Ferramenta PMT

Giovanni Antonio Araujo de Barros e Silva - gaabs@cin.ufpe.br
Pedro Henrique Sousa de Moraes - phsm@cin.ufpe.br

Sumário

1. Identificação

1.1 Equipe

Giovanni Antonio Araujo de Barros e Silva - gaabs@cin.ufpe.br
Pedro Henrique Sousa de Moraes - phsm@cin.ufpe.br

1.2 Contribuições

Giovanni

- Implementação dos algoritmos de busca exata Boyer-Moore, Aho-Corasick e Rabin Karp
- Implementação dos algoritmos de busca aproximada Sellers e Ukkonen
- Otimização, correção e refatoração do código
- Desenvolvimento e execução de testes
- Escrita de documentação e relatório

Pedro

- Implementação da interface em linha de comando
- Implementação dos algoritmos de busca exata Brute Force, Knuth-Morris-Pratt e Shift-Or
- Otimização, correção e refatoração do código
- Desenvolvimento e execução de testes
- Escrita de documentação e relatório

2. Implementação

A ferramenta PMT foi implementada usando a linguagem C++ (cross-platform) e possui o conjunto de opções definidas na especificação do projeto.

As opções da ferramenta são:

- **-a, --algorithm:** Usada para especificar o algoritmo de busca a ser usado. Os valores possíveis são "bf" (Força-Bruta), "kmp" (Knuth-Morris-Pratt), "bm" (Boyer-Moore), "ac" (Aho-Corasick) e "so" (Shift-Or) para buscas exatas, e "se" (Sellers) e "uk" (Ukkonen) para buscas aproximadas.

- **-c, --count:** Exibe apenas a quantidade de padrões encontrados em cada arquivo. Como não efetua operações de escrita por linha, a ferramenta pode ficar mais eficiente em casos de muitos matchings no texto.
- **-p, --pattern:** Especifica um arquivo de padrões, cada linha do arquivo é tratado como um padrão. Se essa opção é usada, o primeiro argumento da ferramenta é tratado como um arquivo de texto e não como um padrão, dessa forma um segundo argumento não é obrigatório.
- **-e, --edit:** Especifica a distância de edição para algoritmos de busca aproximados, sendo obrigatória quando um algoritmo aproximado é selecionado.
- **-h, --help:** Mostra informações sobre o uso e as opções da ferramenta.

2.1 Seleção Automática dos Algoritmos de Busca Exata

Caso nenhum algoritmo seja especificado, a ferramenta seleciona a melhor opção baseada nos parâmetros que melhoram o desempenho dos algoritmos segundo os testes realizados.

Nos testes realizados, o algoritmo Boyer-Moore se mostrou o mais rápido para padrões (>10), isso tambem deve-se ao tamanho do alfabeto usado nos testes que foi de 256 caracteres, aumentando o efeito das heurísticas do Boyer-Moore. De qualquer forma, alfabetos de tamanho menor degradam o desempenho de vários algoritmos, pois aumentam a probabilidade de matching durante as buscas, mas afetam ainda mais o Boyer-Moore uma vez que reduz o efeito de sua heurística do mau caractere.

O Shift-Or apresentou desempenho estável independentemente do tamanho do padrão, mas muito lento para ser usado. O Força-Bruta se mostrou um pouco mais rápido que o Boyer-Moore para padrões pequenos (<=10). O Aho-Corasick também apresentou desempenho estável assim como o Shift-Or mas foi em média quatro vezes mais lento que o Boyer-Moore para a busca de um único padrão de qualquer tamanho. O KMP se mostrou muito lento em relação ao Boyer-Moore e Força-Bruta para padrões pequenos, se aproximando apenas quando o tamanho dos padrões ultrapassa setenta. O algoritmo Força-Bruta foi em média duas vezes mais lento que o Boyer-Moore, mas com melhor

Devido às características e resultados obtidos dos algoritmos, a forma de combinação final foi a seguinte:

- Se mais de quatro padrões são usados, Aho-Corasick é selecionado;
- Se o tamanho médio dos padrões é menor ou igual a 10, Força-Bruta é selecionado;
- Caso contrário, é selecionado o Boyer-Moore.

