Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

# Relatório Científico Final

# Problemas de Transformação de Strings por Operações de Rearranjo

*Número do Processo FAPESP:* 2019/13312-7

*Período de Vigência da Bolsa:* 01/06/2020 a 31/05/2021

*Período coberto pelo presente Relatório:* 01/11/2020 a 31/05/2021

*Beneficiário:*
Gustavo da Silva Teixeira

*Responsável:*
Carla Negri Lintzmayer

Junho/2021

# 1 Resumo das atividades desenvolvidas no período

Como consta no relatório científico parcial, nos primeiros cinco meses de pesquisa, de 01/06/2020 a 31/10/2020, nossos esforços se concentraram em estudar os resultados existentes para o problema de Transformação de Strings por Reversões (TSbR), com o objetivo de desenvolver algoritmos práticos para este problema. No período coberto pelo relatório anterior, desenvolvemos, testamos, analisamos e comparamos seis algoritmos, que foram desde estratégias muito simples, como uma baseada em um algoritmo de ordenação, até a implementação de uma meta-heurística/algoritmo genético.

O presente relatório compreende as atividades realizadas no período de 01/11/2020 a 31/05/2021, que podemos dizer que se dividiram em duas grandes partes. A primeira parte consistiu no desenvolvimento, teste e análise de mais quatro algoritmos para o TSbR, sendo dois deles variações de algoritmos desenvolvidos por nós anteriormente, e os outros dois sendo novas propostas, que utilizam um algoritmo para um problema relacionado, o problema da Partição Comum Mínima em Strings (MCSP). Já a segunda parte consistiu em um estudo mais aprofundado de alguns artigos importantes sobre dois problemas relacionados ao TSbR, o já citado MCSP e o problema do Mapeamento de Strings com Preservação Máxima de Duos (MPSM), com o intuito de entender os resultados algorítmicos já existentes para estes dois problemas, a fim de explorar aspectos estruturais das entradas, tentando encontrar alguma estratégia que nos levasse a um algoritmo de aproximação com fator bem definido para o TSbR, o que seria um resultado teórico interessante.

No que diz respeito aos algoritmos apresentados neste relatório, testamos experimentalmente como um algoritmo simples para um problema relacionado se comporta na prática quando combinado com duas estratégias diferentes para lidar com o problema de ordenação de permutações: um algoritmo exato para ordenação de permutações com sinais e um algoritmo de aproximação para permutações sem sinais.

Em relação às novas variações do algoritmo de mapeamentos aleatórios e da implementação da meta-heurística BRKGA, testamos como estas heurísticas se comportam quando abrimos mão de qualidade nas soluções devolvidas em troca de um menor tempo de execução, utilizando uma estratégia baseada em *breakpoints* de permutações.

Com o estudo mais aprofundado sobre os resultados algorítmicos para os dois problemas relacionados, apesar de não termos chegado a nenhum novo resultado teórico para o TsBR, como um algoritmo de aproximação ou algum limitante para a distância de reversão, foi possível compreender melhor algumas propriedades dos problemas e ganhar mais ferramentas para buscar resultados do tipo futuramente.

Destacamos, também, que o bolsista apresentou trabalhos referentes a esta pesquisa em

simpósios de iniciação científica e a pesquisa resultou em uma publicação em uma revista de iniciação científica.

O restante desde relatório está dividido como segue: na Seção 2, descrevemos as duas variações propostas para dois algoritmos desenvolvidos no primeiro semestre de pesquisa; na Seção 3, descrevemos o Problema da Partição Comum Mínima em Strings (MCSP), destacamos os principais resultados existentes e comentamos brevemente os resultados algorítmicos de alguns artigos estudados; na Seção 4, descrevemos como utilizamos resultados para o MCSP reverso para propor dois algoritmos para o Problema de Transformação de Strings por Reversões (TSbR); na Seção 5, descrevemos como foram realizados os testes experimentais entre os quatro algoritmos aqui propostos, assim como uma análise comparativa entre os resultados obtidos destes testes; na Seção 6, descrevemos o Problema de Mapeamento de Strings com Preservação Máxima de Duos (MPSM), destacamos os principais resultados existentes e comentamos brevemente os resultados algorítmicos de alguns artigos estudados; na Seção 7, fazemos algumas considerações finais sobre o trabalho desenvolvido durante o período da bolsa, citando também os eventos em que o trabalho foi apresentado e a publicação resultante da pesquisa; no Apêndice A, trazemos algumas definições importantes sobre os problemas tratados, necessárias para compreender alguns resultados relacionados com o que consta no relatório anterior; no Apêndice B, trazemos partes importantes dos artigos estudados sobre os problemas relacionados, MCSP e MPSM, que descrevem detalhadamente as estratégias e demonstrações que levaram aos resultados já existentes para eles.

# 2 Variações de Mapeamentos Aleatórios e BRKGA

Nesta seção, descrevemos duas variações de algoritmos propostos na primeira etapa de pesquisa, descritos no relatório científico parcial, sendo uma para a estratégia dos mapeamentos aleatórios e outra para implementação da meta-heurística BRKGA.

Em ambos os casos, a ideia é utilizar o fato de um menor número de breakpoints entre duas permutações geralmente estar diretamente associado a uma distância de reversão menor entre estas permutações.

## 2.1 Mapeamentos aleatórios e breakpoints

O primeiro algoritmo é uma variação do algoritmo MAPAS-1 que, em vez de calcular um número de reversões para cada mapeamento usando KS95, calcula o número de breakpoints de reversão das permutações geradas pelos mapeamentos. No final, ele devolve o número de

reversões calculado por KS95 para a permutação que possui o menor número de breakpoints de reversão.

Como já citado, essa ideia é baseada no fato de que haver menos breakpoints, em geral, implica em distâncias menores, e aplicar KS95 consome muito mais tempo do que calcular o número de breakpoints. Este algoritmo será denominado MAPAS-B2.

A geração de cada um dos mapeamentos aleatórios leva tempo $O(n)$ e, para cada mapeamento, o cálculo do número de breakpoints leva tempo $O(n)$.

## 2.2 BRKGA e breakpoints

Da mesma forma que fizemos com os mapeamentos, propusemos uma variação, que consome menos tempo, para o algoritmo BRKGA-2. Ela consiste em alterar a função de aptidão de modo que o valor de aptidão seja o número de breakpoints de reversão do mapeamento associado. Quanto menor for o número de breakpoints de reversão, maior será o valor de aptidão do indivíduo.

O restante do algoritmo permanece inalterado, exceto pelo fato de que ele retorna o número de reversões calculadas por KS95 para um indivíduo com o maior valor de aptidão, ou seja, o menor número de breakpoints de reversão. Este algoritmo se chamará BRKGA-B3. Agora, observe que o cálculo do valor de aptidão de um indivíduo leva tempo $O(n)$, em vez do antigo tempo $O(n^2)$ necessário para calcular um número de reversões por KS95.

# 3 Estudo do Problema da Partição Comum Mínima em Strings (MCSP)

Nesta seção, descrevemos o Problema da Partição Comum Mínima em Strings (MCSP), trazendo as definições necessárias, os principais resultados existentes na literatura e, nas subseções posteriores, comentando as ideias por trás dos algoritmos propostos nos artigos selecionados para estudo.

## 3.1 Definições e resultados existentes sobre o MCSP

Uma *partição* de uma string $S = s_1 \dots s_n$ é uma sequência $\mathcal{P} = (P_1, P_2, \dots, P_m)$ de strings cuja concatenação é igual a $S$, isto é, $P_1 P_2 \dots P_m = S$. As strings $P_i$, com $1 \leq i \leq m$, são chamadas de *partes* de $\mathcal{P}$, e o número de partes em uma partição é o seu *tamanho*, denotado por $|\mathcal{P}|$.

Seja uma partição $\mathcal{P} = (P_1, P_2, \dots, P_m)$ de uma string $S = s_1 \dots s_n$. Se $\ell = \sum_{i=1}^{j} |P_i|$

para algum $j \in \{1, 2, \ldots, m-1\}$, dizemos que o par $\ell, \ell+1$ é uma *quebra da partição* $\mathcal{P}$, e que $s_\ell s_{\ell+1}$ é um *duo quebrado* da partição $\mathcal{P}$.

Seja uma string $S = s_1 s_2 \ldots s_{n-1} s_n$, com ou sem sinais. A *string reversa* de $S$, que denotamos por $S^R$, é a string que possui os mesmos elementos de $S$, mas em ordem inversa (e, no caso de strings com sinais, com o sinal de cada um dos elementos também inverso), i. e., $S^R = s_n s_{n-1} \ldots s_2 s_1$.

Duas strings $S = s_1 \ldots s_n$ e $T = t_1 \ldots t_n$ são *idênticas*, o que denotamos por $S = T$, se $s_i = t_i$ para cada $i \in \{1, \ldots, n\}$. Dizemos que as strings $S$ e $T$ são *congruentes*, o que denotamos por $S \cong T$, se $S = T$ ou $S = T^R$.

Sejam duas strings compatíveis $A$ e $B$. Dizemos que $S$ é uma *substring comum, no que diz respeito à relação de equivalência* ($=$), se $S$ é substring de $A$ e é substring de $B$. Dizemos que $S$ é uma *substring comum, no que diz respeito à relação de congruência* ($\cong$), se $S$ é substring de $A$ e existe uma string $R$, substring de $B$, tal que $S \cong R$, ou se $S$ é substring de $B$ e existe uma string $R$, substring de $A$, tal que $S \cong R$.

Dadas uma partição $\mathcal{P} = (P_1, \ldots, P_m)$ de uma string $S$ e uma partição $\mathcal{Q} = (Q_1, \ldots, Q_m)$ de uma string $T$, dizemos que o par $\pi = (\mathcal{P}, \mathcal{Q})$ é uma *partição comum* de $S$ e $T$, no que diz respeito à relação $\texttt{Rel} \in \{=, \cong\}$, se existe uma permutação $\sigma$ de $(1, \ldots, m)$ tal que $(P_i, Q_{\sigma(i)}) \in \texttt{Rel}$, para cada $i \in \{1, \ldots, m\}$.

O *Problema da Partição Comum Mínima em Strings* (do inglês *Minimum Common String Partition Problem*, e que chamaremos de *MCSP*) consiste em encontrar uma partição comum entre $S$ e $T$ com tamanho mínimo, denotado por $\texttt{MCSP}(S, T)$. Quando só é permitido que cada símbolo ocorra no máximo $k$ vezes em cada string, chamamos o problema de *k-MCSP*.

Na versão *sem sinais* do problema, chamada apenas de *MCSP*, a entrada consiste em duas strings sem sinais, e a relação utilizada é a equivalência ($=$). Na versão *com sinais* do problema, chamada de *SMCSP*, a entrada consiste em duas strings com sinais, e a relação utilizada é a congruência ($\cong$).

Ambas as versões do problema foram propostas por Chen *et al.* [7] e, independentemente, por Swenson *et al.* [17].

Chen *et al.* [7] propuseram o SMCSP como uma ferramenta para lidar com o problema da Transformação de Strings com Sinais por Reversões e propuseram uma 1.5-aproximação para o 2-SMCSP. Eles observaram que, para quaisquer duas strings compatíveis com sinais $A$ e $B$, os valores de $\texttt{MCSP}(A, B)$ e $d_\rho(A, B)$ diferem apenas por um fator multiplicativo constante: dada uma partição $(P_1, \ldots, P_m)$ de $A$, uma partição $(Q_1, \ldots, Q_m)$ de $B$ e a permutação $\sigma$ de $[m]$, tal que $P_i \cong Q_{\sigma(i)}$ para cada $i \in [m]$, é possível mover a parte $P_{\sigma^{-1}(1)} \cong Q_1$ ao início de $A$ com uma reversão e, se necessário, revertê-la com mais uma reversão; da mesma forma, é possível mover a parte $P_{\sigma^{-1}(2)} \cong Q_2$ para sua posição correta na primeira string com

no máximo duas reversões sem afetar a parte $P_{\sigma^{-1}(1)} \cong Q_1$ no início da string, e assim por diante. Por outro lado, uma reversão "quebra", ou "separa", no máximo dois pares de letras consecutivas na string e, portanto, a partir de uma sequência de $m$ reversões, derivamos uma partição comum com no máximo $2m$ quebras. Uma observação análoga se aplica a strings compatíveis sem sinais e aos problemas RMCSP e TSbR.

Goldstein *et al.* [12] mostraram que o ambos os problemas são NP-difíceis e APX-difíceis, mesmo quando $k = 2$, onde $k$ é o número máximo de ocorrências de cada símbolo em cada strings. No mesmo trabalho, os autores propuseram uma 1.1037-aproximação para $k = 2$ e uma 4-aproximação para $k = 3$, também para ambas as versões.

Chrobak *et al.* [8] propuseram uma heurística gulosa válida para as duas versões, provando que: para $k = 2$, o fator de aproximação da heurística é exatamente 3; o limitante inferior para $k = 4$ da heurística é igual a $\Omega(log\ n)$; e para o caso geral, para qualquer $k$, o fator de aproximação está entre $\Omega(n^{0.43})$ e $O(n^{0.67})$.

Ainda em resultados válidos para as duas versões do problema, para qualquer valor de $k$, Kolman [14] propôs uma $O(k^2)$-aproximação com tempo de execução $O(nk)$, resultado que foi melhorado por Kolman e Walen [16], que propuseram uma $O(k)$-aproximação, com tempo de execução $O(n)$. Para a versão sem sinais, a melhor aproximação é uma $4k$-aproximação, proposta com Kolman e Walen [16].

O MCSP também foi demonstrado ser tratável (FPT) por parâmetro fixo por Damaschke [9] e, mais tarde, Bulteau e Komusiewicz [5] propuseram um algoritmo de parâmetro fixo apenas na quantidade máxima de ocorrências $k$.

Para strings sem sinais, há outra variação do problema, proposta por Kolman [14], chamada de *MCSP reverso (RMCSP)*, na qual strings sem sinais são comparadas utilizando a relação de congruência ($\cong$). Para esta versão, o melhor resultado é uma $O(k)$-aproximação [16].

A seguir, mostraremos algumas relações conhecidas sobre as soluções para estes problemas [11]. Considere que $S$ e $T$ são duas strings compatíveis.

Sejam $rd(S,T)$ e $\texttt{RMCSP}(S,T)$, respectivamente, a distância de reversão e o tamanho de uma partição comum ótima para o RMCSP entre $S$ e $T$ sem sinais. Então, vale que $rd(S,T) \leq 2\,\texttt{RMCSP}(S,T) \leq 4\,rd(S,T) + 2$.

Sejam $td(S,T)$ e $\texttt{MCSP}(S,T)$, respectivamente, a distância de transposição e o tamanho de uma partição comum ótima para o MCSP entre $S$ e $T$ sem sinais. Então, vale que $td(S,T) \leq \texttt{MCSP}(S,T) \leq 2\,td(S,T) + 2$.

Sejam $srd(S,T)$ e $\texttt{SMCSP}(S,T)$, respectivamente, a distância de reversão e o tamanho de uma partição comum ótima para o SMCSP entre $S$ e $T$ com sinais. Então, vale que $srd(S,T) + 1 \leq \texttt{SMCSP}(S,T) \leq 2\,srd(S,T) + 1$ [7].

## 3.2 Algoritmo Greedy

O primeiro artigo que estudamos com mais atenção foi o de Chrobak *et al.* [8], onde se propõe um algoritmo guloso chamado GREEDY para o MCSP, definido como no Algoritmo 1.

**Algoritmo 1:** GREEDY

**Entrada:** duas strings compatíveis $A$ e $B$

1 inicialmente, todas as letras em $A$ e $B$ são não marcadas e $\mathcal{P}$ e $\mathcal{Q}$ estão vazias
2 **enquanto** *existem letras não marcadas em* $A$ **faça**
3 $S \leftarrow$ a substring comum mais longa entre $A$ e $B$ que não contém letras marcadas (em caso de empate, escolha arbitrariamente)
4 $S^A$, $S^B \leftarrow$ ocorrências de $S$ em $A$ e em $B$, respectivamente
5 designe $S^A$ como uma parte de $\mathcal{P}$ em $A$ e $S^B$ como uma parte de $\mathcal{Q}$ em $B$
6 marque todas as letras em $S^A$ e $S^B$

**Saída:** $(\mathcal{P}, \mathcal{Q})$

É fácil entender a ideia deste algoritmo: dadas duas strings compatíveis, marcaremos, a cada iteração, a substring comum mais longa (ainda não marcada) e a designaremos como uma parte na partição comum entre as strings. Também é fácil notar que o algoritmo é correto, devolvendo uma partição comum válida ao final de sua execução, já que cada parte designada é uma substring comum a ambas as strings.

Com relação ao algoritmo GREEDY, vale o seguinte teorema:

**Teorema 1.**

*(a) O fator de aproximação de* GREEDY *para* MCSP *está entre* $\Omega(n^{0,43})$ *e* $O(n^{0,69})$.

*(b) Para o* 4-MCSP*, o fator de aproximação de* GREEDY *é pelo menos* $\Omega(\log n)$.

*(c) Para o* 2-MCSP*, o fator de aproximação de* GREEDY *é igual a* 3.

O foco de nossos estudos sobre este artigo foi o item $(a)$ do teorema, uma vez que nosso interesse se voltou à versão geral do problema, sem fixarmos um tamanho do alfabeto sobre o qual as strings de entrada são descritas.

