O fantástico mundo da distância de edição

Bruno Maciel Fonseca Davi de Castro Reis

03 de julho de 2002

Resumo

Apresentamos neste trabalho o problema de casamento em cadeias de caracteres permitindo erros, também chamado simplesmente de casamento aproximado de caracteres, juntamente com o problema correlato da distância de edição. São mostrados e discutidos os algoritmos existentes para os problemas e alguns experimentos da literatura.

1 Introdução

Apresentamos neste trabalho o problema de casamento em cadeias de caracteres permitindo erros, também chamado simplesmente de casamento aproximado de caracteres. Na sua forma mais geral o problema pode ser descrito como a tarefa de encontrar num texto um determinado padrão, permitindo um número limitado de "erros" entre o texto original e o padrão.

As primeiras referências deste problema remontam à década de 60, onde as principais motivações eram relacionadas a área de computação biológica, processamento de sinais e recuperação de informação. Hoje a área cresce com o surgimento de aplicações em outras áreas, como reconhecimento de letras escritas a mão, detecção de vírus e invasão em computadores, compressão de imagens, mineração de dados, reconhecimento de padrões, comparação e correção de textos, entre outras.

O principal objetivo deste trabalho é mostrar o estado da arte entre os algoritmos de casamento aproximado de padrões. Infelizmente, a natureza deste tipo de problema é extremamente dependente do tipo de erro considerado. Então este trabalho é focado num subconjunto entre os vários modelos de erros possíveis. Comumente considera-se um modelo de erros onde são permitidas a remoção, a inserção ou a substituição de caracteres de ambos os textos, e a cada operação associa-se um custo. A distância de edição é o custo do conjunto de operação de custo mínimo necessário para transformar um texto em outro. Considerando-se o custo unitário para qualquer operação, temos que, por exemplo, a distância de edição entre as palavras *soneca* e *senegal* é 3.

A distância de edição é bem útil porque sua versão mais geral é poderosa o suficiente para uma quantidade muito grande de aplicações. Apesar do fato dos algoritmos aqui descritos serem essencialmente utilizados para cálculo da distância de edição entre dois textos, muitos outros problemas, como detecção de padrões de RNA, processamento de imagens ou inferência de linguagens podem ser modelados sobre técnicas de distância de edição. Esta propriedade de se apresentar como um modelo geral para diversas aplicações em computação é que torna fantástico o trabalho com a distância de edição.

1.1 Conceitos básicos

Antes de introduzirmos os algoritmos, alguns conceitos são importantes para o bom entendimento deles. Neste trabalho usamos as letras s, x, y, z, v, w para representar cadeias de caracteres (*strings*) e as letras $a, b, c, ...$ representam os caracteres. Uma sequência de *strings* ou/e letras representa concatenação. Representamos o alfabeto pela letra grega Σ. Para qualquer *string* $s \in \Sigma$ o seu comprimento é dado por $|s|$. Além disso s_i representa o iésimo caracter de s, sendo que $1 \leq i \leq |s|$. Chamamos de $s_{i..j} = s_i s_{i+1} ... s_j$ ou *string* vazia de $i > j$. A *string* vazia é representada por ϵ.

A partir disso, nós definimos distância de edição da seguinte forma: a distância $d(x, y)$ entre duas *strings* x e y é o menor custo de uma sequência de operações que transforma x em y. O custo de uma sequência de operações é a soma dos custos de operações individuais. As operações são um conjunto finito de regras da forma $\sigma(z, w) = t$, onde z e w são *strings* diferentes e t é um número real não negativo representando o custo da operação.

Um problema relacionado freqüêntemente tratado é o casamento aproximado de padrões com k erros, que pode ser definido como: seja Σ um alfabeto finito de tamanho $|\Sigma| = \sigma$. Seja $T \in \Sigma^*$ um texto de tamanho $n = |T|$ e $P \in \Sigma^*$ um padrão de tamanho $m = |P|$. Seja $k \in \mathbb{N}$ o maior número de erros permitidos. Seja $d : \Sigma^* \times \Sigma^* \rightarrow \mathbb{N}$ a função de distância de edição. O problema é dado por: dado T, P, k e $d(.)$, retornar o conjunto de todas as posições j no texto tal que exista um i tal que $d(P, T_{i..j}) \leq k$.

