Submission date: 24/06/2017
Resubmission date: 18/10/2018
Camera ready submission date: 18/01/2019

1st round notification: 27/07/2018
2nd round notification: 11/01/2019
Available online: 19/01/2019
Publication date: 01/03/2019

Section: application

Um sistema para auxiliar na seleção de regiões específicas de sequências biológicas

Title: A software to help in selection of specifics regions in biological sequences

Jean Alexandre Dobre[1], **Prof. Dr. Said Sadique Adi** [1]

[1]FACOM – Universidade Federal do Mato Grosso do Sul (UFMS)
79.070-900 – Mato Grosso do Sul – MS – Brasil

Abstract. *The Specific Segment Selection Problem takes as input two or more DNA sequences and gives as output the shortest substring in one of them that has at least k differences from any substring of the other sequences. This problem is recurrent in Biology, and its solution can be used, for example, in the design of specific primers, that allows an accurate amplification of specific regions of DNA in laboratory under a PCR protocol. Given the importance of these specific regions in the detection and diagnosis of infectious diseases, we present in this paper the development of a WEB client/server software that can help people interested in the selection of specific substring from genome sequences. Our software uses two algorithms that solve the k Difference Primer Problem, chosen experimentally among four that were coded and evaluated by us.*

Keywords. *Specific segments, Amplification, PCR, Specific region, Specific Primer, Probe.*

Resumo. *O Problema da Seleção de Segmentos Especıficos recebe como entrada duas ou mais sequências de DNA e devolve como saıda o menor segmento de uma das sequências que possui pelo menos k diferenças com relação a todos os segmentos das outras sequências. Esse problema é recorrente na Biologia, e sua solução pode ser utilizada, por exemplo, no desenho de primers especıficos, que permite, usando a técnica de PCR, uma amplificação exata de regiões especıficas de DNA em laboratório. Dada a importância dessas regiões especıficas na detecção e no diagnóstico de doenças infecciosas, nós apresentamos neste artigo o desenvolvimento de uma aplicação WEB cliente/servidor que pode ajudar pessoas interessadas na seleção de segmentos especıfico de sequências genômicas. Nossa aplicação usa dois algoritmos que resolvem o Problema do Primer com k diferenças, escolhidos experimentalmente dentre quatro que foram implementados e avaliados por nós.*

Palavras-Chave. *Segmentos específicos, Amplificação, PCR, Região específica, Primer específico, Sonda.*

Cite as: Dobre, Jean A. & Adi, Said S.. (2019). A software to help in selection of specifics regions in biological sequences (Um sistema para auxiliar na seleção de regiões específicas de sequências biológicas). iSys: Revista Brasileira de Sistemas de Informação (Brazilian Journal of Information Systems), 11(3), 127-151.

1. Introdução

Problemas que envolvem sequências são estudados frequentemente em Ciência da Computação e muitos algoritmos relacionados a esses problemas podem ser aplicados em outras áreas, principamente Biologia Molecular, dando suporte a técnicas e procedimentos executados em laboratórios de análises de DNA. Dentre essas técnicas estão os testes para detecção de doenças ou de agentes causadores de doenças, que é uma tarefa difícil de ser executada por conta da necessidade de se identificar regiões específicas em sequências de DNA. Dado o fato dos algoritmos existentes para identificação de regiões específicas possuirem tempo linear (com um pré-processamento das sequências de entrada), o tamanho considerável das sequências de DNA e o número de segmentos que elas possuem, a identificação de regiões específicas pode consumir um tempo proibitivo mesmo com ajuda computacional.

Através de algoritmos específicos, é possível comparar, computacionalmente, duas ou mais sequências de DNA no intuito de encontrar regiões específicas na sequência alvo, ou seja, na sequência que se deseja estudar. Regiões específicas do genoma são trechos dessa sequência que têm uma quantidade mínima de diferenças com relação aos segmentos de outra sequência[1]. Quanto maior for a região, mais específica ela é. O problema biológico para seleção de regiões específicas em sequências de DNA é conhecido como Problema da Seleção de Regiões Específicas e pode ser modelado computacionalmente através de um problema denominado Problema do *Primer* com k Diferenças.

Em laboratórios de biologia molecular, a PCR é uma técnica básica que permite aos biólogos realizar uma ampla variedade de procedimentos, dentre eles a detecção e o diagnóstico de doenças infecciosas e hereditárias. Por ser um método direto e de alta sensibilidade, ele é capaz de identificar patógenos (bactéria, fungo, parasita), fornecendo informações precisas de tipo, quantidade e presença na amostra analisada. Identificar o tipo correto de patógeno permite um tratamento seletivo e precoce em casos de doenças cujos sintomas ainda não se manifestaram, evitando o uso indevido de medicamentos e até mesmo a morte do organismo.

Quando um espécime é infectado, células do agente infeccioso podem ser encontradas em seu fluído. Logo, é possível encontrar pelo menos dois DNAs no fluído do espécime: o seu próprio e o do agente infeccioso. É possível extrair os DNAs das células por meio de técnicas específicas. Para identificar a infecção é necessário amplificar uma região específica do DNA do agente infeccioso para possibilitar sua visualização em gel de agarose. Assim, é necessário conhecer uma região específica na sequência de DNA do agente infeccioso, ou seja, uma região que não seja igual a outra região de outro DNA presente no fluído.

Neste trabalho, propomos um estudo dos principais algoritmos que resolvem o Problema do *Primer* com k Diferenças, que estão fortemente baseados na solução do Problema do Casamento Inexato com até k Diferenças. Foi implementado as abordagens encontradas na literatura e submetida a testes e análise experimental, visando selecionar a melhor delas. Também propomos o desenvolvimento de um sistema WEB, com diversos

[1]Regiões específicas também podem ser definidas em termos de uma quantidade máxima de similaridade, mas esse tipo de definiçao é menos usada na literatura sobre o problema.

filtros e opções, que permite aos usuários interessados em selecionar regiões específicas o uso do melhor algoritmo implementado para suas entradas. Vale ressaltar que, até onde sabemos, não existe na literatura uma análise experimental como a proposta neste trabalho, e tão pouco um sistema WEB para seleção de segmentos específicos. No que tange ao sistema, o que existem são ferramentas que permitem o desenho de *primers* a partir de regiões específicas, e são dessa forma, complementares ao nosso sistema [Ye et al. 2012], [Marshall 2004], [Abd-Elsalam 2003].

Este artigo está estruturado da seguinte forma: na Seção 2 é apresentado as definições formais da Biologia e Computação necessárias à compreensão do trabalho; na Seção 3 é detalhado o problema biológico e seu modelo computacional, as abordagens para soluciona-lo e os algoritmos implementados; na Seção 4 é apresentado a análise experimental e os resultados obtidos dos testes realizados; na Seção 5 é apresentado detalhadamente o projeto e o desenvolvimento do sistema WEB e na Seção 6 concluímos sugerindo trabalhos futuros.

2. Definições preliminares

2.1. DNA, replicação e PCR

Os **ácidos nucléicos** são moléculas gigantes (macromoléculas), formadas por unidades monoméricas menores conhecidas como nucleotídeos. O ácido desoxirribonucléico, ou simplesmente DNA (do inglês, *deoxyribonucleic acid*) e o ácido ribonucléico, ou simplesmente RNA (do inglês, *ribonucleic acid*), são os dois tipos de ácidos nucléicos presentes na célula [Ribeiro 2009], [Alberts et al. 2009]. Cada nucleotídeo é formado por um radical "fosfato", uma pentose com cinco átomos de carbono e uma base. As bases que compõem o DNA são a (**A**) adenina, (**C**) citosina, (**G**) guanina e (**T**) timina, enquanto que as bases que compõem o RNA são a (**A**) adenina, (**C**) citosina, (**G**) guanina e (**U**) uracila.

Uma **fita** é uma sequência de nucleotídeos unidos quimicamente pelos carbonos 3' e 5' da pentose, através do fosfato. Quando a ligação é feita entre o carbono 5' de um nucleotídeo e o carbono 3' do próximo nucleotídeo da fita, dizemos que o sentido da fita é 5'$\rightarrow$ 3'. Da mesma forma, quando a ligação é feita entre o carbono 3' de um nucleotídeo e o carbono 5' do próximo nucleotídeo da fita, dizemos que o sentido da fita é 3'$\rightarrow$ 5' [Ribeiro 2009], [de Lima 2013]. A estrutura do DNA consiste de duas fitas de nucleotídeos dispostas de forma antiparalela, enoveladas uma sobre a outra em proteínas chamadas histonas, com aspecto de dupla hélice, composta por milhares de nucleotídeos.

A replicação de fragmentos de DNA pode ser realizada em laboratório por uma técnica chamada de *reação em cadeia da polimerase*, ou simplesmente PCR (do inglês, *Polymerase Chain Reaction*) [Alberts et al. 2009]. O DNA de interesse que contém a região a ser replicado é chamado de **DNA alvo** ou **DNA molde**. A PCR ocorre em tubos de ensaio contendo o DNA e mais alguns compostos, como *primers* ou sondas e a enzima taq DNA polimerase (DNA polimerase resistente à altas temperaturas) que faz a replicação do DNA. A PCR é essencialmente utilizada para replicar regiões específicas em determinadas sequências de DNA. Por meio dela, tendo uma amostra do material genético de um espécime (animal, planta ou micro-organismo), é possível diagnosticar se há infecção no espécime e identificar qual é o agente causador da infecção [Mullis et al. 1995].

O processo de replicação *in vitro* de um DNA consiste em três fases ou etapas. A primeira fase é a *desnaturação*, em que o DNA alvo é desnaturado por aquecimento da solução (a cerca de $95°C$) por aproximadamente 30 segundos. Como a estrutura em dupla hélice do DNA é mantida através de pontes de hidrogênio existentes entre as bases pareadas, o calor causa a separação das suas duas fitas; a segunda fase é o anelamento ou *hibridização*, em que os *primers* se ligam por hibridação às respectivas regiões complementares em cada fita. Essa etapa ocorre a uma temperatura, não fixa, entre $45°$ e $65°C$, por aproximadamente um minuto [Mullis et al. 1986]; a fase final é a extensão, ou *polimerização*, em que a taq DNA polimerase usa a extremidade 3' dos *primers* para formar uma nova cadeia com bases complementares ao molde. Essa fase ocorre a uma temperatura de $72°C$, por aproximadamente um minuto. O processo todo é então reiniciado e pode ser repetido muitas vezes até que se tenha a quantidade de fragmentos desejado [Mullis et al. 1995], [Montera and Nicoletti 2008].

Para o sucesso de uma PCR, é necessário um par de *primers* selecionados para delimitar o início e fim da região alvo da sequência de DNA que se quer amplificar. Chamamos de *primer forward* o *primer* que se liga à região inicial de uma das fitas molde e de *primer reverse*, o *primer* que se liga à região final da outra fita molde. Cada ciclo da PCR duplica apenas a região que está entre os *primers forward* e *reverse* da sequência de DNA em estudo. Conforme o procedimento é realizado, os fragmentos sintetizados servem como modelo para novos fragmentos, de forma que, com alguns ciclos, o pedaço de DNA predominante na solução é idêntico à região alvo. Assim, após um número n de ciclos de PCR, tem-se idealmente 2^n regiões idênticas à região do DNA original compreendida entre os dois *primers*.