Seja $\pi$ uma partição comum mínima de $A$ e $B$. No primeiro passo de GREDDY, temos garantia de encontrar uma substring de comprimento pelo menos o comprimento máximo de uma parte em $\pi$. Assim, para a análise deste algoritmo, gostaríamos de ter uma estimativa semelhante para todos os passos posteriores também. Porém, no segundo passo do algoritmo já não temos garantia de que GREEDY encontre uma substring tão longa quanto a segunda parte mais longa em $\pi$, uma vez que este parte pode se sobrepor à substring marcada no primeiro passo e agora pode estar parcialmente marcada.

Sabendo disso, a ideia para provar o fator de aproximação para o MCSP é utilizar as chamadas *partições comuns de referência*, que são partições definidas indutivamente, a cada passo de GREEDY, partindo inicialmente de uma partição mínima comum $\pi$, e que vão se

deteriorando a cada passo, isto é, vão se distanciando da partição comum mínima conforme o algoritmo designa partes que não constituem a solução ótima. Porém, utilizando esta abordagem, é possível estimar quão danosas às partições comuns de referência são essas designações erradas, de modo que se consiga estima, consequentemente, quão longe da solução ótima a solução de GREEDY estará ao fim da execução.

O resumo da parte estudada deste artigo, contendo as demonstrações completas, se encontra no Apêndice B.1.

## 3.3 Algoritmo Refined Greedy

Baseados na estratégia gulosa do algoritmo GREEDY, de Chrobak *et al.* [8], Kolman e Walen [15] propuseram um algoritmo que produz uma $O(k^2)$-aproximação para o $k$-MCSP, denominado REFINED GREEDY e descrito como no Algoritmo 2.

**Algoritmo 2:** REFINED GREEDY

**Entrada:** duas strings compatíveis $A$ e $B$

$\mathcal{A} \leftarrow (A), \mathcal{B} \leftarrow (B)$

**enquanto** *existem partes não marcadas em* $\mathcal{A}$ *e* $\mathcal{B}$ **faça**

- $S \leftarrow$ a substring comum mais longa de $\mathcal{A}$ e $\mathcal{B}$ que não sobrepõe partes marcadas anteriormente
- marca $S^A$ em $\mathcal{A}$ e $S^B$ em $\mathcal{B}$
- corta os duos de fronteira de $S^A$ em $\mathcal{A}$ e os duos de fronteira de $S^B$ em $\mathcal{B}$
- corta, nas partes não marcadas de $\mathcal{A}$ e $\mathcal{B}$, *todas* as ocorrências dos duos $\delta \in \Phi$, onde $\Phi$ é o conjunto de duos de fronteira de $S^A$ e $S^B$

**Saída:** $(\mathcal{A}, \mathcal{B})$

A ideia por trás das escolhas deste algoritmo é evitar uma propagação de escolhas erradas na estratégia gulosa de apenas escolher a substring comum mais longa a cada iteração, adicionando cortes adicionais a cada passo do algoritmo.

Não é difícil ver que o algoritmo devolve uma solução viável: independente da quantidade e de quais são os duos cortados em cada iteração, uma vez que as strings de entrada são compatíveis, cortar os mesmos duos em cada uma delas sempre fará com que se mantenha uma partição comum.

Para provar o fator de aproximação do algoritmo, os autores partem de uma partição comum mínima desconhecida $\pi$, cujo conjunto de duos de fronteira é $\Delta$ e, cortando em $\pi$ todas as ocorrências dos duos de $\Delta$, geram uma partição comum $\pi_1$. A partir desta partição, a estratégia é gerar partições comuns $\pi_{i+1}$, derivadas de partições comuns $\pi_i$, com $i \geq 1$, cortando duos em cada uma destas partições a cada iteração do algoritmo e, ao final, estimar um limitante superior para a quantidade de blocos da partição $\pi_T$, se o algoritmo fez $T$ iterações.

A conclusão a que se chega é que Refined Greedy produz, em tempo polinomial, uma $4k^2$-aproximação para o $k$-TSbR com sinais e uma $8(2k-1)^2$-aproximação para o $k$-TSbR sem sinais.

Se implementarmos do modo que está descrito no Algoritmo 2, o tempo será $O(n^3)$, mas os autores também propõem uma implementação rápida que produz os mesmos fatores de aproximação para os problemas, mas em tempo linear.

Como nosso interesse estava voltado ao fator de aproximação para o MCSP, esta implementação rápida não foi estudada a fundo.

O resumo da parte estudada deste artigo, contendo as demonstrações completas, se encontra no Apêndice B.2.

## 3.4 MCSP a partir do Problema do *Hitting Set* Mínimo

Propostos pelos mesmos autores do Refined Greedy, os próximos dois algoritmos utilizam um procedimento para outro problema, o do *Hitting Set* Mínimo, como parte do algoritmo para o MCSP [16].

No *Problema do Hitting Set Mínimo*, temos um conjunto $U$ e uma coleção $\mathcal{S}$ de subconjuntos de $U$, isto é, $\mathcal{S} = \{S_1, \ldots, S_k\}$ tal que $S_i \subseteq U$ para $i = 1, \ldots, k$. A tarefa é encontrar um *hitting set mínimo* para $\mathcal{S}$, que é o menor conjunto $H \subseteq U$ tal que $H \cap S_i = \emptyset$, para cada $i \in \{1, \ldots, k\}$. O Problema de *Hitting Set* Mínimo é equivalente ao Problema de Cobertura de Conjuntos Mínima.

A ideia por trás do algoritmo é a seguinte. Dadas as strings $A$ e $B$ e uma string $X$ tal que que o número de ocorrências de $X$ em $A$ seja maior (ou menor, respectivamente) do que o número de ocorrências de $X$ em $B$, sabemos que mesmo na partição comum mínima de $A$ e $B$ pelo menos um duo em (uma ocorrência de) $X$ em $A$ (ou em $B$, respectivamente) deve ser quebrada. O algoritmo visa "cortar" todas as substrings de $A$ e $B$ que possuem um número diferente de ocorrências. Isso motiva a seguinte definição.

Para duas strings $A$ e $X$, seja $\#substr(A, X)$ o número de todas as ocorrências da substring $X$ na string $A$. Para uma partição $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ e uma string $X$, denotamos por $\#partes(\mathcal{P}, X)$ o número de partes $P_i = X$ em $\mathcal{P}$. Denotamos por $duos(S)$ o conjunto de todos os duos de uma string $S$, lembrando que um duo é uma substring de comprimento 2.

O primeiro algoritmo proposto é denominado HS e seu pseudocódigo é descrito no Algoritmo 3.

Sobre o algoritmo HS, os autores concluem que, caso um procedimento exato para o Problema do *Hitting Set* Mínimo esteja disponível, o algoritmo é uma $2k$-aproximação para a partição comum mínima. Com isso, se substituirmos o procedimento ótimo para o

**Algoritmo 3:** HS

**Entrada:** duas strings compatíveis $A$ e $B$

`/* construa uma instância` $(U, \mathcal{S})$ `para o Problema do` ***Hitting Set*** `Mínimo: */`

1 $U \leftarrow duos(A) \cup duos(B)$

2 $T \leftarrow \{X \in \Sigma^* : \#substr(A, X) \neq \#substr(B, X)\}$

3 $\mathcal{S} \leftarrow \{duos(X) : X \in T\}$

`/* resolva (aproximadamente) o Problema do` ***Hitting Set*** `Mínimo: */`

4 $\Phi \leftarrow$ um *hitting set* para $(U, \mathcal{S})$

`/* transforme o` ***hitting set*** `em uma partição comum: */`

5 $\mathcal{A}, \mathcal{B} \leftarrow$ para cada duo $xy \in \Phi$, corte todas as ocorrências de $xy$ nas strings $A$, $B$

**Saída:** $(\mathcal{A}, \mathcal{B})$

Problema do *Hitting Set* Mínimo por uma $\alpha$-aproximação, o algoritmo HS torna-se uma $2k\alpha$-aproximação para o MCSP. Porém, o Problema do *Hitting Set* Mínimo é difícil de aproximar, sendo necessária uma solução alternativa.

A solução alternativa encontrada é o algoritmo Fast HS, que tem seu pseudocódigo descrito no Algoritmo 7. A notação necessária para compreender este algoritmo seria um pouco extensa para definir nesta seção, então reservamos as definições, provas e o pseudocódigo apenas para o Apêndice B.

A ideia do Fast HS é que, cortando os duos de strings que satisfazem certa condição, conseguimos obter um *hitting set* de tamanho no máximo duas vezes o tamanho do *hitting set* mínimo, o que nos dá um fator de aproximação bem definido, sabendo da relação entre os fatores de aproximação para o Problema do *Hitting Set* Mínimo e para o MCSP.

Assim, a conclusão a que se chega sobre o algoritmo Fast HS é que ele calcula uma $4k$-aproximação da partição comum mínima de $A$ e $B$.

Neste artigo, os autores também propõem uma implementação para o algoritmo Fast HS com complexidade de tempo $O(n)$. Como o foco de nossos estudos foi a análise das demonstrações sobre o fator de aproximação, não tratamos desta implementação de tempo linear.

O resumo da parte estudada deste artigo, contendo as demonstrações completas, se encontra no Apêndice B.3.

# 4 Utilizando o RMCSP em algoritmos para o TSbR

Nesta seção, descrevemos como foram implementados os algoritmos que utilizam o RMCSP, mais especificamente o algoritmo Greedy para este problema, proposto por Chrobak *et al.* [8], para encontrar partições comuns e manipulá-las a fim de resolver o problema de Transformação de Strings por Reversões.

## 4.1 Selection Sort a partir do RMCSP

O terceiro algoritmo aqui proposto consiste, inicialmente, em encontrar uma partição comum através do algoritmo Greedy para o RMCSP. A ideia é, a partir da partição comum $\pi = (\mathcal{P}, \mathcal{Q})$ encontrada, ordenar as partes usando, novamente, um algoritmo semelhante ao *Selection Sort*.

Começando com $i = 1$, sempre que $P_i \neq Q_i$, temos duas possibilidades, para algum $j > i$. Ou $P_j = Q_i^R$, caso em que uma reversão sobre $S$ coloca $P_j$ em sua posição correta, ou $P_j = Q_i$ , caso em que duas reversões sobre $S$ colocam $P_j$ em sua posição final. No final, teremos $\mathcal{P} = \mathcal{Q}$ e, consequentemente, $S = T$. Este algoritmo será denominado MCSP-1.

Por exemplo, para $S = 30132210$ e $T = 30231021$, a partição comum dada por Greedy é $\pi = (\mathcal{P}, \mathcal{Q})$, onde $\mathcal{P} = (3, 0132, 21, 0)$ e $\mathcal{Q} = (3, 0, 2310, 21)$. A primeira reversão aplicada coloca a parte 0 de $\mathcal{P}$ na segunda posição, ou seja, aplica 3<u>0132210</u>. Com isso, temos $\mathcal{P}' = (3, 0, 12, 2310)$. Agora, devemos colocar a parte 2310 de $\mathcal{P}'$ na terceira posição, o que requer duas reversões. Ou seja, 30<u>122310</u> e 30<u>1322</u>1. Agora, temos $\mathcal{P}'' = (3, 0, 2310, 21) = \mathcal{Q}$, então os algoritmos param após três reversões.

O algoritmo Greedy leva tempo $O(n^3)$ para encontrar uma partição comum entre as strings. Dada a partição comum, o algoritmo que coloca as partes nas posições corretas leva tempo $O(n^2)$. Assim, MCSP-1 tem tempo total $O(n^3)$.

## 4.2 Permutações com sinais a partir do RMCSP

O quarto algoritmo também começa com uma partição comum $\pi = (\mathcal{P}, \mathcal{Q})$ retornada pelo algoritmo Greedy e, então, transforma cada uma em uma permutação com sinais. Cada parte $Q_i$ de $\mathcal{Q}$ é renomeada para $+i$, o que resultará em $\mathcal{Q}$ sendo igual à permutação de identidade $\iota = (+1, +2, \ldots, +n)$, e uma parte $P_j$ em $\mathcal{P}$ é renomeada para $+i$ se $P_j = Q_i$, ou para $-i$ se $P_j = Q_i^R$.

Considere o mesmo exemplo de strings $S = 30132210$ e $T = 30231021$ de antes e sua partição comum $\pi = (\mathcal{P}, \mathcal{Q})$ dada por Greedy. Depois de converter $\pi$ em um par de permutações com sinais, teremos a permutação de identidade $\iota = (+1, +2, +3, +4)$ para $\mathcal{Q}$ e permutação $\pi = (+1, -3, +4, +2)$ para $\mathcal{P}$.

Após este processo, teremos uma instância do problema SbR com sinais, que podemos resolver otimamente [13]. Em seguida, retornamos a distância encontrada. Este algoritmo será denominado MCSP-2.

Renomear a partição comum como uma permutação com sinais leva tempo $O(n^2)$. O cálculo da distância de reversão para permutações com sinais (sem termos de fato a sequência ótima de reversões) leva tempo $O(n)$. Portanto, MCSP-2 também tem tempo total $O(n^3)$,

devido ao algoritmo GREEDY.

# 5 Resultados Experimentais

Nesta seção, tratamos dos resultados experimentais dos quatro algoritmos propostos neste relatório. Ressaltamos que os conjuntos de instâncias utilizados foram exatamente os mesmos descritos no relatório parcial.

Para o algoritmo MAPAS-B2, o único parâmetro adicional é o número de mapeamentos aleatórios gerados para a string $S$, denotado por $N$. Assim como em MAPAS, utilizamos $N = 100000$. Para o algoritmo BRKGA-B3, mantemos os parâmetros utilizados no algoritmo BRKGA-2, descrito do relatório anterior, por ter retornado melhores resultados em relação à combinação de parâmetros utilizada no algoritmo BRKGA-1. Assim, os parâmetros utilizados aqui foram: um tamanho da população $p = 1000$, a quantidade de gerações $g = 100$, a quantidade de indivíduos na elite a cada geração $q_e = 0.3p$, a quantidade de mutantes a cada geração $q_t = 0.3p$, e a probabilidade de um descendente herdas as características do pai de elite $\rho_e = 0.6$. Para algoritmo MCSP-2, utilizamos a ferramenta GRIMM [19] para calcular as soluções ótimas para o SbR com sinais.

A tabela 1 mostra os resultados experimentais obtidos para cada um dos 11 conjuntos de strings, um por linha. As primeiras quatro colunas representam os valores que definem a classe de equivalência das strings em cada conjunto: o comprimento $n$, o tamanho $k$ do alfabeto, o número de ocorrências $f$ de cada um dos símbolos e o número *opt* de reversões aplicadas a uma das strings de cada par.

O erro de um algoritmo é dado por $(d_{alg} - opt)/opt$, onde $d_{alg}$ é a distância média estimada pelo algoritmo para as strings de um determinado conjunto, que é o valor em cada célula.

A média dos erros, dada na linha MÉDIA ERRO, representa o quão longe de *opt* o algoritmo estimou a distância para as 1100 instâncias testadas.

Quanto menor o valor em MÉDIA ERRO, melhor será o desempenho do algoritmo nos conjuntos testados, e um valor negativo nesta linha significa que o algoritmo retornou, em média, soluções melhores (de menor valor) que *opt*, como ocorreu com o algoritmo BRKGA-2 na primeira etapa do projeto.

Analisando os resultados dos quatro novos algoritmos, e comparando-os com os resultados do relatório anterior, podemos ver que ambas as versões de MCSP têm resultados melhores que todas as versões de MAPAS, com MCSP-1 tendo um resultado muito próximo ao de BREAKS. Além disso, vemos que entre as duas versões de MCSP há uma diferença enorme na qualidade dos resultados, onde MCSP-1 só foi melhor que as versões de MAPAS, enquanto MCSP-2 só não se saiu melhor que as versões do BRKGA. Isso com certeza se deve ao fato

Tabela 1: Resultados experimentais dos algoritmos propostos.

| $n$ | $k$ | $f$ | *opt* | MCSP-1 | MCSP-2 | MAPAS-B2 | BRKGA-B3 |
|---|---|---|---|---|---|---|---|
| 20 | 4 | 5 | 10 | 11,14 | 7,86 | 8,49 | 6,85 |
| 30 | 6 | 5 | 10 | 16,29 | 11,38 | 15,50 | 9,80 |
| 40 | 8 | 5 | 15 | 24,40 | 16,78 | 24,15 | 14,99 |
| 50 | 10 | 5 | 15 | 27,37 | 18,94 | 32,00 | 17,51 |
| 60 | 12 | 5 | 20 | 35,73 | 24,52 | 41,43 | 23,13 |
| 70 | 14 | 5 | 20 | 38,39 | 25,82 | 49,87 | 24,21 |
| 80 | 16 | 5 | 25 | 47,37 | 32,08 | 59,00 | 30,65 |
| 90 | 18 | 5 | 25 | 48,39 | 32,18 | 67,06 | 31,42 |
| 100 | 20 | 5 | 25 | 51,30 | 33,60 | 75,81 | 33,71 |
| 50 | 25 | 2 | 15 | 26,89 | 18,18 | 19,36 | 15,74 |
| 100 | 50 | 2 | 25 | 49,85 | 32,02 | 46,93 | 27,71 |
| MÉDIA ERRO | | | | 77,90% | 20,27% | 99,55% | 10,80% |

de MCSP-2 utilizar um algoritmo ótimo para as permutações com sinais geradas a partir das partições comuns, mesmo que haja uma perda de qualidade no resultado (em relação ao ótimo) por conta da transformação da partição em permutação com sinais.

