Ferramenta PMT

1. Identificação

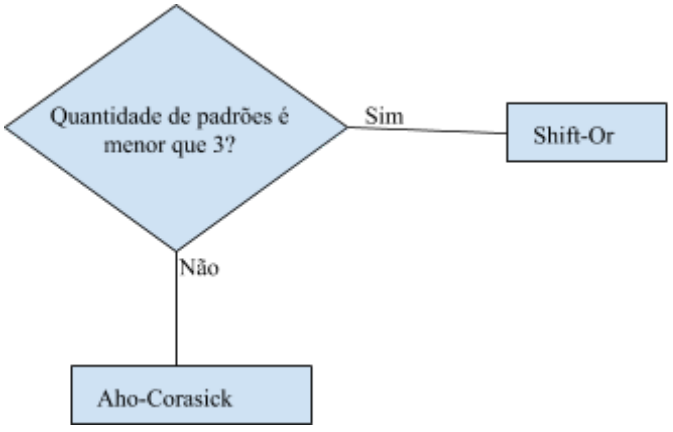
A equipe é formada por Jônatas de Oliveira Clementino (joc) e por Valdemiro Rosa Vieira Santos (vrvs). Jônatas ficou responsável pela implementação dos algoritmos Shift-Or e Wu-Manber e Valdemiro ficou responsável pela implementação dos algoritmos Aho-Corasick e Ukkonen. As outras partes ambos os alunos contribuíram.

2. Implementação

Para o desenvolvimento da ferramenta pmt foi implementado quatro algoritmos de casamento exato e aproximado de padrões, dois para cada categoria. A linguagem de programação escolhida foi C/C++ com o objetivo de alcançar maior eficiência em questão de tempo de execução.

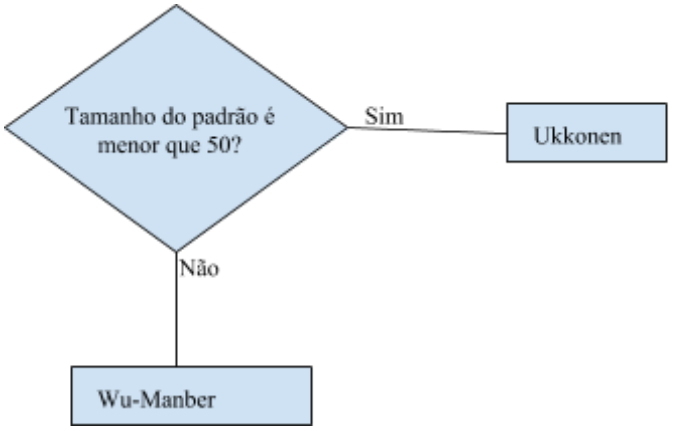
2.1. Algoritmos de Busca Exata

Para o casamento exato de padrões no texto foi escolhido dois algoritmos, sendo eles o Shift-Or e o Aho-Corasick. O Shift-Or foi implementado de tal forma que aceita padrões de tamanhos variados não sendo restrito a um tamanho máximo. Sendo assim, pôde-se comparar os dois algoritmos com diferentes tamanhos de padrão e com diferentes quantidades de padrão. Para padrões de tamanhos variados o Shift-Or se mostrou melhor que o Aho-Corasick, porém ao analisar com mais de 2 padrões, pôde-se notar uma queda de desempenho do Shift-Or em comparação ao Aho-Corasick. Sendo assim, foi utilizado a seguinte heurística para decidir quando utilizar um dos algoritmos (partindo do pressuposto de que a opção de busca aproximada já está desativada):



2.2. Algoritmos de Busca Aproximada

Para o casamento aproximado de padrões no texto foi escolhido dois algoritmos, sendo eles o Wu-Manber e o Ukkonen. O Wu-Manber foi implementado de tal forma que aceita padrões de tamanhos variados não sendo restrito a um tamanho máximo. Sendo assim, pôde-se comparar os dois algoritmos com diferentes tamanhos de padrão e com diferentes quantidades de padrão. Podemos notar que para padrões com um tamanho menor o Ukkonen se mostrou uma opção mais eficiente. No entanto, o Ukkonen tem um crescimento mais acentuado do tempo de execução do que o Wu-Manber a medida que o tamanho do padrão aumenta. Então, a partir de um certo ponto, o Ukkonen passa a ficar mais lento que o Wu-Manber. Sendo assim, foi utilizado a seguinte heurística para decidir quando utilizar um dos algoritmos:



