficazmente o conteúdo nessas plataformas é fundamental para alcançar o sucesso em estratégias de marketing digital e comunicação.

O ”Problema de Gerenciamento de Conteúdo em Plataformas de Mídia Social” emerge como um desafio complexo e multidimensional que exige soluções inteligentes e eficientes. Este problema envolve a otimização de recursos de conteúdo para atingir métricas-chave, como alcance, relevância e engajamento do público. À medida que a quantidade de conteúdo disponível nas mídias sociais continua a crescer exponencialmente, a necessidade de sistemas de gerenciamento de conteúdo eficazes se torna cada vez mais evidente.

Neste trabalho, será explorada a complexidade inerente ao ”Problema de Cobertura de Conjuntos” (SCP) e sua relevância em diversas aplicações, incluindo o ”Problema de Gerenciamento de Conteúdo em Plataformas de Mídia Social”. O foco principal será na análise da complexidade algorítmica associada ao SCP, demonstrando que pertence

à classe NP-completo, e apresentando algoritmos de aproximação e heurísticas para sua resolução.

Essa pesquisa é motivada pela necessidade de compreender e enfrentar eficientemente o desafio do SCP, que transcende o âmbito específico das mídias sociais. O sucesso na resolução desse problema pode ter impactos significativos nas estratégias de otimização de recursos, proporcionando uma visão mais abrangente sobre a eficácia das organizações em lidar com conjuntos complexos de informações.

A seção seguinte apresentará uma revisão bibliográfica que estabelecerá a base teórica e contextual para o estudo. Posteriormente, será explorada a metodologia utilizada para abordar o problema, seguida de uma análise dos resultados obtidos e de uma conclusão que resumirá as descobertas e destacará a relevância do trabalho em contextos além do gerenciamento de mídias sociais.

2. Revisão Bibliográfica

A complexidade associada ao ”Problema de Gerenciamento de Conteúdo em Plataformas de Mídia Social” e ao ”Problema de Cobertura de Conjuntos” (SCP) é multidisciplinar, envolvendo conceitos de otimização, algoritmos e marketing digital. Nesta seção, será apresentada uma revisão da literatura que estabelece a base teórica para ambos os problemas.

2.1. Gerenciamento de Conteúdo em Mídias Sociais

Para compreender o contexto do gerenciamento de conteúdo em mídias sociais, é fundamental abordar estratégias e desafios. Autores como [Kaplan and Haenlein 2010] exploram o uso de mídias sociais para marketing, destacando a importância de conteúdo relevante e engajador. Além disso, [Tuten and Solomon 2018] discutem a evolução do marketing nas mídias sociais, enfatizando a necessidade de abordagens eficazes de gerenciamento de conteúdo.

2.2. Complexidade Computacional

A teoria da complexidade computacional desempenha um papel fundamental na análise do ”Problema de Gerenciamento de Conteúdo em Plataformas de Mídia Social”. [Garey and Johnson 1979] estabeleceram a base teórica ao definir classes de complexidade, como P, NP e NP-completo. Foi também apresentado o ”Problema da Cobertura de Conjuntos” (Set Cover Problem) como um exemplo de problema NP-completo, frequentemente servindo como base para demonstrar a NP-completude de outros problemas.

2.3. Teorema de Cook-Levin e NP-Completude

O Teorema de Cook-Levin estabelece a NP-completude do ”Problema de Satisfatibilidade Booleana” (SAT). Esse problema consiste em determinar se uma fórmula booleana pode ser satisfeita por alguma atribuição de valores às suas variáveis. Cook demonstrou que o SAT é NP-completo, o que implica que qualquer problema em NP pode ser reduzido, em tempo polinomial, a uma instância do SAT.

Essa redução sugere que, se encontrarmos um algoritmo polinomial para resolver o SAT, poderíamos eficientemente resolver qualquer problema em NP. Contudo, até o momento, não foi desenvolvida uma solução eficiente para o SAT, mantendo em aberto

a questão central P versus NP na teoria da computação. O Teorema de Cook-Levin oferece uma estrutura teórica que destaca a complexidade intratável de uma ampla classe de problemas de decisão, contribuindo para a compreensão dos limites fundamentais da computação em termos de eficiência algorítmica.

2.4. Algoritmos de Aproximação e Heurísticas

Além da complexidade, é importante examinar abordagens para resolver problemas de otimização, como os apresentados. Autores como [Vazirani 2001] (Approximation Algorithms) e [Michalewicz 2004] (Heuristic Algorithms) exploram técnicas de aproximação e heurísticas para encontrar soluções próximas ao ótimo em problemas NP-completos.

2.5. Problemas NP-Completos Relacionados

A NP-completude é uma classificação crítica na teoria da complexidade. A redução de problemas NP-completos é um método comum para demonstrar a dificuldade de novos problemas. Problemas bem conhecidos, como o "Problema do Caixeiro Viajante" (TSP) e o "Problema da Mochila" (Knapsack Problem), servem como exemplos de problemas NP-completos, amplamente estudados em contextos de otimização.

Esta revisão bibliográfica estabelece um sólido alicerce teórico para o estudo do "Problema de Gerenciamento de Conteúdo em Plataformas de Mídia Social" e do "Problema de Cobertura de Conjuntos". A partir deste ponto, a metodologia abordará a análise de complexidade dos algoritmos, a demonstração da NP-completude do problema e o desenvolvimento de algoritmos de aproximação e heurísticas para suas resoluções.

3. Metodologia

Nesta seção, iremos formalizar que o Problema de Gerenciamento de Conteúdo de Redes Sociais pode ser solucionado pelo Problema de Cobertura de Conjuntos(SCP), fornecendo a definição formal do SCP e destacando sua relevância e aplicações práticas. Além disso, realizaremos uma análise da complexidade dos algoritmos envolvidos e demonstraremos que o problema é classificado como NP-Completo. Para abordar o "Problema de Gerenciamento de Conteúdo de Mídias Sociais", recorremos ao Problema de Cobertura de Conjuntos, onde mapeamos o grafo da rede social para conjuntos e subconjuntos do Problema de Cobertura de Conjuntos. Cada vértice no grafo é associado a um conteúdo específico, e os subconjuntos representam estratégias de distribuição desses conteúdos nas plataformas de mídia social. Ao resolver eficientemente o Problema de Cobertura de Conjuntos, otimizamos as escolhas estratégicas para maximizar o alcance, relevância e engajamento do conteúdo.