Um ***primer*** é um pequeno segmento sintetizado quimicamente a partir de uma região específica do DNA que está sendo estudado. Chamamos de ***primers* específicos**, os *primers* que se pareiam apenas com a região do qual foi selecionado na fita do DNA molde e com nenhuma outra sequência de DNA presente na solução da PCR. Uma **sonda** (do inglês, *probe*) é um segmento complementar de DNA de tamanho variável (de 100 a 1.000 pares de bases), produzido sinteticamente e usado principalmente para detectar a presença de determinadas sequências de nucleotídeos no DNA alvo [Mullis et al. 1995].

2.2. Sequências: estruturas de dados, problemas e algoritmos relacionados

Para melhor compreensão do problema aqui estudado e dos algoritmos que serão abordados, apresentaremos nesta seção algumas definições de computação relacionadas ao Problema do *Primer* com k Diferenças. Os conceitos desta seção foram retirados, basicamente, de [Gusfield 1997], [Masek and Paterson 1980], [Ristad and Yianilos 1998], [Ito et al. 1995], [Landau and Vishkin 1989], [Cormen et al. 2012].

Uma **sequência** ou **cadeia** s construída sobre um alfabeto Σ, que corresponde a um conjunto finito de caracteres, nada mais é do que uma concatenação ordenada e finita de caracteres de Σ. O **índice** i de uma sequência s representa a posição de um caractere nessa sequência, e denotado por s_i o i-ésimo caractere dela. O **tamanho** de uma sequência s é representado por $|s|$ e corresponde à quantidade de caracteres que ela possui. Uma sequência t de tamanho n é denotado por $t_{1..n}$. Se uma sequência é vazia, ou seja, se $|s| = 0$, ela é representada por ϵ e chamada de **cadeia vazia**. Uma **subcadeia** ou

segmento de uma sequência s é um trecho de s com zero ou mais caracteres consecutivos. Se o primeiro caractere de uma subcadeia de s corresponde ao seu i-ésimo caractere e o último ao seu j-ésimo caractere, a subcadeia é denotada por $s_{i..j}$. Se $i > j$, o segmento corresponde à cadeia vazia ϵ.

Um **prefixo** de uma sequência s é um segmento $s_{1..j}$ tal que $1 \leq j \leq |s|$. Se $j < 1$, tem-se um **prefixo vazio**, que corresponde à cadeia vazia. De forma semelhante, um **sufixo** de uma sequência s é um segmento $s_{i..|s|}$ tal que $1 \leq i \leq |s|$. Se $i > |s|$, tem-se um **sufixo vazio**, que também corresponde à cadeia vazia. O **i-ésimo** sufixo de s, denotado por $s_{i..n}$, representa um sufixo iniciando na posição i de s.

O **maior prefixo comum** ou LCP (do inglês, *Longest Commom Prefix*) entre posições de duas sequências, $p_{1..m}$ e $t_{1..n}$, é o maior segmento $s_{i..k}$, com $0 \leq k \leq m, n$, tal que $s_1 = p_1 = t_1$, $s_2 = p_2 = t_2$ ... $s_k = p_k = t_k$. Se $k = 0$ então $s = \epsilon$. É denotado por $LCP_{p,t}(i, j)$ o maior prefixo comum entre $p_{i..m}$ e $t_{j..n}$.

A **maior extensão comum** ou LCE (do inglês, *Longest Common Extension*) entre posições de duas sequências, $p_{1..m}$ e $t_{1..n}$, é o tamanho do maior prefixo comum entre $p_{i..m}$ e $t_{j..n}$, ou seja, $|LCP_{p,t}(i, j)|$. É denotado por $LCE_{p,t}(i, j)$ a maior extensão comum entre $p_{i..m}$ e $t_{j..m}$.

Uma **árvore de sufixo** é uma árvore de dados construída a partir de sufixos de um determinado texto ou sequência. Dada uma sequência p, de tamanho n, construída sobre um alfabeto Σ, a árvore de sufixo ST da sequência p, concatenado com o símbolo \$, é uma árvore com $n + 1$ nós folhas, enumerados de 1 a $n + 1$. Cada sufixo $p_{i..n}$ de p define exatamente uma folha da árvore, ou seja, cada nó folha recebe um índice de p como rótulo que representa o sufixo $p_{i..n}$ de p. Cada nó interno pode ter dois ou mais filhos e cada aresta de ST representa um segmento não vazio de p [Ukkonen 1995]. O grau de cada nó da árvore é zero, quando o nó é uma folha, ou maior ou igual a dois quando o nó é interno.

O **vetor de sufixo** (do inglês, *suffix array*), é um vetor com os índices de todos os sufixos de uma sequência organizados de forma a expô-los em ordem lexicográfica. Dada uma sequência $t_{1..n}$, construída sobre um alfabeto Σ, onde $t_n = \$$, e \$ não está contido em Σ e precede qualquer outro símbolo contido em Σ, o vetor de sufixo $sa[1..n]$ de t é um vetor de n números inteiros j, que representam os sufixos de t, organizados em ordem lexicográfica. Ou seja, $t_{sa[1]..n} < t_{sa[2]..n} < .. < t_{sa[n]..n}$ [Huynh et al. 2006].

A **árvore de sufixo comprimida** (do inglês *compressed suffix trees* ou *CST*) é considerada uma estrutura de dados de indexação própria, com tamanho reduzido, que representa uma árvore de sufixo, através de vetores, simulando as principais funcionalidades convencionais da árvore de sufixo [Välimäki et al. 2009].

A **distância de edição** é uma métrica que determina a menor quantidade de operações de *inserção*, *remoção* e *substituição* necessárias para transformar uma sequência u em outra sequência v [Gusfield 1997], [Masek and Paterson 1980], [Ristad and Yianilos 1998]. Essas operações são chamadas de **operações de edição** e elas atuam sobre caracteres individuais de uma sequência. Como exemplo, tomando-se s = TACTG e t = ACGCT, a distância de edição entre elas corresponde a 4, relacionada às

operações de remoção de s_1 = T, substituição de s_4 = T por G, substituição de s_5 = G por C e inserção de T ao final de s, totalizando 4 diferenças.

Dada uma sequência p, de tamanho m, uma sequência t, de tamanho n, e um inteiro $k \geq 1$, o **Problema do Casamento Inexato com até k Diferenças (PCIkD)** consiste em encontrar todas as ocorrências de p em t com até k diferenças [Landau and Vishkin 1989]. Para exemplificar, dadas as sequências p = ACT e t = AGTTTTC, há ocorrência de p em t terminando na posição 3 de t com uma diferença (caractere C substituído por G) e terminando na posição 4 de t com duas diferenças (caracteres AC substituídos por GT).

Um algoritmo para resolver o PCIkD usando a distância de edição como medida de diferença entre duas sequências baseia-se na programação dinâmica e consiste em preencher uma matriz $D[0..m; 0..n]$, de acordo com a seguinte recorrência:

$$D[i,j] = \begin{cases} 0 & \text{se } i = 0 \\ j & \text{se } j = 0 \\ min \begin{cases} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1)+1, \quad [s_i \neq t_i] \end{cases} & \text{se } i,j > 0 \end{cases} \tag{1}$$

Cada posição $D[i,j]$ armazena o menor número de diferenças entre $p_{1..i}$ e qualquer segmento do texto terminando em t_j. Após a matriz ser preenchida, se $D[m,j] \leq k$, para algum j entre 1 e n, então existe uma ocorrência com até k diferenças de $p_{1..m}$ finalizando em t_j. Caso contrário, pode-se afirmar que tal ocorrência não existe. Tal algoritmo tem a complexidade de tempo $O(mn)$ e pode ser implementada consumindo espaço $O(n)$.

Um algoritmo alternativo para resolver o PCIkD [Landau and Vishkin] simula o preenchimento da matriz D através de suas diagonais. Uma **diagonal** d da matriz D consiste de todos os $D[i,l]$ tal que $l - i = d$. Para um número de diferenças e e uma diagonal d, definimos uma matriz $L[-(k+1)..n+1; -1..k]$, onde $L[d,e]$ denota a *maior linha* i tal que $D[i,l] = e$ e $D[i,l]$ está na diagonal d. A definição de $L[d,e]$ implica que e é o menor número de diferenças entre $p_{1..L[d,e]}$ e qualquer segmento terminando em $t_{d+L[d,e]}$. Outra implicação é que $p_{L[d,e]+1} \neq t_{L[d,e]+d+1}$. Cada posição $L[d,e]$ é computada da seguinte forma:

$$L[d,e] = \begin{cases} -1 & \text{se } d = n+1 \text{ ou } e = -1 \\ |d|-2 & \text{se } e = |d|-2 \\ |d|-1 & \text{se } e = |d|-1 \\ row + LCE_{p,t}(row+1, d+row+1) & \text{se } e > 0 \text{ e } d \geq -e, \end{cases} \tag{2}$$

onde:

$$row = max \begin{cases} L[d,e-1]+1 \\ L[d-1,e-1] \\ L[d+1,e-1]+1, \end{cases}$$

para toda diferença e de 0 a k e toda diagonal d de $-e$ até n.

Note que $d + L[d, e]$ corresponde exatamente a l. Assim, nessa alternativa de solução do PCIkD, deve se procurar por algum $L[d, e]$ igual a m tal que e seja $\leq k$, o que significa que o padrão $p_{1..m}$ ocorre no texto finalizando em t_{d+m} com e diferenças. Em caso de não existir nenhum $L[d, e] = m$ tal que $e \leq k$, pode ser afirmado que não há ocorrência de p em t com no máximo k diferenças [Landau and Vishkin 1989]. Esse algoritmo possui complexidade de tempo $O(kmn)$ e espaço $O(kn)$, usando um algoritmo ingênuo para a computação da maior extensão comum entre p e t, ou seja, incrementando a variável row enquanto $p_{row+1} = t_{row+f+1}$ e row não alcançou os tamanhos de p e t. Tal algoritmo, claramente, consome tempo $O(n)$.

Contudo, é possível determinar o valor de $LCE_{s,t}(i, j)$ em tempo $O(1)$ empregando uma estrutura de dados capaz de armazenas todos os LCEs possíveis e devolvê-los, sob consulta, em tempo constante. Tais estruturas de dados, como *árvores de sufixo* [Gusfield 1997], [Ito et al. 1995] e *vetores de sufixo* [de Castro Miranda et al. 2005], são construídas em tempo hábil, numa etapa de pré-processamento, e permitem melhorar a complexidade de tempo do algoritmo alternativo que resolve o PCIkD para $O(kn)$.

