Trabalho Final - Projeto e Análise de Algoritmos

Gabriel Samarane [**Pontifícia Universidade Católica de Minas Gerais (PUCMG)** | ***samarane.gabriel@gmail.com***]
João Lucas Curi [**Pontifícia Universidade Católica de Minas Gerais (PUCMG)** | ***joaolucascuri@gmail.com***]
João Madeira [**Pontifícia Universidade Católica de Minas Gerais (PUCMG)** | ***joaomadeira1208@gmail.com***]
Sophia Carrazza [**Pontifícia Universidade Católica de Minas Gerais (PUCMG)** | ***sophiacarrazza7@gmail.com***]

Departamento de Ciência da Computação (DCC) – Pontifícia Universidade Católica de Minas Gerais (PUCMG)
Caixa Postal 2071 – 30.535-901 – Belo Horizonte – MG – Brasil

Abstract. Visando a influência de métodos para a medida de distância de edição de árvores (TED), este estudo apresenta a implementação, em C++, do algoritmo de TED proposto por Zhang Shasha (1989) e de um segundo método guloso proposto pelo grupo. Comparamos os custos computacionais (em tempo de execução e uso de memória) de ambos algoritmos conforme o número de vértices das árvores de teste aumenta.

Keywords: Distância de Edição de Árvores, Algoritmo de Zhang Shasha, Complexidade de Tempo e Espaço

1 Introdução

A distância de edição de árvores (TED) permanece reconhecida como uma técnica revolucionária que modernizou áreas como bioinformática e análise de imagens, fornecendo uma métrica quantitativa para dissimilaridade estrutural hierárquica.

Dados estruturados em árvore constantemente são encontrados na bioinformática. Estruturas secundárias de RNA, glicanos e árvores filogenéticas geralmente possuem estruturas em árvore [1], por exemplo. Dessa forma, essa medida foi capaz de impactar o estudo dessas estruturas, na identificação de similaridades de árvores e problemas relacionados. A metodologia desenvolvida por Zhang and Shasha, permanece referência por seu equilíbrio entre eficiência e precisão. Alternativas como AP-TED (sequencial) e MC-TED (multicore) buscam otimizações [3], mas enfrentam desafios de dependência de dados e uso de recursos.

Este estudo, portanto, propõe uma aplicação da solução proposta pelo trabalho de Zhang and Shasha, em linguagem C++, e uma solução gulosa, além de uma comparação entre ambos algoritmos.

2 Background

2.1 Distância de edição de árvores (TED)

Uma árvore é um grafo $G = (V, E)$ onde V é um conjunto finito de vértices, $E \subseteq V \times V$ é um conjunto de arestas e G é não-direcionado, conexo e acíclico.

Uma TED entre duas árvores T1 e T2 pode ser definida como o número mínimo de vértices que precisam ser substituídos, excluídos ou inseridos em T1 para obter T2.

2.2 TED e Aplicações na Bioinformática

A representação de dados biológicos como árvores ordenadas é intrínseca a própria natureza biológica, justificando o uso de algoritmos de Distância de Edição de Árvores (TED). Essa forma estrutural permite comparação e análise estrutural de entidades biológicas de alta complexidade, já que os dados biológicos se encaixam naturalmente nessa representação. O RNA secundário, por exemplo, representa a estrutura bidimensional de uma molécula de RNA, caracterizada por regiões de pareamento de bases e regiões de fita simples. Os pares de bases são modelados como nós internos e regiões de fita simples se tornam folhas [Bertrand Marchand and Ouangraoua]. As Filogenias, por outro lado, descrevem as relações evolutivas entre diferentes espécies, genes ou outras entidades biológicas. Elas são geralmente árvores com folhas rotuladas por espécies [4].