É importante notar que, embora semelhantes, há uma diferença fundamental entre o problema da distância de edição e o casamento aproximado de padrões. O primeiro problema exige encontrar um *ótimo global* que miniza a distância entre as duas cadeias. O segundo problema percorre a cadeia sendo buscada marcando *todas as posições em que a distância de edição entre as cadeias está dentro de um limiar*. Por isso, freqüêntemente, algoritmos eficientes para um dos problemas podem ser ruins para o outro, ou até mesmo impossíveis de se adaptar sem grandes modificações.

Na maioria das aplicações, nós temos as seguintes operações entre as cadeias são consideradas:

- Inserção $\sigma(e, a)$: inserir um caracter.
- Remoção $\sigma(e, a)$: remover um caracter.
- Substituição $\sigma(a, b)$: substituir um caractere.

O restante desse trabalho se organiza como se segue. Em 2 os algoritmos baseados na técnica de programação dinâmica. A Seção 3 mostra os algoritmos que trabalham com autômatos. Algoritmos mais recentes, que utilizam filtragem são mostrados em 4. Os resultados experimentais do trabalho de pesquisa mais extenso da área [4] são discutidos na Seção 5. Uma rápida introdução à uma generalização do problema para grafos, que têm sido bastante estudada nos últimos anos é mostrada em 6. Finalmente, as conclusões obtidas são apresentadas em 7.

2 Algoritmos baseados em programação dinâmica

Os algoritmos mais antigos da área são baseados na técnica de programação dinâmica. Essa técnica consiste em resolver incrementalmente problemas menores até se atingir uma solução ótima global para as cadeias sendo separadas. A Figura 1 mostra a matriz de programação dinâmica, estrutura sobre a qual os algoritmos trabalham.

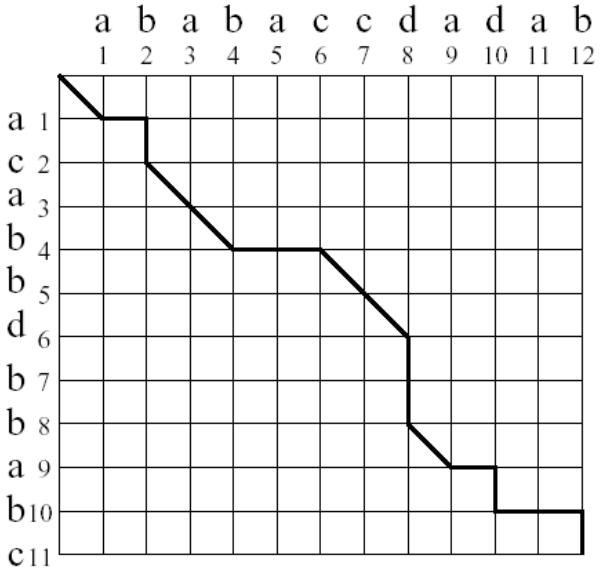

Figura 1: Matriz de programação dinâmica. O caminho demarcado representa o conjunto mínimo de operações que transforma a cadeia que rotula as linhas na cadeia que rotula as colunas.

A matriz da Figura 1 é percorrida pelos algoritmos do topo à esquerda, em direção à base à direita. Cada posição (i, j) da matriz contém a distância de edição entre a cadeia $P_{0,i}$ e a cadeia $M_{0,j}$. O caminho de menor custo entre o topo à esquerda e a base à direita da matriz representa o conjunto de operações que transformam P em M. Uma linha vertical representa a inserção de um caractere em P e uma linha horizontal representa a inserção a remoção de um caractere em M. As substituições ocorrem nas linhas diagonais que percorrem caracteres diferentes.

O algoritmo direto a partir da técnica de programação dinâmica é $O(|n||m|)$, e pode ser usado para o cálculo da distância de edição ou para descobrir o conjunto de operações que transforma uma cadeia em outra. Adaptações triviais ao algoritmo também permitem sua utilização para a busca aproximada de padrões.