Uma instância do Problema de Cobertura de Conjuntos consiste em um conjunto finito X e uma família F de subconjuntos de X, garantindo que cada elemento de X pertença a pelo menos um subconjunto em F. Cada subconjunto $S \in F$ possui um custo associado, representado por $custo(S)$. Elementos podem ser compartilhados por diversos subconjuntos, e um subconjunto $S \in F$ é considerado uma cobertura de seus próprios elementos.

O objetivo central é encontrar o subconjunto $C \subseteq F$ cujos membros constituam uma cobertura de X com o menor custo possível. A dimensão de C é determinada pela quantidade de conjuntos que o compõem. Esse problema é classificado como NP-Difícil, abstraindo muitos desafios combinatórios que surgem em contextos diversos.

3.1. Contexto

O Problema de Cobertura de Conjuntos (SCP) emerge como uma ferramenta essencial na resolução de desafios complexos relacionados à alocação eficiente de recursos. Sua importância permeia diversas áreas, incluindo logística, design de redes, assistência médica e até mesmo no contexto dinâmico do Gerenciamento de Conteúdo em Plataformas de Mídia Social.

O SCP, ao modelar conjuntos e elementos, proporciona uma abordagem matemática robusta para a tomada de decisões estratégicas. Essa versatilidade na representação e resolução de problemas faz do SCP uma peça-chave em cenários nos quais a otimização da distribuição de recursos é crucial para o sucesso das operações.

3.2. Aplicações do problema

O Problema de Cobertura de Conjuntos (SCP) destaca-se por sua aplicabilidade em uma variedade de contextos do mundo real, desempenhando um papel crucial em otimizar a alocação de recursos e a tomada de decisões estratégicas.

No campo da logística, o SCP é empregado para otimizar o transporte e a distribuição de mercadorias. Ao representar conjuntos como diferentes rotas ou métodos de transporte e elementos como itens específicos a serem entregues, o SCP permite a seleção eficiente de conjuntos de transporte que cubram toda a demanda, minimizando custos e maximizando a eficiência da cadeia de suprimentos.

Em redes de comunicação, o SCP auxilia na seleção de conjuntos de transmissores ou pontos de acesso para garantir uma cobertura completa da área desejada. Isso é crucial em cenários como a expansão de redes de telefonia móvel, onde a escolha estratégica de locais para torres de transmissão é vital para assegurar a conectividade ideal.

Na área da saúde, o SCP é aplicado na otimização de recursos médicos, como a distribuição de equipes de atendimento, materiais cirúrgicos e salas de cirurgia. A representação de conjuntos como diferentes configurações de recursos e elementos como necessidades específicas de pacientes permite a tomada de decisões informadas para garantir a melhor cobertura e utilização eficiente de recursos médicos.

O SCP encontra uma aplicação contemporânea e dinâmica no gerenciamento de conteúdo em plataformas de mídia social. Ao modelar conjuntos como estratégias de distribuição de conteúdo e elementos como o próprio conteúdo a ser gerenciado, o SCP é fundamental para selecionar estrategicamente conjuntos que maximizem a visibilidade e o engajamento online, considerando restrições de tempo e recursos.

Esses exemplos demonstram a versatilidade do SCP, ilustrando como essa abordagem matemática pode ser adaptada para resolver uma ampla gama de desafios práticos em diferentes setores.

3.3. Análise de Complexidade dos algoritmos

Para efetuarmos uma análise de complexidade completa do problema de cobertura de conjuntos, será analisado 2 algoritmos distintos sobre o problema: Algoritmo de força bruta e Algoritmo Guloso.

3.3.1. Algoritmo de força bruta

O algoritmo de força bruta enfrenta o Problema de Cobertura de Conjuntos (SCP) por meio de uma abordagem exaustiva, testando todas as combinações possíveis de subconjuntos para encontrar a solução de menor custo.

O algoritmo começa inicializando as variáveis, onde n representa o número de subconjuntos, min_cost é a variável para armazenar o custo mínimo encontrado, e min_set guarda o conjunto de subconjuntos formando a cobertura mínima. Em seguida, um laço é executado, gerando todas as combinações possíveis de subconjuntos, variando o tamanho de 0 a n. Dentro do laço, a cobertura é atualizada combinando os elementos dos subconjuntos escolhidos, e o custo total dos subconjuntos selecionados é calculado.

Posteriormente, verifica-se se a cobertura abrange todo o universo, atualizando o conjunto de subconjuntos e o custo mínimo se a condição for satisfeita. Por fim, o algoritmo retorna o conjunto de subconjuntos e o custo mínimo encontrado.

A complexidade temporal do algoritmo de força bruta é exponencial, mais especificamente $O(2^n)$, onde n é o número de subconjuntos. Isso ocorre devido à necessidade de avaliar todas as 2^n combinações possíveis. Embora o algoritmo seja garantido para encontrar a solução ótima, sua viabilidade prática é limitada para conjuntos de dados grandes devido à alta complexidade.

3.3.2. Algoritmo Guloso

O algoritmo ganancioso aborda o Problema de Cobertura de Conjuntos (SCP) por meio de escolhas locais otimizadas, priorizando subconjuntos mais eficientes em termos de custo por elemento não coberto.

O algoritmo começa inicializando variáveis, onde $universe$ representa o conjunto de todos os elementos do universo, $subset\_indices$ é a lista de índices dos subconjuntos, e $used\_subsets$ armazena os subconjuntos selecionados. Em seguida, inicia um laço que é executado iterativamente até que todos os elementos do universo sejam cobertos. Dentro desse laço, um segundo laço (guloso) itera sobre os índices dos subconjuntos para calcular a efetividade de cobertura, calculando o custo por elemento não coberto para determinar a efetividade de cada subconjunto.