3. O problema da seleção de segmentos específicos

A tarefa de seleção de regiões específicas do DNA alvo tem um papel fundamental no sucesso da PCR e é imprescindível para se evitar o desperdício de tempo e recursos envolvidos no processo. Na prática é uma tarefa difícil de ser executada por conta do tamanho das sequências e da quantidade de possibilidades.

Informalmente, tendo em mãos duas ou mais sequências de DNA, é necessário encontrar o menor segmento dessa sequência que tenha k diferenças em relação a qualquer segmento das outras sequências. Esse segmento específico, se encontrado, é considerado uma região específica e pode ser utilizado na PCR como região alvo a ser replicada. Além disso, o segmento específico pode ser utilizado como *primer* específico ou sonda, dependendo do seu tamanho. Podemos formalizar o problema biológico de Seleção de Segmentos Específicos, da seguinte forma:

> **O Problema da Seleção de Segmentos Específicos**: dadas duas ou mais sequências de DNA, encontrar o menor segmento em uma delas que tenha pelo menos k diferenças com relação a todos os segmentos das outras sequências.

Esse problema prático pode ser modelado para o problema computacional conhecido como Problema do Primer com k Diferenças. Mais detalhadamente, dadas duas sequências α e β e um inteiro $k > 0$, o **Problema do *Primer* com k Diferenças (PPkD)** consiste em encontrar, para cada índice j de α, o menor segmento γ de α que se inicia em j e que tenha pelo menos k diferenças com relação a qualquer segmento de β [Gusfield 1997].

Para exemplificar, dadas duas sequências α = CCCGGCCC, β = CCCGTGCCC e um valor k = 1, na posição 1 de α tem-se o resultado γ = CCCGG, na posição 2 o resultado γ = CCGG, na posição 3 o resultado γ = CGG, na posição 4 o resultado γ =

Tabela 1. Exemplo de uma matriz D da abordagem 1.

			A	**A**	**A**	**C**	**G**	**T**	**A**	**A**
		0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
A	1	1	0	0	0	1	1	1	0	0
A	2	2	1	0	0	1	2	2	1	0
T	3	3	2	1	1	1	2	2	2	1
A	4	4	3	2	1	2	2	3	2	2

GG, e da posição 5 até a posição 8 a resposta é ε. Já para o parâmetro k = 2 a resposta é ε para todas as posições de α, pois não há nenhum segmento de α que tenha pelo menos 2 diferenças com relação a qualquer segmento de β.

Para resolver o PPkD foi utilizado os algoritmos apresentados na Seção 2 para resolver o PCIkD invertendo-se o ponto de vista associado aos seus resultados. De forma mais detalhada, o método consiste em três passos. No primeiro passo, para cada índice j de α, com $1 \leq j \leq (m - k)$, são passados os parâmetros $\alpha_{j..m}$ como p, β como t e k. No segundo passo, executa-se um dos dois algoritmos que resolvem o PCIkD e, no terceiro passo, analiza-se a matriz de programação dinâmica resultante da execução do algoritmo para encontrar o menor prefixo da sequência $\alpha_{j..m}$ (se existir) que tenha distância de edição de pelo menos k com relação à β.

Chamaremos de **abordagem 1** a análise de dados da matriz D, resultante da execução do algoritmo que resolve a Recorrência 1, e de **abordagem 2** a análise de dados da matriz $L[d, e]$, resultante da execução do algoritmo que resolve a Recorrência 2. Cada abordagem possui tempo de execução e uso de memória diferente, mas seguem a mesma estratégia, definida nos três passos descritos anteriormente. Nelas, a ocorrência de segmento específico para cada posição j da sequência α é representado por uma determinada linha, ou seja, o valor devolvido da execução de uma das abordagens é o tamanho do segmento específico.

3.1. Abordagem 1

Após a matriz D ser computada, a abordagem 1 consiste em percorrer todas as linhas de D procurando a primeira linha r da matriz D tal que $D[r, l] \geq k$ para todo $1 \leq l \leq n$ e, devolver $\gamma = p_{1..r}$ se essa linha existir. Se $r = m$ e houver algum valor menor que k nesta linha, pode ser afirmado que não existe um segmento de α começando em j com pelo menos k diferenças com relação à β. Se α tem tamanho m e β tem o tamanho n, então o PPkD pode ser resolvido pela abordagem 1 em tempo $O(m^2n)$ e uso de memória $O(mn)$.

Para exemplificar o uso desse algoritmo, dadas as sequências p = AATA e t = AAACGTAA, o valor k = 2 e a matriz $D[0..4, 0..8]$ apresentada na Tabela 1, observe que existem ocorrências da sequência p na sequência t com até k diferenças terminando em t_2, t_3, t_4, t_5, t_7 e t_8.

3.2. Abordagem 2

Após a matriz $L[d, e]$ ser computada, a abordagem 2 consiste primeiramente em percorrer todas as células $L[d, e]$, tal que $e < k$, procurando pelo valor m. Se existe algum $L[d, e]$ tal que $e < k$ com o valor m, isso significa que existe uma ocorrência de $p_{1..m}$, terminando em $t_{L[d,e]+d}$, com menos que k diferenças e pode ser afirmado que não existe um segmento de α começando em j com pelo menos k diferenças com relação à β. Se não existe nenhum par de valores d e e tais que $L[d, e] = m$ e $e < k$, então há ocorrência de segmento específico. Nesse caso, percorremos a coluna $k - 1$ de L no intuito de obter a maior linha r armazenada em algum $L[d, k - 1]$, para $-k \leq d \leq n$, incrementá-la em uma unidade fazendo-se $r = r + 1$ e devolver $\gamma = p_{1..r}$. Isso é necessário pois cada $L[d, e]$ representa uma ocorrência de $p_{1..L[d,e]}$ finalizando em $t_{d+L[d,e]}$, com exatamente e diferenças. Por definição, sabemos que $p_{L[d,e]+1} \neq t_{L[d,e]+d+1}$, logo $p_{1..L[d,e]+1}$ tem $e + 1$ diferenças com relação a qualquer segmento finalizando em $t_{L[d,e]+d+1}$.

Se α tem tamanho m e β tem o tamanho n, então o PPkD pode ser resolvido pela abordagem 2 em tempo $O(km^2n)$ e espaço $O(nk)$, com o algoritmo ingênuo para determinação do LCE. Quando o LCE é determinado em tempo constante, utilizando uma estrutura de dados como árvores de sufixo ou vetor de sufixo, o tempo total do algoritmo passa a ser $O(kmn)$ [Gusfield 1997].

Um exemplo de matriz L preenchida com base nas diagonais de D pode ser visto na Tabela 2. Nesse exemplo, são utilizadas as mesmas sequências do exemplo anterior, ou seja p = AATA, t = AAACGTAA e k = 2. A diagonal 0, com 2 diferenças, representada por $L[0, 2] = 4$, corresponde à ocorrência de $p_{1..4}$ terminando em t_{4+0}. A diagonal 5, com 1 diferença, representada por $L[5, 1] = 3$ corresponde à ocorrência de $p_{1..3}$ terminando em t_8. A diagonal -1 alcança a linha m com apenas uma diferença, ou seja, $L[-1, 1] = 4$, que corresponde à ocorrência de $p_{1..4}$ terminando em t_3. Perceba que o tamanho desta nova matriz é relativamente menor do que a matriz D.

Tabela 2. Exemplo de matriz $L[d, e]$ apresentando valores computados com base nas diagonais da matriz D.

	-1	**0**	**1**	**2**
-3	-	-	1	2
-2	-	0	1	4
-1	-1	0	4	-
0	-1	2	3	4
1	-1	2	3	4
2	-1	1	2	3
3	-1	0	1	4
4	-1	0	1	4
5	-1	0	3	-
6	-1	2	-	-
7	-1	1	-	-
8	-1	0	-	-
9	-1	-1	-1	-1

Observe que as soluções propostas para o PPkD envolvem somente duas sequências. Para a solução do Problema da Seleção de Segmentos Específicos quando se tem uma sequência alvo e um número c de sequência de comparação, as abordagens propostas precisam ser executadas c vezes, uma vez para cada sequência de comparação (tendo sempre a mesma sequência alvo). A solução final corresponde ao maior segmento encontrado dentre todas as c soluções devolvidos ao final de cada execução. Essa estratégia aumenta a complexidade de tempo das abordagens descritas por um fator de c.

3.3. Implementação

Ao todo, foram implementados cinco programas diferentes baseados nos algoritmos para resolver o PPkD. São quatro programas baseado na abordagem 2, onde cada um deles com um algoritmo diferente que determina o LCE entre duas sequências, e um programa baseado na abordagem 1. Atribuímos a sigla KDP, de *k-difference primer*, seguida de um número sequencial, para cada implementação.

Todos os programas foram implementados na linguagem C++, com base em [Loudon 2000], [Deitel and Deitel 2005], [Sutter 2006], que permite uma estruturação de classes e reaproveitamento de objetos. Com relação às estruturas de dados mais complexas, no caso as árvores de sufixo, árvores de sufixo comprimidas e vetores de sufixo, fizemos uso de bibliotecas prontas, de outros autores, que em geral costumam ser implementações eficientes, testadas e estáveis, prontas para serem utilizadas.

A Tabela 3 apresenta alguns detalhes dos cinco programas desenvolvidos com detalhes sobre eles. Vale ressaltar que na versão final de nosso sotware foi utilizado apenas as duas implementações consideradas as melhores, com base no tempo de execução e espaço de memória alocado.

Tabela 3. Diferentes implementações realizadas

Nome	**Algoritmo**	**Estrutura de dados**	**Complexidade**	
			Tempo	**Espaço**
KDP1	Abordagem 1	-	$O(m^2n)$	$O(n)$
KDP2	Abordagem 2	-	$O(km^2n)$	$O(k+n)$
KDP3	Abordagem 2	Árvore de sufixo comprimida	$O(kmn \log n)$	$O(k+n+\|CST\|)$
KDP4	Abordagem 2	Vetor de sufixo	$O(kmn)$	$O(k+n+4n \log n)$
KDP5	Abordagem 2	Árvore de sufixo	$O(kmn)$	$O(k+n+\|ST\|)$
Observações: m representa o tamanho da sequência α, n o tamanho da sequência β, k é a quantidade de diferenças, CST corresponde a $O(n \log \|\Sigma\|) + 6n + o(n)$ e ST corresponde a $20v + 4n + 12n$, onde v representa o número de nós da árvore;				

4. Análise experimental e resultados

Para fazer a análise experimental dos programas implementados, foram utilizados dados gerados aleatoriamente para criar sequências artificiais e dados reais de sequências do *Homo sapiens*. O uso desses dois tipos diferentes de sequência permite uma análise do comportamento dos programas em diversas situações. Em cada execução, foram passados

como parâmetros duas sequências e um valor inteiro positivo de diferenças. Os testes experimentais foram realizados em um servidor Intel Xeon(R) E5-4650 2.7 GHZ, com 20 MB de memória *cache*, 95 GB de memória RAM e 8 núcleos de processamento, dedicado exclusivamente para os testes. O sistema operacional utilizado foi o Debian GNU/Linux 8 (jessie).