Os algoritmos baseados em programação dinâmica não são competitivos para a busca aproximada em relação aos algoritmos mais recentes que combinam paralelismo de bits com filtragem. Porém, é importante levá-los em consideração por diversos aspectos. O principal deles é a *flexibilidade em relação ao custo das operações*. Esses algoritmos permitem facilmente modificar o custo de cada

operação, e ainda mais, definir custos diferentes para cada operação em função da posição de ocorrência da operação.

Outra característica dos algoritmos baseados em programação dinâmica é a facilidade de se recuperar o conjunto de operações que foi utilizado, em geral com uma penalidade de espaço. Algoritmos de outras classes dificilmente podem ser adaptados para obter essa saída adicional.

Além disso deseja também descobrir apenas se a distância de edição entre duas cadeias está abaixo de um dado limiar k, existem algoritmos com o excelente custo (k^2) baseados em programação dinâmica.

3 Algoritmos baseados em autômatos

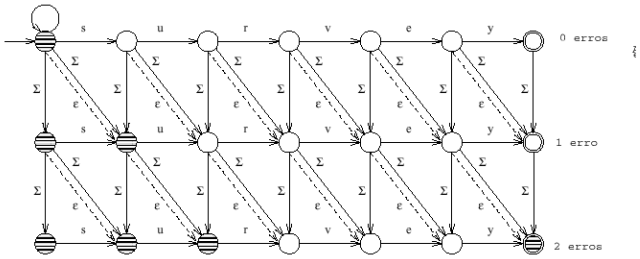

Figura 2: Autômato para casamento aproximado

Uma maneira muito simples é modelar o problema como um autômato não determinístico. Considerando o autômato para $k = 2$ erros da Figura 2. Cada linha representa o número de erros, cada coluna representa um casamento de prefixo. Linhas horizontais representam um casamento de caracteres, todas as outras incrementam o número de erros. Linhas verticais inserem um caracter no padrão, linhas diagonais contínuas substituem um caracter e linhas pontilhadas removem um caracter do padrão.

Este autômato pode ser transformado facilmente em um automâto determinístico para se obter o melhor pior caso $O(n)$ de tempo para o problema de casamento aproximado. O principal problema é que o autômato tem potencialmente um grande número de estados que necessitam ser armazenados na memória. A construção deste autômato depende do tamanho de m e k, tendo complexidade de espaço exponencial. Isto torna inviável a implementação da forma determinística do autômato.

As principais soluções para o problema de espaço são através da simulação do autômato, em vez de transformá-lo em determinístico. Mas com uma penalidade muito grande, com um pior caso de tempo $O(nkm)$. Uma solução bastante eficiente é a implementação através de paralelismo de bits que tem um custo no pior caso de $O(k\lceil m/w \rceil n)$ (sendo w o tamanho da palavra). Quando $m < w$ a complexidade cai em $O(kn)$. É interessante notar que estes custos são somente quando desejamos fazer casamento aproximado entre o texto e o padrão, onde este tipo de implementação é muito eficiente na maioria dos casos.

Uma característica desta classe de algoritmo é a capacidade dele retornar o conjunto de operações e a distância de edição. Mas com algumas penalidades:

a primeira é um incremento de espaço para se armazenar as operações já realizadas. Outra é que o algoritmo só é capaz de retornar as operações e a distância de edição se esta é menor do que o maior número de erros permitidos no autômato. Uma solução simples é construir um autômato capaz de computar *strings* com m erros, onde m é o tamanho do padrão, aumentando o custo de espaço e tempo de execução.

O principal problema, entrentanto, está relacionado ao cálculo a distância de edição e das operações que é muito alto nesta classe de algoritmo. Como o algoritmo faz o casamento de maneira gulosa, percorrendo todo o padrão e todo o texto no autômoto, estes resultados são sempre retornados no pior caso. O que torna os automâtos ineficientes para este tipo de aplicação.

4 Algoritmos que filtram o texto

Os algoritmos que filtram o texto são essencialmente projetados e utilizados para casamento aproximado e não para cálculo da distância de edição, portanto iremos nos restringir à discussão sobre casamento aproximado. A idéia principal por trás da filtragem é que é muito mais fácil dizer que *não* há um casamento em um trecho do texto que afirmar que há um casamento. Por exemplo, se não houver o texto *son* nem o texto *eca* no texto sendo casado, também não será possível encontrar o padrão *soneca* permitindo um erro.