Posteriormente, o algoritmo atualiza o conjunto de subconjuntos selecionados, remove os elementos cobertos do universo e ajusta os índices dos subconjuntos. Por fim, retorna a lista de subconjuntos selecionados e o custo total.

A complexidade temporal do algoritmo ganancioso é principalmente governada pelo processo de escolha gananciosa, aproximadamente $O(n^2)$, onde n é o número de subconjuntos. O loop externo itera sobre todos os elementos não cobertos, enquanto o loop interno percorre todos os subconjuntos para calcular a efetividade de cobertura. Embora os algoritmos gananciosos ofereçam soluções aproximadas, sua eficiência em termos de tempo muitas vezes supera abordagens exaustivas, como o algoritmo de força bruta.

3.3.3. Algoritmo Cobertura de escalonamento de custos

O algoritmo começa inicializando a variável $num\_subsets$ como o número total de conjuntos n e $remaining\_elements$ como um conjunto que contém todos os elementos do universo. Em seguida, realiza um pré-processamento calculando os custos escalonados para cada conjunto e armazenando-os na lista $scaled\_costs$.

O loop principal é projetado para executar enquanto ainda existirem elementos não cobertos, ou seja, enquanto o conjunto $remaining\_elements$ não estiver vazio. Dentro deste loop, um conjunto é selecionado com base em um critério de menor custo escalonado por elemento.

Dentro do loop interno, que itera sobre todos os conjuntos não selecionados, são realizadas operações para calcular o tamanho da interseção entre o conjunto e os elementos restantes. Além disso, é calculado o custo escalonado atual, e o melhor conjunto é atualizado se o custo escalonado atual for menor.

Após a seleção do melhor conjunto no loop interno, há uma etapa de atualização e redução. Nesta etapa, a lista de conjuntos selecionados é atualizada, os elementos cobertos pelo conjunto selecionado são removidos do conjunto de elementos restantes, e o custo total é atualizado.

A condição de parada do algoritmo ocorre quando não há mais conjuntos a serem selecionados, encerrando assim o loop principal.

A complexidade do algoritmo $cover\_cost\_scaling$ é aproximadamente $O(nlogn)$, onde n é o número de subconjuntos. A componente $O(n)$ efere-se à busca linear entre os subconjuntos não selecionados, contribuindo para a complexidade total. A quantidade de iterações do laço principal, que pode ser aproximadamente $O(logn)$, também influencia na complexidade geral do algoritmo. Isso ocorre porque, em cada iteração, o número de elementos não cobertos é reduzido significativamente, proporcionando eficiência ao processo. Assim, a combinação desses fatores define a complexidade temporal do algoritmo.

3.3.4. Prova de que o problema é NP Completo

Os problemas NP (Não Determinísticos) são classificados pela dificuldade em encontrar soluções ótimas em tempo polinomial. Para estabelecer a natureza NP de um novo problema, recorremos à técnica de redução, transformando-o em um problema NP previamente reconhecido. Essa abordagem permite generalizar a complexidade computacional de diferentes desafios.

Utilizando a técnica de redução, reduziremos o SCP a um problema conhecido, o Problema do Vertex Cover (VC).

Precisamos responder se existe uma cobertura de conjunto C para um conjunto X (contendo os elementos a serem cobertos), com no máximo k conjuntos em C.

Utilizando um certificado C, onde cada conjunto em C cobre pelo menos um elemento em X, a verificação de C pode ser feita em tempo polinomial. A validação inclui verificar que $|C| \leq k$ e que C cobre X em $O(|X|^2)$, o que prova que o problema está em NP.

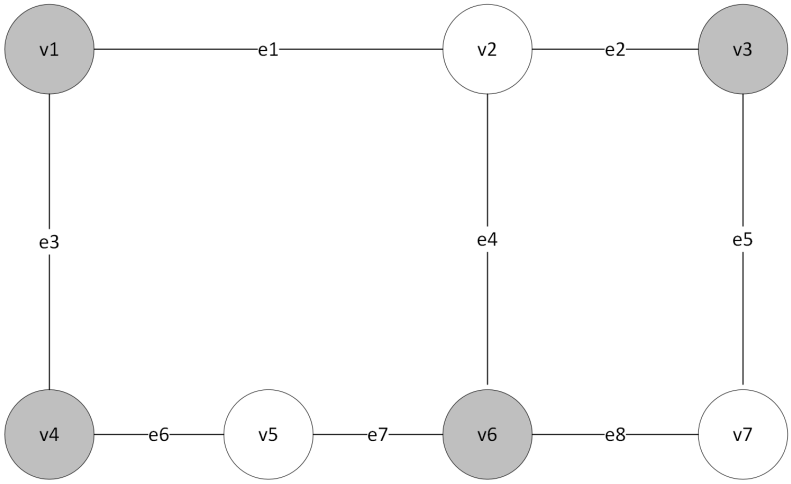
A prova é feita por uma redução do Problema do Vertex Cover (que é NP-Completo) para o Problema de Cobertura de Conjuntos.

Uma cobertura de vértices em um grafo não direcionado $G = (V, E)$ é transformada em uma instância do Problema de Cobertura de Conjuntos. Os conjuntos em C representam os vértices da cobertura, e F representa os conjuntos de arestas adjacentes a cada vértice. Essa redução é feita em tempo polinomial.

A prova mostra que se existe uma cobertura de vértices V' para G, então existe uma cobertura de conjuntos C para X e F construída de acordo com a redução. A correspondência é estabelecida entre $|V'| = |C|$.

Exemplificando com o grafo a seguir, exibindo um Vertex Cover de tamanho 4 (com nós sombreados em V').

Figure 1. Exemplo de Vertex Cover



Fonte: Autor próprio

O resultado da execução da redução é:

$$F = \{S_{v_1} = \{e_1, e_3\}, S_{v_3} = \{e_2, e_5\}, S_{v_4} = \{e_3, e_6\}, S_{v_6} = \{e_4, e_7, e_8\}\}$$

A conclusão é uma validação da redução, mostrando que resolver o Problema de Cobertura de Conjuntos para a instância reduzida é equivalente a resolver o Problema de Cobertura de Vértices para G. Isso estabelece a NP-Completude do Problema de Cobertura de Conjuntos.