2.2 Seleção Automática dos Algoritmos de Busca Aproximada

Para que a busca aproximada seja usada, a opção --edit deve ser usada. Quando o algoritmo não é especificado pelo usuário, ele é escolhido automaticamente.

Nos testes, vimos que o tempo de execução do algoritmo Sellers cresce substancialmente de acordo com tamanho do padrão e o tempo de inicialização do algoritmo Ukkonen cresce bastante de acordo com a distância de edição e o tamanho do padrão.

O algoritmo Sellers se mostrou mais rápido que o Ukkonen para padrões pequenos e para distâncias de edição grandes, pois a medida que o tamanho destes aumenta, a geração da máquina de estados do Ukkonen se torna mais custosa, passando a ser menos eficiente.

Devido às características e resultados obtidos dos algoritmos, a forma de combinação final foi a seguinte:

- Se tamanho do padrão menor ou igual a 5 ou a distância de edição permitida for maior que 3, Sellers é selecionado;
- Caso contrário, é selecionado o Ukkonen.

2.3 Estruturas de dados utilizadas e observações

- Para o algoritmo Shift-Or, foi utilizada uma implementação própria de uma estrutura de dados para simular um array de bits, chamada de BitArray.
- Dado que certos algoritmos utilizam o alfabeto na busca e pré-processamento, foi fixado um tamanho de alfabeto de 256 caracteres.

- No geral, buscou-se fazer uso de estruturas de acesso direto, como vetores e evitou-se o uso de hashmaps. No entanto, devido a complexidade de representar transições por vetores no algoritmo Ukkonen, fez-se uso de hashmap nesse caso.
- Quanto a leitura das entradas, para tanto fez-se uso da API de streams do C++, uma vez que ela gerencia automaticamente o tamanho dos buffers. Isso facilita a implementação dos algoritmos, mas ocasiona numa pequena perda de performance.

3. Testes e Resultados Obtidos

3.1 Ambiente de testes e ferramentas utilizadas

Os testes e comparações da ferramenta e dos algoritmos foram feitos através de scripts em Python. Os scripts executam múltiplas vezes comandos shell e, através da função timeit provida pela linguagem Python, calculam o tempo médio da execução, que é a métrica usada para avaliar o desempenho dos algoritmos. No total, foram executados dez testes por padrão para cada algoritmo. Para a geração dos gráficos, o pacote matplotlib da linguagem Python foi usado.

Os testes foram conduzidos em um computador com sistema operacional Ubuntu 17.04 com processador Intel core i7 2.1GHz e 8GB de memória RAM.

3.2 Descrição dos dados

Para entrada dos testes, foram utilizados arquivos disponíveis no Pizza&Chili Corpus (http://pizzachili.dcc.uchile.cl/). Foram escolhidos três arquivos de entrada com diferentes tamanhos e categorias de texto de forma a proporcionar uma melhor avaliação do funcionamento da ferramenta em casos mais gerais.

O primeiro arquivo escolhido possui 1.1GB, tendo sido escolhidos, a partir do arquivo, padrões de tamanho variando entre 1 e 98. O texto e os padrões são compostos de palavras e frases em inglês.

O segundo arquivo escolhido possui 200MB, tendo sido escolhidos, a partir do arquivo, padrões de tamanho variando entre 2 e

96. O texto e os padrões são compostos de partes de programas escritos em C, C++, Java, etc.

O terceiro arquivo escolhido possui 100MB, tendo sido escolhidos, a partir do arquivo, padrões de tamanho variando entre 1 e 100. O texto e os padrões são compostos de sequências de proteínas, com um alfabeto bem menor que os dos arquivos anteriores.

3.3 Algoritmos de Busca Exata

Os algoritmos foram testados com diferentes arquivos de entrada e padrões de vários tamanhos. A opção --count foi usada para contar a quantidade de ocorrências reportadas pelos algoritmos. A ferramenta grep também foi incluída nos testes para comparação do desempenho com os algoritmos implementados.