2.3 Programação Dinâmica e Problemas de Otimização

Programação dinâmica (PD) é uma técnica que resolve problemas de otimização quebrando-os em subproblemas sobrepostos e aplicando o *princípio da optimalidade*: a solução ótima global pode ser construída a partir de soluções ótimas de subpartes. Dois paradigmas, comuns para cálculo de distância de edição de árvores, são resoluções naturais de programação dinâmica [3]: o top-down (memoização), resolvendo subproblemas sob demanda, armazenando resultados em tabela para evitar recomputação, e o bottom-up (iterativo), preenchendo a tabela em ordem não-decrescente de tamanho, garantindo que cada entrada dependa apenas de valores já calculados.

No contexto da TED, cada subproblema corresponde ao menor custo para converter uma subárvore de T_1 em outra de T_2. O algoritmo de [7] define uma recorrência sobre pares de índices de pós-ordem e preenche uma matriz de $|T_1| \times |T_2|$ posições; a resposta final é recuperada na célula $(|T_1|, |T_2|)$. Essa abordagem reduz a busca ingênua exponencial para tempo $O(n^3)$ e espaço $O(n^2)$.

Algoritmos recentes, como o APTED [5], mantêm o modelo de PD, porém exploram heurísticas de poda e reuso de subestruturas para diminuir o número efetivo de subproblemas, aproximando o custo de execução de $O(n^2)$ na prática.

Assim, a Programação Dinâmica é essencial para justificar a formulação do algoritmo clássico.

3 Trabalhos Relacionados

No contexto da criação de diversas abordagens propostas para a distância de edição de árvores, alguns trabalhos se destacam para este estudo.

O algoritmo de Zhang and Shasha emprega uma abordagem de programação dinâmica que, ao contrário dos seus predecessores, considera de forma estratégica a distância entre florestas (conjuntos de árvores) ordenadas como uma etapa intermediária para simplificar o cálculo da distância entre árvores. O projeto de Zhang e Shasha também é eficientemente paralelizado, alcançando uma complexidade temporal de O(|T1| + |T2|).

Já o trabalho de [6], propõe a aplicação da TED para o cálculo de similaridade entre frases, com objetivo de detectar similaridade suave entre textos, uma tarefa de Processamento de Linguagem Natural (PLN). Este estudo também comparou a implementação feita por TED com a utilização de N-gramas sintáticos, insuficientes para resolver essas tarefas com precisão. Assim, os resultados demonstraram que a medida TED pode ser utilizada de forma eficaz e vantajosa em aplicações de outras áreas além da biotecnologia.

4 Metodologia

Esta seção descreve a implementação assim como as justificativas associadas ao algoritmo de abordagem gulosa não ótima desenvolvida, em comparação com o método exato de [7](**ZS**). Dividimos a explicação em cinco subseções para facilitar a reprodução dos testes.

4.1 Abordagem Gulosa

Diferente do algoritmo de Zhang ([7]), que percorre as árvores em ordem *pós-ordem*, a abordagem gulosa apresentada visita os nós em ordem *pré-ordem*, ou seja, o nó pai é processado antes de seus filhos.

O algoritmo percorre as árvores recursivamente, comparando os nós em paralelo, sempre na ordem em que os filhos aparecem nas respectivas listas. Para cada par de nós, calcula-se um custo de substituição caso suas etiquetas sejam diferentes. Em seguida, os filhos são comparados um a um, na mesma posição (alinhamento posicional).

Ele é considerado guloso porque toma decisões imediatas, comparando diretamente os filhos de cada nó em posições simétricas, sem reordenar nem avaliar alternativas. As decisões são locais e independentes. Cada par de nós é avaliado com base apenas no estado atual, sem considerar o impacto global ou depender de decisões anteriores.

4.1.1 Análise do Algoritmo

Sejam $n = |T_1|$ e $m = |T_2|$ o número de vértices das árvores T_1 e T_2, respectivamente, e $h = \max\{\text{altura}(T_1), \text{altura}(T_2)\}$.

