Estimação da Distância de Reversão de Genomas Baseada em Técnicas de Aprendizado de Máquina

Nathalia Menini, Sergio Z. Arnosti e Jorge da Silva

Instituto de Computação da Unicamp,
Av. Albert Einstein, 1251 - Cidade Universitária, Campinas - SP
{nathmenini,serza.arnosti,jorge.inatel}@gmail.com

Resumo A Biologia Computacional trouxe muitos avanços na tarefa de comparar dois genomas para se estudar as relações e a evolução entre os genes como, por exemplo, através da distância de rearranjo. Neste trabalho, propõe-se a utilização de técnicas de Aprendizado de Máquina, como a Regressão Linear e Redes Neurais, com o objetivo de estimar a distância de reversão para permutações sem orientação de genes baseando-se em *features* extraídas dessas. O treinamento dos algoritmos considerou permutações de tamanhos reduzidos e, posteriormente, verificou o desempenho em permutações maiores. Esses métodos foram comparados com algoritmos já estabelecidos na literatura como, por exemplo, o *Selection Sort Using Reversals* e *Greedy Reversal Sort*, bem como com a distância exata de reversão. Os resultados mostram que a utilização de Regressão Linear apresenta resultados melhores ou tão bons quanto o algoritmo *Greedy Reversal Sort* (2-aproximação). Além disso, para as permutações de tamanho maior que 10, a Regressão Linear e Rede Neural se mostraram abordagens promissoras para a estimação de distância de reversão.

Keywords: genes, reversão, aprendizado de máquina, permutação, estimação, distância

1 Introdução

Comparar dois genomas é uma tarefa fundamental para se estudar as relações e a evolução entre os genes [1,2]. Nesse caminho, a Biologia Computacional tem se apresentado como uma ótima aliada para os pesquisadores da área, tornando possível descobrir relações entre os genes que, para a percepção do ser humano, poderia ser muito difícil de detectar [2].

Nesse contexto, o conceito de *rearranjo de genomas* se torna essencial. O rearranjo de genomas é uma mutação que ocorre nos genomas mitocondriais [3], de modo que a ordem dos genes está em constante rearranjo. Desse modo, através da estimação da distância de rearranjo entre dois genes, a relação entre eles pode ser estimada [4].

Um dos eventos de mutação mais comuns em genomas, conhecido como *reversão*, ocorre quando um fragmento do filamento de DNA é revertido. Por outro

lado, se dois fragmentos de DNA trocam de posições durante o processo de replicação (mas não sofrem reversão), temos um evento de mutação chamado de *transposição* [2].

A área da Filogenia, que estuda a história evolutiva das relações entre espécies, depende fortemente do Principio da Parcimônia [5]. Basicamente, dado um conjunto de possíveis explicações para um fato, a explicação mais simples é a mais provável de estar correta. Como as mutações são relativamente raras de acontecer, quando os pesquisadores constroem uma árvore filogenética, eles tentam fazer com que as espécies tenham o menor número de antecessores possíveis, o que significa que é mais provável que duas espécies que possuem uma mesma característica tenham evoluído de um mesmo ancestral comum do que essa mesma característica ter evoluído duas vezes de espécies diferentes [2].

Entretanto, determinar o número de mutações necessárias para um genoma se transformar em outro não é um problema fácil de se resolver [2]. Mais especificamente, considerando a permutação identidade $\iota = (1, 2, ..., n)$ de tamanho n como o genoma original, dada qualquer permutação π de tamanho n, o problema consiste em construir um modelo capaz de calcular a distância de rearranjo de π a ι, em que são permitidas apenas operações de reversão.

Uma outra possível representação de genomas é utilizando a *orientação* dos genes. Nesse caso, representamos o genoma com os símbolos + ou − para cada gene como, por exemplo, na permutação $(+1, -2, \cdots, -n)$. Na literatura, essa abordagem é conhecida como *permutações com orientação de genes conhecida*. Por outro lado, se não temos informações sobre a orientação dos genes, os sinais são omitidos e a abordagem passa a ser chamada de *permutação sem orientação de genes conhecida*. Este trabalho utilizará apenas a segunda abordagem, ou seja, sem orientação de genes [6].

Novamente, determinar o número mínimo de reversões necessárias para um genoma se transformar em outro, considerando o contexto de permutações sem orientação, não é um problema fácil de ser resolvido, uma vez que foi provado ser um problema pertencente a classe NP-difícil [7].

Muitas abordagens já foram propostas para atacar esse problema como, por exemplo, Kececioglu e Sankoff em [8] propuseram um algoritmo de 2-aproximação que remove *breakpoints* por reversão (o conceito de *breakpoint* será descrito posteriormente), Bafna e Pevzner em [9] criaram o algoritmo de 1.75-aproximação utilizando grafos de reversão e Christie [10] apresentou um algoritmo com fator de aproximação 1.5. O atual estado da arte foi desenvolvido por Berman *et al.* [11], que construiu um algoritmo de 1.375-aproximação. Além disso, Berman e Karpinski [12], mostraram que o problema da distância de reversão sem orientação é MAX-SNP-difícil. Abordagens utilizando algoritmos genéticos e técnicas de Aprendizado de Máquina (AM) também já foram propostas por Auyeung e Abraham em [13] e da Silva *et al.* em [2], respectivamente.

Entretanto, a maioria dos algoritmos presentes na literatura buscam, além de encontrar a distância de reversão, encontrar também as reversões que transformam uma permutação em outra. Neste trabalho, propõe-se a utilização de técnicas de AM, como a Regressão Linear e Redes Neurais, com o objetivo de

estimar a distância de reversão baseando-se em *features* obtidas a partir da permutação, no contexto de permutações sem orientação.