2.3. Detalhes de Implementação Relevantes

2.3.1. Shift-Or e Wu-Manber

Para algoritmos como o Shift-Or e o Wu-Manber todas as estruturas estão representadas como um array, isto só é possível pois o tamanho do alfabeto é fixo e pequeno, isso nos dá acesso constante para qualquer letra dentro do alfabeto. Foi utilizado o ASCII como referência para alfabeto. Também, para implementação desses algoritmos foi utilizado um *long* (64 bits) para utilizar as operações lógicas. Como já mencionado, estes dois algoritmos não possuem restrição quanto ao tamanho do padrão, pois além de fazer uso do long 64 bits, eles na verdade possuem um array de valores long para representar o padrão, e as operações lógicas necessárias estão implementadas para o array, permitindo assim uma flexibilidade quanto ao tamanho do padrão. Pequenas otimizações também foram feitas, como por exemplo um método que faz a operação de *shift left* e *or* ao mesmo tempo, ou *loops* que iteram sobre o mesmo *range* foram colocados juntos.

2.3.2. Aho-Corasick e Ukkonen

Com intuito de dar mais eficiência para os algoritmos, foi pensado ao máximo como implementar de maneira eficiente as funções de transição tanto de Aho-Corasick (go_to e failure) e de Ukkonen. Para isso, foi escolhido usar uma estrutura de C++ chamada unordered_map, que é uma hash table. Sabe-se que o custo médio das operações de inserção e busca é constante, porém o pior caso é linear. Para tentar melhorar isso, foi produzida uma tabela com função de busca que tem custo de inserção e busca no pior caso constante, pois não tem colisões. Essa tabela criada comporta até uma certa quantidade de estados nela todas as operações são feitas em tempo constante. Quando essa tabela se enche, aí sim é utilizada o unordered_map de C++. Depois da implementação dessa estrutura bem simples, a execução do código ficou relativamente mais rápida.

2.3.3. Interface do Usuário

Como pmt é uma ferramenta de linha de comando, a interface dela é simples e executada no terminal. Para obter as opções da linha de comando foi utilizada uma biblioteca chamada *optget.h*.

Alguns possíveis erros de entrada do usuário estão sendo tratados. Segue abaixo lista com todos os erros tratados:

- “pmt: The edit number must be an integer.” - É lançado se, na opção *-e (--edit)*, o número de edição passado pelo usuário não corresponde a um número;
- “pmt: The edit number can't be less than zero.” - É lançado se, na opção *-e (--edit)*, o número de edição é menor que zero;
- “pmt: Pattern file doesn’t exist.” - É lançado se, na opção *-p (--pattern)*, o caminho para o arquivo passado como parâmetro não existe;
- “pmt: Algorithm choice doesn’t exist.” - É lançado quando o argumento passado na opção *-a (--algorithm)* não existe;
- “pmt: The pattern pattern_name must be bigger than the edit number...” - É impresso quando a busca é aproximada e o número de edição é maior ou igual ao tamanho do padrão, pois se fosse permitido, bastaria retornar todas as linhas;
- “pmt: file_name: Text file doesn't exist” - Quando o caminho para o arquivo de texto não existe;

3. Testes e Resultados

Todos os testes foram realizados através de arquivos escritos em bash executando chamadas à pmt com os diferentes algoritmos implementados já citados anteriormente, e utilizando a ferramenta time do GNU para cálculo de desempenho. Também foram utilizados arquivos de textos com os tamanhos variando entre 50MB e 200MB, e arquivos com padrões com diferente quantidade de padrões, que variam entre 1,...,9 e 10,20,...,100.

Para efeito de comparação dos testes, foram utilizadas as ferramentas *grep* para comparação com os algoritmos de casamento exato, e *agrep* para comparação com os algoritmos com casamento aproximado. Para análise de tempo foi utilizado o parâmetro *-c* que faz com que só imprima na tela a quantidade de ocorrências, fazendo com que o tempo de impressão das linhas não afetem o tempo. Como a opção *-c* nesta ferramenta imprime a quantidade de ocorrências nos arquivos de texto, e a opção *-c* das ferramentas *grep* e *agrep* imprimem a quantidade de linhas retornada pela ferramenta, para fazer o teste de corretude dos algoritmos implementados, foi utilizado a impressão das linhas do texto em um arquivo externo para cada ferramenta e foi feito uma comparação entre estes arquivos com um comparador de texto online para verificar se os resultados estão corretos.