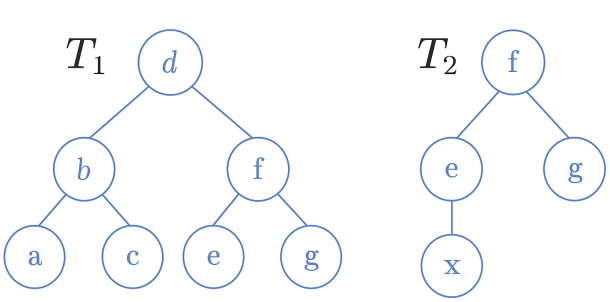

Figure 1. Contra-exemplo de Otimalidade do Algoritmo Guloso

- **Complexidade de tempo:** $\mathcal{O}(n + m)$.
- **Complexidade de espaço:** $\mathcal{O}(h)$.
- **Limitação:** não garante a distância de edição mínima; produz apenas uma estimativa d_{gulosa}.

Para constatar a complexidade de tempo, analisou-se o comportamento do algoritmo em relação ao caminhamento realizado nas árvores. Como a estratégia gulosa apenas caminha para frente, ou seja, verifica um vértice após o outro sem retornar nem reavaliar subárvores ou decisões anteriores, os vértices de ambas as árvores são completamente percorridos. Assim, conclui-se que o número de operações realizadas pelo algoritmo é proporcional à soma da quantidade de vértices das duas árvores.

No que diz respeito à complexidade de espaço, a análise considera o uso da pilha de chamadas recursivas. Como a recursão acompanha a estrutura das árvores e avança por seus ramos, a profundidade máxima da pilha será limitada pela altura da árvore mais alta. Dessa forma, a complexidade de espaço do algoritmo é proporcional à altura das árvores envolvidas.

4.2 Não Otimalidade da Abordagem Gulosa

Para demonstrar a não otimalidade do algoritmo guloso implementado, utilizou-se como referência o algoritmo ótimo descrito em Zhang and Shasha. A abordagem consistiu em executar ambos os algoritmos sobre as mesmas instâncias de árvores e comparar os resultados obtidos em termos da distância de edição calculada.

As árvores utilizadas para o experimento estão representadas na imagem 1.

O algoritmo ótimo retornou um custo total de 15 operações, enquanto o algoritmo guloso produziu um resultado de 17 operações. Esse exemplo evidencia, por meio de um contraexemplo concreto, que o algoritmo guloso não garante a obtenção da distância mínima, confirmando assim sua natureza não ótima.

4.3 Métricas Avaliadas

Esta seção descreve as métricas de avaliação entre os dois algoritmos.

Para os testes, os algoritmos receberam como entrada pares de árvores geradas aleatoriamente, restringidas em 50

vértices no máximo e seguindo uma distribuição normal. Foram testadas 10.000 pares distintos de árvores.

1. **Erro relativo de distância**

$$\text{RE} = \frac{d_{\text{gulosa}} - d_{\text{ZS}}}{d_{\text{ZS}}} \times 100\%$$

2. **Tempo Médio de Execução**: Comparação entre tempo médio de execução dos dois algoritmos sob as mesmas árvores de teste.
3. **Uso Médio de Memória**: Comparação entre uso médio de memória dos dois algoritmos sob as mesmas árvores de teste.
4. **Distância Média de Edição**: Comparação entre as médias de distância de edição dos dois algoritmos sobre as mesmas árvores de teste.

4.4 Justificativas para a Abordagem Gulosa

1. **Complexidade Computacional**: aplicações em, por exemplo, bioinformática frequentemente lidam com milhares de árvores; um algoritmo $O(n^3)$ torna-se inviável. A abordagem gulosa, por sua vez, apresenta complexidade significativamente menor, operando em $O(n)$, tornando-a viável para grandes volumes de dados.
2. **Facilidade de Implementação**: o algoritmo guloso possui estrutura simples e direta, o que facilita sua implementação, manutenção e adaptação para diferentes formatos de dados.
3. **Baixo Consumo de Memória**: a estratégia gulosa exige apenas memória proporcional à altura das árvores, reduzindo o custo espacial e permitindo execução mesmo em ambientes com recursos limitados.