4.1. Dados artificiais

Para a realização dos testes com dados artificiais, foi desenvolvido um *script* escrito na linguagem C++, para gerar as sequências de dados. Este programa utiliza, basicamente, a função *random* para gerar um número aleatório entre 1 e 4, cada qual representando uma letra do alfabeto $\Sigma = \{A, C, G, T\}$ (1 = A, 2 = C, 3 = G e 4 = T). Optou-se por executar os testes artificiais com dezenove sequências, cujos tamanhos variam de 1000 a 10000 caracteres, em intervalos de 1000 caracteres, e de 10000 a 100000 caracteres, em intervalos de 10000 caracteres. O valor de k foi fixado nos valores 30, 300 e 600 diferenças, para compreender o uso do sistema para diferentes tamanhos de regiões específicas.

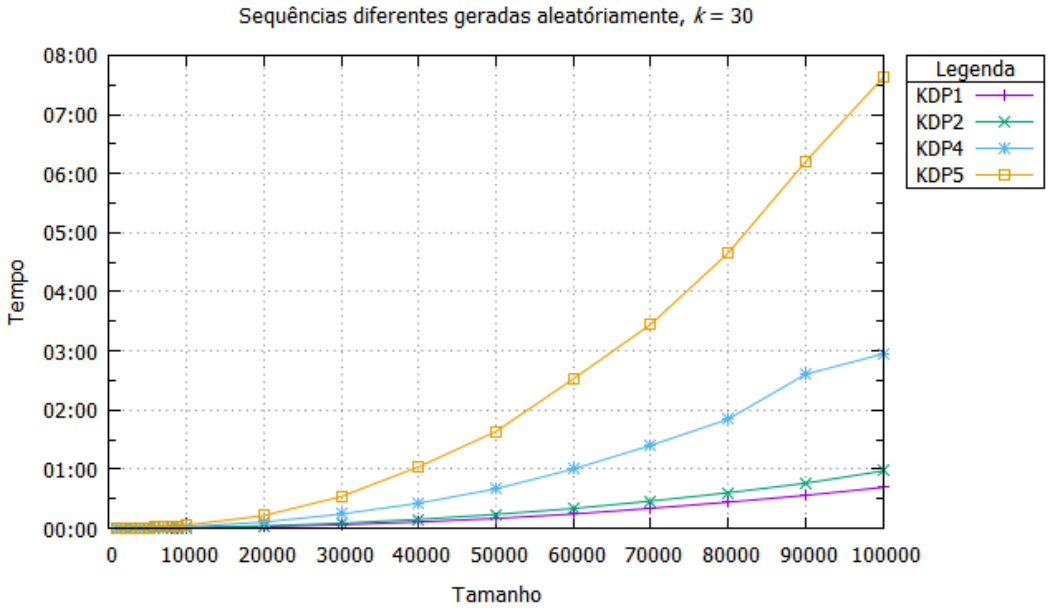

Figura 1. Gráfico de tempo de execução das sequências diferentes, geradas aleatoriamente, $k = 30$**.**

Cada teste foi executado três vezes e extraída a média do tempo de execução e da quantidade de memória utilizada. Nos gráficos 1, 2 e 3 são apresentados os resultados dos testes realizados com sequências diferentes de dados artificiais, para $k = 30$, $k = 300$ e $k = 600$ respectivamente. Observe os tempos absurdamente altos obtidos pelo KDP3 referente ao primeiro teste executado. Como interessa apenas os melhores algoritmos, e percebendo que o programa KDP3 não é competitivo na prática, foi retirado dos demais testes.

Com base nos testes realizados, percebemos que o programa KDP1, que implementa a abordagem 1, é ligeiramente mais rápido que o programa KDP2, que implementa

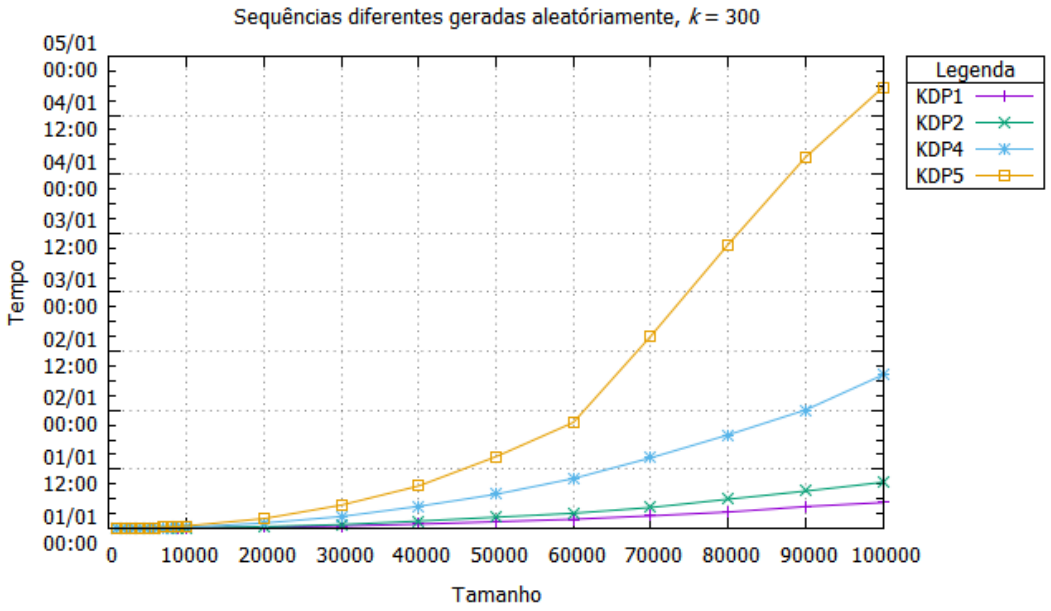

Figura 2. Gráfico de tempo de execução das sequências diferentes, geradas aleatoriamente, $k = 300$**.**

a abordagem 2, o que condiz com a diferença de complexidade teórica desses algoritmos. Mais detalhadamente, ele é cerca de 30% mais rápido, na média, quando $k = 30$, cerca de 61% mais rápido, na média, quando $k = 300$ e cerca de 63% mais rápido, na média, quando $k = 600$. Ambos são muito superiores aos algoritmos KDP4 e KDP5 em cerca de mais de 280% na média.

Acreditamos que a vantagem em tempo obtida pelo programa KDP2 em relação aos programas KDP4 e KDP5 nos testes com sequências completamente diferentes e sequências aleatórias, e igualado nos testes com sequências iguais, pode ser explicada pela organização de memória dos processadores atuais, que devido ao avanço tecnológico obtido com a diminuição dos transistores, possibilitou a adição de memórias de acesso rápido junto à CPU, chamadas de memórias *cache*. Dessa forma, as memórias passaram a ser organizadas por hierarquia de proximidade física ao processador, onde quanto mais próxima estiver, mais rápido é o acesso a ela. O tamanho dessa memória porém é limitado e por conta disso existe um controlador que gerencia os dados nelas armazenadas. Esse controlador sincroniza os dados das memórias *cache* com a memória principal. Caso o dado buscado não seja encontrado na memória *cache*, o controlador busca na memória principal. Como o tempo de sincronização das memórias leva vários ciclos de *clock* do processador, o controlador faz uma previsão de uso dos dados e sempre traz para a memória *cache* registros de memória próximos. Essa organização favorece os programas KDP1 e KDP2, que fazem uso apenas das matrizes de programação dinâmica, onde a próxima célula da matriz quase sempre está na memória *cache* [Hennessy and Patterson 2014].

No caso do KDP4 e do KDP5, suas estruturas de dados possuem uma organização não sequencial na memória do computador e o acesso aos dados é direto para cálculo do

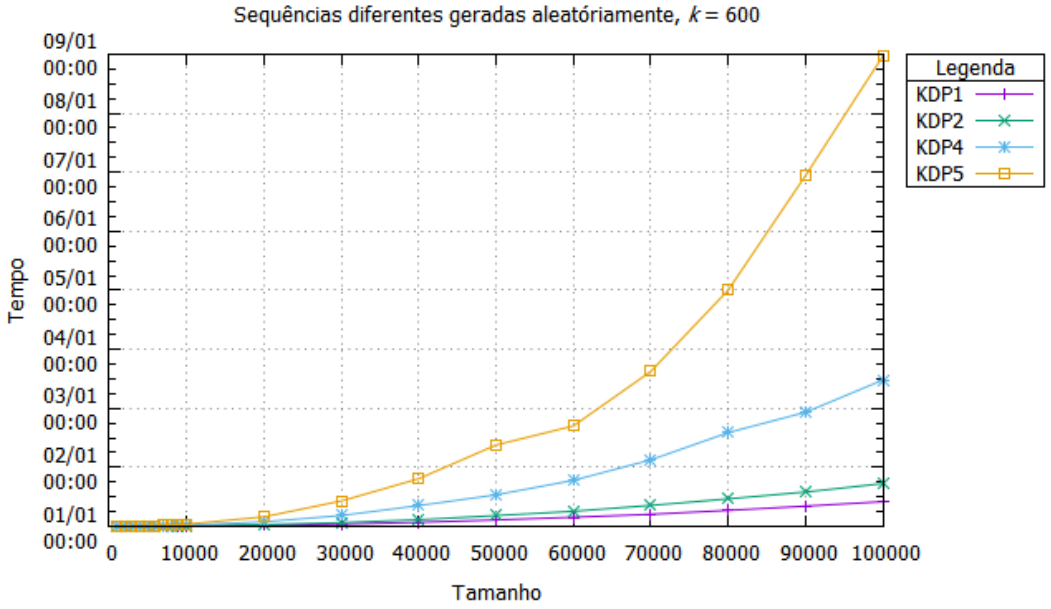

Figura 3. Gráfico de tempo de execução das sequências diferentes, geradas aleatoriamente, $k = 600$**.**

RMQ no vetor de sufixo ou cálculo do LCA na árvore de sufixo, podendo ocasionar vários *cache miss*, onde o dado não está na memória *cache* e o controlador precisa buscar na memória principal, o que prejudica muito o desempenho [Hennessy and Patterson 2014], [Saikkonen and Soisalon-Soininen 2008]. Somando-se a isso a grande quantidade de computação do LCE envolvida no processamento das sequências, fica claro a diferença de tempo considerável entre esses programas e o KDP2.

A média de uso de memória do programa KDP1, é praticamente idêntico à do KDP2 em todos os casos de teste. Quando k = 30, o programa KDP4 utiliza, em média, cerca de 50% mais memória que os programas KDP2 e KDP1 enquanto que o programa KDP5 cerca de 870% mais memória que o programa KDP4. Quando $k = 300$, o programa KDP4 utiliza, em média, cerca de 14% mais memória que os programas KDP2 e KDP1 e o programa KDP5 utiliza cerca de 329% mais memória que o programa KDP4. Finalmente, quando $k = 600$, o programa KDP4 utiliza, em média, cerca de 18% mais memória que os programas KDP2 e KDP1 enquanto que o programa KDP5 utiliza cerca de 177% mais memória que o programa KDP4.