Os algoritmos de filtragem se utilizam dessa propriedade descartando partes do texto onde não é possível ocorrer o padrão dentro do limite de erros possíveis. Para os trechos em que há um potencial de ocorrência do padrão, os algoritmos utilizam algoritmos de outras classes para verificar se o padrão realmente ocorre no trecho.

Embora qualquer algoritmo para casamento aproximado possa ser utilizado em conjunto com os algoritmos de filtragem, normalmente o algoritmo mais simples de programação dinâmica é utilizado pela facilidade de implementação.

O desempenho dos algoritmos de filtragem é bastante sensível ao nível de erro α, uma medida sensível à quantidade de texto que o algoritmo consegue filtrar. Quanto mais texto é filtrado, melhor o desempenho do algoritmo. Dada essa propriedade, existem algoritmos bastante diferentes para diferentes tamanhos de padrão. Os algoritmos voltados para padrões grandes fazem grande esforço para descartar trechos de texto (potencialmente grandes por causa do tamanho do padrão), enquanto para algoritmos voltados para padrões pequenos, muitas vezes é melhor tentar localizar o casamento aproximado no trecho que se esforçar para tentar descartá-lo.

O primeiro algoritmo dessa classe utiliza as técnicas do algoritmo de casamento exato BMH (Boyer-Moore-Horspool), casando os o padrão com o texto da direita para esquerda, de acordo com uma janela que desliza no texto. Assim que são encontrados mais de k *caracteres ruins*, o trecho é descartado. Um *caractere ruim* é aquele que não casa o caractere com o qual está alinhado e também não casa nenhum caractere do padrão à distância menor ou igual à k.

Há diversos algoritmos para filtragem, com diferentes compromissos. Na prática, esses são os melhores algoritmos existentes para busca aproximada. A Figura 3 mostra alguns desses algoritmos. Descrições mais aprofundadas dos algoritmos, juntamente apontadores para literatura relevante podem ser encontrados em [4].

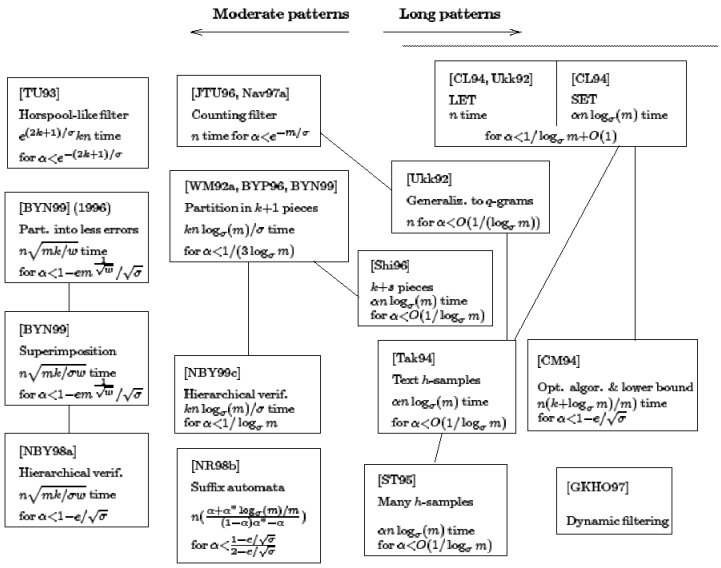

Figura 3: Alguns dos algoritmos de filtragem para casamento aproximado de padrões.

5 Experimentos da literatura

Nesta seção mostramos os experimentos descritos em [4]. Foram feitos testes somente para os algoritmos mais eficientes, excluindo aqueles que tinham alguma contribuição teórica, mas não se mostravam competitivos em relação a outros.

Abaixo apresentamos uma legenda para os gráficos. Uma descrição mais detalhada de cada algoritmo e apontadores para literatura relevante podem ser encontrados em [4].