3.3.5. Algoritmo de Aproximação/Heurísticas

Para o problema da Cobertura de Conjuntos, um exemplo de algoritmo de aproximação é o algoritmo guloso da razão entre o custo e a quantidade de elementos não utilizados. A ideia por trás desse algoritmo é selecionar os subconjuntos com a menor razão custo/elementos não utilizados e colocá-los na solução enquanto houver elementos não utilizados no universo. Essa abordagem intuitivamente prioriza os subconjuntos que oferecem o menor custo.

No entanto, embora o algoritmo guloso funcione bem em muitos cenários, ele pode levar a soluções subótimas em alguns casos. Para ilustrar, consideremos seguinte cenário:

$$Universo = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 \tag{1}$$

$Subconjunto1 = \{2, 3, 5, 13, 14, 16, 19, 24\}$ com custo = 9
$Subconjunto2 = \{8, 9, 13, 26, 27, 29\}$ com custo = 10
$Subconjunto3 = \{1, 4, 5, 10, 15, 20, 25, 27, 28, 30\}$ com custo = 6
$Subconjunto4 = \{6, 15, 16, 19, 21, 22, 28, 29, 30\}$ com custo = 3
$Subconjunto5 = \{1, 6, 10, 12, 13, 14, 25, 29\}$ com custo = 5
$Subconjunto6 = \{8, 9, 11, 12, 17, 20\}$ com custo = 1
$Subconjunto7 = \{2, 6, 7, 17, 26\}$ com custo = 1
$Subconjunto8 = \{1, 2, 4, 5, 8, 10, 11, 20, 23\}$ com custo = 5
$Subconjunto9 = \{11, 14, 18, 19, 22, 25, 27\}$ com custo = 6
$Subconjunto10 = \{2, 8, 11, 16, 17, 19, 25\}$ com custo = 10

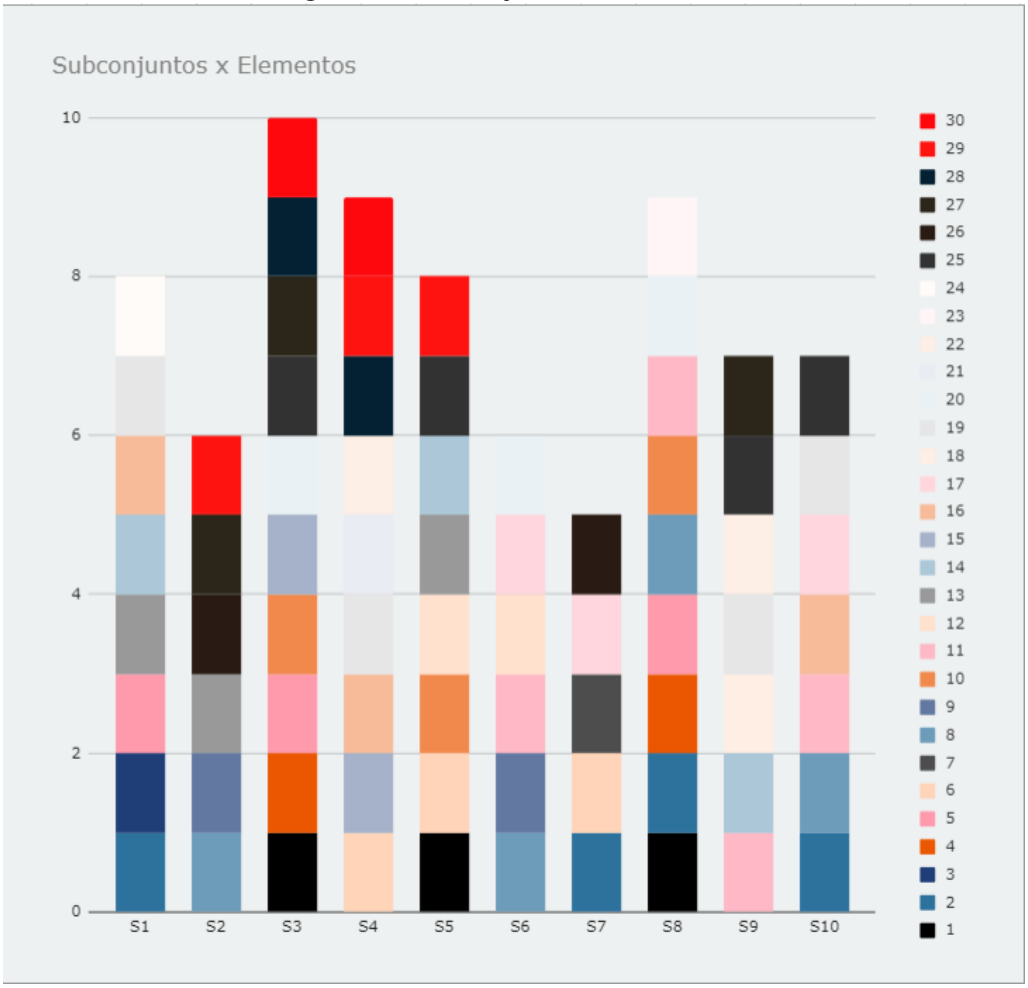
Primeiramente, calcularemos o custo por elemento:
$S1 = 9/8 = 1,125$
$S2 = 10/6 = 1,66$
$S3 = 6/10 = 0,6$
$S4 = 3/9 = 0,33$
$S5 = 5/8 = 0,625$
$S6 = 1/6 = 0,166$
$S7 = 1/5 = 0,2$
$S8 = 5/9 = 0,55$
$S9 = 6/7 = 0,85$
$S10 = 10/7 = 1,42$

Logo, verificamos que o subconjunto que possui menor custo por elemento é o S6. Incluiremos S6 na solução, removeremos do universo os elementos contemplados nesse subconjunto e os marcaremos como utilizados.