4.2. Dados reais

Os testes realizados com sequências artificiais geradas de forma aleatória apresentaram resultados importantes mas que não abrangem todos os casos possíveis. Isso pois, sabemos que as sequências de DNA de seres vivos não se enquadram em nenhum dos três testes realizados com dados aleatórios. Esta bateria de testes com dados reais é importante para avaliar o comportamento dos programas com sequências reais. Para executar os testes com dados reais, foi utilizado dez sequências do *Homo sapiens*, cada uma delas contendo um gene específico, de diversos tamanhos, a saber, *hsarhgdig* com 4398 carac-

teres, *hsascl2* com 4455 caracteres, *hsankrd43* com 5457 caracteres, *hsa1bg* com 8694 caracteres, *hsalg10b* com 14972 caracteres, *hsbad* com 16877 caracteres, *hsankr10* com 38530 caracteres, *hsascc2* com 51655 caracteres, *hsasz1* com 66302 caracteres e *hsaff4* com 90284 caracteres. Foram executadas três séries de testes, variando o valor de k e comparando uma das sequências com todas as outras. Cada série de testes foi executado três vezes e extraída a média do tempo de execução e da quantidade de memória utilizada. Os resultados dos testes com o *hsarhgdig* são apresentados nos Gráfico 4 para $k = 30$, Gráfico 5 para $k = 300$ e Gráfico 6 para $k = 600$.

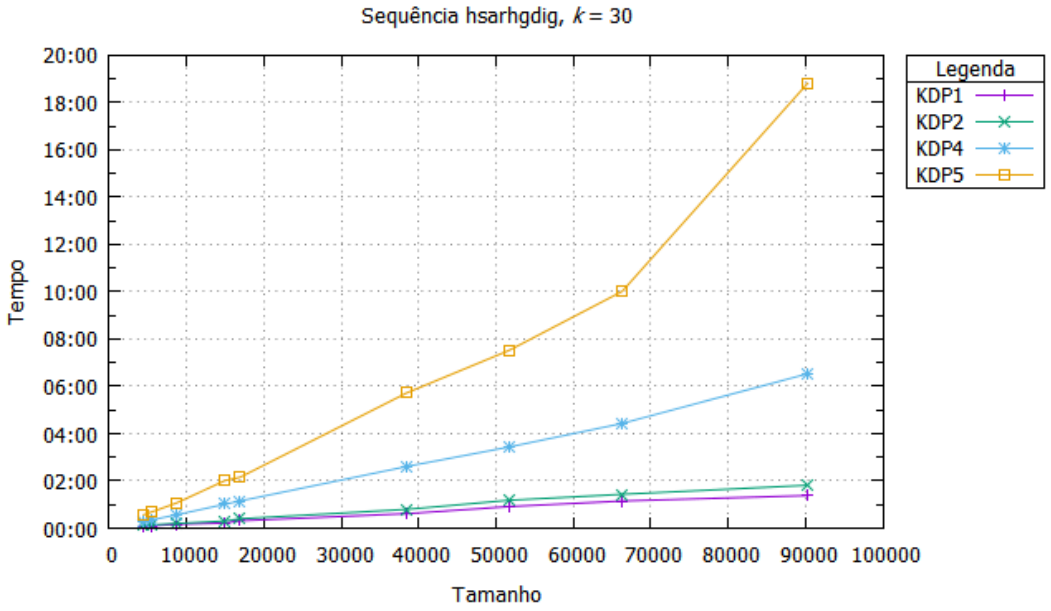

Figura 4. Gráfico de tempo de execução da sequência hsarhgdig, $k = 30$.

Foi verificado que o comportamento dos programas é similar àquele obtido com dados artificias gerados de forma aleatória, e com isso conclusões parecidas podem ser obtidas. Perceba que, para k = 30, o programa KDP1 é cerca de 30% mais rápido, na média, que o programa KDP2. Esse, por sua vez, é cerca de 295%, mais rápido, na média, que o programa KDP4. Quando k = 300, o programa KDP1 é cerca de 57% mais rápido que o programa KDP2, na média, e o programa KDP2 é cerca de 360% mais rápido que o programa KDP4, na média. Quando k = 600, o programa KDP1 é cerca de 59% mais rápido que o programa KDP2, que é cerca de 380% mais rápido que o programa KDP4. Assim, foi selecionado os programas KDP1 e KDP2 para fazer parte do sistema WEB.

Foi percebido que o uso médio de memória dos programas nos testes com dados reais é semelhante aos testes com dados artificiais, ou seja, praticamente idênticos entre os programas KDP1 e KDP2, seguidos pelos programas KDP4 e KDP5.

5. Desenvolvimento do sistema

Existem várias abordagens e estratégias diferentes para desenvolvimento de sistemas, que dependem muito do projeto em si e da equipe disponível. Neste projeto optou-se pela me-

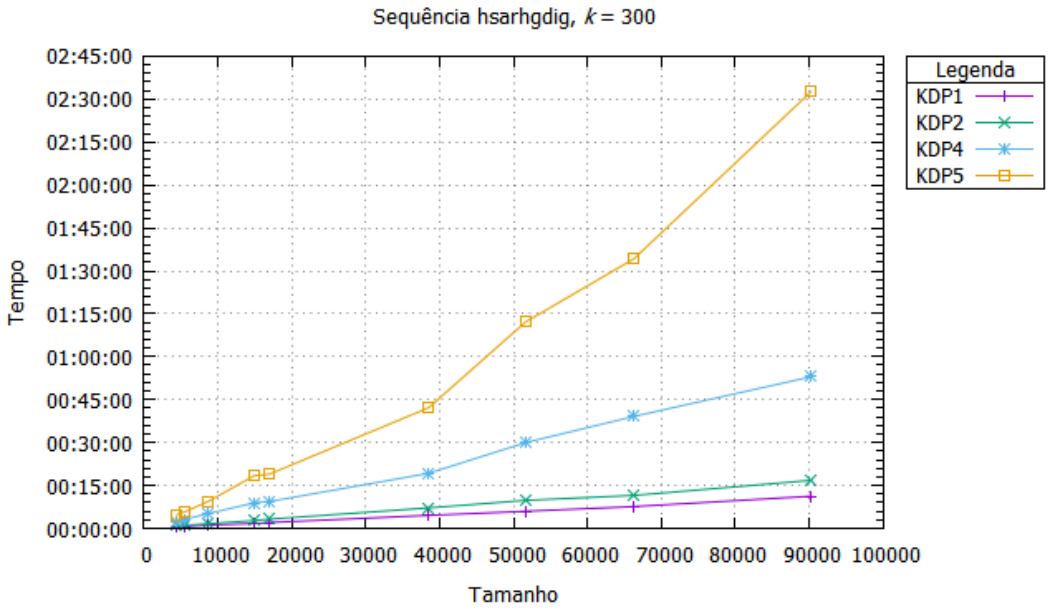

Figura 5. Gráfico de tempo de execução da sequência hsarhgdig, $k = 300$.

todologia ágil de desenvolvimento, que pode ser definida como um conjunto de processos de desenvolvimento de *software* visando simplificar as etapas, com entregas frequentes de *software* funcional e foco na iteração entre indivíduos [Pressman and Maxim 2016]. Isso resulta em uma documentação sucinta, além de etapas do projeto desenvolvidas concorrentemente pela mesma pessoa, que assume diferentes papéis. Foi separado o desenvolvimento do sistema em três etapas principais: projeto de *software*, desenvolvimento e implantação. Decidimos o nome da aplicação como **Web Selection Specific Segment** ou apenas **w3s**, uma alusão ao seu objetivo e a plataforma escolhida.

5.1. Infraestrutura computacional

Para o desenvolvimento do sistema foi utilizado as seguintes ferramentas:

- A ferramenta CASE *Visual Paradgma*, versão 10, para a diagramação de modelos;
- O *framework* PLAY [2], versão 1.4.4, utilizando a linguagem de programação JAVA, versão 8, para desenvolver o aplicativo WEB;
- O *PostgreSQL* [3], versão 9.4, para persistência dos dados;
- O *bootstrap* [4], versão 3.3.7, como *framework* visual;
- O repositório público GIT [5] como controlador de versão de código para salvar os arquivos fontes do sistema;
- O IDE *IntelliJ IDEA* [6], versão 2016.3.4, para codificação e implantação;

[2] https://www.playframework.com/download
[3] https://www.postgresql.org/download/
[4] http://getbootstrap.com/
[5] https://github.com/jeandobre/web-select-primer
[6] https://www.jetbrains.com/idea/download/

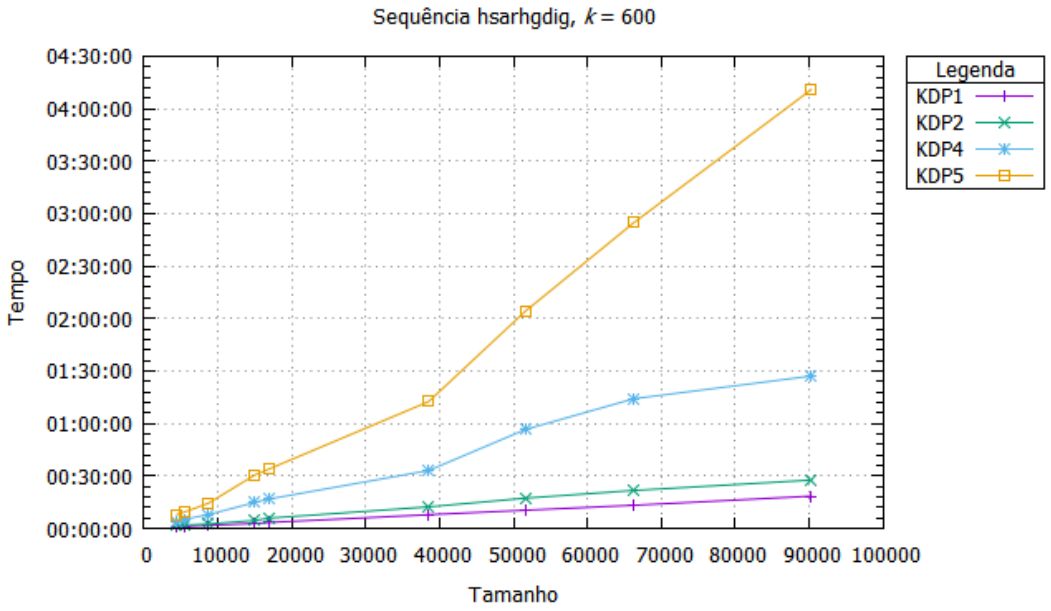

Figura 6. Gráfico de tempo de execução da sequência hsarhgdig, $k = 600$**.**

- O *Bizage Modeler* [7], versão 3.1.0 para modelar o diagrama de atividades do sistema.