Com relação a MAPAS-B2, é fácil notar que este foi o algoritmo que retornou as piores soluções, entre todos os propostos. Se MAPAS já tinha o pior dos resultados entre os algoritmos da etapa anterior, o fato de MAPAS-B2 utilizar os breakpoints entre as strings para ranqueá-las, e não mais os resultados retornados por KS95, fez com que o resultado médio piorasse. Isso se deve ao fato de, apesar da intuição nos levar a crer que um número de breakpoints menor entre duas strings implicar em uma distância menor entre elas, isso nem sempre ser verdade. A única vantagem desta nova variação é o consumo de tempo, pois calcular a quantidade de breakpoints entre 100000 é extremamente menos custoso que calcular 100000 resultados por KS95.

Com relação à nova variação de BRKGA, BRKGA-B3, podemos dizer que ela foi muito interessante em relação ao consumo de tempo, mesmo retornando resultados piores que sua versão anterior, BRKGA-2. Como dito no parágrafo anterior, é muito menos custoso, em relação ao tempo, calcular o número de breakpoints entre duas strings, que leva tempo $O(n)$, do que calcular resultados por KS95, que leva tempo $O(n^2)$. Se levarmos em conta que esta nova variação se saiu melhor que a variação BRKGA-1, mas em um tempo muito menor, este novo algoritmo é uma boa alternativa, pois poderíamos executá-lo com populações maiores e/ou mais gerações e, talvez, obtermos resultados muito próximos aos de BRKGA-2 com muito menos tempo de processamento.

# 6 Estudo do Problema de Mapeamento de Strings com Preservação Máxima de Duos (MPSM)

Nesta seção, assim como fizemos para o MCSP, descreveremos o Problema de Mapeamento de Strings com Preservação Máxima de Duos (MPSM), trazendo as definições necessárias, os principais resultados existentes na literatura e, nas subseções posteriores, comentando as ideias por trás dos algoritmos propostos nos artigos selecionados para estudo.

## 6.1 Definições e resultados existentes sobre o MPSM

Sejam $S = s_1 s_2 \dots s_n$ e $T = t_1 t_2 \dots t_n$ duas strings de comprimento $n$, tal que $T$ é uma permutação de $S$. Um *mapeamento próprio* $\pi$ de $S$ para $T$ é um mapeamento bijetivo com $s_i = t_{\pi(i)}$, para todo $1 \leq i \leq n$. Dizemos que um duo $(s_i, s_{i+1})$ é *preservado* se $s_i$ é mapeado para $t_j$ e $s_{i+1}$ é mapeado para $t_{j+1}$, com $1 \leq i \leq j \leq n$.

Proposto por Chen *et al.* [6], o *Problema do Mapeamento de Strings com Preservação Máxima de Duos* (que chamaremos, daqui em diante, de MPSM – do inglês *Maximum Duo-Preservation String Mapping Problem*) consiste em encontrar um mapeamento próprio entre os símbolos de duas strings $S$ e $T$ que preserve a maior quantidade de duos. É importante destacar que a quantidade de duos preservados em cada string é igual, onde contamos como duos preservados por um mapeamento apenas os duos preservados em uma das strings, e não a soma dos duos preservados em cada uma delas. Assim como o MCSP, também existe uma variação do MPSM onde cada símbolo pode ter, no máximo, $k$ ocorrências em cada uma das strings, e denominamos essa variação do problema como *$k$-MPSM*.

Seja $OPT_{MPSM}(S, T)$ o número de duos preservados em uma única string em uma solução ótima para o MPSM para um par de strings compatíveis $S$ e $T$, ambas de comprimento $n$. Seja $OPT_{MCSP}(S, T)$ a cardinalidade da partição em uma solução ótima do MCSP para as mesmas strings $S$ e $T$. Podemos dizer que os problemas MPSM e MCSP são complementares, pois, sendo $\mathcal{P}$ uma partição de $S$ e $\mathcal{Q}$ uma partição de $T$, tal que o par $(\mathcal{P}, \mathcal{Q})$ é uma solução ótima para o MCSP, vale que $|\mathcal{P}| = |\mathcal{Q}| = OPT_{MCSP}(S, T) = n - OPT_{MPSM}(S, T)$. Em outras palavras, os problemas são complementares pois maximizar o número de duos preservados por um mapeamento próprio entre duas strings é equivalente a minimizar o número de quebras de uma partição comum (que também pode ser vista como um mapeamento próprio) entre estas strings, o que significa minimizar o tamanho de uma partição comum entre elas.

Como citado anteriormente, o MPSM foi proposto por Chen *et al.* [6] no mesmo trabalho em que propuseram dois problemas em grafos, chamados *CMIS* (de *Constrained Maximum Induced Subgraph problem*) e *CNIS* (de *Constrained Minimum Induced Subgraph*

*problem*), para os quais podemos reduzir em tempo polinomial os problemas MPSM e MCSP, respectivamente. Utilizando uma abordagem em Programação Linear, os autores formalizaram uma aproximação para o problema $k$-CMIS (CMIS com cada símbolo podendo ocorrer no máximo $k$ vezes), uma generalização do $k$-MPSM. Para $k = 2$, a solução proposta por eles foi uma 2-aproximação, enquanto que para $k \geq 3$, foi uma $k^2$-aproximação.

Boria *et al.* [2] provaram que o $k$-MPSM é APX-difícil mesmo para $k = 2$, significando que o problema não admite um esquema de aproximação em tempo polinomial (PTAS), a menos que $P = NP$. Além disso, este trabalho melhorou os fatores de aproximação de Chen *et al.* [6], propondo uma 1.6-aproximação para $k = 2$, uma 3-aproximação para $k = 3$, e uma 4-aproximação para o MPSM (onde não importa o valor de $k$), tendo, esta última, uma execução em tempo $O(n^{3/2})$.

Beretta *et al.* [1] estudaram a Tratabilidade por Parâmetro Fixo sobre MPSM e mostraram que o problema é tratável por parâmetro fixo (FPT) quando parametrizado pelo número de duos preservados.

Boria *et al.* [3] melhoraram o fator de aproximação do MPSM propondo um algoritmo de aproximação com fator 3.5, utilizando uma busca local. Brubach [4] apresentou uma 3.25-aproximação, utilizando um algoritmo baseado na correspondência entre *trios* (substrings de comprimento igual à 3).

Como último resultado relevante sobre o MPSM, citamos Dudek *et al.* [10], que provaram que, para qualquer $\epsilon > 0$, existe um algoritmo de $(2 + \epsilon)$-aproximação em tempo polinomial. Além disso, neste mesmo trabalho também foi proposto um algoritmo de 2.67-aproximação para o 3-MPSM, com tempo de execução $O(n^2)$.

## 6.2 Utilizando o emparelhamento de duos

O primeiro algoritmo para o MPSM estudado mais a fundo nesta pesquisa foi a 4-aproximação proposta por Boria *et al.* [2].

A ideia por trás do algoritmo é, primeiramente, construir um grafo bipartido cujo conjunto de vértices de um dos lados é o conjunto de duos da string $A$ e o conjunto de vértices do outro lado é o conjunto de duos da string $B$. Uma vez construído este grafo, nota-se que adicionando arestas entre vértices de lados diferentes, mas que correspondem ao mesmo duo, podemos formar emparelhamentos e que estes emparelhamentos correspondem a mapeamentos próprios dos símbolos de uma string nos símbolos da outra, de modo que o problema de emparelhamento neste grafo seja análogo ao MPSM.

Sendo assim, basta resolver os conflitos entre arestas destes emparelhamentos, uma vez que potencialmente surgem arestas que não podem coexistir em uma mesma solução.

Utilizando propriedades sobre o tamanho de um emparelhamento ótimo, é possível mostrar que o emparelhamento devolvido pelo algoritmo proposto tem tamanho igual a pelo menos um quarto do tamanho do emparelhamento ótimo.

Como os autores demonstram que o tamanho de um emparelhamento máximo no grafo criado é um limitante superior para a quantidade de duos preservado pelo mapeamento correspondente, é provada a 4-aproximação para o MPSM.

O resumo da parte estudada deste artigo, contendo as demonstrações completas, se encontra no Apêndice B.4.

## 6.3 Uma solução mais elegante – Busca local

O próximo algoritmo para o MPSM que foi estudado melhora o fator de aproximação de 4 para $3,5$ e é baseado na ideia de busca local a soluções "vizinhas" de uma solução viável para o problema, buscando sempre encontrar uma solução que seja um ótimo local [3].

É construído um grafo bipartido da mesma forma que no algoritmo anterior. A partir de um emparelhamento que seja uma soluçao viável para este grafo, a ideia é buscar soluções vizinhas à solução atual, que são soluções que contenham todas as arestas da solução atual, exceto uma. Caso uma das soluções vizinhas possua mais arestas do que a solução atual, esta passará a ser a atual, e a próxima busca vai ser em soluções vizinhas a ela. Quando não for mais possível aprimorar a solução, dizemos que chegamos a um ótimo local, que será a solução devolvida pelo algoritmo.

Analisando a quantidade de conflitos que pode haver entre arestas de uma solução ótima para o problema e a solução devolvida pelo algoritmo de busca local, para dada instância, é possível chegar a uma razão de $7/2$, resultando na $3,5$-aproximação.

O resumo da parte estudada deste artigo, contendo as demonstrações completas, se encontra no Apêndice B.5.

## 6.4 Utilizando o emparelhamento de trios

Seguindo a ideia da 4-aproximação proposta em [2], Brubach [4] estendeu a ideia do emparelhamento do grafo de duos, para um emparelhamento em um grafo de trios.

O algoritmo consiste em quatro etapas. Na primeira, se constrói um grafo bipartido ponderado a partir dos trios das strings de entrada, onde arestas referentes a trios formados por dois duos que se correspondem consecutivamente na outra string valem mais que aquelas referentes a trios em que apenas um duo é correspondente na outra string. Na segunda etapa, é encontrado um emparelhamento de peso máximo no grafo bipartido gerado. Na terceira etapa, é construído um grafo bipartido com os duos de ambas as strings, porém,

neste algoritmo, este grafo é ponderado a partir do emparelhamento encontrado para o grafo bipartido gerado com os trios. Na quarta e última etapa, o grafo bipartido ponderado de duos é utilizado para criar um mapeamento próprio entre os caracteres das strings de entrada, de modo que as arestas do grafo são removidas, ao passo que são mapeados caracteres de uma string na outra, seguindo algumas regras com relação ao peso as arestas.

Por fim, o mapeamento devolvido é uma solução viável para o MPSM e, a partir do peso das arestas removidas do grafo bipartido ponderado de duos, na última etapa do algoritmo, o autor prova que o número de duos preservado pelo mapeamento gerado é $4/13$ o peso total das arestas removidas do grafo de duos, e isso corresponde a uma $13/4 = 3,25$-aproximação.

O resumo da parte estudada deste artigo, contendo as demonstrações completas, se encontra no Apêndice B.6.

# 7 Considerações finais

Neste projeto de iniciação científica estudamos o Problema de Transformação de Strings por Reversões (TSbR), propusemos, implementamos e testamos vários algoritmos práticos para este problema. Além disso, estudamos resultados existentes para problemas relacionados, a fim de entender melhor a estrutura do TSbR.

Também estudamos algumas classes de instâncias para as quais podemos encontrar soluções ótimas, aplicando um algoritmo de força bruta. A ideia era encontrar soluções ótimas para algumas classes de strings de comprimento pequeno e tentar explorar propriedades destas entradas já sabendo qual a solução ótima. Porém, pelo fato do algoritmo de força bruta ser muito custoso, estas classes de strings puderam ser apenas de comprimento muito pequeno, e a análise dos resultados obtidos não levou a nenhum resultado relevante.

Com relação aos algoritmos implementados, os objetivos foram atingidos, pois a ideia era propor algoritmos práticos, a partir de estratégias gulosas e heurísticas, e comparar a qualidade das soluções devolvidas por eles. A descrição de todos os algoritmos desenvolvidos, bem como a análise comparativa, foi descrita detalhadamente nos dois relatórios científicos.

Por fim, destacamos que o bolsista apresentou trabalhos referentes a esta pesquisa nos seguintes eventos:

- Primeira Etapa do 28º Simpósio Internacional de Iniciação Científica e Tecnológica da USP (SIICUSP), em outubro de 2020;
- XIII Simpósio de Iniciação Científica da UFABC, em novembro de 2020;
- Etapa Internacional do 28º SIICUSP, em março de 2021.

Além disso, um artigo referente ao trabalho foi submetido e aceito no 40º Concurso de

Trabalhos de Iniciação Científica (CTIC), da Sociedade Brasileira de Computação (SBC), sendo publicado na Revista Eletrônica de Iniciação Científica em Computação (REIC), em junho de 2021 [18].

# Referências


[1] S. Beretta, M. Castelli, and R. Dondi. Parameterized tractability of the maximum-duo preservation string mapping problem. *Theoretical Computer Science*, 646:16 – 25, 2016.

[2] N. Boria, A. Kurpisz, S. Leppänen, and M. Mastrolilli. Improved approximation for the maximum duo-preservation string mapping problem. In D. Brown and B. Morgenstern, editors, *Algorithms in Bioinformatics*, pages 14–25, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44753-6.

[3] N. Boria, G. Cabodi, P. Camurati, M. Palena, P. Pasini, and S. Quer. A 7/2-Approximation Algorithm for the Maximum Duo-Preservation String Mapping Problem. In R. Grossi and M. Lewenstein, editors, *27th Annual Symposium on Combinatorial Pattern Matching (CPM 2016)*, volume 54 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:8, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. ISBN 978-3-95977-012-5.

[4] B. Brubach. Further improvement in approximating the maximum duo-preservation string mapping problem. In M. Frith and C. N. Storm Pedersen, editors, *Algorithms in Bioinformatics*, pages 52–64, Cham, 2016. Springer International Publishing.

[5] L. Bulteau and C. Komusiewicz. Minimum Common String Partition Parameterized by Partition Size is Fixed-parameter Tractable. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2014)*, pages 102–121, Philadelphia, PA, USA, 2014. Society for Industrial and Applied Mathematics.

[6] W. Chen, Z. Chen, N. F. Samatova, L. Peng, J. Wang, and M. Tang. Solving the maximum duo-preservation string mapping problem with linear programming. *Theoretical Computer Science*, 530:1 – 11, 2014.

[7] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of Orthologous Genes via Genome Rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(4):302–315, 2005.

[8] M. Chrobak, P. Kolman, and J. Sgall. The Greedy Algorithm for the Minimum Common String Partition Problem. In K. Jansen, S. Khanna, J. D. P. Rolim, and D. Ron, editors, *Proceedings of the 7th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'2004), and 8th International Workshop on Randomization and Computation (RANDOM'2004)*, pages 84–95, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[9] P. Damaschke. Minimum common string partition parameterized. In K. A. Crandall and

J. Lagergren, editors, *Algorithms in Bioinformatics*, pages 87–98, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87361-7.

[10] B. Dudek, P. Gawrychowski, and P. Ostropolski-Nalewaja. A family of approximation algorithms for the maximum duo-preservation string mapping problem. *arXiv preprint arXiv:1702.02405*, 2017.

[11] G. Fertin, A. Labarre, I. Rusu, É. Tannier, and S. Vialette. *Combinatorics of Genome Rearrangements*. Computational Molecular Biology. The MIT Press, London, England, 2009.

[12] A. Goldstein, P. Kolman, and J. Zheng. Minimum Common String Partition Problem: Hardness and Approximations. In R. Fleischer and G. Trippen, editors, *Proceedings of the 15th International Symposium on Algorithms and Computation (ISAAC'2004)*, pages 484–495, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[13] S. Hannenhalli and P. A. Pevzner. Transforming Cabbage into Turnip: Polynomial Algorithm for Sorting Signed Permutations by Reversals. *Journal of the ACM*, 46(1):1–27, 1999.

[14] P. Kolman. Approximating reversal distance for strings with bounded number of duplicates. In J. Jedrzejowicz and A. Szepietowski, editors, *Mathematical Foundations of Computer Science 2005*, pages 580–590, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31867-5.

[15] P. Kolman and T. Waleń. Approximating Reversal Distance for Strings with Bounded Number of Duplicates. *Discrete Applied Mathematics*, 155(3):327–336, 2007.

[16] P. Kolman and T. Waleń. Reversal Distance for Strings with Duplicates: Linear Time Approximation Using Hitting Set. In T. Erlebach and C. Kaklamanis, editors, *Proceedings of the 4th International Workshop on Approximation and Online Algorithms (WAOA'2006)*, pages 279–289, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[17] K. M. Swenson, M. Marron, J. V. Earnest-Deyoung, and B. M. E. Moret. Approximating the true evolutionary distance between two genomes. *J. Exp. Algorithmics*, 12, Aug. 2008.

[18] G. d. S. Teixeira and C. N. Lintzmayer. Algorithms for transforming strings by reversals. *Revista Eletrônica de Iniciação Científica em Computação*, 19(2), jun. 2021. URL https://sol.sbc.org.br/journals/index.php/reic/article/view/2077.

[19] G. Tesler. GRIMM: Genome Rearrangements Web Server. *Bioinformatics*, 18(3):492–493, 2002.