$$Universo = 1, 2, 3, 4, 5, 6, 7, 10, 13, 14, 15, 16, 18, 19, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30 \tag{2}$$

$Solução = \{S6, \}$
Em seguida, uma nova execução é feita, recalculando o custo por elemento. Como os elementos do subconjunto 6 foram utilizados, há alteração no custo por elemento de subconjuntos que compartilham tais elementos. Ou seja:
$S1 = 9/8 = 1,125$
$S2 = 10/4 = 2,5$
$S3 = 6/9 = 0,66$

Figure 2. Subconjuntos x Elementos



Fonte: Autor próprio

$S4 = 3/9 = 0,33$
$S5 = 5/7 = 0,714$
$S7 = 1/4 = 0,25$
$S8 = 5/6 = 0,833$
$S9 = 6/6 = 1,0$
$S10 = 10/4 = 2,5$

O subconjunto com menor custo é o S7. Adicionaremos tal subconjunto na solução, marcaremos os elementos como utilizados e os removeremos do universo.

$$Universo = 1, 3, 4, 5, 10, 13, 14, 15, 16, 18, 19, 21, 22, 23, 24, 25, 27, 28, 29, 30$$

$$Solução = \{S6, S7, \}$$

Recalculando os custos, temos:
$S1 = 9/7 = 1,285$
$S2 = 10/3 = 3,33$
$S3 = 6/9 = 0,66$

$S4 = 3/8 = 0,375$
$S5 = 5/6 = 0,833$
$S8 = 5/5 = 1,0$
$S9 = 6/6 = 1,0$
$S10 = 10/3 = 3,33$

Adicionando S4 à solução, teremos:
$Universo = 1, 3, 4, 5, 10, 13, 14, 18, 23, 24, 25, 27$
$Solução = \{S6, S7, S4\}$
$S1 = 9/5 = 1,8$
$S2 = 10/2 = 5,0$
$S3 = 6/6 = 1,0$
$S5 = 5/5 = 1,0$
$S8 = 5/5 = 1,0$
$S9 = 6/4 = 1,5$
$S10 = 10/1 = 10,0$

Adicionando S3 à solução, teremos:
$Universo = 3, 13, 14, 18, 23, 24,$
$Solução = \{S6, S7, S4, S3\}$
$S1 = 9/4 = 2,25$
$S2 = 10/2 = 1 = 5 = 10,0$
$S5 = 5/2 = 2,5$
$S8 = 5/1 = 5,0$
$S9 = 6/2 = 3,0$
$S10 =$
Com a inclusão do S3, todos os elementos de S10 já estarão na solução, já serão marcados como utilizados.

Adicionando S1 à solução, teremos:
$Universo = 18, 23$
$Solução = \{S6, S7, S4, S3, S1\}$
$S2 =$
$S5 =$
$S8 = 5/1 = 5,0$
$S9 = 6/1 = 6,0$
Com a inclusão do S1, todos os elementos de S2 e S5 já estarão na solução, já foram utilizados.

Adicionando S8 à solução, teremos:
$Universo = 18, 23$
$Solução = \{S6, S7, S4, S3, S1, S8\}$
$S9 = 6/1 = 6,0$

E, por fim, adicionando S9 à solução, contemplaremos todos os elementos com os seguintes subconjuntos:
$Solução = \{S1, S3, S4, S6, S7, S8, S9\}$

O custo total será de: $9 + 6 + 3 + 1 + 1 + 5 + 6 = 31$

O resultado é subótimo, já que podemos obter, pelo algoritmo de força bruta, a seguinte solução:

$$Solução = \{S1, S4, S6, S7, S8, S9\}$$

O custo total será de: $9 + 3 + 1 + 1 + 5 + 6 = 25$

Apesar dessa limitação, o algoritmo guloso da razão custo/quantidade de elementos não utilizados apresenta algumas vantagens, como sua velocidade e facilidade de implementação. Em muitos casos, ele fornece soluções suficientemente boas e próximas da solução ótima.

3.4. Implementação dos algoritmos

Como mencionado anteriormente na seção de análise de complexidade, utilizaremos dois algoritmos para analisar de maneira abrangente o problema da cobertura de conjuntos. Os algoritmos escolhidos são: um algoritmo que implementa uma abordagem de força bruta e um algoritmo guloso de aproximação. A seguir, apresentaremos os trechos de código correspondentes a cada um deles.

```
# Brute Force Algorithm
def brute_force(universe, subsets, costs):
    n = len(subsets)
    min_cost = float('inf')
    min_set = None

    for combo in chain(*map(lambda x: combinations(range(n)
       , x), range(0, n + 1))):
        cover = set().union(*(subsets[i] for i in combo))
        cost = sum(costs[i] for i in combo)

        if cover == set(range(1, universe + 1)) and cost <
           min_cost:
            min_cost = cost
            min_set = combo

    return min_set, min_cost
```

Listing 1. Função do Algoritmo Força Bruta

```
def greedy(universe, subsets, costs):
    universe = set(range(1, universe + 1))
    subset_indices = list(range(len(subsets)))
    used_subsets = []

    while universe:
        min_cost_per_element = float('inf')
        chosen_subset = None

        for i in subset_indices:
```

```
            subset = subsets[i]
            cost = costs[i]
            unused_elements = len(subset.intersection(
               universe))
            if unused_elements == 0:
                continue
            cost_per_element = cost / unused_elements
            if cost_per_element < min_cost_per_element:
                min_cost_per_element = cost_per_element
                chosen_subset = i

        used_subsets.append(chosen_subset)
        universe -= subsets[chosen_subset]
        subset_indices.remove(chosen_subset)

    return used_subsets, sum(costs[i] for i in used_subsets
       )
```