Todos os testes aqui citados foram executados utilizando o sistema MacOS 10.14 Mojave que possui um SSD de 128GB, um processador Intel Core i5 de 1.8GHz, e por fim uma memória LPDDR3 de 8 GB com 1600 MHz.

3.1. Casamento Exato

Para os testes de desempenho para casamento exato de padrão(ões) no texto, foi utilizado dois arquivos de texto, um com 50MB e outro com 200MB, e os algoritmos Shift-Or, Aho-Corasick e a ferramenta *grep*.

3.1.1. Padrão Único

Para realizar o teste para busca por um padrão único, foi realizado testes com tamanho de padrões variando de 1-9 e depois de 10-100 com um salto de 1 e 10 respectivamente. Como pode-se observar na figura 1 os algoritmos Shift-Or e Aho-Corasick se mostram bem estáveis. O Shift-Or apresenta um desempenho bom e constante, mesmo com a variação de tamanho do padrão, enquanto que a ferramenta grep apresenta bons desempenhos para padrões pequenos, porém dá um salto de aumento de tempo gasto à medida que o padrão cresce, e após este salto continua com um baixo porém perceptível crescimento. O Aho-Corasick por sua vez, apesar de possuir crescimentos e quedas apresenta uma variância

pequena, mostrando-se também estável. Como pode ser observado na figura 2 e 3, as curvas se mantiveram parecidas com as da figura 1, a um acréscimo do tempo que pode ser deduzido que se deve ao fato do tamanho do arquivo que cresceu.