Em um primeiro momento, o foco será estimar a distância e não obter as reversões em si, uma vez que, em áreas como a filogenia, é fundamental obter uma métrica de distância entre objetos [5] e, quanto mais precisa for a métrica, melhor será a análise. Além disso, devido a dificuldade de obter dados da distância de reversão exata para permutações grandes, serão utilizados como dados para treinamento dos algoritmos propostos um *dataset* com permutações de tamanhos reduzidos e, posteriormente, será verificado o desempenho em permutações maiores.

2 Definições e Conceitos Básicos

Nessa seção, será explicada a operação de reversão no contexto de permutações sem orientação e, também, alguns conceitos utilizados para o protocolo de extração das *features* utilizadas nos algoritmos de AM.

Os genomas podem ser representados por permutações (n-tuplas, em que n é a quantidade de genes), em que cada gene é representado por um número e que todos os genes são diferentes um dos outros. Ou seja, uma permutação π de tamanho n é representada por $\pi = (\pi_1\ \pi_2\ \cdots\ \pi_n)$, com $\pi_i \in \{1, 2, \cdots n\}$ e $\pi_i \neq \pi_j \iff i \neq j$. A *permutação identidade*, como mencionada anteriormente, é definida como $\iota = (1\ 2\ \cdots n)$. A *composição* entre duas permutações π e σ é dada por $\pi\sigma = (\pi_{\sigma_1}\ \pi_{\sigma_2} \cdots \pi_{\sigma_3})$. Definimos também o *inverso* de uma permutação π como π^{-1}, em que $\pi^{-1}\pi = \iota$.

A *permutação extendida* de uma permutação $\pi = (\pi_1\ \pi_2\ \cdots\ \pi_n)$, denotada por π_e, é definida como $\pi_e = (0\ \pi_1\ \pi_2\ \cdots\ \pi_n\ n+1)$. A definição de permutação extendida é necessária para a descrição de operações que serão abordadas no decorrer do trabalho.

Dadas duas permutações π e σ e um conjunto de operações $\Sigma = \{\rho_1, \rho_2, \cdots, \rho_k\}$, o problema de transformar π em σ consiste em obter no menor número possível de operações de Σ aplicada em π, tal que π seja transformada em σ. O número de operações necessárias é chamado de distância entre π e σ, representado por $d(\pi, \sigma)$. Para fins de notação, será utilizado $d(\pi, \iota) = d(\pi)$.

O problema de encontrar a distância entre as permutações π e σ é equivalente ao problema de encontrar a distância entre alguma permutação π' e ι, em que $\pi' = \sigma^{-1}\pi$. Ou seja, se queremos encontrar a distância entre π e σ, equivale ao problema de ordenação de $\sigma^{-1}\pi$. Por exemplo, se considerarmos $\pi = (5\ 3\ 1\ 6\ 7\ 4\ 2)$ e $\sigma = (6\ 3\ 2\ 7\ 5\ 1\ 4)$, temos que $\sigma^{-1} = (6\ 3\ 2\ 7\ 5\ 1\ 4)$, o que faz com que $\pi' = \sigma^{-1}\pi = (5\ 2\ 6\ 1\ 4\ 7\ 3)$. Então, a distância entre π e σ equivale a distância de ordenação de π'.

2.1 Reversões

Uma operação de reversão $\rho_r(i, j)$, com $1 \leq i < j \leq n$ reverte um fragmento da permutação, ou seja, $\rho_r(i, j)$ aplicado em $\pi = (\pi_1\ \cdots\ \pi_{i-1}\ \underline{\pi_i \cdots \pi_j}\ \pi_{j+1}\ \cdots\ \pi_n)$ gera $\pi\rho_r(i, j) = (\pi_1\ \cdots\ \pi_{i-1}\ \underline{\pi_j \cdots \pi_i}\ \pi_{j+1}\ \cdots\ \pi_n)$.

Considere, por exemplo, $\pi = (1\ 4\ 3\ 2\ 6\ 7\ 5)$ e a operação de reversão $\rho_r(2,4)$. O resultado após aplicar essa operação é $\pi\rho_r(2,4) = (1\ \underline{2\ 3\ 4}\ 6\ 7\ 5)$.

2.2 *Breakpoints*

No contexto de operações de reversão, dada uma permutação estendida de π, um par de elementos π_i e π_{i+1}, para $0 \leq i \leq n$, é definido como uma adjacência se $|\pi_i - \pi_{i+1}| = 1$. Caso contrário, o par de elementos é chamado de *breakpoint*. A permutação identidade é a única que não possui *breakpoints* e a quantidade de *breakpoints* de uma permutação é denotada por $b(\pi)$ [2,14].

Como exemplo, dada a permutação $\pi = (0\ 1\ 2 \bullet 6\ 5 \bullet 3\ 4 \bullet 7)$, tem-se três *breakpoints* – destacados com o símbolo $\bullet$: um entre os elementos 2 e 6, outro entre os elementos 5 e 3 e, por fim, entre os elementos 4 e 7.

2.3 *Strips*

Strips podem ser definidas como fragmentos de uma permutação que não apresentam *breakpoints*. Mais precisamente, uma *strip* $\pi[i \cdots j]$ é um trecho maximal em π tal que todos os pares (π_k, π_{k+1}) são adjacências, para $i \leq k < j$ [2,14].

