

# Algoritmos e heurísticas para comparações exata e aproximada de seqüências

Guilherme P. Telles<sup>1</sup>, Nalvo F. de Almeida Jr.<sup>2</sup> e Fábio H. Viduani Martinez<sup>2</sup>

`gpt@icmc.usp.br, nalvo@dct.ufms.br, fhvm@dct.ufms.br`

<sup>1</sup>ICMC–USP e <sup>2</sup>DCT–UFMS

# Comparação Inexata de Seqüências

- motivação
- alinhamento de seqüências
- programação dinâmica
- heurísticas para comparação aproximada
  - BLAST e FASTA
- métodos para comparação de textos
  - vetor de termos
  - complexidade de Kolmogorov

# Motivação

- Ilustrar o uso de programação dinâmica para resolver um dos problemas mais freqüentes em comparação de textos: comparar seqüências admitindo algumas diferenças entre elas.
- Talvez este seja até mesmo o problema mais freqüente de todos.

# Problema

Comparar duas seqüências para determinar o quão similares ou dissimilares elas são.

# Porque comparação inexata?

- Em biologia molecular, para admitir casamento de seqüências que têm mutações umas em relação às outras.
- Em textos em geral, para ser possível casar variações de palavras.

# Exemplos

- Proteínas de rato e homem (DUSP18):

```
MTSPWSAFPVQIPQPSIRGLSQITKSLFISNGVAANNK...
MT+P  AFPVQ  QPS+  GLSQITKSL+ISNGVAANNK...
MTAPSCAFPVQFRQPSVSGLSQITKSLYISNGVAANNK...
```

- Palavras flexionadas do português:

```
população -urbana-
|||||      |||||
populações urbanas
```

# Comparação exata

Problema resolvido pelo algoritmo KMP de complexidade  $O(n+m)$  para duas seqüências de tamanhos  $n$  e  $m$ .

# Para lidar com algumas diferenças

- Queremos admitir:
  - Substituição de símbolos
  - Inserção de símbolos
  - Remoção de símbolos
- Podemos definir o problema da comparação inexata entre duas seqüências por meio de dois conceitos importantes:
  - alinhamento
  - similaridade

# Conceitos básicos

Seqüência ou cadeia: sucessão ordenada de caracteres de um alfabeto.

Tamanho de uma cadeia  $s$ ,  $|s|$ , é o número de caracteres em  $s$ .

Caracteres de  $s$  indexados de 1 até  $|s|$ , sendo  $s[i]$  o caractere que ocupa a  $i$ -ésima posição.

Exemplo:  $s = \text{ACTGTCGATG}$ ,  $|s| = 10$ ,  $s[3] = T$  e  $s[8] = A$ .

Cadeia vazia: tem tamanho 0 e é notada  $\epsilon$ . Subcadeia de uma cadeia  $s$ : é uma cadeia formada por caracteres consecutivos de  $s$ . Notamos  $s[i..j]$  a subcadeia que contém os caracteres de  $i$  a  $j$  inclusive.

$s[2..5] = \text{CTGT}$ ,  $s[0..0] = \epsilon$ ,  $s[11..10] = \epsilon$

Prefi xo de uma cadeia  $s$ : é uma subcadeia da forma  $s[0..i]$  para

$0 \leq i \leq |s|$ .  $\epsilon$  é prefí xo de  $s$ . Sufi xo de uma cadeia  $s$ : é uma subcadeia da forma  $s[i..|s|]$  para  $1 \leq i \leq |s| + 1$ .  $\epsilon$  é sufi xo de  $s$ .

$s[0..4] = \text{ACTG}$

$s[8..10] = \text{ATG}$

# Alinhamento

Um alinhamento entre duas seqüências  $s$  e  $t$  consiste na inserção de zero ou mais espaços em quaisquer posições de  $s$  e de  $t$  de tal forma que:

- $s$  e  $t$  tenham o mesmo tamanho
- se  $s[i]$  é um espaço então  $t[i]$  não é um espaço e vice-versa.