Os resultados numéricos e a análise detalhada desses critérios são apresentados na Seção Resultados.

4.5 Implementação do Algoritmo Guloso

Esta seção descreve, em pseudo-código, a implementação do algoritmo guloso.

4.5.1 Pseudo-Código

Algorithm 1 TED-Guloso

```
1:  function CompararArvoresGuloso(n1, n2)
2:      if n1 e n2 são nulos then
3:          return 0
4:      else if n1 é nulo then
5:          return custo de inserção + soma das chamadas
    recursivas para os filhos de n2
6:      else if n2 é nulo then
7:          return custo de remoção + soma das chamadas
    recursivas para os filhos de n1
8:      else
9:          custo ← 0 se n1.label = n2.label, senão custo
    de substituição
10:         Alinhar os filhos de n1 e n2 ordenados
11:         Adicionar ao custo o resultado das chamadas re-
    cursivas para:
              • pares de filhos alinhados
              • filhos restantes de n1 (remoções)
              • filhos restantes de n2 (inserções)
12:         return custo
13:     end if
14: end function
15: function TED-Guloso(a1, a2)
16:     return CompararArvoresGuloso(raiz de a1, raiz de
    a2)
17: end function
```

O algoritmo não faz uso de estruturas auxiliares explícitas além da própria definição da árvore. Toda a execução ocorre por meio de chamadas recursivas, utilizando implicitamente a pilha de chamadas da linguagem como mecanismo de controle de fluxo.

4.6 Implementação do Algoritmo de Zhang e Shasha

A implementação do algoritmo de distância de edição de árvores seguiu o modelo proposto por [7], dividido em duas etapas principais: o pré-processamento das árvores e o cálculo da distância propriamente dito.

4.6.1 Pré-processamento

Nesta etapa inicial, as duas árvores de entrada são convertidas para uma numeração em pós-ordem. Para cada nó i, são armazenadas duas informações importantes: o índice de sua folha mais à esquerda, $l(i)$, e seus ancestrais. Em seguida, é calculado o conjunto LR_keyroots(T) para cada árvore T, que identifica as raízes de subárvores distintas. Esse conjunto reduz o número de comparações necessárias, otimizando o desempenho do algoritmo.

4.6.2 Etapa Principal: função `treedist(i, j)`

A parte central do algoritmo envolve dois laços que percorrem os conjuntos LR_keyroots das duas árvores. Para cada

par de nós i e j, a função `treedist(i, j)` calcula a distância entre as subárvores enraizadas nesses nós.

O cálculo é feito por meio de programação dinâmica, preenchendo uma matriz de distâncias entre as subárvores de T_1 e T_2. Quando os nós i e j são as raízes de subárvores completas, a distância entre elas é definida como o mínimo entre as operações de remoção, inserção e substituição de nós. Caso contrário, a função `treedist` é chamada recursivamente para resolver os casos menores.

Essa estratégia permite calcular a distância total entre as duas árvores com um bom equilíbrio entre precisão e eficiência.

A implementação resultou em uma complexidade de tempo proporcional a

$$O(|T_1| \times |T_2| \times \min(\text{profundidade}, \text{número de folhas})),$$

e uma complexidade de espaço quadrática, ou seja, $O(n^2)$, onde n representa o número de nós.

5 Resultados

Esta seção apresenta os resultados comparativos entre o algoritmo de Zhang e Shasha (ZS) e o método guloso proposto. A Tabela 1 sumariza essas comparações de desempenho. O erro relativo calculado para o parâmetro de distância de edição foi de 36,38%. para a o parâmetro distãncia de edição foi calculado em: 36.38%.