Uma *strip* $\pi[i \cdots j]$ é chamada descrescente se e somente se a sequência $\pi_i, \pi_{i+1}, \cdots, \pi_{j-1}, \pi_j$ for descrescente. As *strips unitárias* são sempre definidas como descrescentes, com exceção das *strips* formadas por π_0 e π_{n+1} que são sempre crescentes. Por outro lado, se a sequência for crescente, a *strip* é dita crescente. Considere que o símbolo $\rightarrow$ representa uma *strip* crescente e $\leftarrow$ uma *strip* decrescente, desse modo, temos que as *strips* de π são $\pi = (\overrightarrow{0}\ \overleftarrow{3\ 2}\ \overrightarrow{4\ 5}\ \overleftarrow{1}\ \overrightarrow{6})$ [2,14].

2.4 Maior e Menor *Strip*

Podemos definir como a maior *strip* aquela que, dentre todas, possui a maior quantidade de elementos. Já a menor *strip* podemos definir como aquela que possui a menor quantidade de elementos e que, além disso, ela não é unitária – quando existem *strips* não unitárias. Note que, quando só existem *strips* unitárias, a maior e a menor *strips* terá valor igual a um.

2.5 Ciclos

Os ciclos são um conceito muito importante dentro do estudo da ordenação de permutações. Grande parte das *features* utilizadas no treinamento dos algoritmos de AM estão relacionadas a ciclos.

Desse modo, para definir os ciclos que foram utilizados nesse trabalho, é suficiente definir um *grafo de ciclos alternados* para uma permutação π, denotado por $G(\pi)$, onde $G(\pi) = (V, E_{black} \cup E_{gray})$.

O conjunto (V, E_{black}, E_{gray}) é definido como:

Conjunto de Vértices

$$V = \{+0, -\pi_1, +\pi_1, -\pi_2, +\pi_2, \cdots, -\pi_n, +\pi_n, -(n+1)\}$$

Conjunto de arestas pretas
$E_{black} = \{(-\pi_i, +\pi_{i-1}) \mid 1 \leq i \leq n+1\}$

Conjunto de arestas cinzas
$E_{gray} = \{(+(i-1), -i) \mid 1 \leq i \leq n+1\}$

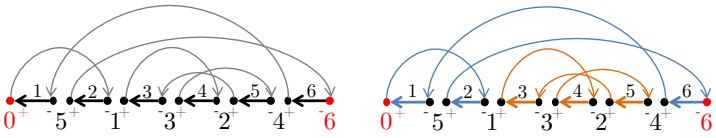
O grafo é chamado alternado, pois ao caminhar pelas arestas alterna-se entre arestas pretas e cinzas, sendo as pretas as que vão definir as características de um ciclo [2,10,15].

Dado um grafo $G(\pi)$ de uma permutação π, as arestas pretas são rotuladas de 1 a $n+1$, dessa maneira um k-ciclo é definido como um ciclo contendo k arestas pretas. Se um ciclo possui um número ímpar de arestas pretas ele é considerado um ciclo ímpar, caso contrário é um ciclo par. Se o ciclo possui apenas uma aresta preta ele é definido como ciclo unitário, e, além disso, a permutação identidade é a única que possui $n+1$ ciclos unitários [15].

Um k-ciclo também pode ser chamado de orientado se as arestas pretas pertencentes a ele são listadas em ordem não-decrescente, caso contrário, são chamados de não-orientados [2,16].

A Figura 1 ilustra um exemplo de *grafo de ciclos alternados* para a permutação $\pi = (5\ 1\ 3\ 2\ 4)$ acrescida dos vértices $\pi_0 = +0$ e $\pi_{(n+1)} = -(n+1) = -6$, onde n é o tamanho da permutação. Na Figura 1a, apresentamos as arestas pretas e cinzas. Já na Figura 1b, temos que a sequência de arestas azuis (6, 1, 2) e também as arestas laranjas (5, 3, 4) representam dois 3-ciclo, que são ímpares e orientados.

Dentro de uma permutação, o total de ciclos é definido como a soma de todos os k-ciclos, sendo o maior ciclo aquele que possui a maior sequência de arestas pretas e o menor aquele que possui a menor sequência, desconsiderando os ciclos unitários. Se existirem apenas ciclos unitários, então o tamanho do maior e menor ciclo é definido como 1.



(a) Arestas pretas e cinzas (b) Ciclos destacados

Figura 1: Grafo de ciclos alternados para a permutação $\pi = (5\ 1\ 3\ 2\ 4)$.

2.6 Algoritmos de Ordenação por Reversão

Utiliza-se uma variação do algoritmo *Selection Sort* para atacar o problema de ordenação por reversão. O Algoritmo 1, chamado de *Selection Sort Using*

Reversals (SSUR), possui um caráter ingênuo, uma vez que realiza sempre o mesmo passo sem se preocupar com outras características da permutação. Além disso, o SSUR não garante a ordenação com o número mínimo de reversões possíveis. Ele inicia pelo elemento 1, verifica a posição atual dele na permutação e faz uma reversão da respectiva posição correta até a posição atual para que o elemento chegue à sua posição correta e, a seguir, repete esses passos para todos os outros elementos até que a permutação esteja ordenada [14].

Por não garantir o menor número de reversões, o SSUR é um algoritmo de aproximação que não garante uma aproximação melhor do que $(n-1)/2$ vezes a solução ótima. Por exemplo, para a permutação $\pi = (5\ 1\ 2\ 3\ 4)$ de tamanho $n = 5$, o algoritmo SSUR realiza 4 reversões para ordenar π, porém são necessárias apenas 2 reversões [14].