3.3.1 Busca por apenas um padrão

Os testes a seguir, apesar de utilizarem vários padrões, englobam resultados da busca de apenas um padrão por vez

Para que a saída do comando grep fosse equivalente a dos algoritmos implementados, ela foi executada da seguinte forma: ‘grep pattern textfile -o | wc -l’.

3.3.1.1 Teste 1: Textos em inglês

A Figura 1 mostra o desempenho de cada algoritmo para os padrões testados.

Os resultados mostram o baixo desempenho do Boyer-Moore para padrões pequenos, além disso seu desempenho é quase idêntico ao do Força-Bruta. O KMP não obteve bons resultados para padrões pequenos, apenas alcançando o Força-Bruta com padrões de tamanho maior que

setenta.

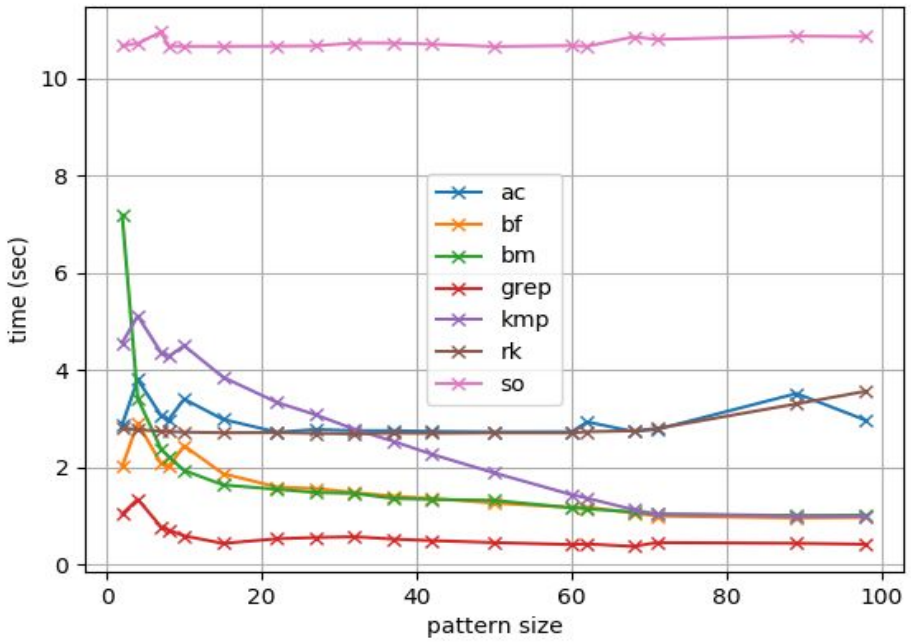

Figura 1: Teste individual dos algoritmos para padrões de tamanho variando entre 1 e 98 em texto de 1.1GB obtido em "http://pizzachili.dcc.uchile.cl/texts/nlang/"

3.3.1.2 Teste 2: Código Fonte

A Figura 2 mostra o desempenho de cada algoritmo para os padrões testados.