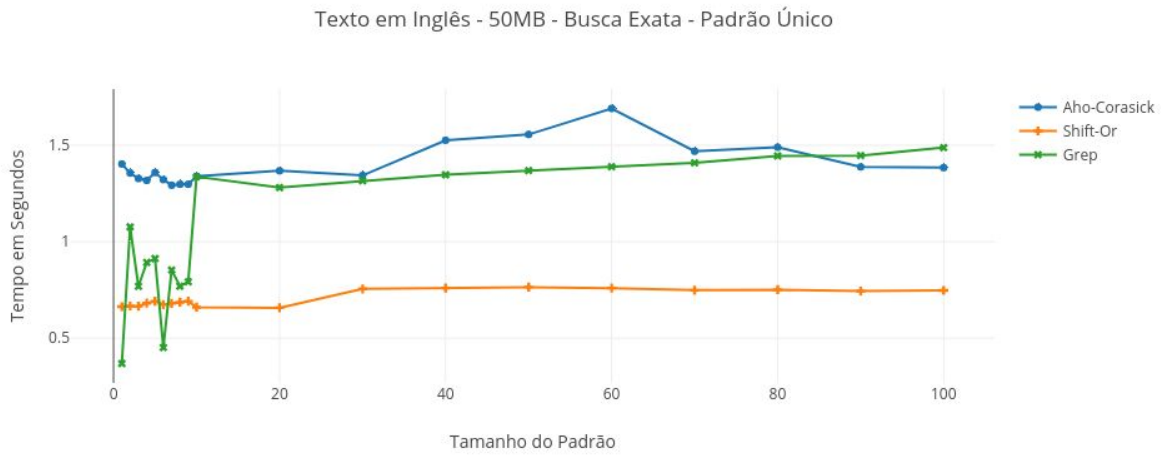


Figura 1 - Busca exata em um único padrão em texto de 50MB

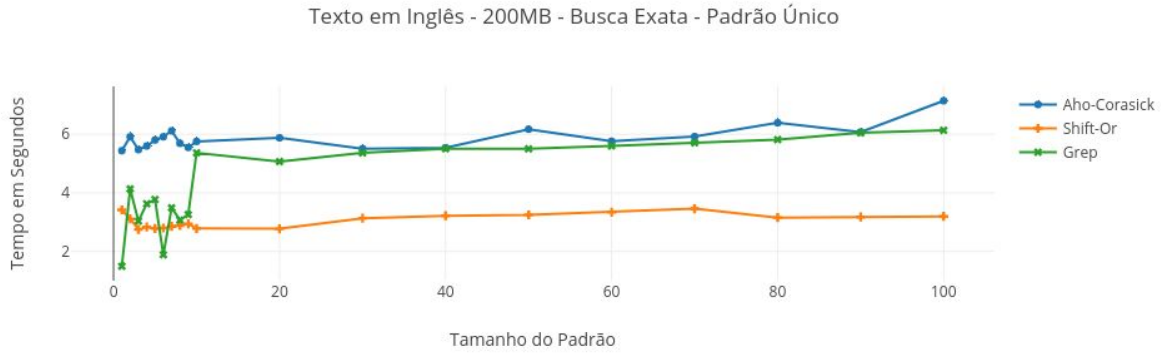


Figura 2 - Busca exata em um único padrão em texto de 200MB

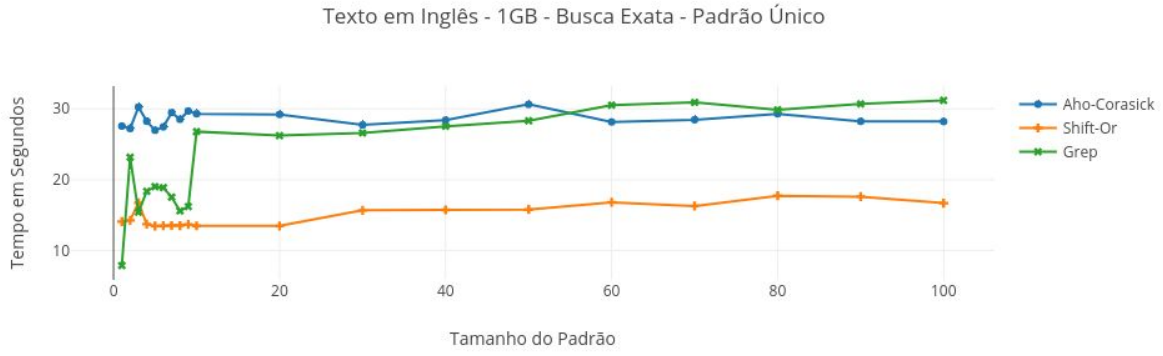


Figura 3 - Busca exata em um único padrão em texto de 1GB

3.1.2. Múltiplos Padrões

Para o casamento de múltiplos padrões no texto, foi-se utilizados arquivos com padrões variando de 1-9 e depois de 10-100 com um salto de 1 e 10, respectivamente, sobre a quantidade de padrões. Todos os padrões utilizados possuíam um tamanho fixo de 10 caracteres. A partir da figura 4 pode-se notar que o Shift-Or e o *grep* tem sem desempenho reduzido (crescimento do tempo) com o aumento da quantidade de padrões, enquanto o Aho-Corasick se mostra bastante estável em relação ao seu desempenho, o que faz sentido, tendo em vista que ele se utiliza de uma árvore *trie*, e a percorre de acordo com texto, fazendo uma busca simultânea por padrões. Além disso, ao observar a figura 5 pode-se notar que o Shift-Or se mostra um pouco mais eficiente que o Aho-Corasick realçando que a quantidade de padrões é uma variável que afeta diretamente o desempenho do Shift-Or. Também, nota-se na figura 6 e 7 que o comportamento das curvas nas figuras 4 e 5 se mantém, com um acréscimo no tempo, que se deve ao fato do tamanho do arquivo de texto.