# A Definições e notação sobre o TSbR

Uma *permutação* é uma tupla $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, de comprimento $|\pi| = n$, com elementos $\pi_i \in \{1, 2, \ldots, n\}$, onde $|\pi_i| \neq |\pi_j| \leftrightarrow i \neq j$, isto é, em que não há elementos repetidos. Em uma *permutação com sinais*, cada elemento tem um sinal "+" ou "−" associado. Em uma *permutação sem sinais*, os sinais são omitidos.

A *permutação identidade* $\iota = (1, 2, 3, \ldots, n)$ é a permutação de $n$ elementos na qual $|\pi_i| < |\pi_j| \leftrightarrow i < j$. Com esta representação, podemos considerar que o problema de transformar uma permutação em outra, de mesmo tamanho, se reduz a transformar uma permutação na permutação identidade, i. e., *ordená-la*.

Dada uma permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, sua *permutação inversa*, denotada por $\pi^{-1}$, é a permutação obtida ao trocarmos o valor de cada um dos elementos $\pi_i$ por suas respectivas posições $i$, com $1 \leq i \leq n$. Isto significa que $\pi^{-1}_{\pi_i} = i$, para $1 \leq i \leq n$.

Definimos o *grupo simétrico* $S_n$ como sendo o conjunto que contém todas as $n!$ permutações sem sinais possíveis de comprimento $n$, ou as $2^n n!$ permutações com sinais possíveis.

Seja uma permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, de comprimento $n$. A versão estendida de $\pi$, denotada por $\pi^l$, é obtida ao adicionarmos a $\pi$ os elementos $\pi_0 = 0$ e $\pi_{n+1} = n + 1$, ou seja, $\pi^l = (0, \pi_1, \pi_2, \ldots, \pi_n, n + 1)$.

Seja $\pi^l$ a versão estendida da permutação $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. Dizemos que um par $(\pi^l_i, \pi^l_{i+1})$, com $0 \leq i \leq n$, é um *breakpoint de reversão* de $\pi$ se $\pi^l_{i+1} - \pi^l_i \neq 1$. Quando $|\pi^l_{i+1} - \pi^l_i| > 1$, dizemos que o par $(\pi^l_i, \pi^l_{i+1})$ é um *breakpoint forte*.

Uma *string* $S = s_1 s_2 \ldots s_n$ é uma sequência de *elementos* $s_i$, com $1 \leq i \leq n$, tirados de um *alfabeto* $\Sigma = \{0, 1, 2, \ldots\}$ de *símbolos*, que pode conter símbolos repetidos. Se o alfabeto tem tamanho $|\Sigma| = k$, uma string descrita sobre este alfabeto é chamada de *string k-ária*. Uma *string com sinais* é uma string onde cada elemento possui um sinal "+" ou "−" associado. Quando não há informação sobre a orientação dos elementos, os sinais são omitidos e a string é dita *sem sinais*. Note que, por definição, uma permutação é uma string na qual os elementos do alfabeto não se repetem. O *comprimento* de uma string $S = s_1 s_2 \ldots s_n$ é denotado por $|S|$ e representa a quantidade de posições para símbolos que ela possui, i.e., $|S| = n$.

Uma *substring* de uma string $S$ de comprimento $n$, é uma string $S' = s_i s_{i+1} \ldots s_{j-1} s_j$, onde $1 \leq i \leq j \leq n$, e $s_i s_{i+1} \ldots s_{j-1} s_j$ aparece consecutivamente em $S$.

Um *prefixo* de uma string $S$ é uma substring de $S$ que contém seu primeiro elemento. O *maior prefixo comum* entre duas strings $S$ e $T$ é a substring maximal $L = s_1 s_2 \ldots s_{p-1} s_p$ tal que, para todo $1 \leq i \leq p$, $s_i = t_i$. O comprimento do maior prefixo comum entre $S$ e $T$ é denotado por $lcp(S, T)$.

Um *sufixo* de uma string $S$ é uma substring de $S$ que contém seu último elemento. O *maior sufixo comum* entre duas strings $S$ e $T$ é a substring maximal $L = s_j s_{j+1} \dots s_{n-1} s_n$ tal que, para todo $j \leq i \leq n$, $s_i = t_i$. O comprimento do maior sufixo comum entre $S$ e $T$ é denotado por $lcs(S, T)$.

Denotamos por $f(S, x)$ a quantidade de ocorrências do símbolo $x$ na string $S$. Da mesma forma, denotamos por $f(S, S')$ a quantidade de ocorrências da *substring* $S'$ na string $S$.

Duas strings $S$ e $T$ são ditas *compatíveis* se $(i)$ ambas têm o mesmo comprimento $n$, $(ii)$ ambas são descritas sobre o mesmo alfabeto $\Sigma$, e $(iii)$ $f(S, x) = f(T, x)$, para todo $x \in \Sigma$. Nos problemas aqui tratados, sempre consideraremos que as strings analisadas são compatíveis.

Uma *classe de equivalência*, denotada por $\mathcal{L} = (a_0, a_1, \dots, a_{k-1})$, é o conjunto de strings compatíveis com exatamente $a_i$ ocorrências de cada símbolo $i \in \Sigma$, onde $|\Sigma| = k$. Note que, por esta definição, $n = a_0 + a_1 + \dots + a_{k-1}$, i.e., a soma da quantidade de ocorrências de cada símbolo é igual ao comprimento das strings.

Uma substring maximal contendo apenas símbolos $x$, para algum $x \in \Sigma$, é chamada de *bloco* (em strings com sinais, os símbolos devem ter o mesmo sinal). A quantidade de blocos em uma string $S$ é denotada por $bl(S)$. Dadas duas strings compatíveis $S$ e $T$, denotamos $bl_{max} = max\{bl(S), bl(T)\}$ e $bl_{min} = min\{bl(S), bl(T)\}$.

Definimos a *string identidade* $S_\iota$ de uma classe de equivalência como sendo a string $S_\iota = s_1 s_2 \dots s_n$, onde $s_i \leq s_{i+1}$, para $1 \leq i < n$, ou seja, é uma string ordenada de forma não decrescente pelo valor de seus elementos.

Uma *reversão* $\rho(i, j)$, com $1 \leq i < j \leq n$, representa o rearranjo $(1\ 2\ \dots\ i-2\ i-1\ \underline{j\ j-1 \dots i+1\ i}\ j+1\ j+2\ \dots\ n-1\ n)$. Isso significa que, quando aplicada a uma string $S = s_1 s_2 \dots s_n$, a reversão inverte o segmento que vai da posição $i$ à posição $j$, transformando a string $S$ em $S \cdot \rho(i, j) = s_1\ \dots\ s_{i-1}\ \underline{s_j\ s_{j-1} \dots s_{i+1}\ s_i}\ s_{j+1}\ \dots\ s_n$.

Uma *reversão de prefixo*, denotada $\rho_p(j)$, com $1 < j \leq n$, é uma reversão que contém, necessariamente, o primeiro elemento da string, i.e., equivale a $\rho(1, j)$. Uma *reversão de sufixo*, denotada $\rho_s(i)$, com $1 \leq i < n$, é uma reversão que contém, necessariamente, o último elemento da string, i.e., equivale a $\rho(i, n)$.

Uma *reversão com sinais* $\bar{\rho}(i, j)$, com $1 \leq i < j \leq n$, é o rearranjo $(1\ 2\ \dots\ i-2\ i-1\ \underline{-j\ -(j-1) \dots -(i+1)\ -i}\ j+1\ j+2\ \dots\ n-1\ n)$. Isso significa que, quando aplicada a uma string com sinais $S$, a reversão com sinais inverte o segmento que vai da posição $i$ à posição $j$ de $S$, invertendo, também, o sinal de cada elemento deste segmento, transformando a string $S$ em $S \cdot \bar{\rho}(i, j) = s_1\ \dots\ s_{i-1}\ \underline{-s_j\ -s_{j-1} \dots -s_{i+1}\ -s_i}\ s_{j+1}\ \dots\ s_n$.

Uma *reversão de prefixo com sinais*, denotada $\bar{\rho}_p(j)$, com $1 < j \leq n$, é uma reversão com sinais que contém, necessariamente, o primeiro elemento da string com sinais, i.e., equivale a $\bar{\rho}(1, j)$. Já uma *reversão de sufixo com sinais*, denotada $\bar{\rho}_s(i)$, com $1 \leq i < n$, é uma reversão

com sinais que contém, necessariamente, o último elemento da string com sinais, i.e., equivale a $\bar{\rho}(i, n)$.

Seja uma string $S = s_1 s_2 \dots s_n$, com ou sem sinais. A *string reversa* de $S$, que denotamos por $S^R$, é a string resultante da aplicação da reversão $\rho(1,\, n)$ em $S$.

Duas strings $S = s_1 \dots s_n$ e $T = t_1 \dots t_n$ são *idênticas*, o que denotamos por $S = T$, se $s_i = t_i$ para cada $i \in \{1, \dots, n\}$. Dizemos que as strings $S$ e $T$ são *congruentes*, o que denotamos por $S \cong T$, se $S = T$ ou $S = T^R$.

*Problemas de Transformação por Operações de Rearranjo* são formulados, de maneira mais geral, da seguinte forma: dadas duas strings compatíveis e um conjunto de operações de rearranjo permitidas, encontre o menor custo de uma sequência de operações de rearranjo que transforme uma string na outra. O conjunto das operações permitidas é o que faz a diversidade desta classe de problemas.

Seja $c(\ell) = \ell^\alpha$ o custo de um rearranjo de comprimento $\ell$, isto é, o custo de um rearranjo que envolve $\ell$ elementos. Na abordagem tradicional definimos $\alpha = 0$, de modo que todos os rearranjos têm o mesmo custo unitário e o objetivo se resume a encontrar o menor número de rearranjos necessários para transformar uma string em outra.

Um *modelo de rearranjo* $\beta$ é o conjunto de operações rearranjo que são permitidas durante o processo de transformação de strings. Escreveremos $\beta$ como "$(\ |p|ps)(r|\bar{r})$", onde $p$ significa *prefixo*, $ps$ significa *prefixo e sufixo*, $r$ significa *reversão*, $\bar{r}$ significa *reversão com sinais*. Por exemplo, se $\beta = p\bar{r}$, então as operações permitidas são as reversões de prefixo com sinais.

Dadas duas strings compatíveis $S$ e $T$ e um modelo $\beta$, a *distância* $d_\beta(S, T)$ entre $S$ e $T$ é o menor custo de uma sequência de rearranjos em $\beta$ que transforma $S$ em $T$. Portanto, um *Problema de Transformação de Strings* tem como objetivo encontrar a distância entre $S$ e $T$ dadas.

Dado um modelo de rearranjo $\beta$ e um inteiro $n$, definimos o *diâmetro* $D_\beta(n)$ como o maior valor de $d_\beta(S, T)$ entre todos os pares possíveis de strings compatíveis $S$ e $T$ de tamanho $n$.

Consideraremos os *acrônimos* para os problemas da forma (TSB) ( |P|PS) (R|$\bar{\text{R}}$), onde TSB significa "Transformação de Strings por", P significa "Prefixo", PS significa "Prefixo e Sufixo", R significa "Reversões" e $\bar{\text{R}}$ significa "Reversões com Sinais". Por exemplo, TSBPSR é o acrônimo para o problema de *Transformção de Strings por Reversões de Prefixo e Sufixo*.

Um *duo* é uma string de comprimento igual a 2. Geralmente, esta definição é utilizada para nos referirmos a pares de elementos subsequentes em uma string, sendo os *duos de uma string* $S$ cada uma das substrings de comprimento 2 em $S$.

Para uma string $S = s_1 \dots s_n$, denotamos por $duos(S)$ o conjunto de duos da string $S$, isto é, $duos(S) = \{s_i s_{i+1} \mid 1 \leq i < n\}$.

O conceito de *breakpoints de reversão*, muito utilizado em permutações, foi adaptado para

strings considerando seus duos. Suponha que queremos transformar a string $S$ na string $T$. Se $S$ contém mais duos $xy$ que $T$, com $x, y \in \Sigma$, então cada ocorrência de $xy$ que $S$ contém a mais que $T$ deve ser separada em algum ponto da transformação. Cada um destes duos extras, como os $xy$ a mais em $S$ do que em $T$, são denominados *breakpoints de reversão de $S$ com relação a $T$*. Uma diferença importante em relação aos breakpoints em permutações é que, em strings, não identificamos as posições dos breakpoints.

Seja $\delta(x) = x$, se $x > 0$, e $\delta(x) = 0$, caso contrário. Para a identificação e contagem dos breakpoints de reversão entre duas strings, consideramos um elemento especial $w < 0$, adicionado no início de ambas as strings, e um elemento especial $z > |\Sigma| - 1$, adicionado no final de ambas as strings. O *número de breakpoints de reversão* entre duas strings compatíveis $S$ e $T$, denotado por $b_\rho(S, T)$, é dado por

$$b_\rho(S,T) = \sum_{w \leq x < y \leq z} \delta\left(f(S,xy) + f(S,yx) - f(T,xy) - f(T,yx)\right) + \sum_{x \in \Sigma} \delta\left(f(S,xx) - f(T,xx)\right) .$$

# B Resumos dos Artigos Estudados

Neste apêndice, trazemos as partes em que focamos nossos estudos sobre os problemas relacionados MCSP e MPSM, contendo todas as definições e demonstrações completas sobre os fatores de aproximação dos algoritmos. O apêndice visa complementar a definição informal dada no texto do relatório para cada um dos artigos estudados.

## B.1 Chrobak et al. (2005)

Neste artigo [8], os autores estudam um algoritmo guloso para o MCSP, denominado Greedy. O algoritmo basicamente constrói uma partição comum extraindo iterativamente a substring comum mais longa entre as strings de entrada. Mais precisamente, seu pseudocódigo pode ser descrito como no Algoritmo 4.

**Algoritmo 4:** Greedy

**Entrada:** duas strings compatíveis $A$ e $B$

1 inicialmente, todas as letras em $A$ e $B$ são não marcadas e $\mathcal{P}$ e $\mathcal{Q}$ estão vazias
2 **enquanto** *existem letras não marcadas em* $A$ **faça**
3 $S \leftarrow$ a substring comum mais longa entre $A$ e $B$ que não contém letras marcadas (em caso de empate, escolha arbitrariamente)
4 $S^A$, $S^B \leftarrow$ ocorrências de $S$ em $A$ e em $B$, respectivamente
5 designe $S^A$ como uma parte de $\mathcal{P}$ em $A$ e $S^B$ como uma parte de $\mathcal{Q}$ em $B$
6 marque todas as letras em $S^A$ e $S^B$

**Saída:** $(\mathcal{P}, \mathcal{Q})$

Por exemplo, se $A = cdabcdabceab$, $B = abceabcdabcd$, então Greedy primeiro marca a substring comum $abcdabc$, depois $ab$ e, em seguida, três substrings de uma letra $c$, $d$ e $e$, e a partição resultante é

$$\langle (c, d, abcdabc, e, ab), (ab, c, e, abcdabc, d) \rangle,$$

enquanto a partição comum ótima é $\langle (cdabcd, abceab), (abceab, cdabcd) \rangle$. Conforme ilustrado pelo exemplo anterior, a partição comum calculada por Greedy não é necessariamente ótima. A questão a ser estudada, então, é qual o fator de aproximação de Greedy para o MCSP e suas variantes. Os autores provam os seguintes resultados:

**Teorema 2.**

*(a) O fator de aproximação de* Greedy *para* MCSP *está entre* $\Omega(n^{0,43})$ *e* $O(n^{0,69})$.

*(b) Para o* 4-MCSP*, o fator de aproximação de* Greedy *é pelo menos* $\Omega(\log n)$.

*(c) Para o* 2-MCSP*, o fator de aproximação de* Greedy *é igual a* 3.

Estes resultados se estendem à variação *com sinais* do problema, o SMCSP, onde cada letra tem um sinal + ou − associado a ela [7, 12].

Denotamos por $A = a_1 a_2 \dots a_n$ e $B = b_1 b_2 \dots b_n$ as duas strings de entrada arbitrárias, mas fixas, de GREEDY. Sem perda de generalidade, assumimos que $A$ e $B$ são compatíveis. Se $\pi$ é uma partição comum de $A$ e $B$, então o *tamanho* de $\pi$ é o número de partes em $\pi$, que denotaremos por $\#partes(\pi)$. O tamanho de uma partição comum mínima de $A$ e $B$ é denotado por $\texttt{MCSP}(A, B)$.

Normalmente lidamos com ocorrências de letras em strings, em vez das próprias letras. Por *substring* estaremos nos referindo (a menos que indicado de outra forma) a uma ocorrência específica de uma string em outra. Assim, identificamos uma substring $S = a_p a_{p+1} \dots a_{p+s}$ de $A$ pelo conjunto de índices $\{p, p+1, \dots, p+s\}$ e escrevemos $S = \{p, p+1, \dots, p+s\}$, onde $|S| = s + 1$ é o *comprimento* de $S$. Obviamente, a mesma convenção se aplica a substrings de $B$. Se $S$ é uma *substring comum* de $A$ e $B$, usamos as notações $S^A$ e $S^B$ para distinguir as ocorrências de $S$ em $A$ e em $B$, respectivamente.