- **CTF**: heurística de corte Ukkonen.
- **CLP**: algoritmo de particionamento de coluna de Chang e Lampe.
- **DFA**: automato deterministico preguiçoso de Navarro.
- **RUS**: algoritmo dos quatro-russos de Wu et al.
- **BPR**: shift-and de Wu e Manber.
- **BPD**: paralelismo de bits na diagonal de Baeza-Yates e Navarro.
- **BPM**: matriz dinâmica através de paralelismo de bits de Myers.
- **BMH**: BMH modificado para admitir erros.
- **CNT**: filtro de contagem de Jokinen et al.
- **EXP**: particionamento em k+1 pedaços mais verificação hierárquica de Baeza-Yates e Navarro.

- **BPP**: paralelismo de bits usando partição de padrões de Baeza-Yates e Navarro.
- **BND**: algoritmo BNDM adaptado para aceitar erros de Navarro e Raffinot.
- **QG2**: filtro q-gram de Sutinen e Tarhio.
- **QG1**: algoritmo q-gram de Takaoka.

Os experimentos foram feitos em três tipo de textos: um arquivo de 10MB de sequência de DNA, com um alfabeto de tamanho 4. Um arquivo de linguagem natural também de tamanho de 10MB. E um arquivo de de 10MB contendo um discurso a respeito da lei americana na Universidade de Indiana.

A figura 4 mostra os resultados para padrões pequenos ($m = 10$). Considerando algoritmos sem filtro BPD é normalmente 30% mais rápido que o próximo, BPM. Em relação a filtros, EXP é o mais rápido se considerarmos poucos erros. Esse valor de "poucos erros" aumenta para alfabetos maiores.

A figura 5 mostra os resultados para padrões maiores ($m = 30$). As observações são semelhantes quando consideramos padrões pequenos, mas é interessante notar que o algoritmo BPM demonstra uma melhora em relação ao algoritmo BPD, isso se deve ao fato que para o algoritmo BPM comporta todo o problema numa palavra de computador, o que não acontece para o algoritmo BPD. Em relação aos filtros EXP ou BND são os mais rápidos (dependendo do alfabeto) até que um certo nivel de erros é atingido, depois disso BPP vira o mais rápido.

A figura 6 considera o caso de se variar o valor de m. Considerando algoritmos sem filtro os resultados se repetem. BPR é melhor para $k = 1(m = 10)$ e depois BPD é melhor até que um tamanho de padrão é atingido. A diferença neste caso está nos algoritmos de filtro, PEX se tornou o melhor algoritmo quando consideramos textos em Inglês e o texto da fala. Quando consideramos o texto com o DNA para $m \leq 30$, BND é o mais rápido. Depois para $m > 30$ e $m \leq 60$ PEX se tornou o algoritmo mais rápido e com $m > 60$ QG2 foi o campeão.

Interessante notar que os algoritmos QG1 e QG2 melhoram quando m cresce. Logo para m muito grande ($m > 100$) e alfabetos pequenos (de tamanho 4, como sequências de DNA) estes algoritmos devem ser considerados.

6 Distância de Edição e Grafos

Recentemente, a generalização do problema de distância de edição para grafos, onde as operações são remoção, inserção e modificação de rótulos de vértices dos grafos comparados, têm recebido grande atenção [3, 5, 7, 1], e já há resultados que mostram que a distância de edição pode ser utilizada para resolver outros problemas clássicos da área, como máximo subgrafos isomórficos [1, 6, 2].

Em especial, têm havido bastante trabalhos sobre a distância de edição entre árvores. As aplicações exploradas são várias, como análise de estruturas de RNA e extração de informação páginas Web.

7 Conclusões

O problema da distância de edição é ainda um problema com muito a ser explorado. O algoritmo a ser utilizado deve sempre ser escolhido, dentre uma grande variedade de algoritmos, de acordo com as necessidades da aplicação.

Há uma enorme gama de aplicações que necessitam do cálculo da distância de edição, ou casamento aproximado de padrões. A generalização do problema para grafos, embora ainda tenha poucos algoritmos, também é de grande interesse para áreas como bioinformática ou extração de informação.

Referências

[1] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689–694, 1997.

[2] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19(3):225–259, 1998.

[3] Sudarshan S. Chawathe. Comparing hierarchical data in external memory. In *Proceedings of the Twenty-fifth International Conference on Very Large Data Bases*, pages 90–101, Edinburgh, Scotland, U.K., 1999.

[4] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.

[5] Andrew Nierman and H. V. Jagadish. Evaluating structural similarity in XML documents. In *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*, Madison, Wisconsin, USA, June 2002.