Table 1. Análise comparativa de desempenho: Zhang Shasha vs. Guloso

Métrica	**Zhang & Shasha**	**Guloso**
Consumo Médio de Memória(KB)	0.2931	16.75
Média Distância de Edição	76.88	104.83
Tempo Médio de Execução (μs)	308	1.02

A análise dos resultados apresentados na Tabela 1 confirma as expectativas teóricas de desempenho. O guloso se mostrou superior em tempo de execução, sendo significativamente mais rápido. Essa velocidade o torna ideal para lidar com grandes volumes de dados, onde a complexidade cúbica do ZS seria inviável.

Em relação à memória, o ZS consumiu menos para as árvores testadas. Contudo, devido à sua natureza de complexidade quadrática, para árvores muito maiores, o ZS se tornaria insustentável em termos de memória, enquanto o guloso manteria sua eficiência linear $O(\max(|T_1|, |T_2|))$.

A distância de edição média do ZS foi de 76.88, contra 104.83 do guloso, o que quantifica a não-otimalidade do método guloso e mostra que ele sacrifica a precisão pela velocidade.

Em suma, os resultados reforçam o equilíbrio entre otimalidade e eficiência. O ZS é preferível para precisão máxima, enquanto o guloso é a melhor escolha para lidar com grandes volumes de dados, mesmo que com uma solução não-ótima.

6 Conclusão e Trabalhos Futuros

Este trabalho analisou e comparou dois algoritmos para o cálculo da distância de edição entre árvores: o algoritmo ótimo de Zhang e Shasha e uma abordagem gulosa implementada em C++. Os resultados mostraram que, embora o algoritmo clássico produza a distância mínima, ele apresenta alto custo computacional para árvores grandes. O algoritmo guloso, por outro lado, alcança complexidade linear e maior eficiência, embora sem garantir otimalidade.

Essa característica o torna mais adequado para aplicações que envolvem grandes volumes de dados, como na bioinformática. Como trabalhos futuros, propõe-se a aplicação dos algoritmos em outros domínios, uma análise mais aprofundada do uso de memória e o desenvolvimento de implementações paralelas, especialmente para o método guloso.

References

[1] Akutsu, T. (2010). Tree Edit Distance Problems: Algorithms and Applications to Bioinformatics. *IEICE Transactions on Information and Systems*, E93-D(2):208–218. DOI: 10.1587/transinf.E93.D.208.

[Bertrand Marchand and Ouangraoua] Bertrand Marchand, Yoann Anselmetti, M. L. and Ouangraoua, A. Median and Small Parsimony Problems on RNA trees.

[3] Dayi Fan, D., Lee, R., and Zhang, X. (2024). X-TED: Massive Parallelization of Tree Edit Distance. *Proceedings of the VLDB Endowment*, 17(7):1683–1696. DOI: 10.14778/3654621.3654634.

[4] Felsenstein, J. (2004). *INFERRING PHYLOGENIES*. Sinauer Associates, Inc. • Publishers Sunderland, Massachusetts.

[5] Pawlik, M. and Augsten, N. (2015). Tree edit distance: Robust and memory-efficient. *Information Systems*, 56. DOI: 10.1016/j.is.2015.08.004.

[6] Sidorov, G., Gomez-Adorno, H., Markov, I., Pinto, D., and Loya, N. (2015). Computing text similarity using Tree Edit Distance. In *2015 Annual Conference of the North American Fuzzy Information Processing Society (NAFIPS) held jointly with 2015 5th World Conference on Soft Computing (WConSC)*, pages 1–4, Redmond, WA, USA. IEEE. DOI: 10.1109/NAFIPS-WConSC.2015.7284129.

[7] Zhang, K. and Shasha, D. (1989). Simple Fast Algorithms for the Editing Distance between Trees and Related Problems. *SIAM Journal on Computing*, 18(6):1245–1262. DOI: 10.1137/0218082.