# Exemplo

TAGCA

GCATCAT

-TAGCA-

| |

GCATCAT

T-A-GCA

|

GCATCAT

--TAG-CA-

| |

GC-A-TCAT

# Pontuação

Podemos definir uma pontuação com base nas semelhanças e diferenças de letras nas colunas de um alinhamento.

$$\text{score}(s, t) = \sum_{i=1}^{|s|} c_i$$

onde

$$c_i = \begin{cases} 1 & \text{se } s[i] = t[i] \\ -1 & \text{se } s[i] \neq t[i] \\ -2 & \text{se } s[i] = '-' \text{ ou } t[i] = '-' \end{cases}$$

# Exemplo

```
--TAG--CA--  
  |  ||  
GC--A--TCAT
```

$$\text{score} = -2 + -2 + -2 + 1 + -2 + -2 + 1 + 1 + -2 = -9$$

```
--TAGCA--  
  |  ||  
GCATCAT
```

$$\text{score} = -2 + -1 + 1 + -1 + 1 + 1 + -2 = -3$$

# Similaridade

Valor máximo de pontuação obtido por um alinhamento entre duas seqüências.

# Problema

Calcular a similaridade entre duas seqüências e construir um alinhamento com a similaridade calculada.

- Uma solução possível
  - Gerar todos os alinhamentos possíveis entre as duas seqüências, calcular a pontuação de cada um e escolher o melhor.
  - Inviável porque é muito ineficiente.

# Um pouco de intuição

Vamos supor que temos um alinhamento de tamanho  $k$  entre um prefixo  $s[1..i]$  de  $s$  e um prefixo  $t[1..j]$  de  $t$ :

colunas	1	2	3	...	$k$
seqüência $s$	□	□	□	...	□
seqüência $t$	□	□	□	...	□

# Aumentando o alinhamento

Primeira possibilidade: acrescentar um caractere de  $s$  e um caractere de  $t$  ao alinhamento:

colunas	1	2	3	...	$k$	$k + 1$
seqüência $s$	□	□	□	...	□	$s[i + 1]$
seqüência $t$	□	□	□	...	□	$t[j + 1]$

# Aumentando o alinhamento

Segunda possibilidade: Acrescentar um caractere de  $s$  e um espaço ao alinhamento:

colunas	1	2	3	...	$k$	$k + 1$
seqüência $s$	□	□	□	...	□	$s[i + 1]$
seqüência $t$	□	□	□	...	□	-

# Aumentando o alinhamento

Terceira possibilidade: acrescentar um caractere de  $t$  e um espaço ao alinhamento.

colunas	1	2	3	...	$k$	$k + 1$
seqüência $s$	□	□	□	...	□	-
seqüência $t$	□	□	□	...	□	$t[j + 1]$

# Idéia do Algoritmo

- Estender o alinhamento começando com os prefixos vazios.
- Armazenar os valores da similaridade para os prefixos à medida que o alinhamento é estendido (técnica chamada programação dinâmica).

# Algoritmo

- Constrói uma matriz  $M$  em que a célula  $M[i, j]$  tem a similaridade entre  $s[1..i]$  e  $t[1..j]$ , usando aquelas três possibilidades para estender um alinhamento.
- A similaridade aparece na célula  $M[|s|, |t|]$ .

$$M[i, 0] = -2i$$

$$M[0, j] = -2j$$

$$M[i, j] = \max \begin{cases} M[i - 1, j] + (-2) \\ M[i - 1, j - 1] + c(i, j) \\ M[i, j - 1] + (-2) \end{cases}$$

$$c(i, j) = \begin{cases} 1 & \text{se } s[i] = t[j] \\ -1 & \text{se } s[i] \neq t[j] \end{cases}$$

# Exemplo

$s = \text{ACGT}$

$t = \text{AAT}$

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

Diagram illustrating the dynamic programming table for sequence alignment between  $s = \text{ACGT}$  and  $t = \text{AAT}$ . The table shows the alignment scores for each pair of characters. The optimal alignment path is indicated by arrows: (0,0) ← (1,1) ← (2,2) ← (3,3) ← (4,4).

# Alinhamentos

- A partir da matriz podemos construir um ou mais alinhamentos entre as seqüências.
- As setas indicam emparelhamento de um caracter de s com um de t ou adição de espaço a s, ou adição de espaço a t.
- Caminhos alternativos na matriz levam a alinhamentos diferentes, mas com mesma pontuação.

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

Arrows in the original image indicate backpointers: horizontal arrows from right to left, vertical arrows from bottom to top, and diagonal arrows from bottom-right to top-left.

# Tipos de alinhamentos

O mesmo algoritmo pode ser adaptado para produzir alinhamentos globais, locais ou semiglobais.

# Alinhamentos Globais

- As seqüências são comparadas como um todo. É o que vimos até aqui.
- Útil para comparar seqüências aproximadamente de mesmo tamanho que diferem pouco entre si.
- Um alinhamento global pode diluir uma região de boa similaridade local. Alinhamento local

```
ACTGGCG-CTAAT
  ||  ||  ||
TCT-ACGACTGGC
```

```
-----ACTGGCGCTAAT
                |||||
TCTACGACTGGC-----
```

# Alinhamentos Locais

- Um alinhamento local entre duas seqüências  $s$  e  $t$  é um alinhamento entre uma subcadeia de  $s$  e uma subcadeia de  $t$ .
- Útil para comparar um gene contra um cromossomo e para encontrar subcadeias comuns entre duas seqüências.