### B.1.1 Partições como funções

Suponha que seja dada uma bijeção $\xi : [n] \to [n]$, onde $[n] = \{1, 2, \dots, n\}$, que *preserva letras* de $A$ e $B$, isto é, $b_{\xi(i)} = a_i$ para todo $i \in [n]$. Um par de posições consecutivas $i, i+1 \in [n]$ é chamado de *quebra* de $\xi$ se $\xi(i+1) \neq \xi(i) + 1$. Denotaremos por $\#quebras(\xi)$ o número de quebras em $\xi$. Para uma substring comum $S$ de $A$ e $B$, digamos $S = a_p a_{p+1} \dots a_{p+s} = b_q b_{q+1} \dots b_{q+s}$, dizemos que $\xi$ *respeita* $S$ se $\xi$ mapeia letras consecutivas de $S^A$ em letras consecutivas em $S^B$, ou seja, $\xi(i) = i + q - p$ para $i \in S^A$.

Uma bijeção $\xi$ que preserva letras induz uma partição comum (também denotada por $\xi$, para simplificar) cujas partes são substrings de comprimento máximo de $A$ que não contêm quebras de $\xi$. A partição obtida desta forma não possui partes "desnecessárias", isto é, $\#partes(\xi) = \#quebras(\xi) + 1$. E vice-versa, se $\pi = \langle P, Q \rangle$ é uma partição comum de $A$ e $B$, podemos pensar em $\pi$ como uma bijeção que preserva letras $\pi : [n] \to [n]$ que respeita cada parte da partição. Obviamente, temos $\#partes(\pi) \geq \#quebras(\pi) + 1$. Essa relação é utilizada ao longo do artigo, identificando partições comuns com suas bijeções correspondentes.

### B.1.2 Partições de comuns de referência

Seja $\pi$ uma partição comum mínima de $A$ e $B$ (que pode não ser única, mas escolhemos uma das partições comuns mínimas de forma arbitrária). No primeiro passo, GREDDY tem garantia de encontrar uma substring $S_1$ de comprimento pelo menos o comprimento máximo de uma parte em $\pi$. Para a análise de GREEDY, gostaríamos de ter uma estimativa semelhante para todos os passos posteriores também. No entanto, já no segundo passo não há garantia

de que GREEDY encontre uma substring tão longa quanto a segunda parte mais longa em $\pi$, uma vez que esta parte pode se sobrepor a $S_1$ e, neste ponto da execução, pode estar parcialmente marcada (em $A$ ou $B$). Para obter uma estimativa mais justa de $|S_t|$, para $t > 1$, é introduzida uma *partição comum de referência* correspondente de $A$ e $B$, que respeita todas as partes $S_1, \ldots, S_{t-1}$ selecionadas por GREEDY nos passos 1 a $t-1$. Esta partição pode ser gradualmente "deteriorada" (em comparação com a partição comum mínima de $A$ e $B$), ou seja, pode incluir mais partes e suas partes podem ficar mais curtas. Além disso, ela pode não incluir uma partição comum mínima dos segmentos de letras não marcadas. No entanto, as partições comuns de referência fornecem uma estimativa útil do "dano" causado por GREEDY quando faz escolhas erradas (i.e., quando marca substrings que não estão na partição ótima).

Seja $g$ o número de passos de GREEDY sobre $A$ e $B$. Para $t = 0, 1, \ldots, g$, a *partição comum de referência* $\rho_t$ é definida indutivamente como segue. Inicialmente, $\rho_0 = \pi$. Considere qualquer $t = 1, \ldots, g$. Suponha que $S_t^A = \{p, p+1, \ldots, p+s\}$ e $S_t^B = \{q, q+1, \ldots, q+s\}$. Definimos a função $\delta : S_t^A \to S_t^B$ tal que $\delta(i) = i + q - p$ para $i \in S_t^A$. Então, $\rho_t$ é definida por

$$\rho_t(i) = \begin{cases} \delta(i), & \text{para } i \in S_t^A, \\ \rho_{t-1}(\delta^{-1}\rho_{t-1})^{\ell(i)}(i), & \text{para } i \in [n] - S_t^A, \end{cases} \tag{1}$$

onde $\ell(i) = \min\{\lambda \geq 0 : \rho_{t-1}(\delta^{-1}\rho_{t-1})^{\lambda}(i) \notin S_t^B\}$. Mostramos que cada $\rho_t$ é bem definida e também limitamos o aumento do número de quebras de $\rho_{t-1}$ para $\rho_t$:

**Lema 3.** *Para todo $t = 0, 1, \ldots, g$, (a) $\rho_t$ é uma partição comum de $A$ e $B$; (b) $\rho_t$ respeita $S_1, \ldots, S_t$; e (c) se $t > 0$, então $\#quebras(\rho_t) \leq \#quebras(\rho_{t-1}) + 4$.*

*Observação* 1. A parte $(b)$ do lema implica que após cada passo $t$ do algoritmo, toda parte na partição de referência $\rho_t$ ou está completamente marcada ou está completamente desmarcada; as partes do primeiro tipo são chamadas de *marcadas*, enquanto as partes do segundo tipo são chamadas de *não marcadas*.

*Demonstração.* A demonstração do lema é por indução. Para $t = 0$, as partes $(a)$ e $(b)$ são trivialmente verdadeiras. Suponha que $t > 0$ e que o lema seja válido para $t - 1$. Para simplificar a notação, seja $S = S_t$, $\rho = \rho_{t-1}$ e $\rho' = \rho_t$.

Considere um grafo bipartido $G \subseteq [n] \times [n]$, com arestas $(i, \rho(i))$, para $i \in [n]$, e $(i, , \delta(i))$, para $i \in S^A$. Esses dois tipos de arestas são chamados *$\rho$-arestas* e *$\delta$-arestas*, respectivamente.

Sejam $\overline{S}^A = [n] - S^A$ e $\overline{S}^B = [n] - S^B$. Nesta prova, para evitar a introdução de notação adicional, pensaremos em $S^A$ e $\overline{S}^A$ como os conjuntos de vértices da parte superior de $G$ e $S^B$ e $\overline{S}^B$ como os vértices da parte inferior. Então, qualquer vértice em $\overline{S}^A$ ou $\overline{S}^B$ incide em uma

$\rho$-aresta, e cada vértice em $S^A$ ou $S^B$ incide em uma $\rho$-aresta e em uma $\delta$-aresta. Assim, $G$ é uma coleção de caminhos e ciclos disjuntos de vértices cujas arestas alternam entre $\rho$-arestas e $\delta$-arestas. Nós os chamamos de *$G$-caminhos* e *$G$-ciclos*. Todos os $G$-ciclos têm comprimento par e contêm apenas vértices de $S^A$ e $S^B$. Todos os $G$-caminhos máximos têm comprimentos ímpares, começam em $\overline{S}^A$ e terminam em $\overline{S}^B$, e seus vértices internos estão em $S^A$ ou $S^B$. O $G$-caminho começando em $i \in \overline{S}^A$ tem a forma

$$i,\ \rho(i), \delta^{-1}\rho(i),\ \rho\delta^{-1}\rho(i), \ldots,\ \rho(\delta^{-1}\rho)^{\ell(i)}(i).$$

Assim, para $i \in \overline{S}^A$, $\rho'(i)$ é simplesmente o outro extremo do $G$-caminho que começa em $i$. Isso implica que $\rho'$ é um-para-um e preserva letras e, portanto, é de fato uma partição comum. A condição $(b)$ segue imediatamente da hipótese indutiva e da definição de $\rho'$. Resta provar $(c)$.

**Lema 4.** *Suponha que $i$, $i+1$ é uma quebra de $\rho'$. Então, uma das seguintes condições se aplica:*

**(B0)** *Exatamente um entre $i$ ou $i+1$ está em $S^A$.*

**(B1)** *$i, i+1 \in \overline{S}^A$ e existe $\lambda \leq \min\{\ell(i),\ \ell(i+1)\}$ tal que $(\delta^{-1}\rho)^\lambda(i)$, $(\delta^{-1}\rho)^\lambda(i+1)$ é uma quebra de $\rho$.*

**(B2)** *$i, i+1 \in \overline{S}^A$ e existe $\lambda \leq \min\{\ell(i), \ell(i+1)\}$ tal que exatamente um entre $\rho(\delta^{-1}\rho)^\lambda(i)$ ou $\rho(\delta^{-1}\rho)^\lambda(i+1)$ pertence a $S^B$.*

Nos referimos a quebras dos tipos (B0), (B1), (B2), respectivamente, como quebras induzidas pelos extremos de $S^A$, quebras induzidas pelas quebras dentro de $S^A$ (apenas se $i$, $i+1$ é uma nova quebra) e quebras induzidas pelos extremos de $S^B$.

*Demonstração.* Se exatamente um entre $i$ ou $i+1$ está em $S^A$, o caso (B0) vale. Como $i, i+1$ nunca é uma quebra em $\rho'$ se $i$ e $i+1$ estão em $S^A$, assumimos que $i, i+1 \in \overline{S}^A$ para o resto da demonstração.

Considere o maior inteiro $\lambda \leq \min\{\ell(i), \ell(i+1)\}$ para o qual $(\delta^{-1}\rho)^\lambda(i)$ e $(\delta^{-1}\rho)^\lambda(i+1)$ são consecutivos em $S^A$, isto é, $(\delta^{-1}\rho)^\lambda(i+1) = (\delta^{-1}\rho)^\lambda(i) + 1$. Ressaltamos que não é necessariamente verdade que $(\delta^{-1}\rho)^h(i+1) = (\delta^{-1}\rho)^h(i) + 1$ para $h = 1, \ldots, \lambda$; esses índices podem divergir e se encontrar posteriormente, qualquer número de vezes. Seja $j = (\delta^{-1}\rho)^\lambda(i)$. Temos dois subcasos. Se $\rho(j+1) \neq \rho(j)+1$, então $j$, $j+1$ é uma quebra de $\rho$, então a condição (B1) é satisfeita. Se $\rho(j+1) = \rho(j) + 1$, então pelo menos um entre $\rho(j)$ e $\rho(j+1)$ deve estar em $S^B$, caso contrário, $i, i+1$ não seria uma quebra de $\rho'$. Mas também não podemos ter $\rho(j)$, $\rho(j+1) \in S^B$, uma vez que $(\delta^{-1}\rho)^{\lambda+1}(i)$, $(\delta^{-1}\rho)^{\lambda+1}(i+1)$ seriam consecutivos em $S^A$, violando a escolha de $\lambda$. Portanto, o caso (B2) é válido. □

Agora, completamos a demonstração da parte $(c)$ do Lema 3. Não há quebras de $\rho'$ dentro de $S^A$, e temos no máximo duas quebras do tipo (B0) correspondendo aos extremos de $S^A$. Pela disjunção de $G$-caminhos e $G$-ciclos, existem no máximo duas quebras de $\rho'$ do tipo (B2), cada uma correspondendo a um extremo de $S^B$. Da mesma forma, cada quebra de $\rho$ (dentro ou fora de $S^A$) induz no máximo uma quebra de $\rho'$ do tipo (B1). Isso implica em $(c)$, e a prova do lema está completa. □

Observe que, na demonstração, não utiliza-se o fato de $S$ ter comprimento máximo. Portanto, a construção de $\rho_t$ pode ser usada para converter qualquer partição comum $\pi$ em outra partição $\pi'$ que respeite uma determinada substring comum $S$ e que tenha no máximo quatro quebras a mais que $\pi$.

### B.1.3 Limite superior para o MCSP

Nesta seção, é demonstrado que o fator de aproximação de Greedy é $O(n^{0,69})$. A demonstração usa as partições comuns de referência, introduzidas na Seção B.1.2, para controlar o comprimento das substrings comuns selecionadas por Greedy.

Para $p \geq q \geq 1$, definimos $H(p, q)$ como o menor número $h$ com a seguinte propriedade: para quaisquer strings de entrada $A$ e $B$, se em algum passo $t$ de Greedy existem no máximo $p$ símbolos não marcados em $A$ e no máximo $q$ partes não marcadas na partição de referência atual $\rho_t$, então Greedy faz no máximo mais $h$ passos até parar (portanto, sua partição comum final tem no máximo $t + h$ partes). Por conveniência, permitimos $p$ e $q$ não inteiros na definição. Observe que $H(p, q)$ é não descrescente em ambas as variáveis.

Antes de provar o limite superior $O(n^{0,69})$, os autores demonstram um limite ligeiramente mais fraco, mas mais simples, $O(n^{0,75})$. O Lema 3 nos dá imediatamente uma recorrência

$$H(p, q) \leq H\left(p\left(1 - \frac{1}{q}\right), q + 3\right) + 1,$$

sempre que ambos os valores de $H$ estão definidos, já que em um passo de Greedy a substring comum mais longa tem pelo menos $p/q$ letras. Assim, pelo menos $p/q$ letras serão marcadas na próxima partição, e o número de partes não marcadas na partição de referência aumenta em no máximo 3.

Provaremos que, para $p \geq q$ e uma constante $C$ suficientemente grande, temos

$$H(p, q) \leq Cp^{\frac{3}{4}}q^{\frac{1}{4}} - \frac{1}{3}q. \tag{2}$$

Para $q = 1$ isso é trivial, pois Greedy encontra uma única parte não marcada que é uma substring comum. Assumindo $q \geq 2$, seguimos por indução em $p$. Escolhemos uma constante

universal suficientemente grande $C$ tal que, para todo $q \leq p < 5q$, o lado direito é pelo menos $p$, que é um limite superior trivial para $H(p, q)$ . Para $p \geq 5q \geq 10$, temos $p(1-1/q) \geq q+3$; assim, podemos usar a hipótese indutiva (2) e a recorrência para obter

$$\begin{aligned} H(p,\, q) &\leq H\left(p\left(1-\frac{1}{q}\right),\, q+3\right)+1 \\ &\leq Cp^{\frac{3}{4}}\left(1-\frac{1}{q}\right)^{\frac{3}{4}}(q+3)^{\frac{1}{4}}-\frac{1}{3}(q+3)+1 \\ &\leq Cp^{\frac{3}{4}}q^{\frac{1}{4}}-\frac{1}{3}q. \end{aligned}$$

A última desigualdade segue de $(q-1)^{\frac{3}{4}}(q+3)^{\frac{1}{4}} \leq (q-1)^{\frac{1}{2}}[(q-1)^{\frac{1}{4}}(q+3)^{\frac{1}{4}}] \leq (q-1)^{\frac{1}{2}}(q+1)^{\frac{1}{2}} \leq q$. Isso completa o passo de indução e a prova. O limite que provamos implica que $H(p, q) \leq O(p^{0,75})q$. Assim, se a entrada de Greedy consiste em duas strings $A$ e $B$, de comprimento $n$, o número de partes na partição de Greedy é no máximo $H(n, \texttt{MCSP}(A, B)) = O(n^{0,75}) \cdot \texttt{MCSP}(A, B)$.

A ideia da demonstração do limite melhorado é considerar, em vez de um passo de Greedy, um número de passos proporcional ao número de partes na partição ótima original, e mostrar que, durante esses passos, Greedy marca uma fração constante da string de entrada. Isso produz uma recorrência melhorada para $H(p, q)$.

**Lema 5.** *Seja $p \geq q \geq 1$, e suponha que $p$ e $q$ satisfaçam $p \geq 9q/5 + 3$. Então, $H(p, q) \leq H(5p/6, (3q + 5)/2) + (q + 5)/6$.*

*Demonstração.* Considere uma computação de Greedy sobre $A$ e $B$ onde, após algum passo $t$ (ou seja, com $t$ partes já marcadas), existem $p$ símbolos não marcados em $A$, e $q$ partes de referência não marcadas de $\rho_t$. Denotamos esses partes de referência por $R_1$, $R_2, \ldots,$ $R_q$, em ordem não crescente de comprimento, ou seja, $|R_z| \geq |R_{z+1}|$, para $z = 1, \ldots, q - 1$. Analisamos a computação de Greedy começando no passo $t+1$. Seja $g$ o número de passos adicionais que Greedy executa. Nosso objetivo é mostrar que

$$g \leq H\left(\frac{5}{6}p, \frac{3q+5}{2}\right) + \frac{q+5}{6}. \tag{3}$$

Uma vez que o limite é monótono em $p$ e $q$, não precisamos considerar o caso de menos de $q$ partes não marcadas ou menos de $p$ símbolos não marcados. Se $g \leq (q + 5)/6$, a desigualdade (3) é trivialmente válida, então, para o resto da demonstração assumimos que $g > (q + 5)/6$.

Seja $T_i = S_{t+i}$ a substring comum selecionada por Greedy no passo $t + i$. Dizemos que Greedy *atinge* $R_z$ no passo $t + i$ se $T_i$ se sobrepõe a $R_z$, seja em $A$ ou em $B$, i.e., ou se

$T_i^A \cap R_z^A \neq \emptyset$ ou se $T_i^B \cap R_z^B \neq \emptyset$.

*Afirmação* 1. Para todo $j = 1, \ldots, g$, o comprimento total das partes $R_1, \ldots, R_q$ em $A$ que são atingidas por GREEDY nos passos $t+1, \ldots, t+j$ é no máximo $6\sum_{i=1}^{j} |T_i|$.

*Demonstração.* Estimamos o comprimento total das partes $R_z$ em $A$ que são atingidas no passo $t+i$, mas não foram atingidas nos passos $t+1, \ldots, t+i-1$. O comprimento total das partes que estão contidas em $T_i^A$ e $T_i^B$ é no máximo $2|T_i|$. Existem até quatro partes que são atingidas parcialmente, mas pela escolha gulosa de $T_i$, cada uma tem comprimento no máximo $|T_i|$, e a afirmação vale. □

*Afirmação* 2. $6\sum_{i=1}^{\lfloor (q+5)/6 \rfloor} |T_i| \geq p$.