Algoritmo 1: *Selection Sort Using Reversals*

Entrada: permutação π, tamanho da permutação n
Saída: número de reversões num_rev
início
  $num\_rev \leftarrow 0$;
  $i \leftarrow 1$;
  repita
    $pos \leftarrow$ posição atual do elemento i na permutação π (*e.g.* $\pi_{pos} = i$);
    se $pos \neq i$ **então**
      $\pi \leftarrow \pi \cdot \rho(i, pos)$;
      $num\_rev \leftarrow num\_rev + 1$;
    $i = i + 1$
  até $\pi = identidade$ *ou* $i = n - 1$;
fim

Outro algoritmo utilizado, um pouco mais inteligente que o descrito anteriormente, é conhecido por *Greedy Reversal Sort* (GRS), e está detalhado em Algoritmo 2. O GRS considera os conceitos de *breakpoints* e *strips* crescentes e decrescentes descritos nas Seções 2.2 e 2.3, respectivamente.

Este algoritmo, por sua vez, possui caráter guloso, pois sempre tenta remover o máximo possível de *breakpoints* a cada passo. Entretanto, também não garante o número mínimo de reversões. O GRS possui garantias matemáticas de que ele remove, em média, um *breakpoint* a cada reversão, ou seja, é capaz de ordenar qualquer permutação π em, no máximo, $b(\pi)$ reversões. Porém, como o limite inferior de *breakpoints* que podem ser removidos é dois a cada reversão, temos que o algoritmo exato não conseguiria ordenar π em menos do que $b(\pi)/2$ reversões [8]. Portanto, o fator de aproximação do GRS é dado por $\frac{b(\pi)}{b(\pi)/2} = 2$.

2.7 Técnicas de Aprendizado de Máquina

Regressão Linear Regressão Linear (RL) é uma das técnicas mais básicas de regressão que visa quantificar os impactos de variáveis explicativas (ou *features*)

Algoritmo 2: *Greedy Reversal Sort*

```
Entrada: permutação π, tamanho da permutação n
Saída: número de reversões num_rev
início
    num_rev ← 0;
    repita
        se π tem uma strip decrescente então
            k ← menor elemento dentre todas as strips decrescentes;
            ρ ← reversão que corta na posição depois de k e depois de k − 1;
            se π · ρ não possui strips decrescentes então
                l ← maior elemento dentre todas as strips decrescentes em π;
                ρ ← reversão que corta na posição antes de l e antes de l + 1;
        senão
            ρ ← reversão que corta os dois primeiros breakpoints;
        π ← π · ρ;
        num_rev ← num_rev + 1
    até π = identidade;
fim
```

em uma variável resposta. O termo *linear* indica a suposição do modelo em que a relação entre a variável resposta e as *features* se dá através de uma combinação linear. Portanto, RL trata-se de um método de aprendizado supervisionado para realizar predições quantitativas.

O modelo de RL múltiplo [17,18] pode ser definido como

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon, \tag{1}$$

em que Y representa a variável resposta, $x = \{x_1, x_2, \cdots, x_n\}$ representam as n *features*, ϵ representa o erro (estocástico) e $\beta = \{\beta_0, \beta_1, \cdots, \beta_n\}$ são os $n + 1$ coeficientes de regressão. A estimação desses coeficientes pode ser feita através do método de Mínimos Quadrados, Máxima Verossimilhança, Decomposição QR, Descida do Gradiente e outros métodos [17,18].

Redes Neurais Em busca de obtermos ainda melhores resultados em termos de predição, é possível utilizar abordagens mais complexas, como é o caso das *Redes Neurais* (RN) [17,18]. Na Figura 2 apresentamos uma arquitetura de uma RN com uma camada de *input*, uma camada escondida (*hidden*) e uma de *output*. Mais precisamente, os círculos destacados em amarelo representam o *bias* (ou intercepto), e $x = \{x_1, \ldots, x_n\}$ (em vermelho) são as n *features* consideradas na análise. Já na camada *hidden* (azul), temos que

$$a^{(2)} = g(X_{m\times(n+1)} W^1_{(n+1)\times h}), \tag{2}$$

em que $g(.)$ é uma função de ativação, X é a matriz com o intercepto e todas as n *features* para os m exemplos, e W^1 é a matriz com os pesos associados a cada aresta que vai dos $n + 1$ neurônios da camada *input* para os h neurônios

na camada *hidden*. Por fim, na camada *output* (verde) temos que

$$o = g(u_{m\times(h+1)}W^2_{(h+1)\times c}) \tag{3}$$

em que $u = (a^{(2)}, I_{m\times 1})$, ou seja, adicionamos a coluna do *bias* na matriz $a^{(2)}$ e W^2 é a matriz com os pesos associados aos $h+1$ neurônios (já com o intercepto) da camada escondida com os c *outputs*. Note que o número de neurônios na camada *output* está diretamente relacionado com o objetivo da análise que, no caso, seria realizar uma predição e, portanto, teríamos $c = 1$.

Arquiteturas mais complexas (isto é, com um maior número de camadas *hidden*) podem ser facilmente estendíveis a partir do exemplo acima. O processo de aprendizado e estimação dos pesos se dá através da etapa de *forward* e de *backpropagation* [18]. Além disso, técnicas de regularização como o *drop-out* [18] podem ser utilizadas. No *drop-out*, a cada inserção de um novo vetor de dados na rede, ocorre a eliminação temporária de neurônios e respectivas ligações, com uma determinada probabilidade [17].