# Variação do algoritmo

- Podemos encontrar alinhamentos locais usando uma variação do algoritmo para alinhamentos globais.
- A célula  $M[i, j]$  da matriz armazena a melhor pontuação de um alinhamento entre um sufixo de  $s[1..i]$  e um sufixo de  $t[1..j]$ .
- O alinhamento com a cadeia vazia recebe o valor 0.
- A similaridade está em alguma célula de  $M$ .

# Construção de $M$

$$M[i, 0] = 0$$

$$M[0, j] = 0$$

$$M[i, j] = \max \begin{cases} M[i - 1, j] + (-2) \\ M[i - 1, j - 1] + c(i, j) \\ M[i, j - 1] + (-2) \\ 0 \end{cases}$$

$$c(i, j) = \begin{cases} 1 & \text{se } s[i] = t[j] \\ -1 & \text{se } s[i] \neq t[j] \end{cases}$$

# Alinhamento semi-global

- Não penaliza espaços nas extremidades.
- Alinhamento entre um prefixo de  $s$  e um sufixo de  $t$  ou vice-versa.
- Alinhamento em que  $s$  está contido em  $t$  ou vice-versa.

# Variação do algoritmo

- Podemos encontrar alinhamentos semi-globais usando uma variação do algoritmo para alinhamentos globais.
- A inicialização da primeira linha será com zeros para admitir buracos no início de  $s$ .
- A inicialização da primeira coluna será com zeros para admitir buracos no início de  $t$ . A similaridade está em alguma célula da última coluna ou da última linha de  $M$ .

# Complexidade

- Complexidade  $O(nm)$  de tempo e espaço.
- Existem algoritmos que usam espaço  $O(\min(n, m))$ .

# Extensão: mais de uma seqüência

- Alinhamento múltiplo de seqüências.
- Muito usado para alinhar proteínas e identificar regiões de similaridade.

# Generalização do PD

Para  $k$  seqüências de tamanho máximo  $n$ :

- Matriz  $k$ -dimensional.
- Esquema de pontuação para  $k$  letras.
- Algoritmo tem que preenche  $n^k$  células e cada uma depende de  $2^{k-1}$  outras.
- Alinhamento múltiplo de seqüências é NP-completo.

# Recapitulando

- Vimos um algoritmo para comparar seqüências que lida com diferenças entre as seqüências.
- Leva tempo proporcional ao tamanho das seqüências.
- Podemos usar a mesma idéia para produzir alinhamentos globais, locais ou semiglobais.

# Heurísticas para casamento inexato

# Porque usar heurísticas?

Algoritmo exato baseado em PD é muito custoso quando:

- Quando temos um conjunto grande de seqüências para comparar entre si.
- Quando queremos comparar uma seqüência com muitas outras em um banco de seqüências.

# Banco de seqüências

- Conjunto de seqüências de DNA, RNA ou proteína em formato digital. Não está necessariamente associado a um SGBD. Pode ser, por exemplo, um arquivo texto.
- Genbank
  - Banco de seqüências moleculares de acesso público.
  - tinha 44.575.745.176 letras em 40.604.319 seqüências no fim de 2004.
  - [www.ncbi.nlm.nih.gov/Genbank](http://www.ncbi.nlm.nih.gov/Genbank)

# Problema

Comparar uma ou mais seqüências contra todas as outras em um banco de seqüências grande, para encontrar alinhamentos locais.

# BLAST

- Heurística para encontrar alinhamentos locais entre seqüências.
- BLAST é um acrônimo de Basic Local Alignment Search Tool.
- Fundamentada em modelos estatísticos.
- Tem sensibilidade e desempenho bastante bons.

# BLAST

- Família de programas: BLAST (90), gapped-BLAST (97), psi-BLAST, megaBLAST e outros.
- Existem várias implementações disponíveis de BLAST.
- Talvez seja o programa mais usado por bioinformatas atualmente.

# Comparação: PD x BLAST

- Uma proteína contra 2.831 outras
- Algoritmo PD exato (programa swat): 4.19 segundos
- Algoritmo BLAST (programa blastall): 0.17 segundo (+0.34 segundo para pré-processar o banco)

# Observação que motivou o BLAST

- Se um alinhamento entre duas seqüências é estatisticamente significativo (isto é) tem baixa probabilidade de ser aleatório, ele provavelmente contém palavras alinhadas nas duas seqüências com alta similaridade.
- Palavra: cadeia pequena, normalmente de 3 aminoácidos.