*Demonstração.* Escolha $l$ para ser o menor inteiro tal que $6\sum_{i=1}^{l} |T_i| \geq p$. Uma vez que $\sum_{i=1}^{g} |T_i| = p$, temos que $l$ é bem definido e $l \leq g$. Para $j = 1, \ldots, l$, defina $\chi_j$ como o índice maximal para o qual $\sum_{x=1}^{\chi_j} |R_x| \leq 6\sum_{i=1}^{j-1} |T_i|$. Uma vez que $6\sum_{i=1}^{l-1} |T_i| < p = \sum_{x=1}^{q} |R_x|$, todos $\chi_j$ são bem definidos, e $\chi_l < q$. Destacamos também que $\chi_1 = 0$. Para cada $j = 1, \ldots, l$, a Afirmação 1 implica que uma das partes $R_1, \ldots, R_{\chi_j+1}$ não é atingida por nenhuma das partes $T_1, \ldots, T_{j-1}$ e, portanto, pela definição de GREEDY e da ordenação das partes $R_z$, $|T_j| \geq |R_{\chi_j+1}|$. Considerando novamente a ordenação das partes $R_z$, temos $6|T_j| \geq |R_{\chi_j+1}| + \cdots + |R_{\chi_j+6}|$. Concluímos que $\chi_{j+1} \geq \chi_j + 6$, para $j = 1, \ldots, l-1$. Isso, por sua vez, implica que $q \geq \chi_l + 1 \geq 6l - 5$. Portanto, $l \leq (q+5)/6$, e a Afirmação 2 vale, pela escolha de $l$ e o fato de ser um inteiro. □

Pela Afirmação 2, após exatamente $\lfloor (q+5)/6 \rfloor$ passos, GREEDY marca pelo menos $p/6$ letras, então o número de letras não marcadas restantes é no máximo $p' = 5p/6$. Pelo Lema 3, o número de partes de referência não marcadas aumenta em no máximo três em cada passo (uma vez que uma nova parte é marcada), então o número de partes de referência não marcadas induzidas por GREEDY nesses $\lfloor (q+5)/6 \rfloor$ passos é no máximo $3\lfloor (q+5)/6 \rfloor \leq (q+5)/2$. Assim, o número total de partes de referência não marcadas após esses passos é no máximo $q' = q + (q+5)/2 = (3q+5)/2$. A condição no lema garante que $H(p', q')$ é definido, então, por indução, temos que o número total de passos é no máximo $H(p', q') + (q+5)/6$. Isso completa a prova da desigualdade (3) e do lema. □

Finalmente, provamos o limite superior no item $(a)$ do Teorema 2.

**Teorema 6.** GREEDY *é uma* $O(n^\gamma)$*-aproximação para o* MCSP*, onde* $\gamma = \log \frac{3}{2} / \log \frac{9}{5} \approx 0,69$.

*Demonstração.* Provaremos, por indução em $p$, que para $p \geq q$ e uma constante $C$ suficientemente grande,

$$H(p,q) \leq Cp^{\gamma}(q+5)^{1-\gamma} - \frac{1}{3}q.$$

Escolhemos uma constante universal $C$ tal que, para todo $q \leq p < 9q/5+3$, o lado direito é pelo menos $p$ e, portanto, a desigualdade é válida. Para $p \geq 9q/5+3$, pelo Lema 5, pela hipótese indutiva e pela escolha de $\gamma$, temos

$$\begin{aligned} H(p,q) &\leq H\left(\frac{5}{6}p, \frac{3q+5}{2}\right) + \frac{q+5}{6} \\ &\leq C\left(\frac{5}{6}p\right)^{\gamma}\left(\frac{3}{2}(q+5)\right)^{1-\gamma} - \frac{1}{3}\cdot\frac{3q+5}{2} + \frac{q+5}{6} \\ &= Cp^{\gamma}(q+5)^{1-\gamma} - \frac{1}{3}q. \end{aligned}$$

Sejam $A$ e $B$ strings de entrada de comprimento $n$, e $\texttt{MCSP}(A,B) = m$. Então, o número de partes na partição de GREEDY é no máximo $H(n,m) = O(n^{\gamma})m$, e o teorema vale. □

### B.1.4 Limite inferior para o MCSP

Nesta seção, os autores mostram que o fator de aproximação de GREEDY é $\Omega(n^{1/\log_2 5}) = \Omega(n^{0,43})$. Para $i = 0, 1, \ldots$, primeiro construímos as strings $C_i$, $D_i$, $E_i$, $F_i$ como segue. Inicialmente, $C_0 = a$ e $D_0 = b$. Suponha que já temos $C_i$ e $D_i$, e seja $\Sigma^i$ o conjunto das letras usadas em $C_i$, $D_i$. Defina um novo alfabeto $\Sigma'_i$ que tem uma nova letra, digamos $a'$, para cada $a \in \Sigma_i$. Primeiro criamos as strings $E_i$ e $F_i$ substituindo todas as letras $a \in \Sigma_i$ em $C_i$ e $D_i$, respectivamente, pelas letras correspondentes $a' \in \Sigma'_i$. Então, sejam

$$C_{i+1} = C_iD_iE_iD_iC_i \quad \text{e} \quad D_{i+1} = D_iE_iF_iE_iD_i.$$

Para cada $i$, consideramos a instância de strings $A_i = C_iD_i$ e $B_i = D_iC_i$. Por exemplo, $E_0 = c$, $F_0 = d$, $A_0 = ab$, $B_0 = ba$, $C_1 = abcba$, $D_1 = bcdcb$, $A_1 = abcbabcdcb$ e $B_1 = bcdcbabcba$, etc.

Seja $n = 2 \cdot 5^i$. Temos $|A_i| = |B_i| = n$ e $\texttt{MCSP}(A_i, B_i) \leq 2$. Afirmamos que a partição comum de GREEDY de $A_i$ e $B_i$ tem $2^{i+2} - 2 = \Omega(n^{1/\log_2 5})$ partes.

Assumimos aqui que GREEDY não especifica como os empates são decididos, ou seja, sempre que uma substring mais longa pode ser escolhida de duas ou mais maneiras diferentes, podemos decidir qual escolha GREEDY faz.

A demonstração é por indução. Para $i = 0$, GREEDY produz duas substrings, como

afirmado. Para $i \geq 0$,

$$A_{i+1} = C_i D_i E_i D_i C_i D_i E_i F_i E_i D_i \quad \text{e} \quad B_{i+1} = D_i E_i F_i E_i D_i C_i D_i E_i D_i C_i.$$

Existem três substrings comuns de comprimento $5^{i+1}$ : $C_i D_i E_i D_i C_i$, $D_i E_i F_i E_i D_i$, e $E_i D_i C_i D_i E_i$, e não existem mais substrings comuns. Para justificar isso, usamos o fato de que o alfabeto de $C_i$ e $D_i$ é disjunto do alfabeto de $E_i$ e $F_i$. Suponha que $S$ seja uma substring comum de comprimento pelo menos $5^{i+1}$. Para ter este comprimento, $S$ deve conter ou o primeiro ou o segundo $E_i$ de $A_{i+1}$. Agora, temos alguns casos dependendo de qual desses dois $E_i$'s está contido em $S$, e onde está mapeado em $B_{i+1}$, através da ocorrência de $S$ em $B_{i+1}$. Se $S$ contém o primeiro $E_i$, então, pela suposição sobre os alfabetos, este $E_i$ deve ser mapeado em $E_i F_i E_i$ ou no último $E_i$ em $B_{i+1}$. Se estiver mapeado em $E_i F_i E_i$, então $S$ deve ser $E_i D_i C_i D_i E_i$. Se estiver mapeado no último $E_i$ em $B_{i+1}$, então $S$ deve ser $C_i D_i E_i D_i C_i$. No último caso, $S$ contém o segundo $E_i$ em $A_{i+1}$. Pelas mesmas considerações do primeiro caso, é fácil mostrar que $S$ deve ser ou $D_i E_i F_i E_i D_i$ ou $E_i D_i C_i D_i E_i$.

Tendo desempatado, suponha que GREEDY marca a substring $E_i D_i C_i D_i E_i$. As strings modificadas são

$$C_i D_i \overline{E_i D_i C_i D_i E_i} F_i E_i D_i \quad \text{e} \quad D_i E_i F_i \overline{E_i D_i C_i D_i E_i} D_i C_i,$$

onde a linha sobre elas indica as partes marcadas por GREEDY. Na primeira string, os segmentos não marcados são $A_i$ e $A'_i D_i$, e na segunda string os segmentos não marcados são $B_i$ e $D_i B'_i$, onde $A'_i = F_i E_i$ e $B'_i = E_i F_i$ são idênticos a $A_i$ e $B_i$, respectivamente, mas com as letras renomeadas. O argumento no parágrafo anterior e a disjunção dos alfabetos implicam que o comprimento máximo de uma substring comum não marcada é $5^i$. Desempatamos novamente e fazemos GREEDY fazer a correspondência dos dois $D_i$'s em $F_i E_i D_i$ e $D_i E_i F_i$, e as strings resultantes têm a forma

$$A_i \overline{E_i D_i C_i D_i E_i} A'_i \overline{D_i} \quad \text{e} \quad \overline{D_i} B'_i \overline{E_i D_i C_i D_i E_i} B_i.$$

Agora, temos dois pares de substrings não marcadas $\{A_i, B_i\}$ e $\{A'_i, B'_i\}$. Estes dois pares de strings possuem alfabetos disjuntos e serão processados por GREEDY independentemente um do outro. Por indução, GREEDY produz $2^{i+2} - 2$ partes de $A_i$ e $B_i$, e o mesmo número de $A'_i$ e $B'_i$. Portanto, obtemos o total de $2(2^{i+2} - 2) + 2 = 2^{i+3} - 2$ partes.

## B.2 Kolman e Walen (2007a)

Neste artigo [15], os autores mostram que uma modificação simples do algoritmo GREEDY, proposto por Chrobak *et al.* [8], denominada REFINED GREEDY, é uma $O(k^2)$-aproximação para $k$-MCSP, o que também implica em uma $O(k^2)$-aproximação para $k$-TSbR.

Aqui, destacamos algumas definições que serão utilizadas. Se $S$ é uma substring comum de $A$ e $B$, usamos as notações $S^A$ e $S^B$ para distinguir entre as ocorrências de $S$ (ou de $S$ reversa, $S^R$) em $A$ e $B$; se $S$ ocorrer mais de uma vez em $A$, então $S^A$ se refere a uma ocorrência arbitrária, mas fixa, de $S$ em $A$ e uma convenção análoga se aplica à string $B$. Dadas duas partições $\mathcal{A} = (A_1, \ldots, A_m)$ e $\mathcal{B} = (B_1, \ldots, B_{m'})$, uma substring comum de $\mathcal{A}$ e $\mathcal{B}$ é uma string $S$ tal que $S$ é uma substring comum de $A_i$ e $B_j$, para algum par de índices $i$ e $j$. Um *duo* é uma string de comprimento dois. *Cortar* um duo $a_i a_{i+1}$ de uma parte $P = a_j \ldots a_k$ de uma partição de $A$, para $j \leq i < k$, significa substituir a parte $P$ na partição por duas partes $P_1 = a_j \ldots a_i$ e $P_2 = a_{i+1} \ldots a_k$. Para uma substring $S = a_i \ldots a_j$ de $A = a_1 \ldots a_n$, com $i > 1$, dizemos que $a_{i-1}a_i$ é um *duo de fronteira (esquerda)* de $S$, e da mesma forma, se $j < n$, $a_j a_{j+1}$ é um *duo de fronteira (direita)* de $S$.

Para o $k$-MCSP sem sinais, o pseudocódigo de REFINED GREEDY é descrito como no Algoritmo 5.

**Algoritmo 5:** REFINED GREEDY

**Entrada:** duas strings compatíveis $A$ e $B$

1 $\mathcal{A} \leftarrow (A), \mathcal{B} \leftarrow (B)$
2 **enquanto** *existem partes não marcadas em* $\mathcal{A}$ *e* $\mathcal{B}$ **faça**
3 $S \leftarrow$ a substring comum mais longa de $\mathcal{A}$ e $\mathcal{B}$ que não sobrepõe partes marcadas anteriormente
4 marca $S^A$ em $\mathcal{A}$ e $S^B$ em $\mathcal{B}$
5 corta os duos de fronteira de $S^A$ em $\mathcal{A}$ e os duos de fronteira de $S^B$ em $\mathcal{B}$
6 corta, nas partes não marcadas de $\mathcal{A}$ e $\mathcal{B}$, *todas* as ocorrências dos duos $\delta \in \Phi$, onde $\Phi$ é o conjunto de duos de fronteira de $S^A$ e $S^B$

**Saída:** $(\mathcal{A}, \mathcal{B})$

Para estender o algoritmo para o $k$-MCSP com sinais e para o $k$-RMCSP, além de considerar substrings comuns no que diz respeito à relação de equivalência $\cong$, a diferença é que nas etapas de corte, não cortamos apenas todas as ocorrências de $\delta \in \Phi$ mas também todas as ocorrências de $\delta^R$.

Para dar um exemplo, considere uma instância do 4-MCSP,

$$A = abxyuvafxyuvddddhefxyuvebxyuvgggg,$$
$$B = abxyuvddddafxyuvhefxyuvggggebxyuv.$$

O Refined Greedy marca primeiro a substring $S_1 = xyuvdddd$ (neste exemplo, usamos uma linha sobre os símbolos para denotar a marcação) e corta todas as ocorrências não marcadas dos duos de $\Phi = \{fx, dh, bx, da\}$. Na segunda iteração, Refined Greedy procura a substring não marcada mais longa nas partições $\mathcal{A} = (ab,\ xyuvaf,\ \overline{xyuvdddd},\ hef,\ xyuveb,\ xyuvgggg)$ e $\mathcal{B} = (ab,\ \overline{xyuvdddd},\ af,\ xyuvhef,\ xyuvggggeb,\ xyuv)$, marca a substring $S_2 = xyuvgggg$ e corta os duos de $\Phi = \{ge\}$. Na terceira iteração, o algoritmo procura a substring não marcada mais longa nas partições $\mathcal{A} = (ab,\ xyuvaf,\ \overline{xyuvdddd},\ hef,\ xyuveb,\ \overline{xyuvgggg})$ e $\mathcal{B} = (ab,\ \overline{xyuvdddd},\ af,\ xyuvhef,\ \overline{xyuvgggg},\ eb,\ xyuv)$, marca a substring $S_3 = xyuv$ e corta os duos de $\Phi = \{xa, ch\}$. Ao final, Refined Greedy gera a partição comum

$$\mathcal{P} = \langle (ab, xyuv, af, xyuvdddd, hef, xyuv, eb, xyuvgggg),\\ (ab, xyuvdddd, af, xyuv, hef, xyuvgggg, eb, xyuv)\rangle.$$

Porém, a partição comum ótima $\mathcal{P}_{\text{OPT}}$ para esta instância tem seis partes:

$$\mathcal{P}_{\text{OPT}} = \langle (abxyuv, afxyuv, dddd, hefxyuv, ebxyuv, gggg),\\ (abxyuv, dddd, afxyuv, hefxyuv, gggg, ebxyuv)\rangle.$$

Comparado a Greedy, em um muito alto nível, a vantagem de Refined Greedy é que, ao introduzir cortes adicionais a cada passo, o algoritmo restringe a propagação de erros, que são causados pela escolha gulosa de uma substring comum.

**Teorema 7.** Refined Greedy *é uma $2k^2$-aproximação para o $k$-MCSP com e sem sinais, e uma $2(2k-1)^2$-aproximação para o $k$-RMCSP.*

*Demonstração.* A saída do algoritmo é claramente uma partição comum. Só temos que provar o limite em sua qualidade. Para simplificar a apresentação, provamos a afirmação em detalhes para o $k$-MCSP sem sinais e, em seguida, descrevemos resumidamente as modificações necessárias para o $k$-MCSP com sinais e o $k$-RMCSP.

É conveniente estender as noções de partição e partição comum de strings para sequências de strings. A *partição da sequência* de strings $\mathcal{A} = (A_1, \ldots, A_l)$ é uma sequência de strings $A_{1,1}, \ldots, A_{1,k_1}, A_{2,1}, \ldots, A_{2,k_2} \ldots, A_{l,1}, \ldots, A_{l,k_l}$, de modo que $A_i = A_{i,1} \ldots, A_{i,k_i}$ para $i \in [l]$. Para duas sequências de strings, a partição comum é definida de forma análoga à de duas strings.