[6] Gabriel Valiente. Tree edit distance and common subtrees. Research Report LSI-02-20-R, Universitat Politècnica de Catalunya, 2002.

[7] Kaizhong Zhang, Rick Statman, and Dennis Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.

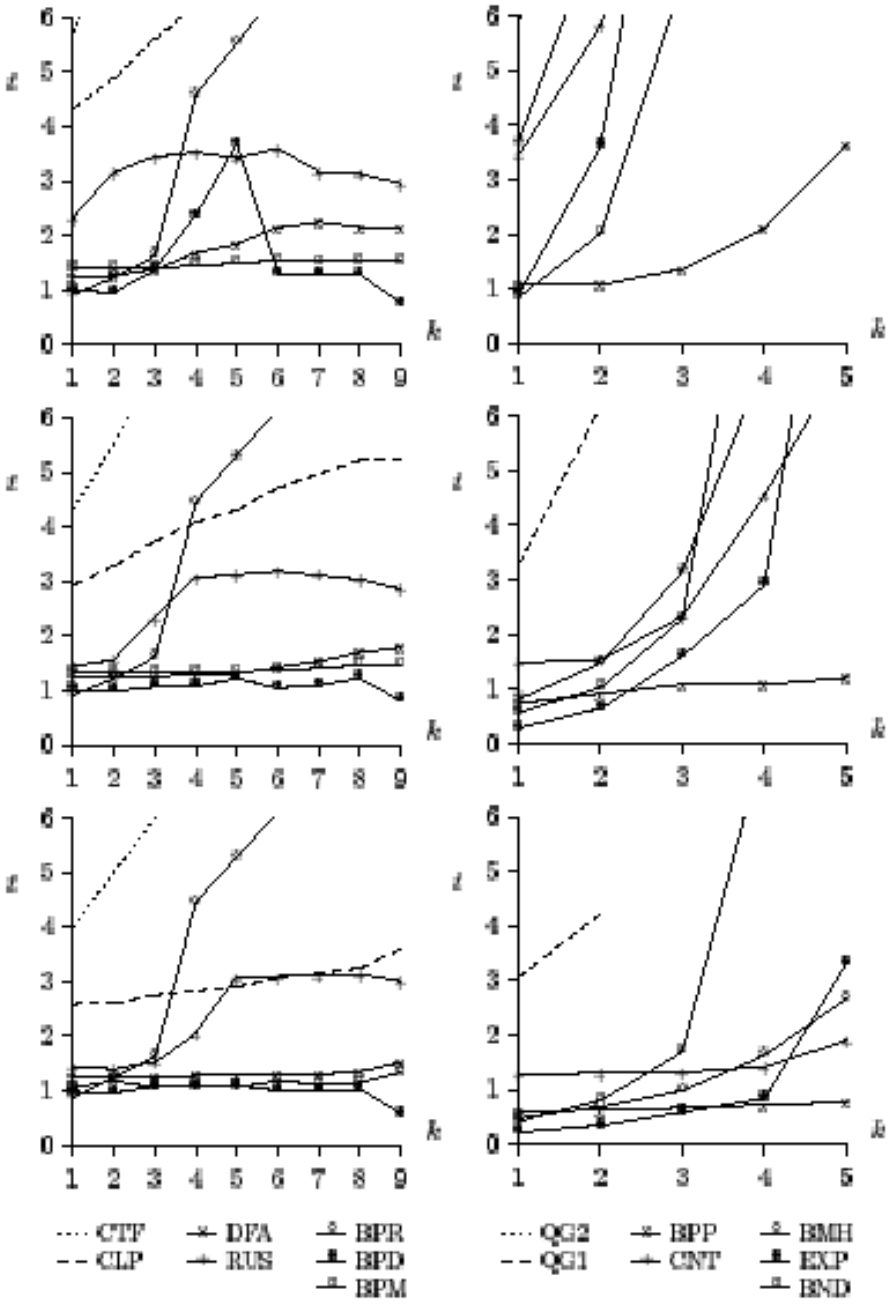

Figura 4: Resultados para $m = 10$ e variando k. Os gráficos a esquerda mostram algoritmos sem filtro e a direita mostram algoritmos com filtros. Linha 1-3 mostra DNA, texto em Inglês e texto de um fala, respectivamente.