# Idéia do BLAST

- Procurar palavras que alinham “bem” nas duas seqüências e tentar fazer esse alinhamento entre as palavras crescer.

# Entrada para o Gapped-Blast

- Duas seqüências  $s$  e  $t$ .
- Uma matriz de substituições  $S$ , que especifica a pontuação para alinhar pares de aminoácidos.

Exemplo de matriz

	A	C	D	...
A	2	-4	-1	...
C	-3	3	1	...
D	-1	2	2	...
...	...	...	...	...

# Passos do Gapped-BLAST

- 1- Compila, em cada seqüência, uma lista de palavras que quando alinhadas com alguma outra podem pontuar acima de um limiar  $T$ .
- 2- Busca alinhamentos entre palavras selecionadas no Passo 1 que satisfaçam certas condições.
- 3- Estende os alinhamentos encontrados no Passo 2.

# 1- Compilação da lista de palavras

O algoritmo percorre as seqüências a partir da primeira letra, uma a uma, verificando se as palavras pontuam acima de  $T$  quando alinhadas sem espaços com uma outra qualquer, de acordo com a matriz  $S$ .

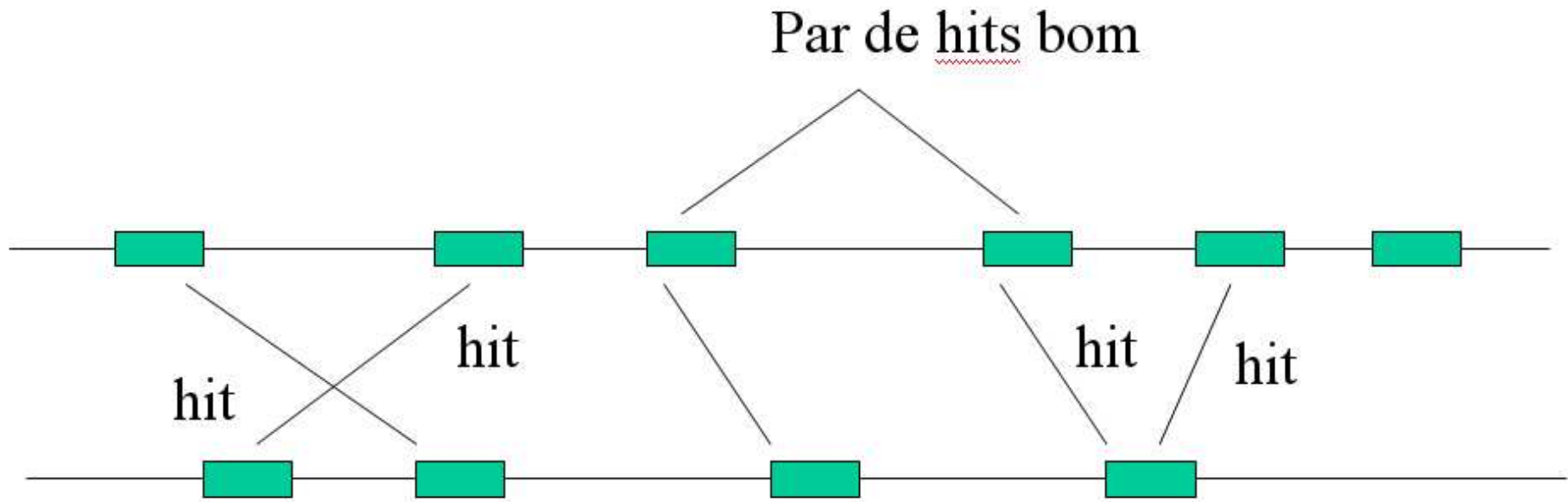
ANMIGZR

palavras: ANM,NMI,MIG,GZR

## 2- Busca de pares de hits

- Hit: uma palavra de  $s$  que alinha com uma de  $t$  com pontuação acima de  $T$ .
- Diagonal de um hit:  $i - j$ , onde  $i$  é a posição da palavra de  $s$  e  $j$  é a posição da palavra de  $t$ .
- O algoritmo procura pares de hits que:
  - Não se sobrepõem.
  - Estão na mesma diagonal.
  - Estão no máximo a uma distância  $A$  entre si.