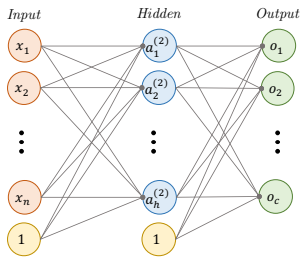


Figura 2: Arquitetura da Rede Neural com uma camada escondida.

3 Metodologia

Dadas todas as definições e nomenclaturas necessárias, a base desse projeto é construir um modelo baseado em técnica de AM que seja capaz de estimar a distância de reversão, baseando-se em algumas *features* extraídas das permutações. O treinamento dos algoritmos será feito considerando permutações de tamanhos reduzidos e, então, avaliaremos a sua qualidade em permutações grandes.

3.1 Conjunto de dados

O conjunto de dados utilizado neste projeto considera todas as permutações de tamanho $n = \{5, 6, \cdots, 10\}$, totalizando 4.037.880 permutações. Para cada permutação, foi calculado um vetor de 13 *features*, bem como a distância exata

de reversão[1]. A Tabela 1 mostra todas as *features* calculadas e a distância exata de reversão, exemplificadas com a permutação $\pi = (5\ 4\ 3\ 1\ 2)$.

Tabela 1: Exemplo das *features* que foram utilizadas para o treinamento dos algoritmos de AM para a permutação $\pi = (5\ 4\ 3\ 1\ 2)$

Feature	Valor observado para π
Quantidade de *breakpoints*	3
Quantidade de *strips* unitárias	2
Tamanho da menor *strip*	2
Tamanho da maior *strip*	3
Quantidade de *strips* crescentes	3
Quantidade de *strips* decrescentes	1
Total de ciclos	2
Quantidade de ciclos impares	2
Quantidade de ciclos unitários	1
Tamanho do maior ciclo	5
Tamanho do menor ciclo	5
Quantidade de ciclos orientados	1
Tamanho da permutação	5
Distância exata	2

Sabe-se que, em técnicas que envolvem AM, problemas a se evitar são o de *underfitting* e *overfitting*. Como uma tentativa de evitar e avaliar tais problemas, podemos dividir o nosso conjunto de dados em: treino, validação e teste [17,18]. Inicialmente, o *dataset* foi dividido em dois, uma parcela de 80% e outra de 20%. A parcela de 20% é reservada para o conjunto de teste, enquanto que os outros 80% são dividimos novamente em 80% para treinamento e 20% para validação.

3.2 *Workflow* da Metodologia

A Figura 3 ilustra o fluxograma deste trabalho. Inicialmente, para todas as permutações de tamanho $n = \{5, 6, \cdots, 10\}$, foram extraídas as *features* apresentadas na Tabela 1, bem como a distância exata de reversão. Em seguida, esse conjunto de dados foi dividido em treino, validação e teste, de modo que os conjuntos de treino e validação foram utilizados para o treinamento e análise de *overfitting* dos modelos. Posteriormente, após os melhores modelos serem escolhidos, em termos de algumas medidas estatísticas como, por exemplo, o *Root Mean Squared Error* (RMSE), aplicamos o conjunto de testes e também um conjunto de permutações de tamanho $n = \{11, 20, 30, \cdots, 100\}$ sendo que, para cada n, consideramos uma amostra de 5000 permutações. Por fim, comparamos os modelos com os Algoritmos 1 e 2 descritos na Seção 2.6.

[1] A distância exata de reversão foi, parte extraída do site `http://mirza.ic.unicamp.br/` e, também, parte fornecida pelo prof. Dr. Zanoni Dias do IC-UNICAMP.

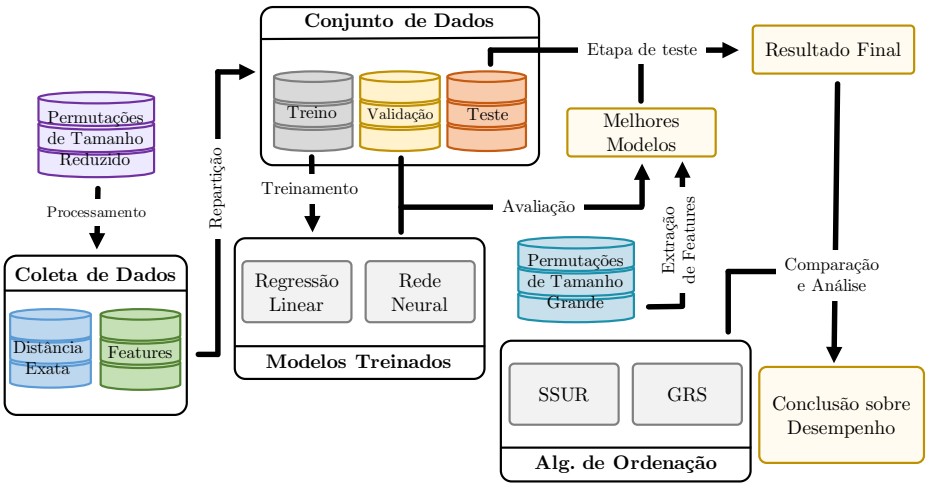


Figura 3: Fluxograma da metodologia do projeto.

4 Resultados e Avaliação

Para a estimação dos parâmetros da RL, foi considerado a decomposição QR. Já na RN, foram consideradas as arquiteturas descritas na Tabela 2, como função de ativação foi utilizada a ReLU [18] e como função de custo o *Mean Squared Error* [17].