5.2. Arquitetura e requisitos

Foi definida, primeiramente, a arquitetura detalhada de interação entre os subsistemas e a apresentada na Figura 7, onde é possível ver a etapa de processamento, que usa diretamente uma implementação do programa KDP e gera um arquivo com os resultados. O banco de dados serve como suporte ao sistema Web, e será utilizado para guardar os dados do processamento. A arquitetura proposta é muito flexível, facilitando sua manutenção e possibilitando a substituição de qualquer componente. Caso surja uma implementação nova e mais eficiente do algoritmo para resolver o PPkD, será possível substituir apenas a camada de processamento, sem necessidade de recodificação.

A tarefa de seleção de segmentos específicos geralmente é guiado pela experiência e conhecimento do biólogo sobre aquela determinada sequência de DNA, de forma que ele pode conhecer uma região específica a ser amplificada na PCR. Porém, na maioria dos casos, é uma tarefa extremamente difícil, que vai além da experiência do biólogo, principalmente quando há mais de duas sequências envolvidas. O sistema w3s vai auxiliá-los nessa tarefa, pois vai entregar uma lista de ocorrências de segmento específico, computadas de forma exata.

Perceba que o w3s não substitui ferramentas computacionais já utilizadas por biólogos. Pelo contrário, vai precisar do suporte delas, pois as ocorrências de segmento específico deverão ser submetidas à uma ferramenta de desenho de *primers*. O sistema w3s é uma ferramenta computacional especializada e auxiliará apenas na tarefa de seleção

[7] http://www.bizagi.com/pt/produtos/bpm-suite/modeler

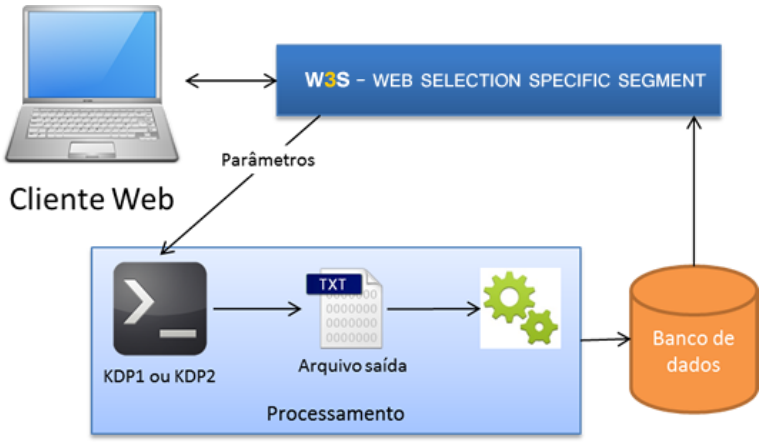

Figura 7. A arquitetura detalhada dos sistemas.

de segmentos específicos. A Figura 8 apresenta o diagrama de atividades com as principais funcionalidades que usuários necessitam do sistema.

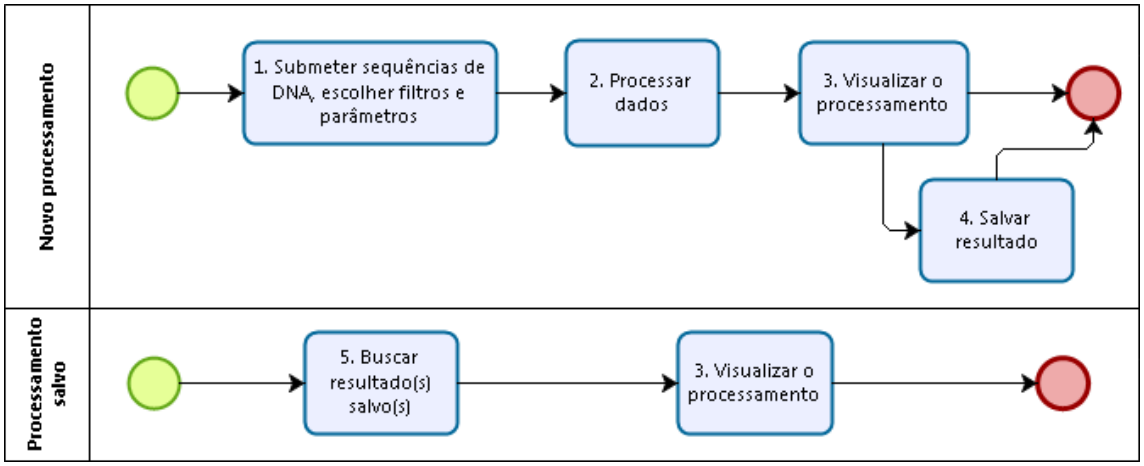

Figura 8. O diagrama de atividades mostrando os fluxos de atividades que devem ser suportados pelo sistema.

Esse diagrama apresenta dois fluxos: novo processamento e processamento salvo. O primeiro é o fluxo principal utilizado na primeira execução do sistema para obtenção dos resultados. O segundo é um fluxo alternativo, e só ocorre quando o usuário salvar os resultados oriundos de uma primeira execução do sistema, para uma busca futura por eles.

Os requisitos foram levantados com a ajuda de um biólogo que trabalha diretamente na área de atuação do sistema. A lista dos principais requisitos do sistema é apresentada a seguir.

1. Submeter sequências de DNA, escolher filtros e parâmetros
    1.1. Na tela inicial do sistema, o usuário carrega arquivos de DNA, ou digita/copia sequências de texto de DNA, no formado FASTA ou texto

contínuo, escolhe os parâmetros e filtros, seleciona um dos programas KDP1 ou KDP2, e submete os dados para processamento;
   i. O usuário pode adicionar mais de uma sequência para comparação com a sequência alvo, ou seja, o sistema deve permitir múltiplas sequências de comparação;
   ii. O usuário pode marcar a opção para ver a maior/menor ocorrência;
   iii. O usuário pode filtrar o tipo de processamento, escolhendo processar todas as posições da sequência alvo ou apenas um determinado intervalo;
   iv. O usuário pode filtrar o resultado mostrando apenas as ocorrências com um certo tamanho de caracteres;
   v. O usuário pode filtrar o resultado mostrando apenas as ocorrências com uma certa distância entre elas.

1.2. O sistema recebe os dados e arquivos de DNA, e carrega os arquivos para o servidor de dados.

2. Processar os dados;
   2.1. O sistema deve fazer a validação dos dados submetidos;
   2.2. O sistema deve converter arquivos FASTA para arquivos de texto;
   2.3. O sistema deve processar os dados usando um dos programas selecionados pelo usuário;
   2.4. O sistema deve guardar os resultados obtidos em banco de dados.
3. Visualizar o processamento;
   3.1. O usuário visualiza os dados do processamento e todas as ocorrências em formato de barras/sequência;
   3.2. O usuário pode visualizar os dados do processamento e todas as ocorrências em forma de lista de dados;
   3.3. O usuário pode copiar uma ocorrência de *primer*.
4. Salvar resultado;
   4.1. Se decidir por salvar o resultado do processamento efetuado, o usuário informa os dados de identificação do processamento e salva o resultado.
5. Buscar resultado(s) salvo(s).
   5.1. Se decidir por buscar o processamento salvo, o usuário informa os dados de identificação do processamento;
   5.2. O sistema lista os processamentos salvos com base nos dados informados pelo usuário;
   5.3. O usuário seleciona um processamento para visualizar seus resultados e o sistema executa o requisito 3.

A seguir é apresentado detalhadamente dois requisitos solicitados pelo usuário e as soluções encontradas parar explorar os resultados decorrentes da solução do PPkD. Para isso, redefinimos a sequência α e a sequência β, dados como entrada do problema, como sequência alvo e sequência de comparação, respectivamente.

No requisito 1.1.i, o sistema deve permitir que o usuário carregue mais de uma sequência de comparação, ou seja, que ele compare a sequência alvo com mais de uma sequência de comparação. Mais detalhadamente, seja B um conjunto com 2 sequências. Neste caso, é necessário comparar a sequência alvo com B_1 e com B_2.

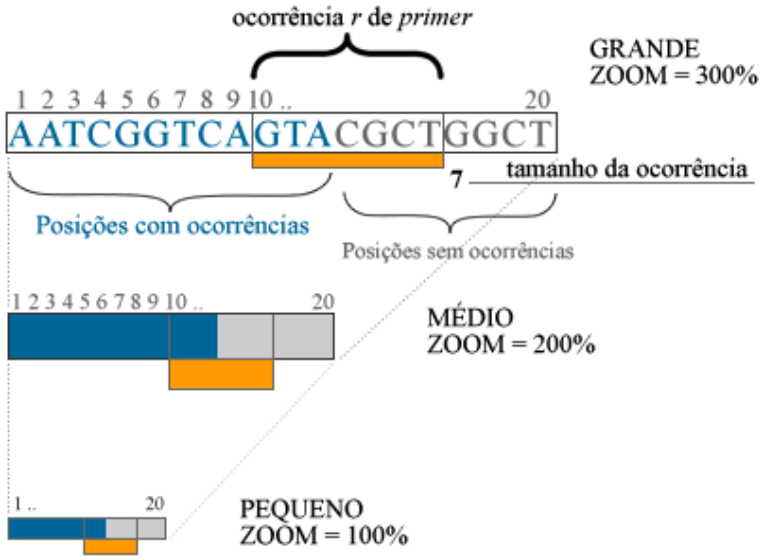

Figura 9. O projeto do componente visual de apresentação da sequência alvo.

Para suportar a funcionalidade mencionada, deve ser executado o algoritmo que resolve o PPkD para cada sequência do conjunto B e guardado a lista de resultados. Após o processamento, é necessário selecionar, para cada posição j da sequência α, a maior ocorrência nessa posição, em cada lista de resultado, desde que essa posição tenha ocorrência em todos os resultados. Se não houver uma ocorrência para a posição j da sequência alvo em todos os resultados, então é possivel afirmar que não há uma ocorrência que inicia na posição j e que tenha k diferenças com relação a todas as sequências do conjunto B.

Para exemplificar, suponha o processamento envolvendo uma determinada sequência alvo e um conjunto B de três sequências. Seja r_1 a lista de ocorrências resultante da comparação da sequência alvo com B_1, r_2 a lista resultante da comparação da sequência alvo com B_2 e r_3 a lista resultante da comparação da sequência alvo com B_3. Usando a ideia apresentada acima, é possível preencher uma lista lst com os valores oriundos dos resultados de cada uma das execuções, representados na Tabela 4. Em negrito estão os valores escolhidos para compor a lista lst. O valor -1 representa uma não ocorrência. Perceba que na posição 1, o maior valor, 18, está na posição 1 de $r3$, na posição 2, o maior valor, 12, está na posição 2 de $r2$, na posição 3, o maior valor, 19, está na posição 3 de $r1$ e a partir da posição 4 não há mais ocorrências.