# 2- Busca de pares de hits



# 3- Extensão dos alinhamentos

Duas fases:

- Sem espaços, mais rápida.
- Com espaços se a sem espaços der um bom alinhamento.

# Extensão dos alinhamentos sem espaços

- O alinhamento do segundo hit do par é estendido sem adicionar espaços.
- Pára de estender quando houver uma variação negativa grande na pontuação em relação à máxima obtida.
- Se o alinhamento obtido tiver pontuação maior que  $S_g$ , então ele passa para a próxima fase.

# Extensão dos alinhamentos sem espaços

\* \* \*

m q n g h s d e e s v r n g s g e s s q s

| | | | | | | | | |

p q q a q p p e l a v r q r s a m l r k q

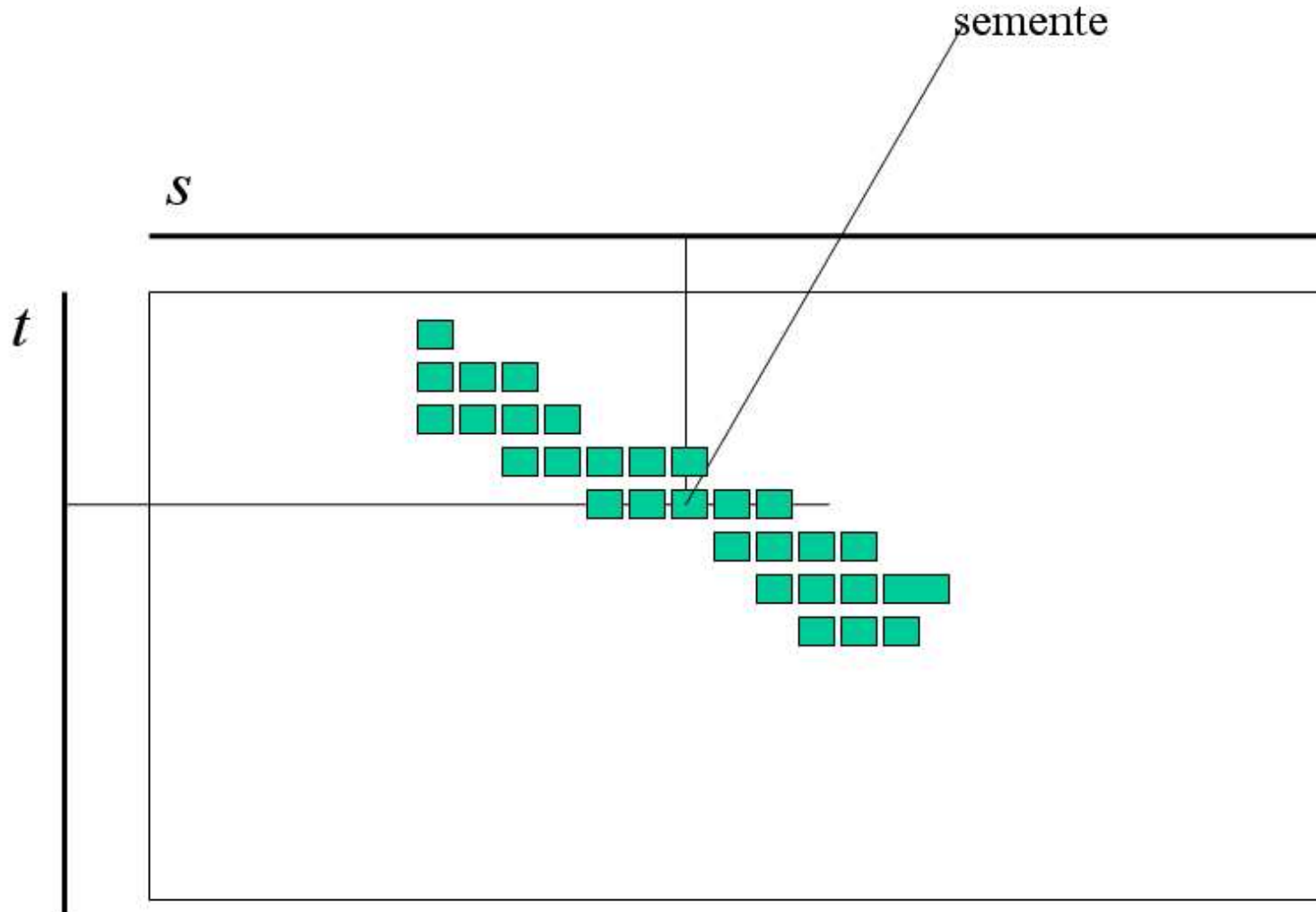
# Extensão dos alinhamentos com espaços

O algoritmo seleciona a semente do alinhamento com espaços: coluna central dentre as 11 colunas consecutivas de maior pontuação no alinhamento sem espaços.