A Tabela 2 também mostra o RMSE dos conjuntos de validação (RMSE_{val}) para cada modelo considerado. Pode-se observar que, dentre todos os modelos, o que obteve menor RMSE_{val} foi a RL e, com relação aos modelos de RN, a rede I-RN B foi a que apresentou melhores resultados. Desse modo, por parcimônia, a análise seguirá com os modelos RL e I-RN B.

A Tabela 3, por sua vez, apresenta os valores de RMSE dos conjuntos de teste (RMSE_{teste}) e novamente o RMSE_{val} para os modelos RL e I-RN B. Como o valor de RMSE_{val} é muito próximo ao valor de RMSE_{teste}, pode-se conjecturar que não houve *overfitting* em nenhum dos modelos.

Na Figura 4 é possível observar o gráfico da distância média *vs* o número de *breakpoints* e também o gráfico da distância média *vs* o tamanho da permutação para os modelos RL, I-RN B e para os algoritmos SSUR (Alg. 1) e GRS (Alg. 2), considerando permutações de tamanho reduzido, ou seja, $n \in \{5,\ 6,\ 7,\ 8,\ 9,\ 10\}$ para o conjunto de teste.

Tabela 2: Modelos utilizados na etapa de treinamento, bem como o valor do RMSE_{val} para cada um dos modelos

	1a *Hidden*		2a *Hidden*		
Modelo	η^1	δ^1	η^2	δ^2	RMSE_{val}
I-RN A	15	0.2	–	–	0.5055
I-RN B	30	0.2	–	–	**0.4866**
II-RN A	30	0.3	15	0.2	0.5184
II-RN B	64	0.3	32	0.2	0.4973
II-RN C	256	0.3	128	0.2	0.4899
RL	–	–	–	–	**0.4814**

Características que não se aplicam estão indicadas com –. Além disso, η^i e δ^i fazem referencia ao número de neurônios e ao valor do *drop-out* na camada escondida i, respectivamente.

Tabela 3: RMSE para o modelo de RL e RN, considerando o conjunto de validação e teste

Modelo	RMSE_{val}	RMSE_{teste}
RL	0.4814	0.4813
I-RN B	0.4866	0.4867

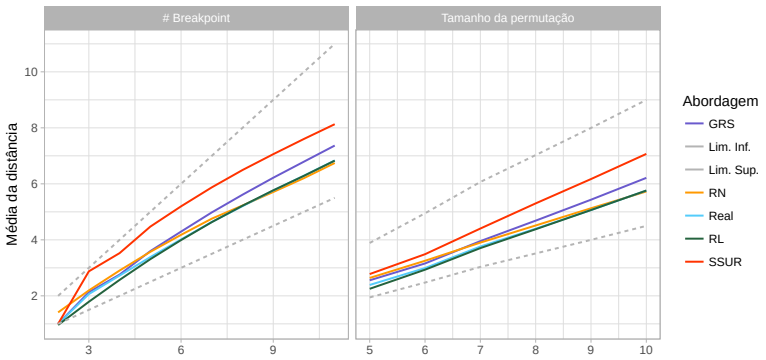


Figura 4: Distância média *vs* o número de *breakpoints*, bem como a distância média *vs* o tamanho da permutação.

Observa-se nos dois gráficos que, para todas as abordagens, a média ficou dentro dos limites inferior e superior. Mais precisamente, nota-se que o algoritmo SSUR foi o que se distanciou mais dos valores exatos de distância de reversão, representados pela reta Real (cor azul) e que o algoritmo de 2-aproximação (GRS) manteve-se sempre próximo do valor exato. Por outro lado, percebe-se

que os melhores resultados estão nas abordagens propostas de AM, sendo que a RL foi a que se manteve mais próxima da reta dos valores exatos, embora a RN também tenha apresentado resultados satisfatórios. Por fim, é possível observar que, à medida em que o número de *breakpoints* e o tamanho da permutação cresce, maior é a distância média e, mais próximo do valor exato as abordagens de AM ficam.

A Figura 5 apresenta a diferença, em módulo, das curvas da Figura 4 em relação a curva Real. Os resultados corroboram com o que foi discutido anteriormente, ou seja, o algoritmo GRS, a RL e a RN foram as abordagens que apresentaram resultados mais próximos do valor exato de distância de reversão, fato observado pelos menores valores de diferença entre as retas. Percebe-se, também, que a RL e a RN foram as que obtiveram a menor distância e que, conforme o número de *breakpoints* e o tamanho da permutação aumenta, menor é a diferença para esses dois modelos.

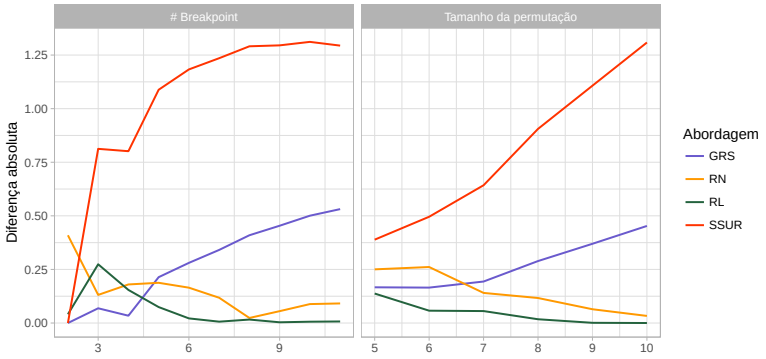


Figura 5: Diferença, em módulo, das curvas da Figura 4 com a curva real.