**Observação 1.** *Seja $(\mathcal{Q}, \mathcal{R})$ uma partição comum das sequências de strings $\mathcal{A}$ e $\mathcal{B}$, e seja $\delta$ qualquer duo que aparece em $\mathcal{Q}$ e $\mathcal{R}$. Seja $\mathcal{Q}'$ a partição de $\mathcal{A}$ que é obtida de $\mathcal{Q}$ cortando todas as ocorrências do duo $\delta$, e seja $\mathcal{R}'$ a partição de $\mathcal{B}$ que é obtida de $\mathcal{R}$ cortando todas as ocorrências do duo $\delta$. Então, $(\mathcal{Q}', \mathcal{R}')$ é uma partição comum de $\mathcal{A}$ e $\mathcal{B}$.*

*Demonstração.* Como $\mathcal{Q}$ é uma permutação de $\mathcal{R}$, cada parte $P$ de $\mathcal{Q}$ que contém $\delta$ também aparece em $\mathcal{R}$, e vice-versa. Assim, se cortarmos todas as ocorrências de $\delta$ em $\mathcal{Q}$ e $\mathcal{R}$, as novas partições resultantes $\mathcal{Q}'$ e $\mathcal{R}'$ serão novamente permutações uma da outra. □

Seja $\pi = (\mathcal{P}, \mathcal{Q})$ uma partição comum mínima de $A$ e $B$, sendo $m$ seu tamanho e $\Delta$ o conjunto de todos os duos de fronteira das partes em $\mathcal{P}$ e em $\mathcal{Q}$. Seja $T$ o número de passos do algoritmo. Iremos construir iterativamente partições comuns $\pi_i$ de $A$ e $B$ que nos ajudarão a estimar o tamanho da partição comum encontrada por Refined Greedy. Definimos $\pi_1$ como a partição comum derivada de $\pi$ cortando *todas* as ocorrências de todos os duos em $\Delta$ (o fato de que $\pi_1$ é um partição segue da Observação 1). As quebras em $\pi_1$ são chamadas de quebras *iniciais*. Para instâncias de $k$-MCSP, o número de partes em $\pi_1$ é no máximo $k$ vezes maior do que o número de partes em $\pi$ (se uma letra $a$ aparecer como letra mais à esquerda em uma parte de $\pi$, então há no máximo $k$ partes em $\pi_1$ com $a$ como a letra mais à esquerda) e, portanto, o número de quebras iniciais é no máximo $km - 1$.

Seja $S_i$ a substring que Refined Greedy usou na iteração $i$ e seja $\Phi_i$ o conjunto de duos de fronteira de $S_i^A$ e $S_i^B$. Para uma iteração $i \geq 1$ de Refined Greedy, definimos $\pi_{i+1}$ como a partição comum derivada de $\pi_i$ cortando todas as ocorrências de todos os duos em $\Phi_i$. Para facilitar a referência, denotamos por $\mathcal{A}_i$ e $\mathcal{B}_i$ os conjuntos $\mathcal{A}$ e $\mathcal{B}$ no início da iteração $i$, por $s_i$ a primeira posição de $S_i^A$ em $A$, por $t_i$ a última posição de $S_i^A$ em $A$, por $s_i'$ a primeira posição de $S_i^B$ em $B$, e por $t_i'$ o última posição de $S_i^B$ em $B$.

**Observação 2.** *Para toda iteração $i$ e para todo $0 \leq l < |S_i| - 1$: o par $s_i + l$, $s_i + l + 1$ é uma quebra inicial de $A$ se e somente se o par $s_i' + l$, $s_i' + l + 1$ é uma quebra inicial de $B$.*

*Demonstração.* A observação segue da definição de $\pi_1$ e quebras iniciais: se uma ocorrência de um duo é cortada em $\pi_1$, então todas as ocorrências deste duo são cortadas. □

Dada uma quebra $l$, $l + 1$ de uma partição de $A$ e uma substring $S = a_i \dots a_j$ de $A$, dizemos que a substring $S$ *passa sobre a quebra* $l, l+1$ se $i \leq l < j$. A Observação 2 pode ser declarada informalmente da seguinte forma: se a parte $S_i^A$ passa sobre uma ou mais quebras iniciais, então a parte $S_i^B$ passa sobre o mesmo número de quebras iniciais e, além disso, as posições relativas das quebras iniciais em $S_i^A$ e $S_i^B$ são as mesmas.

Sejam $\mathcal{A}_i' \subseteq \mathcal{A}_i$ e $\mathcal{B}_i' \subseteq \mathcal{B}_i$ os subconjuntos de strings não marcadas de $\mathcal{A}_i$ e $\mathcal{B}_i$, respectivamente, no início da fase $i$, e seja $\pi_i'$ a restrição de $\pi_i$ a $\mathcal{A}_i'$ e $\mathcal{B}_i'$. A Observação 2 implica na seguinte afirmação importante.

**Observação 3.** *Para todo $i$, $\pi_i'$ é uma partição comum de $\mathcal{A}_i'$ e $\mathcal{B}_i'$.*

*Demonstração.* A demonstração é por indução. Para $i = 1$, nada é marcado, então $\mathcal{A}_1' = \{A\}$, $\mathcal{B}_1' = \{B\}$, $\pi_1' = \pi_1$ e a afirmação é óbvia. Para $i > 1$, as Observações 1 e 2 implicam

que as partes de $\pi_i$ correspondentes à parte recém-marcada $S^A_{i-1}$ são iguais às partes de $\pi_i$ correspondentes à parte recém-marcada $S^B_{i-1}$. Observando que fora de $S^A_{i-1}$ e $S^B_{i-1}$, cortes dos mesmos duos (i.e., duos de $\Phi_{i-1}$) são usados para obter $\pi'_i$ a partir de $\pi'_{i-1}$ e $(\mathcal{A}'_i, \mathcal{B}'_i)$ a partir de $(\mathcal{A}'_{i-1}, \mathcal{B}'_{i-1})$, a demonstração está concluída. □

**Lema 8.** *Para todo $i$,*

- *a parte $S_i = a_{s_i} \dots a_{t_i}$ é uma parte inteira em $\pi_i$; ou*
- *a parte $S_i$ passa sobre uma quebra inicial.*

*Demonstração.* O lema segue da Observação 3 e da natureza gulosa de Refined Greedy: para cada substring comum $S$ de $\mathcal{A}'_i$ e $\mathcal{B}'_i$ que não satisfaz nenhuma das condições do lema, existe outra substring comum mais longa $S'$ de $\mathcal{A}'_i$ e $\mathcal{B}'_i$ tal que $S$ é uma substring própria de $S'$. □

O Lema 8 nos fornece uma ferramenta para limitar o número de quebras na partição comum $\pi_T$ após o último passo do algoritmo. Se Refined Greedy escolhe para $S_i$ uma parte inteira de $\pi_i$, então $\pi_{i+1} = \pi_i$ e não há novas quebras em $\pi_{i+1}$, com relação a $\pi_i$. Se Refined Greedy escolhe para $S_i$ uma substring que passa sobre uma quebra inicial, então precisamos de no máximo $2k$ cortes (na string $A$) para obter $\pi_{i+1}$ a partir de $\pi_i$ e carregamos todos eles até a quebra inicial sobre a qual $S^A_i$ passa. Após o último passo do algoritmo, existem no máximo $(2k+1)(km-1)$ quebras em $\pi_T$ (lembrando de que $km-1$ é um limite superior do número de quebras iniciais).

Por construção, a partição comum $\pi_T$ é um refinamento da partição comum $\langle \mathcal{A}_T, \mathcal{B}_T \rangle$ computada por Refined Greedy. Assim, o número de partes em $\pi_T$ é um limite superior do número de partes em $\langle \mathcal{A}_T, \mathcal{B}_T \rangle$ e o fator de aproximação do algoritmo é no máximo $k(2k+1)$. Para melhorar um pouco este fator, observamos que se Refined Greedy escolheu em $L$ passos substrings que passam sobre as quebras iniciais (note que $L \leq km-1$), então existem no máximo $2kL + (km-1-L) \leq 2k^2m - 1$ quebras em $\langle \mathcal{A}_T, \mathcal{B}_T \rangle$, o que implica no fator de aproximação desejado.

Para o $k$-MCSP com sinais e o $k$-RMCSP, só precisamos ajustar a demonstração para refletir o fato de que agora uma substring $S$ de $A$ pode ter correspondência com uma substring $T$ de $B$ mesmo se $S \neq T$, mas $S = T^R$. Assim, na Observação 1, cortamos não apenas todas as ocorrências do duo $\delta$, mas também todas as ocorrências do duo $\delta^R$. Para obter a partição comum $\pi_1$ a partir de $\pi$, para cada $\delta \in \Delta$ cortamos todas as ocorrências de $\delta$ assim como todas as ocorrências de $\delta^R$; para o $k$-MCSP com sinais, o número de quebras em $\pi_1$ aumenta novamente no máximo $k$ vezes, para o $k$-RMCSP aumenta no máximo $2k-1$ vezes. Na Observação 2, distinguimos se $S^A_i = S^B_i$ ou se $S^A_i = (S^B_i)^R$. No último caso, contamos as posições relativas das quebras iniciais em $S^B_i$ de trás para frente (ou seja, a afirmação é: $s_i + l$,

$s_i + l + 1$ é uma quebra inicial de $A$ se e somente se o par $t'_i - l - 1$, $t'_i - l$ é uma quebra inicial de $B$); o primeiro caso é como antes. Para $k$-MCSP com sinais, o número de duos cortados em $A$ em uma iteração é no máximo $2k$, para $k$-RMCSP é no máximo $2(2k-1)$. □

Considerando a relação entre o MCSP com sinais e o TSbR com sinais, e entre o RMCSP e o TSbR sem sinais, obtemos o seguinte teorema.

**Teorema 9.** *Existe uma $4k^2$-aproximação de tempo polinomial para o $k$-TSbR com sinais, e uma $8(2k-1)^2$-aproximação de tempo polinomial para o $k$-TSbR sem sinais.*

Com relação ao tempo de execução do Refined Greedy, apenas ressaltamos que uma implementação simples e ingênua do algoritmo é executada em tempo $O(n^3)$.

Neste mesmo artigo, os autores propõem um algoritmo chamado Educated Greedy que também é uma $O(k^2)$-aproximação para o $k$-MCSP e suas variantes, mas que pode ser implementado em tempo polinomial. Como nosso foco foi o estudo da prova do fator de aproximação, nos concentramos apenas no algoritmo Refined Greedy.

## B.3 Kolman e Walen (2007b)

No *Problema do Hitting Set Mínimo*, temos um conjunto $U$ e uma coleção $\mathcal{S}$ de subconjuntos de $U$, isto é, $\mathcal{S} = \{S_1, \ldots, S_k\}$ tal que $S_i \subseteq U$ para $i = 1, \ldots, k$. A tarefa é encontrar um *hitting set mínimo* para $\mathcal{S}$, que é o menor conjunto $H \subseteq U$ tal que $H \cap S_i = \emptyset$, para cada $i \in 1, \ldots, k$. O Problema de Hitting Set Mínimo é equivalente ao Problema de Cobertura de Conjuntos Mínima.

Neste artigo [16], o autores utilizam um algoritmo para o Problema do Hitting Set Mínimo como um procedimento para lidar com o MCSP. A ideia por trás do algoritmo é simples. Dadas as strings $A$ e $B$ e uma string $X$ tal que que o número de ocorrências de $X$ em $A$ seja maior (ou menor, respectivamente) do que o número de ocorrências de $X$ em $B$, sabemos que mesmo na partição comum mínima de $A$ e $B$ pelo menos um duo em (uma ocorrência de) $X$ em $A$ (ou em $B$, respectivamente) deve ser quebrada. O algoritmo visa atingir (i.e., cortar) todas as substrings de $A$ e $B$ que possuem um número diferente de ocorrências. Isso motiva a seguinte definição.

Para duas strings $A$ e $X$, seja $\#substr(A, X)$ o número de todas as ocorrências da substring $X$ na string $A$. Para uma partição $\mathcal{P} = (P_1, P_2, \ldots, P_m)$ e uma string $X$, denotamos por $\#partes(\mathcal{P}, X)$ o número de partes $P_i = X$ em $\mathcal{P}$.

O primeiro algoritmo proposto no artigo é denominado HS e seu pseudocódigo é descrito no Algoritmo 6.

**Algoritmo 6:** HS

**Entrada:** duas strings compatíveis $A$ e $B$

construa uma instância $(U, \mathcal{S})$ para o Problema do Hitting Set Mínimo:
    $U \leftarrow duos(A) \cup duos(B)$
    $T \leftarrow \{X \in \Sigma^* \mid \#substr(A, X) \neq \#substr(B, X)\}$
    $S \leftarrow \{duos(X) \mid X \in T\}$
resolva (aproximadamente) o Problema do Hitting Set Mínimo:
    $\Phi \leftarrow$ um hitting set para $(U, \mathcal{S})$
transforme o hitting set em uma partição comum:
    $\mathcal{A}, \mathcal{B} \leftarrow$ para cada duo $xy \in \Phi$, corte todas as ocorrências de $xy$ nas strings $A$, $B$

**Saída:** $(\mathcal{A}, \mathcal{B})$

**Lema 10.** *A partição* $(\mathcal{A}, \mathcal{B})$ *computada pelo algoritmo* HS *é uma partição comum das strings* $A$ *e* $B$.

*Demonstração.* A demonstração é por contradição. Suponha que exista uma parte $X \in \mathcal{A}$ tal que $\#partes(\mathcal{A}, X) \neq \#partes(\mathcal{B}, X)$; se houver várias dessas partes, considere $X$ a mais longa delas. Uma vez que a parte $X$ não é cortada por nenhum duo de $\Phi$, temos $duos(X) \cap \Phi = \emptyset$, e uma vez que $\Phi$ é uma resposta válida para o Problema do Hitting Set, vale que $duos(X) \notin \mathcal{S}$. Concluímos que $\#substr(A, X) = \#substr(B, X)$. Pretendemos obter uma contradição inferindo uma igualdade para $\#partes(\mathcal{A}, X)$ e $\#partes(\mathcal{B}, X)$.

Explorando o fato de que $X$ não é cortado por nenhum duo de $\Phi$, é possível calcular os valores $\#partes(\mathcal{A}, X)$ e $\#partes(\mathcal{B}, X)$ pela seguinte fórmula (denotamos por $X \sqsubseteq Y$ que $X$ é uma substring de $Y$ e por $X \sqsubset Y$ que $X$ é uma substring própria de $Y$):

$$\#partes(\mathcal{A}, X) = \#substr(A, X) - \sum_{Y \sqsubseteq A, X \sqsubset Y} \#substr(Y, X) \cdot \#partes(\mathcal{A}, Y)$$

$$\#partes(\mathcal{B}, X) = \#substr(B, X) - \sum_{Y \sqsubseteq B, X \sqsubset Y} \#substr(Y, X) \cdot \#partes(\mathcal{B}, Y)$$

Por nossa escolha, $X$ é a parte mais longa com $\#partes(\mathcal{A}, X) \neq \#partes(\mathcal{B}, X)$ (informalmente, uma parte "errada"); portanto, para todas as strings $Y$ que satisfazem $X \sqsubset Y$, temos $\#partes(\mathcal{A}, Y) = \#partes(\mathcal{B}, Y)$. Concluímos que $\#partes(\mathcal{A}, X) = \#partes(\mathcal{B}, X)$, o que é uma contradição. □

**Lema 11.** *O algoritmo* HS *encontra uma* $2k$*-aproximação da partição comum mínima (se um procedimento exato para o Problema do Hitting Set Mínimo estiver disponível).*

*Demonstração.* Considere qualquer partição comum $(\mathcal{A}', \mathcal{B}')$ de $A$ e $B$. Então, todo duo em um hitting set mínimo para a instância $(U, \mathcal{S})$ deve aparecer como um duo quebrado em $\mathcal{A}'$ ou $\mathcal{B}'$. Ou seja, a metade do tamanho do hitting set mínimo é um limite inferior para o

tamanho da partição comum mínima. Observando que o algoritmo corta no máximo $k$ duos para cada duo do conjunto $\Phi$, a afirmação é válida. □

Observe que, substituindo o procedimento ótimo para o Problema do Hitting Set Mínimo por um procedimento de $\alpha$-aproximação, o algoritmo HS encontra uma $2k\alpha$-aproximação para o MCSP. Infelizmente, o Problema do Hitting Set Mínimo é difícil de aproximar; assim, para obter um bom fator de aproximação, precisamos investigar propriedades especiais da instância $(U, \mathcal{S})$.

### B.3.1 Uma O(k)-aproximação para o k-MCSP

Seja $(\mathcal{A}_o, \mathcal{B}_o)$ uma partição comum mínima das strings $A$ e $B$ (se houver várias partições comuns mínimas, escolhemos uma delas arbitrariamente); dizemos que as quebras em $\mathcal{A}_o$ e $\mathcal{B}_o$ são as *quebras ótimas*. Nesta partição comum mínima, existem $2|\mathcal{A}_o| - 2$ quebras ótimas. Dizemos que uma substring $X = a_i \dots a_j$ (resp., $X = b_i \dots b_j$) *passa sobre* uma quebra ótima se houver uma quebra ótima $l$, $l+1$ em $\mathcal{A}_o$ (resp., em $\mathcal{B}_o$) tal que $i \leq l < j$.

Lembre-se da definição do conjunto $T = \{X \in \Sigma^* \mid \#substr(A, X) \neq \#substr(B, X)\}$; informalmente, $T$ é o conjunto de todas as substrings erradas. Observe que na instância do Problema do Hitting Set, a maioria das substrings em $T$ são redundantes. Para ser mais específico, se $X, Y \in T$ e $X$ é uma substring própria de $Y$, então podemos remover $Y$ do conjunto $T$, e um hitting set para $\{duos(X) \mid X \in T \setminus \{Y\}\}$ ainda será um hitting set para $\mathcal{S}$. Usando esta observação é possível reduzir substancialmente o tamanho do conjunto $\mathcal{S}$. Em particular, a relação $\sqsubseteq$ induz uma ordem parcial no conjunto $T$; seja $T_{\min} \subseteq T$ o conjunto de todos os elementos minimais de $T$, no que diz respeito à relação $\sqsubseteq$. Então $T_{\min}$ satisfaz a propriedade desejada:

**(P)** se $X, Y \in T$, e $X$ é uma substring própria de $Y$, então $Y \notin T_{\min}$,

e, ao mesmo tempo, um hitting set para o conjunto $\mathcal{S}' = \{duos(X) \mid X \in T_{\min}\}$ é um hitting set para $\mathcal{S}$.