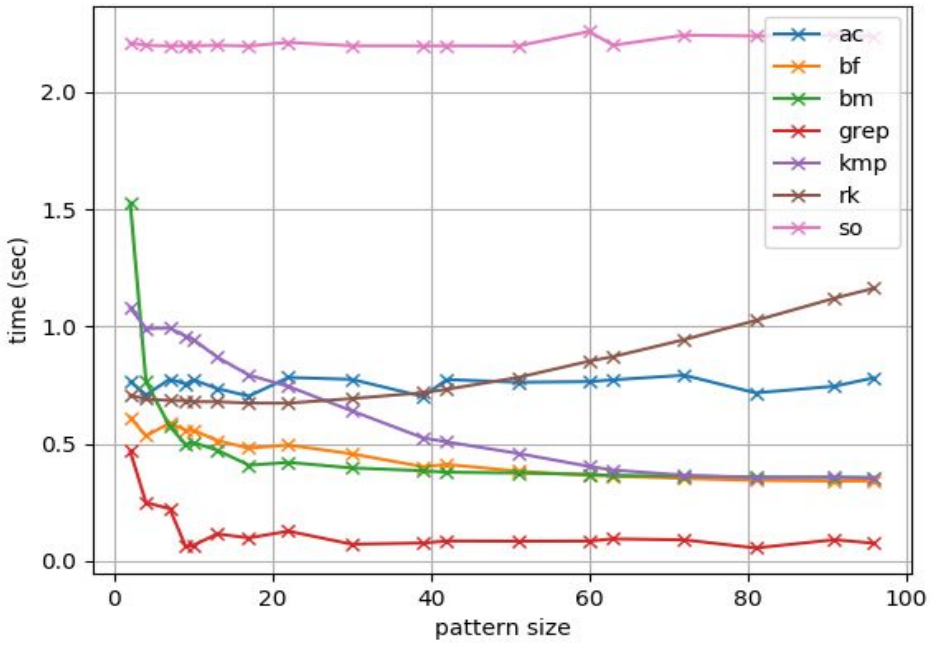

Figura 2: Teste individual dos algoritmos para padrões de tamanho variando entre 2 e 96 em texto de 200MB obtido em "http://pizzachili.dcc.uchile.cl/texts/code/"

Os resultados são muito parecidos com os do teste 1. Isso já era esperado, dada a natureza do arquivo de texto, pois este também possui muitas palavras em inglês e um alfabeto grande.

3.3.1.3 Teste 3: Proteinas

A Figura 3 mostra o desempenho de cada algoritmo para os padrões testados.

Os resultados obtidos neste teste diferem bastante dos anteriores, principalmente para os algoritmos KMP, Força-Bruta e Boyer-Moore, que possuem desempenhos bem distintos, sendo o KMP mais lento que o Aho-Corasick.

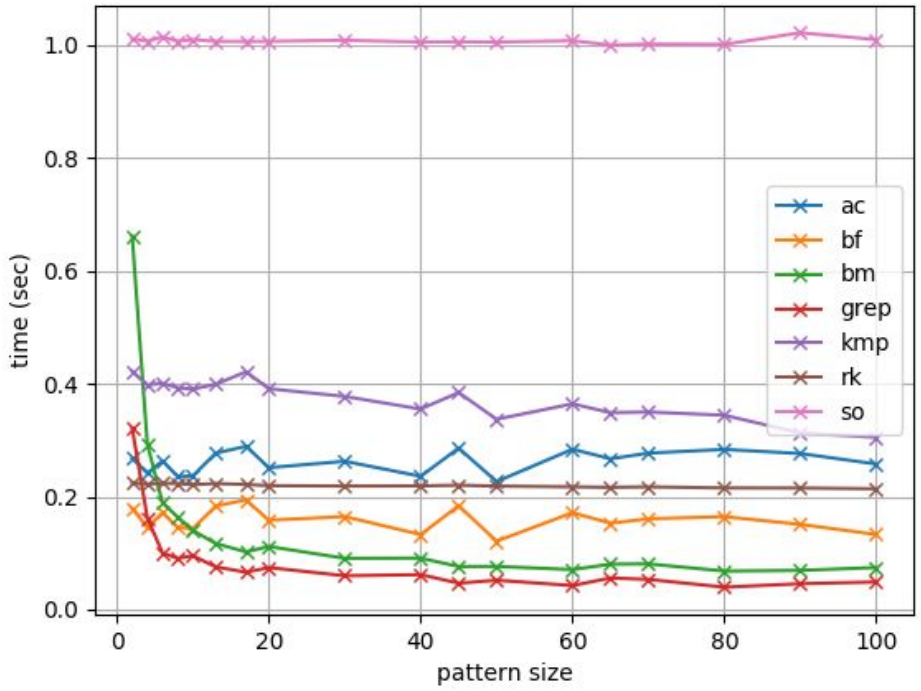

Figura 3: Teste individual dos algoritmos para padrões de tamanho variando entre 1 e 100 em texto de 100MB obtido em "http://pizzachili.dcc.uchile.cl/texts/protein/"