Visto que os resultados das abordagens de AM foram satisfatórios, torna-se interessante verificar a proporção de distâncias que possam ter sido superestimadas ou subestimadas. A Tabela 4 mostra a proporção da distância de reversão que foi calculada fora dos limites inferior ($\frac{b(\pi)}{2}$) e superior ($b(\pi)$) para todas as abordagens considerando apenas as permutações de tamanho reduzido. Como era esperado, o GRS foi o único que não apresentou resultados fora dos limites. Já o SSUR, apresentou uma quantidade significativa de estimações fora do limite superior, evidenciando que esse algoritmo superestima a distância de reversão, ou seja, realiza muito mais operações do que a quantidade necessária. Nas abordagens de AM, embora não houvesse garantia nenhuma de obter estimações dentro dos limites, nota-se que, apesar de existirem casos fora dos limites, a proporção observada de ocorrência desses é de fato muito baixa, indicando novamente uma boa qualidade na estimação de permutações com tamanho reduzido.

Tabela 4: Proporção da distância de reversão que foi calculada fora dos limites inferior ($\frac{b(\pi}{2}$) e superior ($b(\pi)$) para as permutações de tamanho reduzido

	Abordagem			
Limite	GRS	SSUR	RL	RN
Inferior	0	0	2×10^{-5}	1×10^{-6}
Superior	0	0.0208	0	0

Considerando os resultados interessantes obtidos para as permutações de tamanho reduzido, decidiu-se por analisar o comportamento das abordagens para permutações de tamanhos maiores, considerando $n \in \{11, 20, 30, \cdots, 100\}$. A Figura 6 apresenta a distância média *vs* o número de *breakpoints*, bem como a distância média *vs* o tamanho da permutação para as permutações maiores. De modo geral, nota-se que todas as abordagens permaneceram dentro ou próximo dos limites superior e inferior. Novamente, observa-se que o SSUR apresentou o maior valor para a distância média, seguido do GRS. A RN apresentou valores bem menores do que os outros métodos, sinalizando uma possível subestimação, ou seja, estimando valores menores do que eles deveriam ser. Em relação a RL, observou-se que as estimações ficam próximas da média entre os limites superior e inferior. É importante ressaltar que, para uma melhor análise, seria necessário comparar os resultados obtidos com os valores exatos da distancia de reversão, uma tarefa que, computacionalmente, se torna inviável.

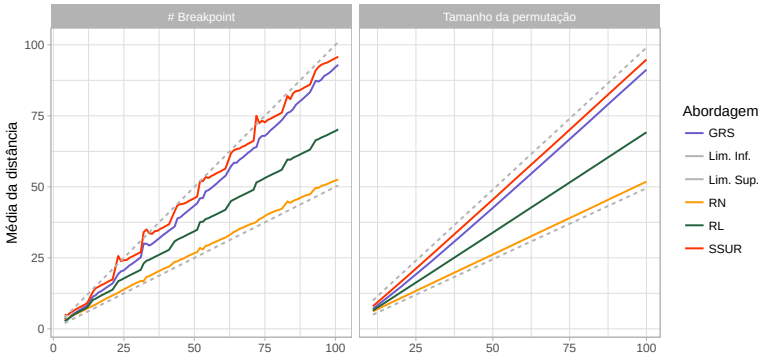


Figura 6: Distância média *vs* o número de *breakpoints*, bem como a distância média *vs* o tamanho da permutação, para permutações maiores.

Por fim, a Tabela 5 apresenta a proporção da distância de reversão que foi calculada fora dos limites inferior ($\frac{b(\pi)}{2}$) e superior ($b(\pi)$) para todas as abordagens considerando as permutações de tamanho maior. Novamente, como era esperado,

o GRS não apresentou resultados fora dos limites. Já o SSUR, apresentou estimações fora do limite superior, evidenciando que esse algoritmo superestima a distância de reversão, ratificando o que foi discutido anteriormente. Nas abordagens de AM, a RL apresentou todos os resultados dentro dos limites. Já na RN, vemos uma quantidade não desprezível de estimações fora do limite inferior, comprovando uma possível subestimação.

Tabela 5: Proporção da distância de reversão que foi calculada fora dos limites inferior ($\frac{b(\pi}{2}$) e superior ($b(\pi)$) para as permutações com tamanho maior

	Abordagem			
Limite	GRS	SSUR	RL	RN
Inferior	0	0	0	0.0333
Superior	0	0.0096	0	0

5 Problemas encontrados

Neste trabalho, tentou-se implementar o algoritmo de Christie [10] com o objetivo de utilizá-lo na comparação e até mesmo extrair das permutações outras *features* para os modelos de AM.

Apesar do algoritmo de 1,5-aproximação apresentar menor complexidade de entendimento se comparado ao de 1,375, enfrentou-se grande dificuldade em implementar algumas partes do algoritmo, porque alguns passos são omitidos, por exemplo, o passo que determina como é realizada a conexão de arestas pretas e cinzas no grafo de ciclos; outros passos não estão explicados de forma clara, como o que define as reversões que os vértices no grafo de reversões representam.

Apesar de Christie ter publicado uma *errata* [19] para o seu artigo original [10] melhorando a explicação de alguns detalhes, os passos citados anteriormente continuaram sem uma definição clara. Além disso, Soncco-Álvarez e Ayala-Rincón [20], em 2012, identificaram e corrigiram um problema no principal lema que define a transformação do grafo de reversões.

Considerando os empecilhos encontrados e o vasto tempo despendido tentando implementar tal algoritmo, seria relevante aprofundar os conhecimentos na área a fim de conseguir solucionar completamente as imprecisões do algoritmo em questão e assim implementá-lo de forma correta.