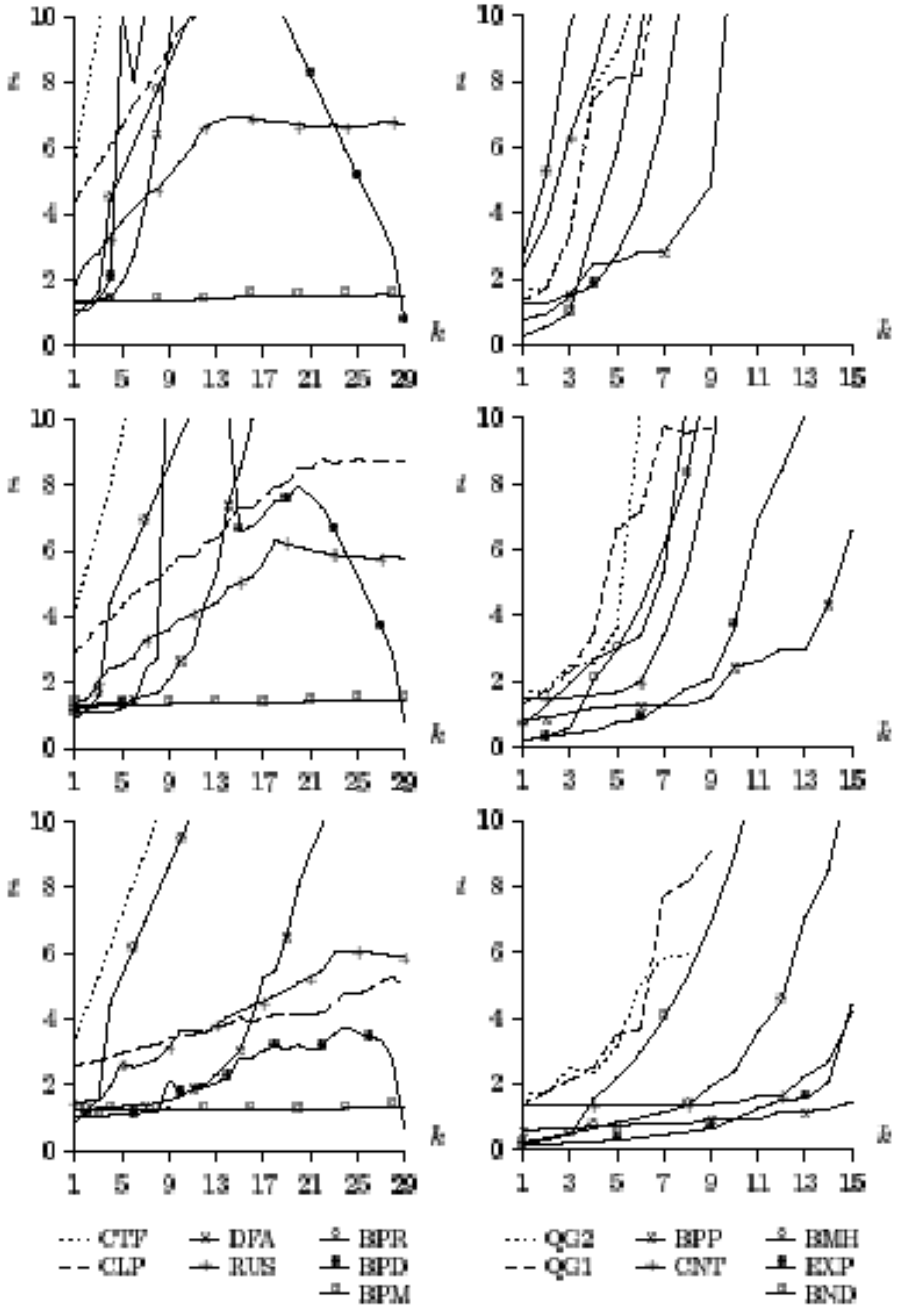

Figura 5: Resultados para $m = 30$ e variando k. Os gráficos a esquerda mostram algoritmos sem filtro e a direita mostram algoritmos com filtros. Linha 1-3 mostra DNA, texto em Inglês e texto de um fala, respectivamente.