3.3.1.4 Conclusões

Com base nos testes feitos usando os algoritmos exatos, é possível concluir que:

- Boyer-Moore se mostra muito versátil para a maioria dos casos, deve-se tomar cuidado apenas com padrões pequenos, menores que dez, pois independentemente do algoritmo usado, os resultados podem variar bastante quando um teste único é feito.
- O Aho-Corasick é em média de duas a quatro vezes mais lento que o Boyer-Moore, tornando seu uso válido quando mais de quatro padrões devem ser buscados.
- O uso do Força-Bruta é interessante em alguns casos, principalmente quando os padrões são pequenos, pois isso aproxima outros algoritmos como o KMP e o Boyer-Moore de seus piores casos.
- O Shift-Or tem uma performance muito ruim. Isso já era esperado, pois como ele faz muitas operações por posição do texto, mesmo que seja uma quantidade constante, possui resultado pior que o

Força-Bruta, que na maioria dos casos encontra mismatches e rapidamente avança para as próximas letras.

3.3.2 Busca por vários padrões

Os testes a seguir englobam resultados da busca de vários padrões por vez. Quando o algoritmo não suporta a busca simultânea, é realizada uma busca para cada padrão.

Para que a saída do comando grep fosse equivalente a dos algoritmos implementados, ela foi executada da seguinte forma: ‘grep -f patternfile textfile -o | wc -l’.

3.3.2.1 Teste 4: Proteínas com lista de padrões

A Figura 4 mostra o desempenho de cada algoritmo para os padrões testados.

Os resultados mostram que, como esperado, o algoritmo Aho-Corasick teve uma melhora de desempenho considerável na busca por vários padrões. O grep, por sua vez, teve uma piora considerável, uma vez que executa uma busca por padrão.

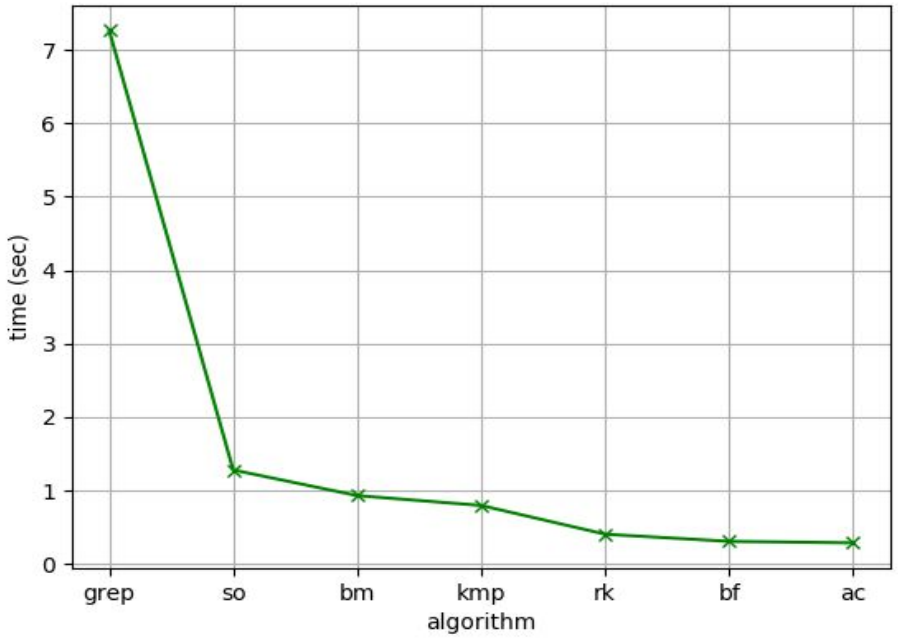

Figura 4: Teste individual dos algoritmos para 18 padrões de tamanho variando entre 1 e 100 em texto de 100MB obtido em “http://pizzachili.dcc.uchile.cl/texts/protein/”

3.3.2.2 Conclusões

- Observou-se que quando utiliza-se um conjunto de padrões, o Aho-Corasick possui vantagem de desempenho considerável.
- Os outros algoritmos da ferramenta também apresentaram bom desempenho se comparado ao grep, já que foram feitos para fazer o reconhecimento das linhas do texto com vários padrões ao mesmo tempo.