# Extensão dos alinhamentos com espaços

- Preenche uma matriz de programação dinâmica a partir da semente nas duas direções.
- Preenche apenas as células em que a pontuação não diminui mais que um certo limiar  $X_g$ .

# Extensão dos alinhamentos com espaços



# Saída do Gapped-BLAST

- Alinhamentos com espaços que tenham significância estatística aceitável.
- Pontuação normalizada: permite comparar alinhamentos que usam esquemas de pontuação diferentes.
- Outros alinhamentos com boa pontuação que não pode ser aumentada pela extensão ou redução do alinhamento (HSPs).

# Sensibilidade

A probabilidade de que o BLAST perca um alinhamento com pontuação maior que 40 é de menos de 4% para  $w = 3$  e  $T = 11$ .

# Complexidade

- No pior caso  $O(nm)$ .
- Na prática, muito aquém desse limite.

# Recapitulando

- BLAST é uma heurística para encontrar alinhamentos locais entre seqüências.
- Baseado na extensão do alinhamento entre palavras pequenas.
- Usa uma variação do algoritmo de PD que preenche apenas uma região da matriz.
- Desenvolvido para ser usado para DNA ou proteínas, pode ser ajustado para textos em geral.

# FASTA

- Heurística para encontrar alinhamentos locais entre uma seqüência e várias outras em um banco de seqüências.
- Criada por Lipman e Pearson.
- Aprimoramento da heurística Fastp, dos mesmos autores.

# Entrada para o FASTA

- Uma seqüências  $s$  e um banco de seqüências.
- Uma matriz de substituições  $S$ , que especifica a pontuação para alinhar pares de letras.

# Passos do FASTA

- Identificação de alinhamentos exatos e regiões.
- Pontuação das regiões.
- Pontuação de união de regiões.
- Produção de um alinhamento.

# Identificação de alinhamentos exatos

O FASTA identifica alinhamentos exatos e sem inserção de espaços entre subsequências de tamanho  $k$ , **k-tuplas**, comuns à seqüência consultada  $s$  e à seqüência  $t$  no banco.

# Exemplo

- Sejam  $s$  e  $t$  seqüências de proteínas e  $k = 3$ .
- o FASTA constrói um vetor  $A$  de tamanho  $20^3 = 8000$ ,
- armazena em  $A$  as posições de todas as triplas de  $s$  e
- varrendo a seqüência  $t$ , identifica as posições em  $s$  e  $t$  onde há coincidência entre triplas.

# $k$ -tuplas na mesma diagonal

- Suponha que duas  $k$ -tuplas em  $s, s'$  e  $s''$ , são iguais a duas  $k$ -tuplas em  $t, t'$  e  $t''$ .
- Se a diferença entre a posição inicial de  $s'$  e  $t'$  é igual à diferença entre a posição inicial de  $s''$  e  $t''$ , então os alinhamentos  $s't'$  e  $s''t''$  estão na mesma diagonal da matriz.

# Identificação de regiões

- Uma região é formada por uma ou mais  $k$ -tuplas na mesma diagonal que não estejam muito distantes.
- Cooresponde a um alinhadamento sem espaços.
- As dez regiões de melhor pontuação são selecionadas.

# Pontuação das regiões

- A pontuação das dez melhores regiões é recalculada usando uma matriz de substituições, considerando apenas alinhamentos sem espaços.
- Cada região é podada, conservando apenas sua sub-região de pontuação maximal.
- Apenas regiões com pontuação acima de um certo limiar são consideradas nos próximos passos do algoritmo.
- O programa registra a região de melhor pontuação entre  $s$  e cada seqüência  $t_i$  no banco.