Listing 2. Função do Algoritmo Guloso

```
def cover_cost_scaling(universe, subsets, costs):
    num_subsets = len(subsets)
    remaining_elements = set(range(1, universe + 1))
    scaled_costs = [costs[i] / len(subsets[i]) for i in
       range(num_subsets)]
    selected_subsets = []
    total_cost = 0

    while remaining_elements:
        best_subset = None
        min_scaled_cost = float('inf')

        for i in range(num_subsets):
            if i not in selected_subsets:
                intersection_size = len(subsets[i].
                   intersection(remaining_elements))
                current_scaled_cost = scaled_costs[i] /
                   intersection_size if intersection_size >
                    0 else float('inf')

                if current_scaled_cost < min_scaled_cost:
                    min_scaled_cost = current_scaled_cost
                    best_subset = i

        if best_subset is not None:
            selected_subsets.append(best_subset)
            remaining_elements -= subsets[best_subset]
            total_cost += costs[best_subset]
```

```
        else:
            break

    return selected_subsets, total_cost
```

Listing 3. Função do Algoritmo Cover Cost Scaling

```
import time
import random
from itertools import chain, combinations


# Brute Force Algorithm O(n^2)
def brute_force(universe, subsets, costs):
    n = len(subsets)
    min_cost = float('inf')
    min_set = None

    for combo in chain(*map(lambda x: combinations(range(n), x), range(0, n + 1))):
        cover = set().union(*(subsets[i] for i in combo))
        cost = sum(costs[i] for i in combo)

        if cover == set(range(1, universe + 1)) and cost < min_cost:
            min_cost = cost
            min_set = combo

    return min_set, min_cost

# Greedy Algorithm O(n^2)
def greedy(universe, subsets, costs):
    universe = set(range(1, universe + 1))
    subset_indices = list(range(len(subsets)))
    used_subsets = []

    while universe:
        min_cost_per_element = float('inf')
        chosen_subset = None

        for i in subset_indices:
            subset = subsets[i]
            cost = costs[i]
            unused_elements = len(subset.intersection(universe))
            if unused_elements == 0:
                continue
            cost_per_element = cost / unused_elements
```

```
            if cost_per_element < min_cost_per_element:
                min_cost_per_element = cost_per_element
                chosen_subset = i

        used_subsets.append(chosen_subset)
        universe -= subsets[chosen_subset]
        subset_indices.remove(chosen_subset)

    return used_subsets, sum(costs[i] for i in used_subsets)

# Approximation by Cost Algorithm O(log n)
def cover_cost_scaling(universe, subsets, costs):
    num_subsets = len(subsets)
    remaining_elements = set(range(1, universe + 1))
    scaled_costs = [costs[i] / len(subsets[i]) for i in range(num_subsets)]
    selected_subsets = []
    total_cost = 0

    while remaining_elements:
        best_subset = None
        min_scaled_cost = float('inf')

        for i in range(num_subsets):
            if i not in selected_subsets:
                intersection_size = len(subsets[i].intersection(remaining_elements))
                current_scaled_cost = scaled_costs[i] / intersection_size if intersection_size > 0 else float('inf')

                if current_scaled_cost < min_scaled_cost:
                    min_scaled_cost = current_scaled_cost
                    best_subset = i

        if best_subset is not None:
            selected_subsets.append(best_subset)
            remaining_elements -= subsets[best_subset]
            total_cost += costs[best_subset]
        else:
            break

    return selected_subsets, total_cost

# Data sets for testing
```

```
def test_data(i):
    if i == 0:
        universe = 5
        subsets = [ set([4, 1, 3]), set([2, 5]), set([1, 4,
            3, 2]) ]
        costs = [5, 10, 3]

        return universe, subsets, costs

    if i == 1:
        universe = 5
        subsets = [set([1, 2, 3]), set([3, 4]), set([4, 5])
           , set([1, 2, 5])]
        costs = [2, 3, 4, 5]

        return universe, subsets, costs

    if i == 2:
        universe = 7
        subsets = [set([1, 2, 3]), set([3, 4]), set([4, 5])
           , set([1, 2, 5]), set([6, 7])]
        costs = [2, 3, 4, 5, 1]

        return universe, subsets, costs

    if i == 3:
        universe = 4
        subsets = [set([1, 2]), set([2, 3]), set([3, 4]),
           set([1, 4])]
        costs = [2, 3, 4, 1]

        return universe, subsets, costs

    if i == 4:
        universe = 4
        subsets = [set([1, 2]), set([2, 3]), set([3]), set
           ([1, 4])]
        costs = [2, 3, 1, 5]

        return universe, subsets, costs

    if i == 5:
        universe = 5
        subsets = [frozenset([1, 2]), frozenset([3, 4]),
           frozenset([1, 3]), frozenset([2, 4]), frozenset
           ([5])]
```

```
        costs = [1, 2, 3, 4, 5]

        return universe, subsets, costs

    if i == 6:
        universe = 50
        subsets = [frozenset([i, i+1]) for i in range(1,
           51, 2)]
        costs = [1 for _ in range(25)]

        return universe, subsets, costs

    if i == 50:
        universe = 100
        subsets = [frozenset([i, i+1]) for i in range(1,
           101, 2)]
        costs = [1 for _ in range(50)]

        return universe, subsets, costs

    if i == 7:
        universe = 30

        # Generate random subsets
        num_subsets = 10
        subsets = [frozenset(random.sample(range(1,
           universe + 1), random.randint(5, 10))) for _ in
           range(num_subsets)]

        # Generate random costs
        costs = [random.randint(1, 10) for _ in range(
           num_subsets)]

        return universe, subsets, costs

    if i == 60:
        universe = 50

        # Generate random subsets
        num_subsets = 30
        subsets = [frozenset(random.sample(range(1,
           universe + 1), random.randint(5, 10))) for _ in
           range(num_subsets)]

        # Generate random costs
        costs = [random.randint(1, 10) for _ in range(
```

```
            num_subsets)]

        return universe, subsets, costs

for i in range(0, 8):
    universe, subsets, costs = test_data(i)

    while True:
        try:
            start = time.time()
            result = brute_force(universe, subsets, costs)
            end = time.time()

            print("Brute Force result: ", result, "Elapsed time: ", end - start, "(", i, ")")

            start = time.time()
            result = greedy(universe, subsets, costs)
            end = time.time()

            print("Greedy result: ", result, "Elapsed time: ", end - start, "(", i, ")")

            start = time.time()
            result = cover_cost_scaling(universe, subsets, costs)
            end = time.time()

            print("Approximation by Cost result: ", result, "Elapsed time: ", end - start, "(", i, ")")

            print("--------------------")
            print("Universe: ", universe)
            print("Subsets: ", subsets)
            print("Costs: ", costs)
            print("--------------------")

            break
        except Exception as e:
            print(f"An unexpected error occurred: {e}")
            print("Trying again with new data set...")
            print("--------------------")

            universe = 30

            # Generate random subsets
```

```
199             num_subsets = 10
200             subsets = [frozenset(random.sample(range(1,
                    universe + 1), random.randint(5, 10))) for _
                     in range(num_subsets)]
201
202             # Generate random costs
203             costs = [random.randint(1, 10) for _ in range(
                    num_subsets)]
```