3.4 Algoritmos de Busca Aproximada

Os algoritmos de busca aproximada foram testados com diferentes arquivos de entrada e padrões de vários tamanhos. A opção --count foi usada para contar a quantidade de ocorrências reportadas pelos algoritmos. Além disso, foram testadas várias configurações de distância máxima de edição.

Os testes a seguir, apesar de utilizarem vários padrões, englobam resultados da busca de apenas um padrão por vez.

3.4.1 Teste 5: Proteinas Com Distancia de Edição 2, 5 e 10

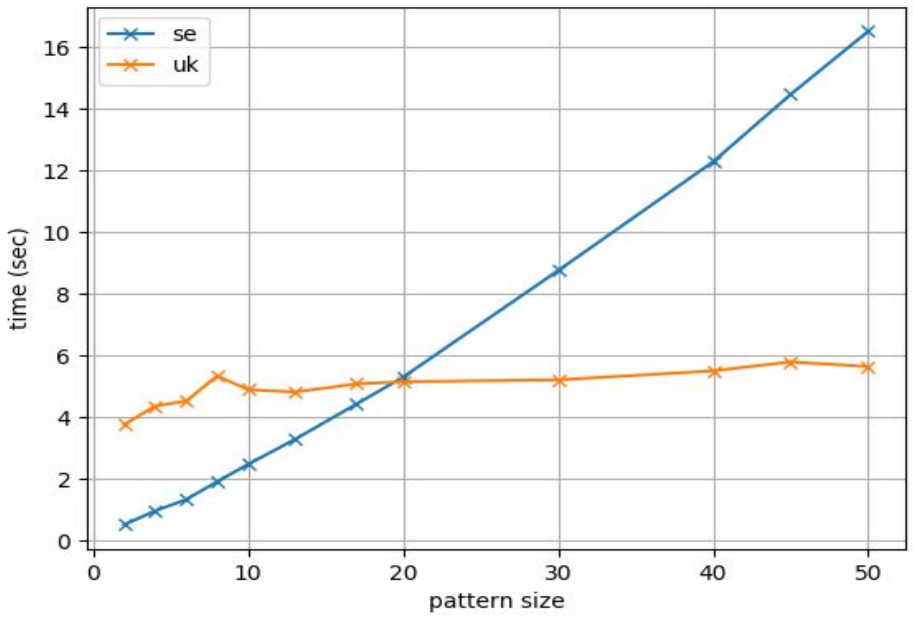

Figura 5: Teste individual dos algoritmos de busca aproximada com distância máxima de edição 2 para padrões de tamanho variando entre 2 e 50 em texto de 100MB obtido em "http://pizzachili.dcc.uchile.cl/texts/protein/"