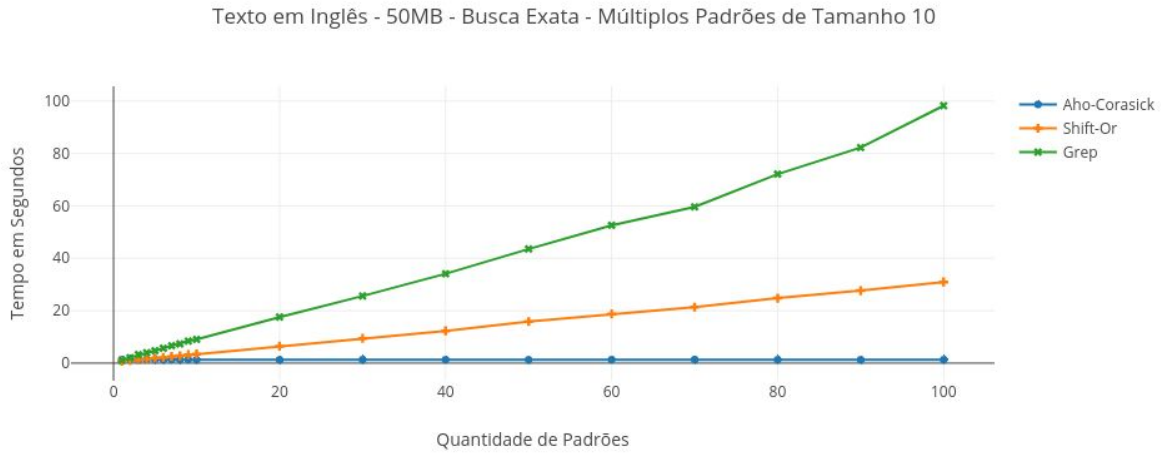


Figura 4 - Busca exata em múltiplos padrões em texto de 50MB

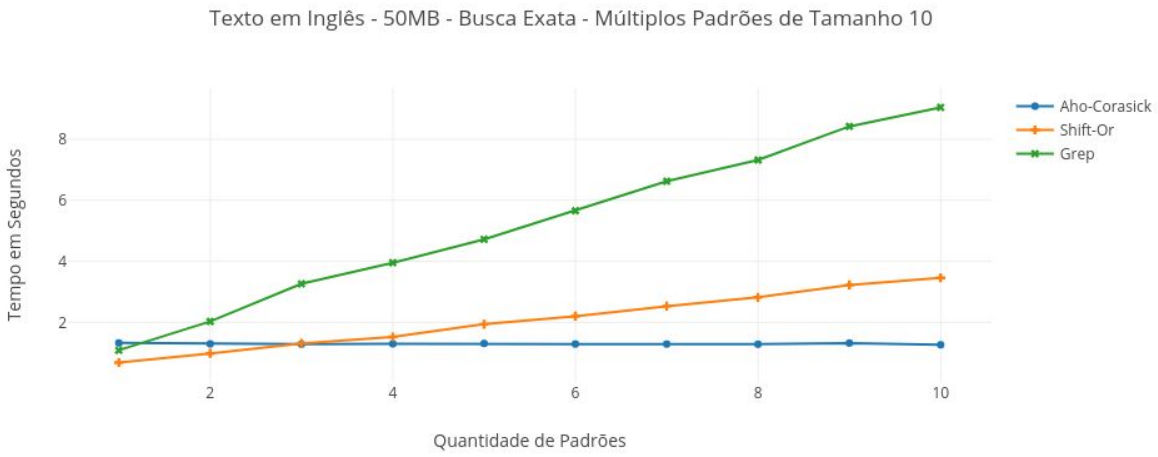


Figura 5 - Zoom da Figura 4

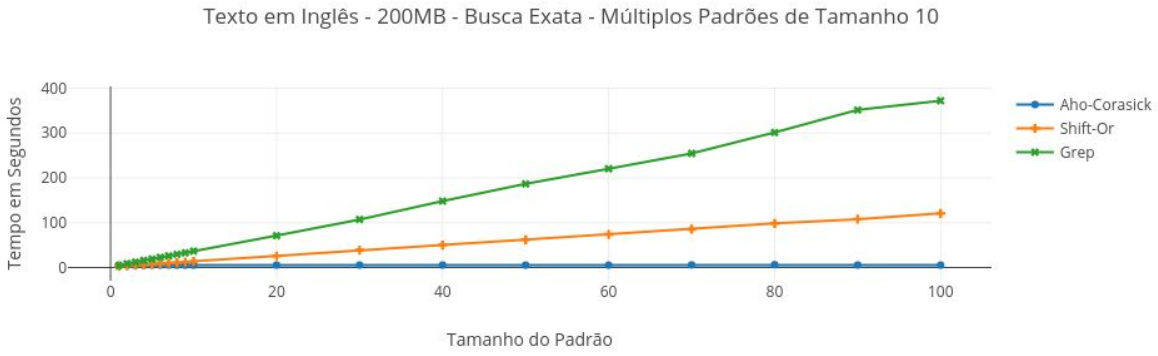


Figura 6 - Busca exata em múltiplos padrões em texto de 200MB

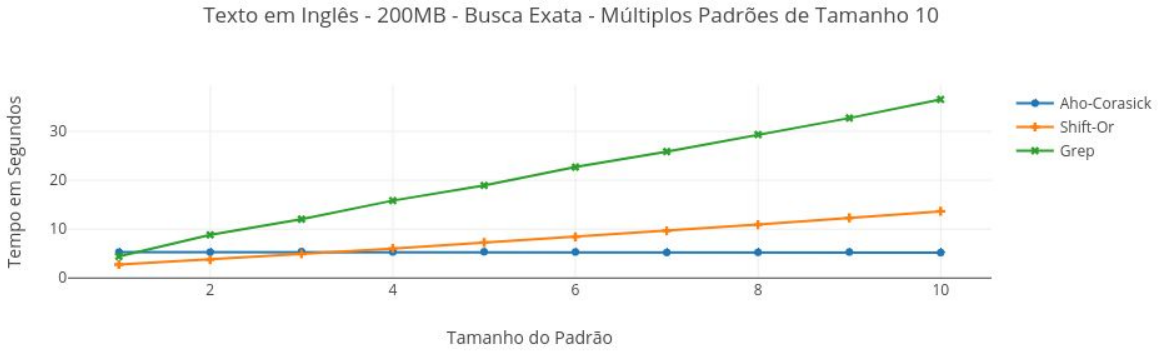


Figura 7 - Zoom da Figura 6

3.2. Casamento Aproximado

Para o casamento aproximado, foi utilizado os dois arquivos já mencionados anteriormente, com padrão único, variando o erro (distância de edição) de 1-3. Em relação a este experimento, pode-se notar o Wu-Manber é um algoritmo que se mostra bastante estável apesar do baixo desempenho, ao observar as figuras de 8-13 observa-se que ele não apresenta nenhuma relação constante de crescimento e apesar de apresentar algumas variações, elas são poucas e geralmente locais. Em contrapartida ao observar o Ukkonen nas mesmas figuras, fica evidente que ele é um algoritmo que tem um bom desempenho, porém ele apresenta crescimentos constante, em alguns casos um crescimento suave, em outras, mais rústico. Como o Wu-Manber não apresenta muita variação, e um desempenho alto, ele pode ser usado para padrões consideravelmente grandes, enquanto o Ukkonen pode ser utilizado com padrões de tamanhos razoáveis. Além disso, a ferramenta *agrep*, se mostrou bastante eficiente em todos os experimentos. Contudo, ela não suporta padrões muito grandes, o que os outros algoritmos suportam.