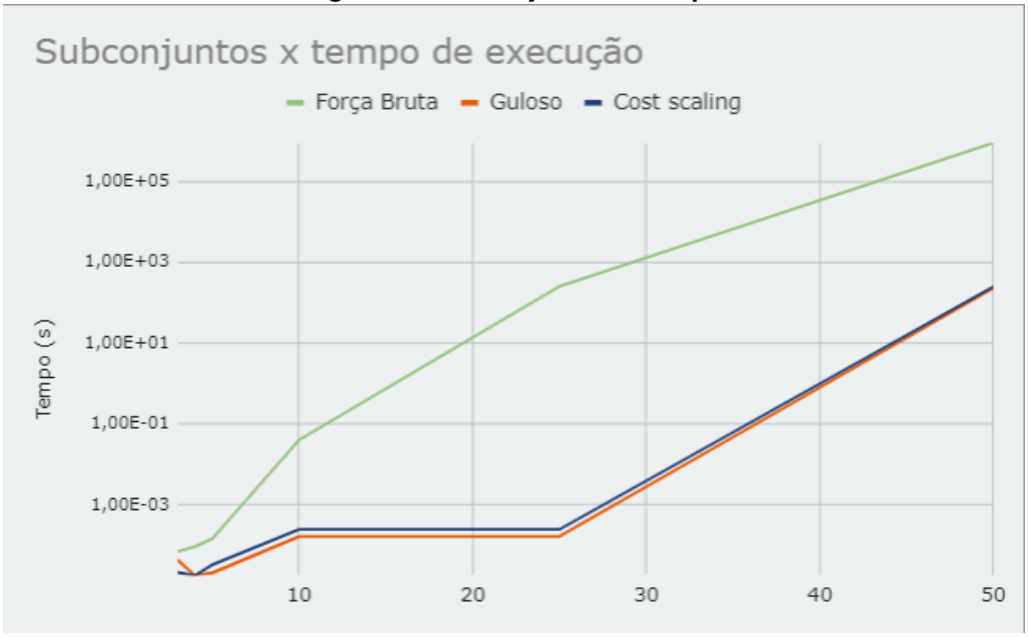
Listing 4. Algoritmo completo

4. Resultados

Foram realizados vários testes de tempo de execução do problema da cobertura de conjuntos para verificar, na prática, a análise de complexidade dos algoritmos e sua viabilidade de execução. Para isso, calculamos a diferença de tempo antes e depois da execução do algoritmo. Para analisar o comportamento dos algoritmos, variamos os valores de n - que corresponde ao número de subconjuntos - e calculamos a média dos valores de tempo. Esses resultados serão plotados em um gráfico para observar se o comportamento do algoritmo coincide com a teoria.

4.1. Comparativo dos resultados

Figure 3. Subconjuntos x Tempo



Fonte: Autor próprio

4.2. Análise dos resultados

A análise do tempo de execução destes algoritmos fornece informações valiosas sobre a eficiência de cada abordagem. O algoritmo de força bruta, como indicado pelos tempos de execução exponenciais, revela sua complexidade computacional significativamente alta.

A complexidade exponencial implica que o tempo de execução aumenta de maneira exponencial com o tamanho do conjunto de dados, tornando-o impraticável para problemas de escala moderada ou grande.

Em contraste, os algoritmos guloso e cost cover scaling demonstram tempos de execução notavelmente mais baixos. Essa eficiência relativa sugere que esses métodos possuem uma complexidade computacional inferior, tornando-os mais adequados para lidar com conjuntos de dados extensos.

Além disso, a semelhança nos tempos de execução entre os algoritmos guloso e cost cover scaling destaca uma equivalência prática em termos de eficiência. Ambos os métodos conseguem encontrar soluções de maneira eficaz, e a escolha entre eles pode depender de considerações adicionais, como a natureza específica do problema e a facilidade de implementação.

Em resumo, a análise do tempo de execução e da complexidade dos algoritmos reforça a noção de que, em termos práticos, os algoritmos guloso e cost cover scaling oferecem uma eficiência superior, enquanto o algoritmo de força bruta se torna impraticável para conjuntos de dados maiores devido à sua complexidade exponencial.

Na análise das soluções geradas pelos algoritmos, fica evidente que, em casos menos complexos, os resultados podem ser comparáveis entre os algoritmos. No entanto, a verdadeira disparidade entre eles se manifesta quando lidamos com conjuntos de dados consideravelmente maiores.

A eficiência dos algoritmos guloso e cost cover scaling se destaca em sua capacidade de encontrar soluções praticamente idênticas em um período de tempo razoável, enquanto o algoritmo de força bruta apresenta não apenas tempos de execução proibitivos, mas também pode gerar soluções distintas devido à sua abordagem exaustiva.

Assim, embora os resultados possam convergir para conjuntos de dados menores, a diferença nos tempos de execução reforça a importância de escolher métodos mais eficientes para garantir a resolução eficaz de problemas mais desafiadores.

5. Conclusão

Com base nas análises detalhadas e nos resultados obtidos ao explorar diferentes abordagens para o problema da cobertura de conjuntos, é possível chegar a conclusões importantes que têm implicações na escolha de algoritmos para essa classe de problemas NP-completos.