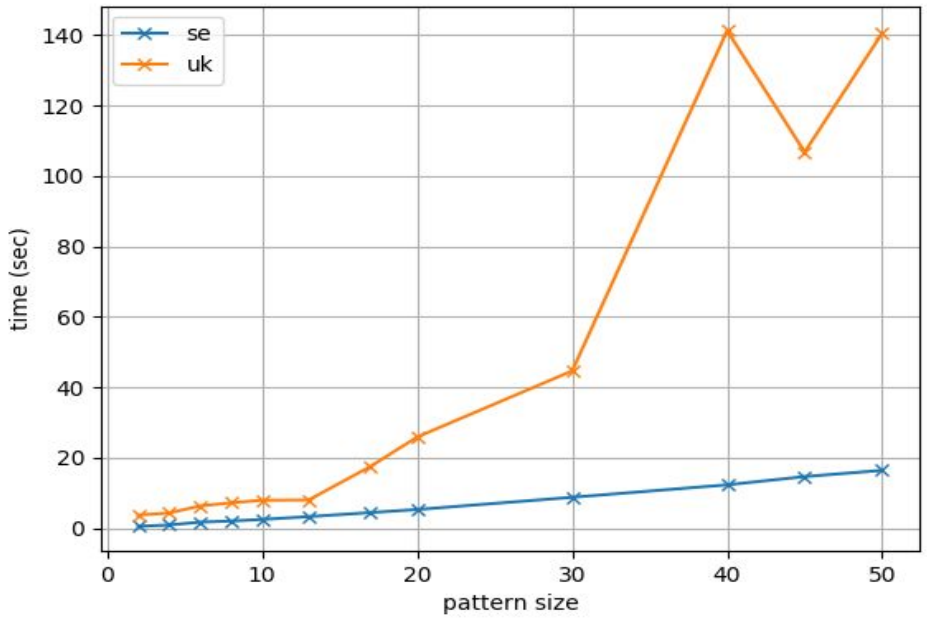

Figura 6: Teste individual dos algoritmos de busca aproximada com distância máxima de edição 5 para padrões de tamanho variando entre 2 e 50 em texto de 100MB obtido em "http://pizzachili.dcc.uchile.cl/texts/protein/"

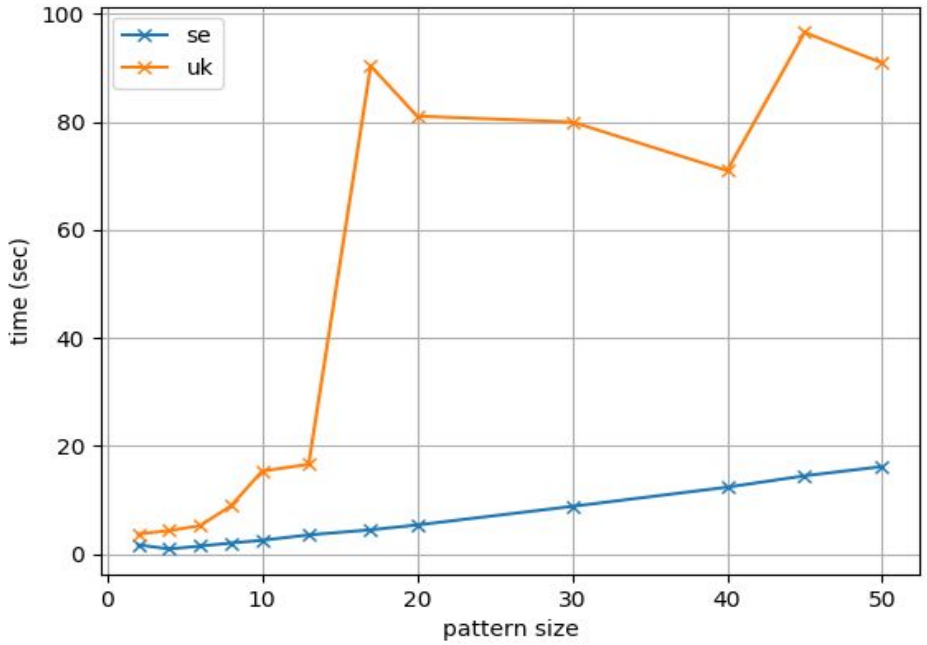

Figura 7: Teste individual dos algoritmos de busca aproximada com distância máxima de edição 10 para padrões de tamanho variando entre 2 e 50 em texto de 100MB obtido em "http://pizzachili.dcc.uchile.cl/texts/protein/"

3.4.2 Conclusões

Com base nos testes feitos usando os algoritmos de busca aproximada, é possível concluir que:

- O tempo de execução do algoritmo Sellers cresce bastante com o aumento do tamanho do padrão. Apesar disso, é relativamente eficiente no caso de padrões pequenos.
- O algoritmo Ukkonen tem o tempo de inicialização maior, mas seu tempo de execução varia pouco, justificando seu uso no caso de padrões maiores.
- O aumento da distância máxima de edição pouco altera o tempo de execução do Sellers. No Ukkonen, por outro lado, há um aumento visível do tempo de execução, uma vez que também há um grande crescimento do número de estados.