A demonstração da correção do algoritmo é simples. Ainda utilizando a Tabela 4 perceba que, escolhendo como resultado final o valor 15, que corresponde ao tamanho do menor segmento começando na posição 1 da sequência alvo com k diferenças em relação à sequência B_1, ele não corresponde a uma solução do problema ao considerarmos a sequência B_3. Esse segmento não possui pelo menos k diferenças com relação a B_3, pois o menor segmento iniciando na posição 1 com k diferença em relação a B_3 tem tamanho 18. Com isso, selecionar o tamanho da maior ocorrência para a posição j da sequência

Tabela 4. Exemplo de ocorrências de múltiplos resultados, onde cada coluna representa uma posição j de α e cada linha um resultado da comparação de α com uma sequência do conjunto B.

	1	2	3	4	5	6	7	8
$r1$	15	11	**19**	19	18	19	**-1**	**-1**
$r2$	16	**12**	15	**-1**	**-1**	**-1**	-1	-1
$r3$	**18**	11	17	16	-1	-1	-1	-1
lst	18	12	19	-1	-1	-1	-1	-1

alvo garante que todos os outros segmentos tenham pelo menos k diferenças com relação a qualquer sequência do conjunto B.

Um algoritmo para resolver essa variação do PPkD, utilizando uma das abordagens já discutidas nos capítulos anteriores, tomaria tempo $O(l(abd) + o)$, onde l representa a quantidade de sequências do conjunto B, abd o tempo da abordagem aplicada e o o maior número de ocorrências. Utilizando a abordagem 1, a complexidade é $O(l(m^2n)+o)$ e utilizando a abordagem 2, a complexidade é $O(l(km^2n) + o)$, onde m é o tamanho da sequência alvo, n é o tamanho de cada sequência em B e k é a quantidade de diferenças.

Dado a sua importância no contexto do sistema, o requisito 3.1 merece uma explicação mais detalhada. Visando facilitar a visualização dos resultados oriundos do processamento e sabendo que pode haver muitas ocorrências para uma sequência alvo, estabelecemos como requisito adicional o desenvolvimento de um componente que apresenta a sequência alvo em formato de barra e permite visualizar cada candidato ao selecionar a posição inicial daquela ocorrência. De forma mais detalhada, o componente inclui os seguintes requisitos:

- Apresentar a sequência alvo em forma de barra, com opção de rolagem horizontal, quando a sequência extrapolar o tamanho da largura da tela;
- Cada posição da sequência alvo que tiver uma ocorrência deve ser marcada com uma cor diferente da sequência;
- Ao clicar na posição marcada de cor diferente, o componente deve mostrar o tamanho da ocorrência e o segmento para esta posição;
- Permitir a visualização do resultado em três tamanhos, pequeno, médio e grande, onde cada letra da sequência, no tamanho grande, corresponde a 10 *pixels*[8] de largura na tela, em tamanho médio, corresponde a 2 *pixels* e em tamanho pequeno, a 1 pixel.

Um esquema visual, para facilitar o entendimento e compreensão das funcionalidades do componente, é apresentado na Figura 9.

Como o sistema tem o requisito de salvar o resultado, para posterior recuperação, é fundamental o uso de banco de dados. A Figura 10 apresenta o modelo de entidade e relacionamento que contempla todos os requisitos levantados para este sistema. A entidade *configuracoes* serve apenas para armazenar dados de configuração do sistema e não se relaciona a nenhuma outra entidade.

[8]Um pixel é o menor elemento num dispositivo de exibição.

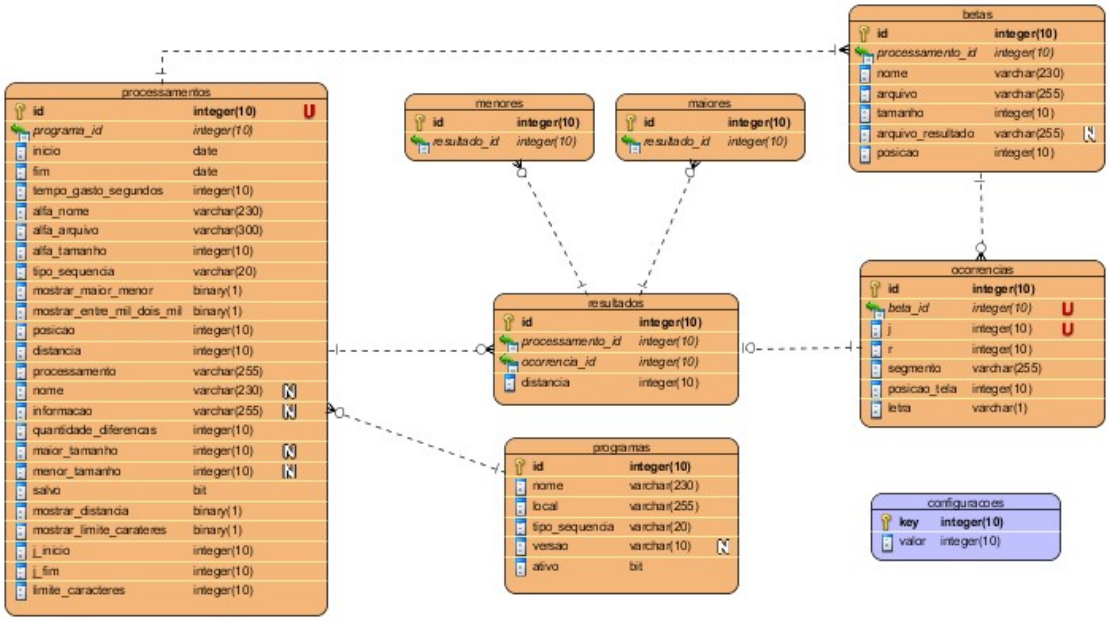

Figura 10. Diagrama de entidade e relacionamento do sistema.

5.3. Desenvolvimento

Como o sistema w3s é basicamente uma *interface* para uso de um dos programas KDP1 e KDP2, desenvolvidos em C++, os requisitos funcionais detalhados, tem como base o resultado gerado pela execução de qualquer um desses programas. Assim, o sistema desenvolvido em JAVA executa os algoritmos desenvolvidos em C++, passando os devidos parâmetros para essa execução e, ao final, abre o arquivo de resultados gerado. Perceba que a integração entre o sistema e o programa acontece aqui, onde a comunicação entre eles ocorre primeiramente, por parâmetros usados na execução dos algoritmos em *prompt* de comando e, posteriormente, ao termino da execução, pelo arquivo de resultados, escrito no padrão: posição j de α; tamanho do segmento; segmento. Para exemplificar, suponha uma execução com os seguintes parâmetros: *-a* CCCGGCCC *-b* CCCGTGCCC *-k* 1 *-sf* teste.txt. O resultado escrito no arquivo teste.txt será:

$$
\begin{aligned}
&j; tamanho; sequencia\\
&0; 5; CCCGG\\
&1; 4; CCGG\\
&2; 3; CGG\\
&3; 2; GG
\end{aligned}
$$

A partir desse arquivo, fazemos o carregamento das ocorrências para uma lista, chamada aqui de lst. Perceba que lst terá tamanho máximo $m-k$, dado que m é tamanho da sequência alvo e k a quantidade de diferenças. No exemplo dado anteriormente, lst tem 4 elementos, onde a posição 0 do arquivo corresponde à posição 1 da sequência α

no sistema, que apresenta corretamente ao usuário a sequência a partir da posição 1. A primeira linha do arquivo é o cabeçalho e será ignorada.

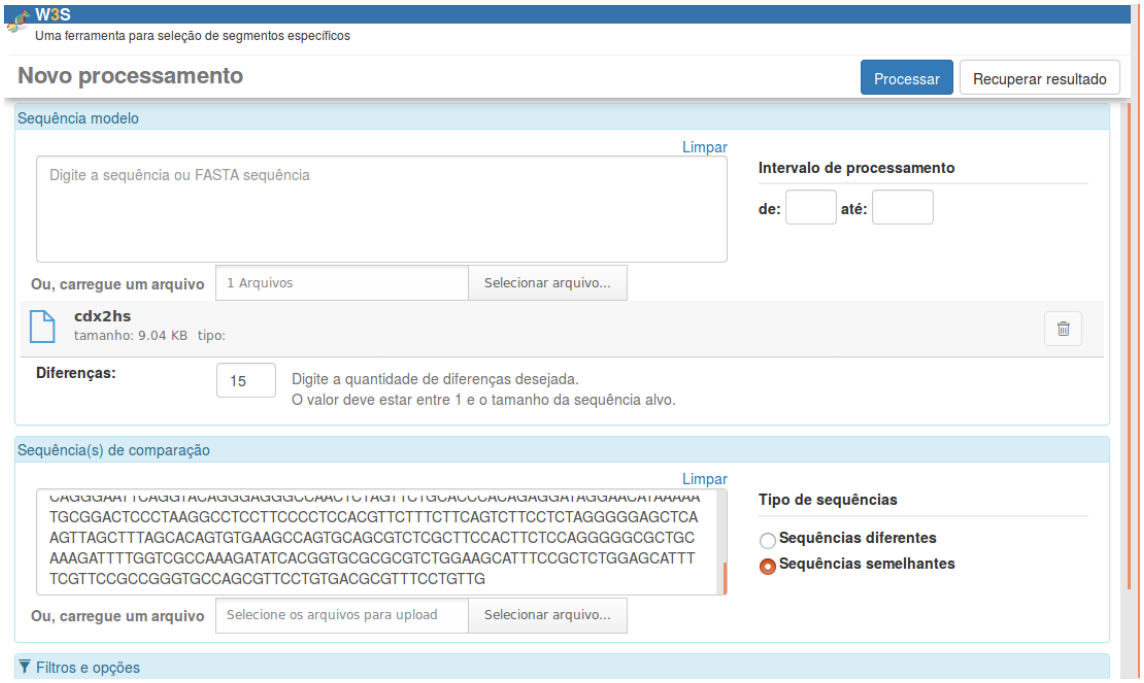

Figura 11. Tela inicial do sistema.

Na Figura 11 é apresentado a tela inicial do sistema, referente as atividades de submeter sequências de DNA, escolher filtros e parâmetros. Na Figura 12 é apresentado a tela de visualizar o processamento, mostrando o resultado de um processamento realizado. Perceba nela, o componente visual de apresentação das ocorrências e a opção de salvar o resultado para consulta posterior.

5.4. Implantação

Foi separada as funcionalidades do sistema em versões usuais de *software*, ou seja, a cada versão, o sistema deve ser passível de uso, mesmo com um conjunto pequeno de requisitos implementados. Assim, dividimos a lista de requisitos em cinco versões do sistema. Ao final da quarta versão, foi realizado um teste de aceitação juntamente a todos os envolvidos do projeto do sistema. Como o sistema estava com quase todas as funcionalidades implementadas, testadas e operantes, foi possível que o usuário final do sistema executasse um procedimento completo de seleção de segmentos específicos. Isso foi de fundamental importância, visto que o usuário detectou algumas funcionalidades necessárias que o sistema não disponibilizava e pôde validar outras funcionalidades do sistema, mostrando pontos de acerto e pontos a serem melhorados.