Como esperado, ficou claramente demonstrado que o problema de Set Cover é NP-completo, destacando a intratabilidade de encontrar soluções ótimas em tempo polinomial para instâncias gerais do problema.

Entre os algoritmos examinados, o algoritmo guloso destaca-se pela sua eficiência, apresentando tempos significativamente menores em comparação com outras abordagens, como a força bruta. No entanto, é crucial ressaltar que a eficiência do algoritmo guloso não vem sem um compromisso: apesar de oferecer uma execução rápida, não há garantia de uma solução ótima em todos os casos. A natureza heurística do algoritmo de aproximação pode resultar em soluções aproximadas que podem se desviar do ótimo global, dependendo da instância do problema.

Por outro lado, o algoritmo de força bruta, apesar de garantir a solução ótima, é prejudicado por tempos de execução impraticáveis para conjuntos de dados de tamanho considerável. A busca exaustiva por todas as combinações possíveis torna-se inviável, destacando a necessidade de abordagens mais eficientes para problemas práticos.

Ao introduzir o algoritmo de cover cost, observamos que esta abordagem oferece um equilíbrio interessante. Seu desempenho intermediário entre o guloso e o de força bruta sugere uma potencial aplicação em cenários onde a busca por soluções ótimas é crucial, mas os recursos computacionais são limitados. A estratégia de custo de cobertura adiciona uma camada de flexibilidade, permitindo que o algoritmo ajuste seu comportamento com base nas características específicas do problema.

Além disso, ao considerar o contexto do Gerenciamento de Conteúdo em Plataformas de Mídia Social, percebemos que a aplicabilidade prática desses algoritmos se estende a cenários modernos e dinâmicos. A capacidade de lidar eficientemente com conjuntos de dados representativos de conteúdo em redes sociais destaca a relevância dessas abordagens em contextos de aplicação específicos.

Em resumo, a escolha do algoritmo apropriado para o problema de cobertura de conjuntos é uma ponderação cuidadosa entre tempo de execução e qualidade da solução. O algoritmo guloso oferece eficiência, o algoritmo de força bruta garante otimalidade, e o algoritmo de cover cost apresenta uma abordagem intermediária, refletindo as complexidades inerentes à resolução deste desafiador problema NP-completo, com considerações específicas para o gerenciamento de conteúdo em plataformas de mídia social.

6. Referências Bibliográficas

[Smith 2020] fornece insights sobre estratégias de gerenciamento de conteúdo em mídias sociais, destacando a relevância do problema abordado neste trabalho.

O estudo de [Johnson 2018] analisa o impacto do conteúdo gerado pelo usuário na interação em mídias sociais, relacionando-se diretamente ao tema de gerenciamento de conteúdo em plataformas sociais.

[Rodriguez and Chen 2019] abordam algoritmos de otimização de conteúdo, um aspecto fundamental do gerenciamento de conteúdo em mídias sociais.

A pesquisa de [Wang and Lee 2017] analisa práticas de curadoria de conteúdo em mídias sociais, relevante para estratégias de gerenciamento de conteúdo.

[Garcia and Patel 2016] discutem a eficiência na distribuição de conteúdo, um aspecto crítico do gerenciamento de conteúdo em mídias sociais.

[Brown and Thomas 2019] avaliam a importância de conteúdo visual em campanhas de marketing nas mídias sociais, relacionando-se ao conteúdo gerenciado.

[Martinez and Clark 2018] exploram abordagens heurísticas para escalonamento de conteúdo em marketing nas mídias sociais, relevante para estratégias de gerenciamento.

O estudo de [Kim and Smith 2017] realiza uma análise de sistemas de recomendação de conteúdo, um elemento importante em estratégias de gerenciamento de conteúdo em mídias sociais.

[Liu and Yang 2016] contribuem para a compreensão do envolvimento do usuário, um fator-chave no sucesso das estratégias de gerenciamento de conteúdo.

[Chen and Rodriguez 2020] exploram abordagens algorítmicas para prever o desempenho de conteúdo, um componente crucial do gerenciamento de conteúdo em mídias sociais.

7. Anexos

References

Brown, D. and Thomas, E. (2019). Evaluating the role of visual content in social media marketing campaigns. *International Journal of Marketing Studies*, 11(3):104–118.

Chen, C. and Rodriguez, A. (2020). An algorithmic approach to content performance prediction in social media marketing. *Journal of Digital Strategy*, 13(4):267–281.

Garcia, L. and Patel, R. (2016). Towards efficient content distribution in social media platforms. *Journal of Social Media Management*, 4(1):17–29.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

Johnson, A. (2018). The impact of user-generated content on social media engagement. *International Journal of Communication*, 12:3421–3440.

Kaplan, A. M. and Haenlein, M. (2010). Users of the world, unite! the challenges and opportunities of social media. *Business Horizons*, 53(1):59–68.

Kim, H. and Smith, P. (2017). A survey of content recommendation systems in social media platforms. *Journal of Interactive Advertising*, 17(2):87–101.

Liu, W. and Yang, Q. (2016). Understanding user engagement in social media through content analysis. *International Journal of Social Media and Interactive Learning Environments*, 4(1):45–59.

Martinez, S. and Clark, M. (2018). Heuristic approaches to content scheduling in social media marketing. *Journal of Marketing Optimization*, 6(2):79–95.

Michalewicz, Z. (2004). *Heuristic Algorithms for Scheduling and Network Design*. Springer.

Rodriguez, M. and Chen, H. (2019). A comparative analysis of content optimization algorithms for social media marketing. *Journal of Online Advertising*, 21(4):56–71.

Smith, J. (2020). Content management strategies in social media platforms. *Journal of Digital Marketing*, 8(2):123–140.

Tuten, T. L. and Solomon, M. R. (2018). *Social Media Marketing*. SAGE Publications.

Vazirani, V. V. (2001). *Approximation Algorithms*. Springer.

Wang, Q. and Lee, K. (2017). An analysis of social media content curation practices in e-commerce. *Journal of Information Science*, 45(3):318–332.