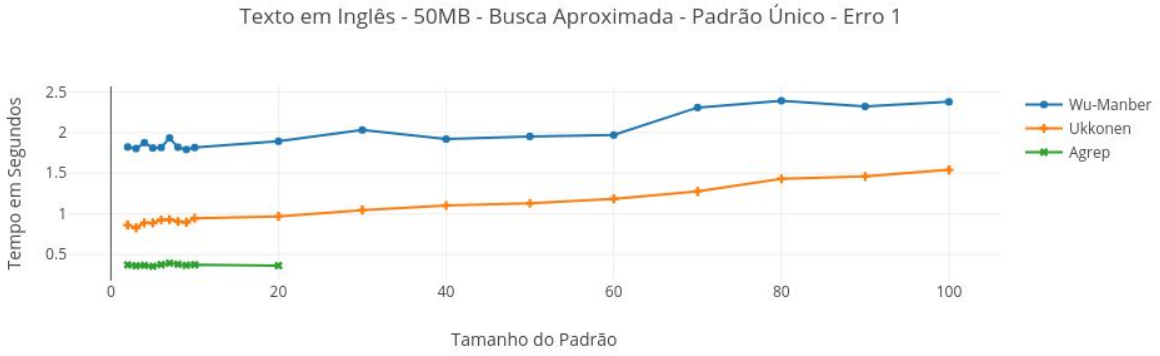


Figura 8 - Busca aproximada com padrão único e erro igual a 1 em texto de 50MB

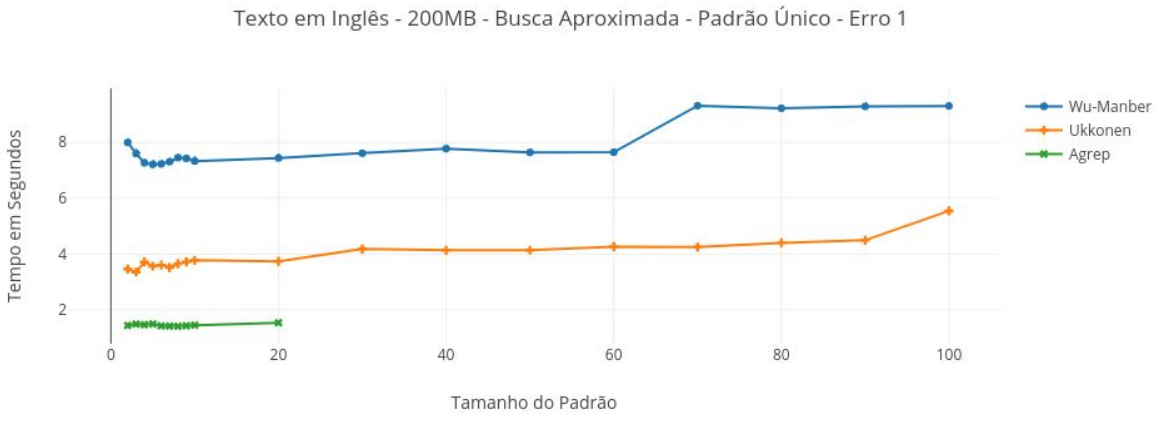


Figura 9 - Busca aproximada com padrão único e erro igual a 1 em texto de 200MB

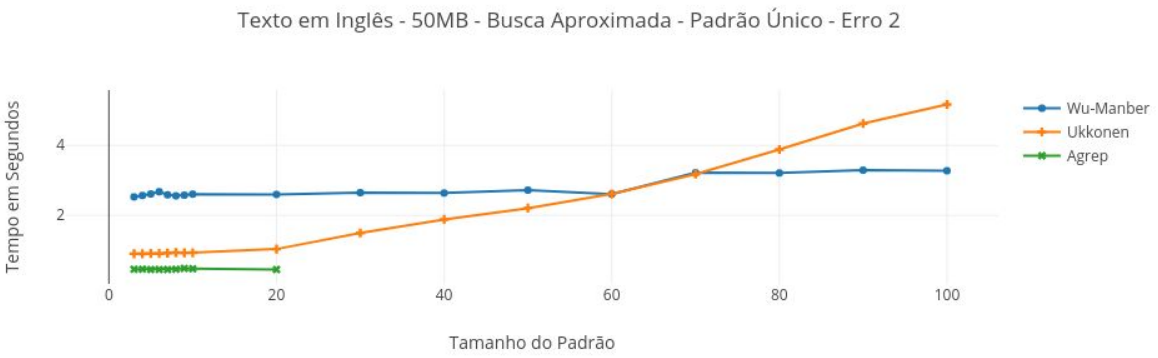


Figura 10 - Busca aproximada com padrão único e erro igual a 2 em texto de 50MB

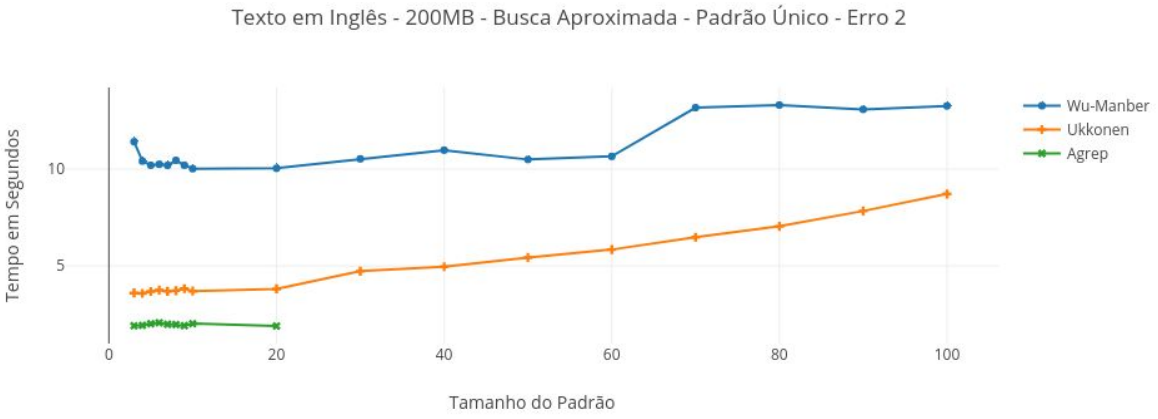


Figura 11 - Busca aproximada com padrão único e erro igual a 2 em texto de 200MB

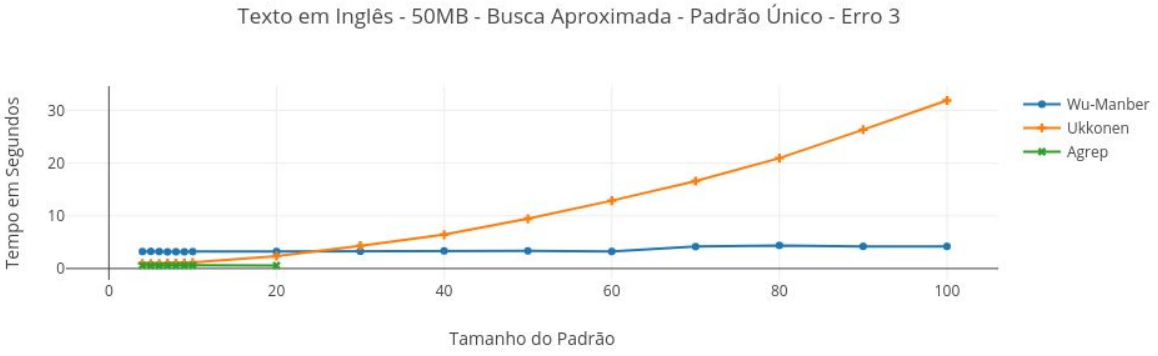


Figura 12 - Busca aproximada com padrão único e erro igual a 3 em texto de 50MB

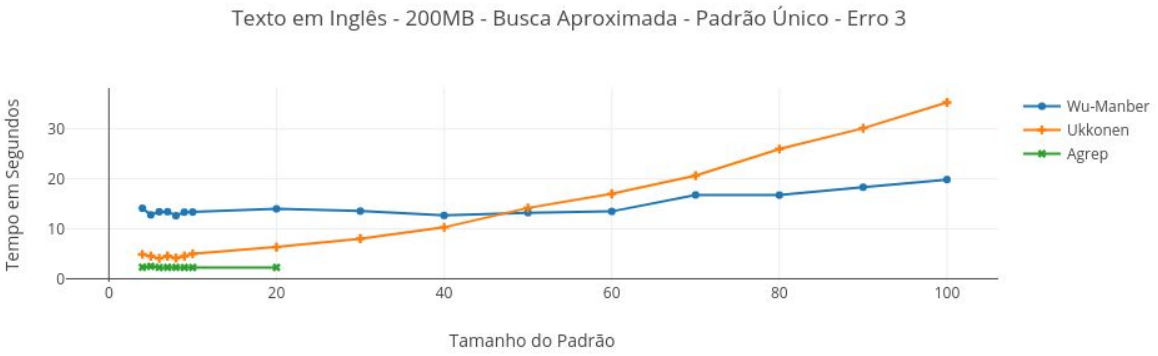


Figura 13 - Busca aproximada com padrão único e erro igual a 3 em texto de 200MB

4. Conclusão

Observando os testes rodados para analisar os algoritmos, pode-se perceber que o algoritmo Aho-Corasick se sai muito bem com múltiplos padrões para consulta, se mostrando muito estável à medida que o número de padrões aumenta e bem mais eficiente que a ferramenta grep. Por outro lado, quando se tem apenas um padrão, o Shift-Or se mostrou uma melhor opção, se mostrando superior a ferramenta grep também.

Já nos algoritmos de busca aproximada, foi observado que nos padrões de tamanho menor o Ukkonen tinha um desempenho bem melhor que Wu-Manber. Porém, a medida que o tamanho dos padrões iam aumentando, se pode perceber que o Ukkonen tinha um crescimento bem mais acentuado que Wu-Manber em termos de tempo de execução. Sendo assim, foi observado que o Wu-Manber seria uma opção melhor para padrões muito grandes. Aqui na busca aproximada, a ferramenta *agrep* se saiu melhor que os algoritmos implementados.