6 Conclusões

Este trabalho analisou a aplicação de métodos de Aprendizado de Máquina, mais especificamente Regressão Linear (RL) e Redes Neurais (RN), na estimação da distância de reversão para permutações sem orientação de genes, problema que foi comprovado pertencer a classe NP-Difícil. Esses métodos foram comparados

com algoritmos já estabelecidos na literatura (SSUR e GRS) e também com a distância exata de reversão.

Os resultados descritos na seção anterior mostraram que a utilização de RL apresenta resultados melhores ou tão bons quanto o algoritmo de 2-aproximação (GRS). Para as permutações de tamanhos 8, 9 e 10, a RL praticamente se equiparou a curva de distância exata a menos de uma diferença ínfima observada nos gráficos da Figura 5. Já a abordagem de RN superestimou a distância de reversão para permutações de tamanho 5 e 6 se comparado ao algoritmo GRS, mas para os outros tamanhos de permutação, ou seja, 7, 8, 9 e 10, esta abordagem mostrou uma melhor estimação que o GRS e pior ou no máximo semelhante à abordagem de RL.

Para as permutações de tamanho maior que 10, a RL e RN se mostraram promissoras, porém acredita-se que os modelos necessitam de mais *features* para a etapa de treinamento, bem como maior potencial computacional para arquiteturas de RN mais complexas. Em RN, uma quantidade significativa de estimações ficou fora do limite inferior, o que confirma a afirmação anterior. Além disso, uma comparação com os valores exatos das permutações grandes poderia permitir uma análise mais precisa.

A utilização de modelos de AM, se mostrou uma interessante ferramenta para se estimar a distância de reversão entre permutações. Para melhorar a análise, como trabalho futuro seria interessante testar outros modelos de AM e também adicionar à comparação um algoritmo melhor que o SSUR e GRS, como o de 1,5-aproximação proposto por Christie [10] ou até mesmo o de 1,375-aproximação proposto por Berman *et al.* [11].

Referências

1. Lou, X.W., Zhu, D.M.: Sorting unsigned permutations by weighted reversals, transpositions, and transreversals. Journal of Computer Science and Technology **25**(4) (Jul 2010) 853–863
2. da Silva, F.A.M., Oliveira, A.R., Dias, Z.: Machine Learning Applied to Sorting Permutations by Reversals and Transpositions. Technical Report IC-PFG-17-03, Institute of Computing, University of Campinas (July 2017) In English, 20 pages.
3. Russell, P.: IGenetics: A Molecular Approach. Benjamin Cummings (2010)
4. Pevzner, P.: Computational Molecular Biology: An Algorithmic Approach. A Bradford book. CogNet (2000)
5. Podani, J., Csontos, P., Tamás, J.: Additive trees in the analysis of community data. Community Ecology **1**(1) (2000) 33–41
6. Setubal, J., Meidanis, J.: Introduction to computational molecular biology. PWS Publishing (1997)
7. Caprara, A.: Sorting by reversals is difficult. In: Proceedings of the First Annual International Conference on Computational Molecular Biology. RECOMB '97, New York, NY, USA, ACM (1997) 75–83
8. Kececioglu, J., Sankoff, D.: Exact and approximation algorithms for sorting by reversals, with application to genome rearrangement. Algorithmica **13**(1) (Feb 1995) 180

9. Bafna, V., Pevzner, P.A.: Genome rearrangements and sorting by reversals. SIAM Journal on Computing **25**(2) (1996) 272–289
10. Christie, D.A.: A 3/2-approximation algorithm for sorting by reversals. In: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '98, Philadelphia, PA, USA, Society for Industrial and Applied Mathematics (1998) 244–252
11. Berman, P., Hannenhalli, S., Karpinski, M. In: 1.375-Approximation Algorithm for Sorting by Reversals. Springer Berlin Heidelberg, Berlin, Heidelberg (2002) 200–210
12. Berman, P., Karpinski, M. In: On Some Tighter Inapproximability Results (Extended Abstract). Springer Berlin Heidelberg, Berlin, Heidelberg (1999) 200–209
13. Auyeung, A., Abraham, A.: Estimating genome reversal distance by genetic algorithm. In: Evolutionary Computation, 2003. CEC '03. The 2003 Congress on. Volume 2. (Dec 2003) 1157–1161 Vol.2
14. Jones, N., Pevzner, P.: An Introduction to Bioinformatics Algorithms. A Bradford book. London (2004)
15. Rahman, A., Shatabda, S., Hasan, M.: An approximation algorithm for sorting by reversals and transpositions. Journal of Discrete Algorithms **6**(3) (2008) 449 – 457
16. Hannenhalli, S., Pevzner, P.A.: Transforming cabbage into turnip: Polynomial algorithm for sorting signed permutations by reversals. J. ACM **46**(1) (January 1999) 1–27
17. Neter, J., Kutner, M.H., Nachtsheim, C.J., Wasserman, W.: Applied Linear Statistical Models. Irwin, Chicago (1996)
18. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
19. Christie, D.A.: A 3/2-approximation algorithm for sorting by reversals (errata). https://pdfs.semanticscholar.org/b72d/e682534128050914d813184f3bbdc71bf1be.pdf Online; Acessado em 24-11-2017.
20. Soncco-Álvarez, J.L., Ayala-Rincón, M.: A genetic approach with a simple fitness function for sorting unsigned permutations by reversals. In: 2012 7th Colombian Computing Congress (CCC). (Oct 2012) 1–6