Ao final, o código fonte do sistema foi carregado para o repositório GIT nos endereços https://github.com/jeandobre/web-select-primer e https://github.com/jeandobre/k-difference-prime, onde encontram-se disponíveis para implantação e uso.

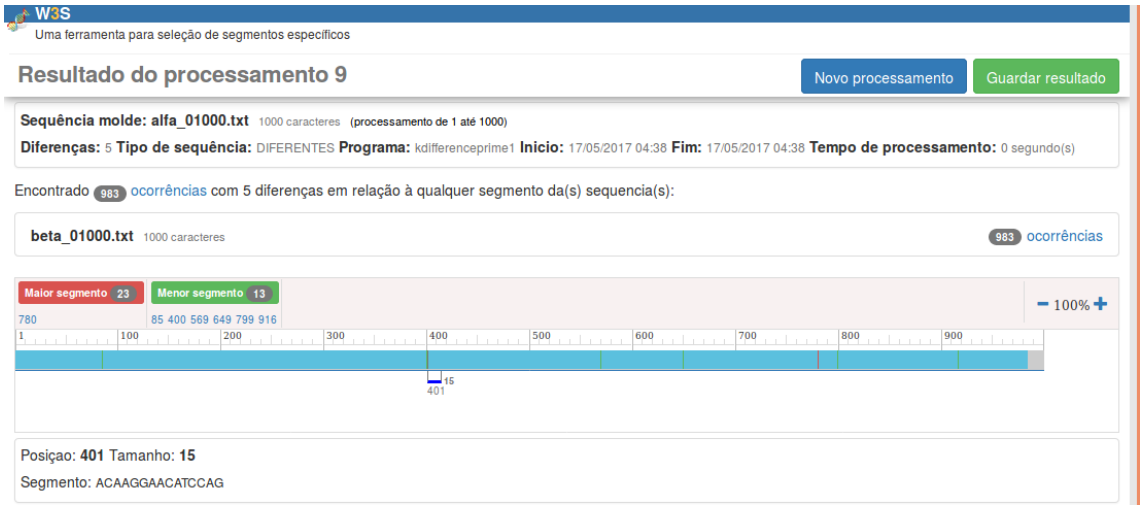

Figura 12. Tela de visualização dos resultados.

6. Conclusão

Neste trabalho abordamos o problema biológico de seleção de segmentos específicos através de algumas soluções existentes para o problema computacional denominado Problema do *Primer* com k Diferenças e depois realizamos uma pesquisa na literatura buscando por algoritmos que resolvem o problema computacional. Propomos então implementar os algoritmos correspondentes, submetendo-os a testes com dados artificiais e reais, e selecionar o melhor deles para o desenvolvimento de uma aplicação WEB cliente/servidor afim de auxilar os biólogos na busca por segmentos específicos para as mais diversas atividades em laboratórios.

Este artigo apresentou sucintamente os resultados obtidos dos testes realizados com dados artificiais e reais durante a pesquisa. Os testes serviram para avaliar, além das abordagens, as diferenças entre as estruturas de dados que servem para a determinação da maior extensão comum entre duas sequências. Com base nos testes, escolhemos os dois melhores programas, KDP1 e KDP2, para serem utilizados na aplicação WEB.

Detalhamos o desenvolvimento da aplicação WEB, que tem o objetivo de auxiliar os usuários na busca por segmentos específicos, com uma arquitetura que permite a evolução do sistema e integração de novas abordagens de forma fácil. Acreditamos que possa se tornar uma ferramenta para auxiliar no processo de seleção de segmentos específicos para as mais diversas atuações da PCR.

Verificamos ao longo do trabalho que existem várias heurísticas (Método BYP em [Baeza-Yates and Perleberg 1992], Método Chang-Lawler em [Chang and Lawler 1990], Método da iteração Múltipla em [Pevzner and Waterman 1995] e Método sublinear de Myers em [Myers 1994]) para o Problema do Casamento Inexato com até k Diferenças que empregam novas abordagens e são, teoricamente, mais rápidos que os algoritmos que foram aqui implementados. Nesse sentido, como trabalhos futuros prevemos a implementação e avaliação dessas heurísticas e também o desenvolvimento de novas soluções para o Problema do *Primer* com k Diferenças.

Uma outra possibilidade de trabalho futuro é a implementação de algoritmos que resolvem o Problema do *Primer* com k Diferenças utilizando outras medidas de comparação de sequências que não a Distância de Edição, como a Distância de *Hamming*. A implementação de tais algoritmos, e a adição deles à ferramenta computacional, pode permitir aos usuários a escolha da medida desejada para comparação das sequências, de acordo com a aplicação e as sequências envolvidas. Na realidade, esse trabalho já está em curso, com um resumo estendido já escrito e aceito em conferência nacional [Rocha et al. 2018], com o próximo passo consistindo de um estudo da viabilidade da inserção do novo algoritmo no sistema.

Referências

[Abd-Elsalam 2003] Abd-Elsalam, K. A. (2003). Bioinformatic tools and guideline for pcr primer design. *african Journal of biotechnology*, 2(5):91–95.

[Alberts et al. 2009] Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P. (2009). *Biologia Molecular da Célula*. Artmed Editora.

[Baeza-Yates and Perleberg 1992] Baeza-Yates, R. A. and Perleberg, C. H. (1992). Fast and practical approximate string matching. In *Annual Symposium on Combinatorial Pattern Matching*, pages 185–192. Springer.

[Chang and Lawler 1990] Chang, W. I. and Lawler, E. L. (1990). Approximate string matching in sublinear expected time. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 116–124. IEEE.

[Cormen et al. 2012] Cormen, T., Leiserson, C., Rivest, R. L., and Stein, C. (2012). *Algoritmos: Teoria e Prática*. Campus, Rio de Janeiro, 3 edition.

[de Castro Miranda et al. 2005] de Castro Miranda, R. C., Ayala-Rincón, M., and Solon, L. (2005). Modifications of the Landau-Vishkin Algorithm Computing Longest Common Extensions via Suffix Arrays and Efficient RMQ Computations.

[de Lima 2013] de Lima, L. I. S. (2013). O Problema do Alinhamento de Segmentos. Master's thesis, FACOM-UFMS.

[Deitel and Deitel 2005] Deitel, M. and Deitel, P. (2005). *C++ como programar. 5a edição*. Editora Pearson, São Paulo.

[Gusfield 1997] Gusfield, D. (1997). *Algoritms on strings, trees, and sequences: Computer Science and Computational Biology*. Cambridge University Press.

[Hennessy and Patterson 2014] Hennessy, J. L. and Patterson, D. A. (2014). *Organização e Projeto de Computadores: a interface hardware/software*, volume 4. Elsevier Brasil.

[Huynh et al. 2006] Huynh, T. N., Hon, W.-K., Lam, T.-W., and Sung, W.-K. (2006). Approximate string matching using compressed suffix arrays. *Theoretical Computer Science*, 352(1):240–249.

[Ito et al. 1995] Ito, M., Shimizu, K., Nakanishi, M., and hashimoto, A. (1995). Polynomial-time Algorithms for Computing Characteristic Strings. *Systems and Computers in Japan*, 26(3):30–38.

[Landau and Vishkin] Landau, G. M. and Vishkin, U. Fast parallel and serial approximate string matching. *Journal of algorithms*, 10(2):157–169.

[Landau and Vishkin 1989] Landau, G. M. and Vishkin, U. (1989). Fast parallel and serial approximate string matching. *Journal of Algorithms*, 10(2):157–169.

[Loudon 2000] Loudon, K. (2000). *Dominando algoritmos com C*. Editora Ciencia Moderna Ltda, São Paulo.

[Marshall 2004] Marshall, O. J. (2004). Perlprimer: cross-platform, graphical primer design for standard, bisulphite and real-time pcr. *Bioinformatics*, 20(15):2471–2472.

[Masek and Paterson 1980] Masek, W. J. and Paterson, M. S. (1980). A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31.

[Montera and Nicoletti 2008] Montera, L. and Nicoletti, M. (2008). The pcr primer design as a metaheuristic search process. *Artificial Intelligence and Soft Computing–ICAISC 2008*, pages 963–973.

[Mullis et al. 1995] Mullis, K. B., Ferré, F., Gibbs, R., and Morley, B. J. (1995). PCR-the polymerase chain reaction. *Trends in Genetics*, 11(6):249–249.

[Mullis et al. 1986] Mullis, K. B. F. F., Faloona, F., Scharf, S., Saiki, R., Horn, G., and Erlich, H. (1986). Specific enzymatic amplification of DNA in vitro: the polymerase chain reaction. In *Cold Spring Harbor symposia on quantitative biology*, volume 51, pages 263–273, NY. Cold Spring Harbor Laboratory Press.

[Myers 1994] Myers, E. W. (1994). A sublinear algorithm for approximate keyword searching. *Algorithmica*, 12(4-5):345–374.

[Pevzner and Waterman 1995] Pevzner, P. A. and Waterman, M. S. (1995). Multiple filtration and approximate pattern matching. *Algorithmica*, 13(1-2):135–154.

[Pressman and Maxim 2016] Pressman, R. and Maxim, B. (2016). *Engenharia de Software-8ª Edição*. McGraw Hill Brasil.

[Ribeiro 2009] Ribeiro, M. C. M. (2009). *Genética Molecular*. Universidade Federal de Santa Catarina. Biologia/EaD/UFSC, Florianópolis.

[Ristad and Yianilos 1998] Ristad, E. S. and Yianilos, P. N. (1998). Learning string-edit distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(5):522–532.

[Rocha et al. 2018] Rocha, L. B., Adi, S. S., and Elói, A. (2018). Specific substring problem: an application in bioinformatics. In *Simpósio Brasileiro de Bioinformática*.

[Saikkonen and Soisalon-Soininen 2008] Saikkonen, R. and Soisalon-Soininen, E. (2008). Cache-sensitive memory layout for binary trees. In *Fifth IFIP International Conference On Theoretical Computer Science–TCS 2008*, pages 241–255. Springer.

[Sutter 2006] Sutter, H. (2006). *Programação avançada em C++*. Pearson-Mahron Books, São Paulo.

[Ukkonen 1995] Ukkonen, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14(3):249–260.

[Välimäki et al. 2009] Välimäki, N., Mäkinen, V., Gerlach, W., and Dixit, K. (2009). Engineering a compressed suffix tree implementation. *Journal of Experimental Algorithmics (JEA)*, 14:2.

[Ye et al. 2012] Ye, J., Coulouris, G., Zaretskaya, I., Cutcutache, I., Rozen, S., and Madden, T. L. (2012). Primer-blast: A tool to design target-specific primers for polymerase chain reaction. *BMC Bioinformatics*, 13(1):134.