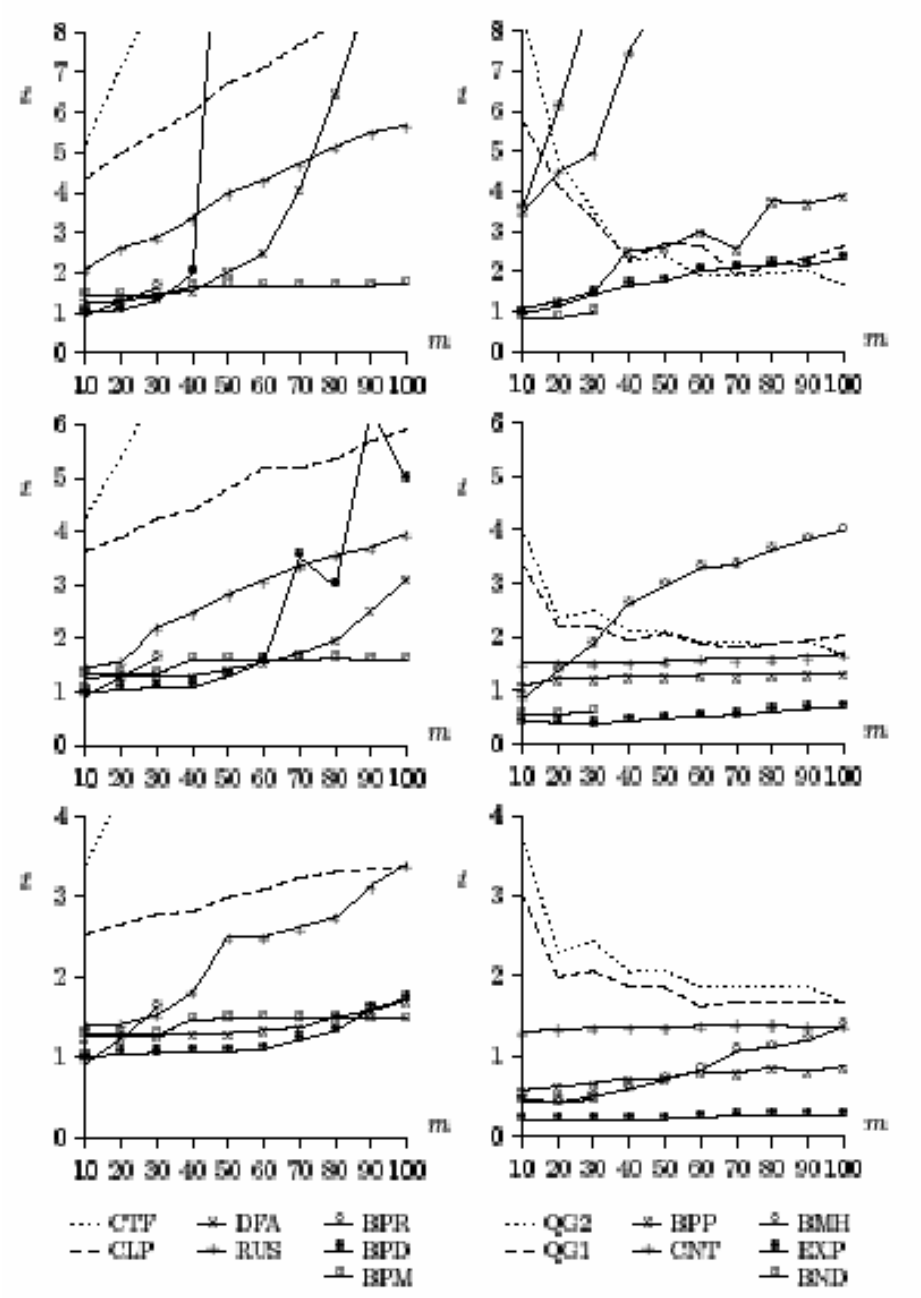

Figura 6: Resultados variando m. Os gráficos a esquerda mostram algoritmos sem filtro e a direita mostram algoritmos com filtros. Linha 1-3 mostra DNA, texto em Inglês e texto de um fala, respectivamente.