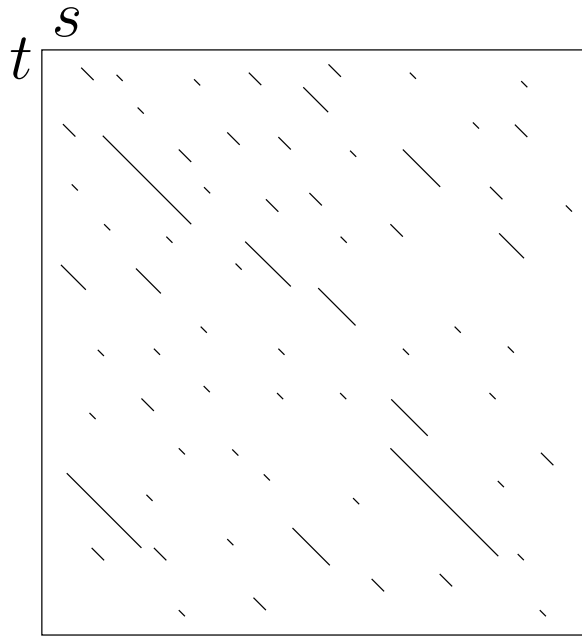
# Pontuação de união de regiões

- as sub-regiões são unidas para produzir um alinhamento com espaços.
- leva em conta a posição das sub-regiões, sua pontuação e uma função de penalidade para uní-las.
- Esta função de penalidade corresponde à inserção de espaços no alinhamento.

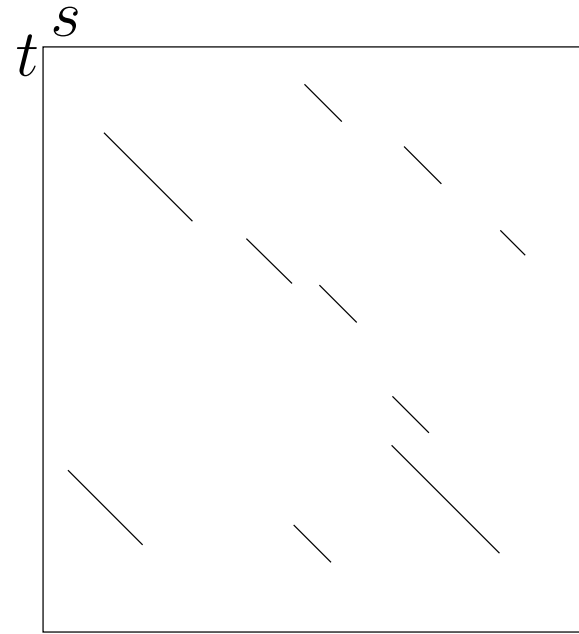
# Produção de um alinhamento

As regiões de melhor pontuação são usadas para produzir um alinhamento entre  $s$  e uma seqüência  $t_i$  do banco usando uma variação do algoritmo de programação dinâmica restrito a uma região em torno do alinhamento obtido no terceiro passo

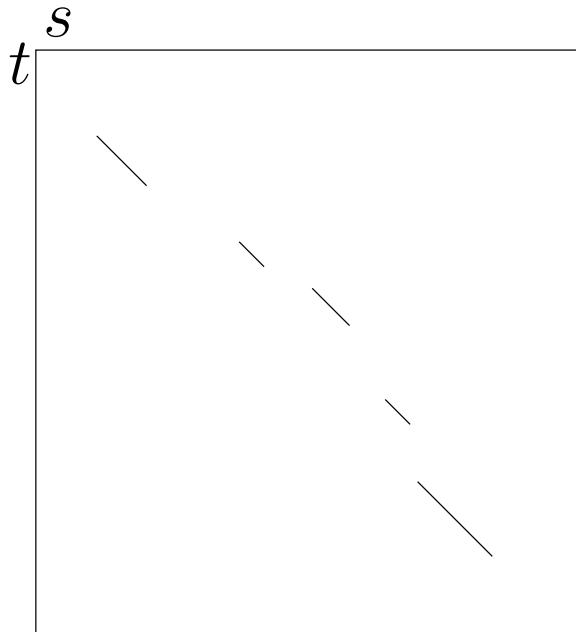
# Graficamente



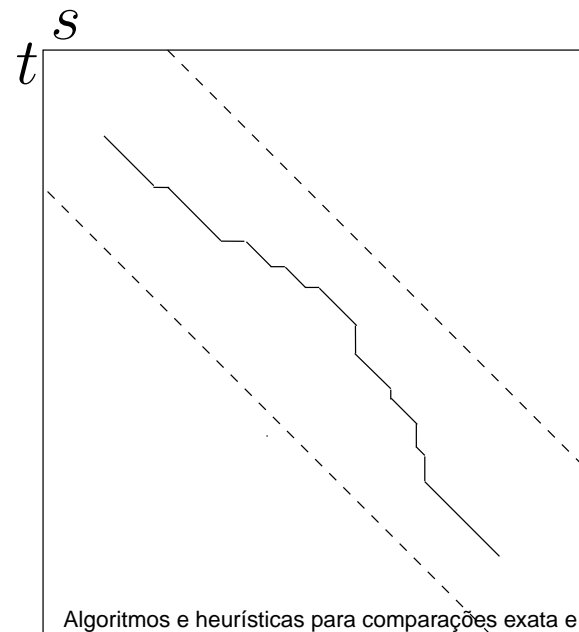
a



b



c



# Comparação de textos

# Problema

Apesar de podermos considerar um texto como uma seqüência, se usarmos um algoritmo de comparação entre dois textos, o resultado provavelmente não será muito satisfatório.

# Exemplo

```
enquanto -e-u trabalho, outros descansam
  |      | |      |      |      |      |
o--utros descansam enquanto eu trabalho--
```

```
enquanto eu trabalho
|||||||||||||||||
enquanto eu trabalho
```

# Vetor de termos

- cada texto do conjunto  $T$  é representado como um vetor multidimensional.
- Cada dimensão dos vetores representa um termo.
- Um termo é uma palavra do texto.

# Escolha dos termos

- vocabulário curado.
- o conjunto de termos que aparecem em algum texto em  $T$ .
- Remoção de artigos, preposições, conjunções e outras palavras pouco discriminativas.
- Redução dos termos aos seus radicais.

# Construção dos vetores

- Seja  $\mathcal{V}$  o vocabulário escolhido.
- Seja  $V_{t_i}$  o vetor de termos para o texto  $t_i$ .
- Cada elemento  $V_{t_i}[x]$  deste vetor contém o número de vezes que o termo  $x$  aparece no texto  $t_i$ .
- O valor de  $V_{t_i}[x]$  pode ser normalizado em função do tamanho dos documentos.

# Similaridade entre textos

A similaridade entre dois textos pode ser computada como um produto entre os vetores

$$\text{sim}(t_i, t_j) = \sum_{\forall x \in \mathcal{V}} V_{t_i}[x] \times V_{t_j}[x].$$

# Considerações

- A representação vetorial dos textos é bastante utilizada.
- Quando a dimensão dos vetores é muito grande em geral é necessário selecionar um conjunto de termos relevantes com poder de discriminação. Processo *ad-hoc*.
- Quando o conjunto de textos se altera, o conjunto de termos pode se modificar o suficiente para que seja necessário recomputar os vetores de todo o conjunto.

# Quantidade de informação

# Complexidade de Kolmogorov

- A complexidade de Kolmogorov de uma cadeia  $s$  é  $K(s|\varepsilon) = K(s)$ , isto é, o tamanho do menor programa que gera  $s$  tendo a cadeia vazia como entrada.

# Intuitivamente

- Intuitivamente, a complexidade de Kolmogorov é a quantidade de informação que uma cadeia contém.

# Complexidade Condicional de Kolmogorov

- Dados dois textos  $s$  e  $t$  quaisquer, a **complexidade condicional de Kolmogorov** de  $s$  dado o texto  $t$ ,  $K(s|t)$ , é o tamanho do menor programa que, tendo  $t$  como entrada, gera  $s$ .

# Intuitivamente

- Intuitivamente, a complexidade condicional de Kolmogorov é a quantidade de informação que a cadeia  $t$  não tem a respeito de  $s$ .
- Quando mais informação  $t$  possui a respeito de  $s$ , menos complicada é a geração de  $s$  a partir de  $t$  e, portanto, menor é a complexidade condicional de Kolmogorov,  $K(s|t)$ .

# Distância entre textos

- Dadas duas cadeias  $s$  e  $t$ , é possível calcular

$$d(s, t) = 1 - \frac{K(s) - K(s|t)}{K(st)},$$

que é uma métrica entre  $s$  e  $t$  a menos de um fator logarítmico aditivo, onde  $st$  é a concatenação de  $s$  e  $t$ .

- Note que  $d(s, t) = 0$  quando  $s = t$  e  $d(s, t)$  tende a 1 quanto mais distintos são  $s$  e  $t$ .

# Distância entre textos

- Poderíamos calcular  $d(s, t)$ , mas a complexidade de Kolmogorov não é computável.
- Podemos aproximar a complexidade de Kolmogorov usando compactação.

# NCD

$$NCD(s, t) = \frac{C(st) - \min\{C(s), C(t)\}}{\max\{C(s), C(t)\}}$$

onde  $C(\cdot)$  é o tamanho da versão compactada de um texto.

- Essa relação também é uma métrica entre  $s$  e  $t$  a menos de um fator logarítmico aditivo.

# Considerações

- Aproximações da complexidade de Kolmogorov podem ser usadas para determinar o relacionamento entre textos com base em seu conteúdo.
- Possibilita uma análise preliminar sem necessidade de pré-processamento ou inspeção manual dos textos.