**Lema 12.** *Se $X \in T_{\min}$ então existe uma ocorrência de $X$ em $A$ ou em $B$ que passa sobre uma quebra ótima.*

*Demonstração.* Considere uma string $X \in T_{\min}$ e suponha que nenhuma ocorrência de $X$ em $A$ e $B$ passa sobre uma quebra ótima. Então, cada ocorrência de $X$ em $A$ ou em $B$ é uma substring de alguma parte na partição comum mínima $(\mathcal{A}_o, \mathcal{B}_o)$. Como $\mathcal{A}_o$ e $\mathcal{B}_o$ consistem no mesmo multiconjunto de partes e nenhuma ocorrência de $X$ passa sobre uma quebra ótima, temos $\#substr(A, X) = \#substr(B, X)$. Isso implica em $X \notin T$, o que é uma contradição. □

Usando o lema, atribuímos a cada string em $T_{\min}$ uma quebra ótima. Em particular, para $X \in T_{\min}$, seja $f(X)$ a quebra ótima que uma ocorrência de $X$ em $A$ ou em $B$ passa sobre; se houver mais de uma quebra ótima, escolhemos uma delas arbitrariamente.

**Exemplo 1.** *Para $A = abaab$, $B = ababa$, a partição comum mínima é $(aba, ab)$, $(ab, aba)$, com $ba \in T_{\min}$, e $f(ba)$ igual à quebra $(2, 3)$ na partição de $B$.*

**Lema 13.** *Se $X, Y \in T_{\min}$, com $X = x_1, \ldots, x_l$ e $f(X) = f(Y)$, então $duos(Y) \cap \{x_1x_2, x_{l-1}x_l\} = \emptyset$.*

*Demonstração.* Uma vez que $X$ e $Y$ passam sobre a mesma quebra ótima, a sobreposição delas tem tamanho pelo menos dois. Além disso, como $X$ não é uma substring própria de $Y$ e vice-versa (pela propriedade **(P)** e as hipóteses do lema), a afirmação vale. □

A consequência do Lema 13 é a seguinte. Sejam $\mathcal{A}$ uma partição de $A$ e $\mathcal{B}$ uma partição de $B$ e seja $X = x_1 \ldots x_l$ uma substring comum de $\mathcal{A}$ e $\mathcal{B}$ tal que $X \in T_{\min}$. Então, cortando todas as ocorrências de $x_1x_2$ e $x_{l-1}x_l$ em $\mathcal{A}$ e $\mathcal{B}$ nós "atingimos" (isto é, cortamos), também, (um duo em) cada string de $T_{\min}$ que passa sobre a quebra ótima $f(X)$. Assim, se escolhermos para cada corte ótimo uma string de $T_{\min}$ que passa sobre ela (se houver tal string para o corte; se não houver tal string, ignoramos este corte) e juntarmos o primeiro e o último duo de cada string, obteremos um hitting set para $T_{\min}$ de tamanho no máximo duas vezes o tamanho do hitting set mínimo. Obviamente, não conhecemos as quebras ótimas e, portanto, temos que construir o hitting set de uma maneira diferente. O Algoritmo 7, denominado FAST HS, faz isso:

**Algoritmo 7:** FAST HS

**Entrada:** duas strings compatíveis $A$ e $B$

```
1 compute um conjunto T' tal que T_min ⊆ T' e T' tem tamanho O(n)
2 Φ ← ∅
3 A ← (A), B ← (B)
4 para cada X ∈ T' em ordem crescente de comprimento faça
5     se duos(X) ∩ Φ = ∅ então
6         adicione o primeiro e o último duo de X a Φ
7         corte todas as ocorrências do primeiro e do último duo de X nas partições A e B
```

**Saída:** $(\mathcal{A}, \mathcal{B})$

**Lema 14.** *Se uma string $X$ satisfaz a condição $duos(X) \cap \Phi = \emptyset$ no algoritmo* FAST HS*, então $X \in T_{\min}$.*

*Demonstração.* Suponha, por contradição, que $X$ passou no teste, mas $X \notin T_{\min}$. Seja $\Phi'$ o conjunto $\Phi$ imediatamente antes de processar a string $X$. A suposição $X \notin T_{\min}$ implica

que existe uma string $X' \in T_{\min}$ tal que $X'$ é uma substring própria de $X$. Uma vez que $|X'| < |X|$, a string $X'$ foi processada antes da string $X$ e, portanto, $duos(X') \cap \Phi' \neq \emptyset$. Além disso, como $duos(X') \subseteq duos(X)$, vale $duos(X) \cap \Phi' \neq \emptyset$ e, portanto, $X$ não pode passar no teste, o que é uma contradição. □

**Teorema 15.** *O algoritmo* Fast HS *calcula uma* $4k$*-aproximação da partição comum mínima de* $A$ *e* $B$*.*

*Demonstração.* Se $X_1$ e $X_2$ são duas strings diferentes para as quais o conjunto $\Phi$ foi aumentado, então, pelo Lema 13, $f(X_1) \neq f(X_2)$. Assim, o conjunto $\Phi$ foi aumentado no máximo $|\mathcal{A}_o| + |\mathcal{B}_o| - 2$ vezes e, portanto, o conjunto final $\Phi$ contém no máximo $2 \cdot (|\mathcal{A}_o| + |\mathcal{B}_o| - 2)$ duos.

Como estamos lidando com uma instância de $k$-MCSP, cada duo do conjunto $\Phi$ introduz no máximo $k$ cortes. Vale que

$$|\mathcal{A}| \leq k \cdot 2 \cdot (|\mathcal{A}_o| + |\mathcal{B}_o| - 2) + 1 \leq 4k \cdot |\mathcal{A}_o|.$$

□

*Observação* 2. O fator de aproximação se aplica mesmo se medirmos o tamanho de uma partição comum não pelo número de partes, mas pelo número de quebras.

**Um limite inferior.** Seja $A = ba\{ab\}^{k-1}$ e $B = \{ab\}^k$. Então, o conjunto $\Phi$ consiste em dois duos $\{aa, ab\}$ e a partição computada pelo algoritmo Fast HS tem tamanho $k+1$, enquanto a partição comum mínima tem tamanho 3. Assim, o fator de aproximação do algoritmo Fast HS é $\Omega(k)$.

Neste artigo, os autores também propõem uma implementação para o algoritmo Fast HS com complexidade de tempo $O(n)$. Mais uma vez, como o foco de nossos estudos foi a análise das demonstrações sobre o fator de aproximação, não tratamos desta implementação de tempo linear.

Pode-se modificar facilmente os algoritmos HS e Fast HS para funcionar também para a relação $\cong$, tanto para strings com sinais como sem sinais. Redefinimos $\#substr(A, S)$ para que conte as ocorrências de $S$ e $S^R$ em $A$; as definições dos conjuntos $T$, $T'$ e $T_{\min}$ permanecem inalteradas. A nova definição de $\#substr$ requer uma pequena mudança no cálculo do conjunto $T'$ na implementação linear. Também precisamos de uma pequena mudança no Lemma 12 e no Lemma 13, respectivamente:

**Lema 16.** *Se* $X \in T_{\min}$*, então existe uma ocorrência de* $X$ *ou* $X^R$ *em* $A$ *ou em* $B$ *que passa sobre uma quebra ótima.*

**Lema 17.** *Se $X, Y \in T_{\min}$, com $X = x_1, \ldots, x_l$ e $f(X) = f(Y)$, então duos$(Y) \cap \{x_1x_2$, $x_{l-1}x_l$, $(x_1x_2)^R, (x_{l-1}x_l)^R\} = \emptyset$.*

Finalmente, sempre que o algoritmo original corta duos $xy$, o algoritmo modificado também corta duos $(xy)^R$. Isso aumenta o fator de aproximação por um fator multiplicativo 2.

**Teorema 18.** *O algoritmo* FAST HS *computa, em tempo linear, uma $\Theta(k)$-aproximação para o k-MCSP com sinais, sem sinais e reverso, e para o k-TSbR com e sem sinais.*

## B.4 Boria et al. (2014)

Neste artigo [2], os autores propõem um algoritmo de aproximação para o MPSM, com fator de aproximação 4, baseado em uma estratégia de construir um grafo bipartido com os duos das strings de entrada e, neste grafo, encontrar um emparelhamento máximo.

Para $i = 1, \ldots, n$, denotamos por $a_i$ o $i$-ésimo elemento da string $A$ e por $b_i$ o $i$-ésimo elemento da string $B$. Também denotamos por $D^A = (D^A_1, \ldots, D^A_{n-1})$ e $D^B = (D^B_1, \ldots, D^B_{n-1})$ os conjuntos de duos de $A$ e de $B$, respectivamente. Para $i = 1, \ldots, n-1$, $D^A_i$ corresponde ao duo $(a_i, a_{i+1})$, e $D^B_i$ corresponde ao duo $(b_i, b_{i+1})$.

Um mapeamento $\pi$ de $A$ para $B$ é dito *próprio* se for bijetivo, e se $a_i = b_{\pi(i)}$, para todo $i = 1, \ldots, n$. Em outras palavras, cada letra do alfabeto em $A$ deve ser mapeada para a mesma letra em $B$ para que o mapeamento seja próprio. Um par de duos $(D^A_i, D^B_j)$ é dito *preservável* se $a_i = b_j$ e $a_{i+1} = b_{j+1}$. Dado um mapeamento $\pi$, um par de duos preservável $(D^A_i, D^B_j)$ é dito *preservado por* $\pi$ se $\pi(i) = j$ e $\pi(i+1) = j+1$. Finalmente, dois pares de duos preserváveis $(D^A_i, D^B_j)$ e $(D^A_h, D^B_l)$ serão chamados de *conflitantes* se não houver um mapeamento próprio que preserve ambos. Esses conflitos podem ser de dois tipos, sem perda de generalidade, suponhamos que $i \leq h$ (resp. $j \leq l$):

**Tipo 1:** $i = h$ (resp. $j = l$) e $j \neq l$ (resp. $i \neq h$);

**Tipo 2:** $i = h - 1$ (resp. $j = l - 1$) e $j \neq l - 1$ (resp. $i \neq h - 1$).

Vamos agora definir formalmente o problema em questão:

**Definição 1.** MAPEAMENTO DE STRINGS COM PRESERVAÇÃO MÁXIMA DE DUOS *(MPSM):*

***Instância:*** *duas strings $A$ e $B$, tal que $B$ seja uma permutação de $A$.*

***Solução:*** *um mapeamento próprio $\pi$ de $A$ para $B$.*

***Objetivo:*** *maximizar o número de duos preservados por $\pi$, denotado por $f(\pi)$.*

Um *mapeamento de duo* $\sigma$ é um mapeamento, que – diferente de um mapeamento $\pi$, que mapeia cada caractere em $A$ para um caractere em $B$ – mapeia um *subconjunto* de duos de $D^A$

para um *subconjunto* de duos de $D^B$. Assim, $\sigma(i) = j$ significa que o duo $D_i^A$ está mapeado para o duo $D_j^B$. Novamente, um mapeamento de duo $\sigma$ é considerado próprio se for bijetivo, e se $D_i^A = D_{\sigma(i)}^B$ para todos os duos mapeados através de $\sigma$. Observe que um mapeamento de duo próprio pode mapear alguns pares de duos conflitantes. Perceba, entretanto, que um mapeamento de duo próprio pode gerar apenas conflitos do Tipo 2. Finalmente definimos o conceito de mapeamento de duo *não conflitante*, que é um mapeamento de duo próprio que não mapeia nenhum par de duos conflitantes.

*Observação* 3. Um mapeamento de duo não conflitante $\sigma$ sobre algum subconjunto de duos de tamanho $f(\sigma)$ imediatamente deriva um mapeamento próprio $\pi$ sobre todo o conjunto de caracteres, com $f(\pi) \geq f(\sigma)$: basta mapear os caracteres mapeados por $\sigma$ da mesma forma que $\sigma$ o fez, e mapear arbitrariamente os caracteres restantes.

### B.4.1 4-aproximação para o MPSM

Nesta seção, os autores apresentam o algoritmo de aproximação para o MPSM, baseado na estratégia de emparelhamento de duos.

**Teorema 19.** *Existe um algoritmo de 4-aproximação para o MPSM que é executado em tempo* $O(n^{3/2})$.

*Demonstração.* Considere as duas strings $A = a_1 a_2 \dots a_n$ e $B = b_1 b_2 \dots b_n$ que desejamos mapear, preservando um número máximo de duos, com $D^A$ e $D^B$ denotando seus respectivos conjuntos de duos. Além disso, seja $\pi^*$ um mapeamento ótimo que preserva um número máximo de duos $f(\pi^*)$.

Construa um grafo bipartido $G$ da seguinte forma: os vértices à esquerda e à direita representam os duos de $D^A$ e $D^B$, respectivamente. Sempre que um duo à direita e um à esquerda são preserváveis (têm as mesmas duas letras na mesma ordem), adicionamos uma aresta entre os dois vértices correspondentes.

Neste ponto, observe que existe uma correspondência de um-para-um entre emparelhamentos em $G$ e os mapeamentos de duo próprios entre $D^A$ e $D^B$. Em outras palavras, existe um emparelhamento em $G$ com $f(\pi^*)$ arestas. De fato, o conjunto de duos preservado por qualquer solução (e, portanto, pela ótima) pode ser representado como um emparelhamento em $G$. Portanto, denotando por $M^*$ um emparelhamento máximo em $G$, isso significa que

$$f(\pi^*) \leq |M^*| \tag{4}$$

Infelizmente, um emparelhamento $M^*$ em $G$ não se traduz imediatamente em um mapeamento próprio que preserva $|M^*|$ duos. No entanto, ele corresponde a um

mapeamento de duo próprio que mapeia $|M^*|$ duos, que, como notado anteriormente, pode gerar apenas conflitos do Tipo 2.

Em $G$, um conflito do Tipo 2 corresponde a dois vértices consecutivos de um lado emparelhados a dois vértices não consecutivos do outro lado. Portanto, para gerar um mapeamento de duo não conflitante $\sigma$ usando um emparelhamento $M^*$, basta particionar o emparelhamento $M^*$ em 4 sub-emparelhamentos, da seguinte maneira: seja $M(\textit{par, ímpar})$ o sub-emparelhamento de $M^*$ contendo todas as arestas cujo extremo esquerdo tem índice par, e o extremo direito tem índice ímpar; defina $M(\textit{ímpar, par})$, $M(\textit{par, par})$ e $M(\textit{ímpar, ímpar})$ da forma análoga. Seja $\hat{M}$ o emparelhamento com a maior cardinalidade entre esses quatro. Obviamente, lembrando que os quatro sub-emparelhamentos definem uma partição de $M^*$, vale $|\hat{M}| \geq |M^*|/4$. Considerando que $\hat{M}$ não contém nenhum par de arestas com extremos consecutivos, o mapeamento de duo $\sigma$ correspondente não tem conflito. Seguindo a Observação 3, $\sigma$ deriva um mapeamento próprio $\pi$ sobre os caracteres, tal que

$$f(\pi) \geq f(\sigma) = |\hat{M}| \geq \frac{|M^*|}{4} \geq \frac{f(\pi^*)}{4}$$

Uma 4-aproximação pode, portanto, ser computada criando o grafo $G$ a partir das strings $A$ e $B$, computando nele um emparelhamento máximo $M^*$, particionando $M^*$ de quatro formas, pela paridade dos índices, e devolvendo a maior partição $\hat{M}$. Em seguida, mapeie os duos emparelhados seguindo as arestas $\hat{M}$ e mapeie todos os outros caracteres arbitrariamente. A complexidade total desta computação é dada pela complexidade de computar um emparelhamento ótimo em $G$, que é $O(n^{3/2})$. □

## B.5 Boria et al. (2016)

Neste artigo [3], os autores propõem um algoritmo, baseado na estratégia de busca local, para lidar com o MPSM. Este algoritmo melhora o fator de aproximação igual a 4, de [2], para um fator de aproximação igual a $3,5$.

### B.5.1 Transformação do problema para um problema em grafos

Em [2], o problema é mostrado como um caso particular do seguinte problema em grafos, que chamamos de Problema do Máximo Emparelhamento Bipartido Consecutivo. Dado um grafo bipartido onde os vértices em ambos os lados são ordenados como $A = (a_1, \ldots, a_n)$ e $B = (b_1, \ldots, b_n)$, o Problema do Máximo Emparelhamento Bipartido Consecutivo consiste em encontrar o emparelhamento máximo $M$ tal que, se $(a_i, b_j) \in M$, então $a_{i+1}$ só pode ser emparelhado com $b_{j+1}$, e $b_{j+1}$ só pode ser emparelhado com $a_{i+1}$. Em outras palavras,

conjuntos de vértices consecutivos em um lado devem ser emparelhados com vértices consecutivos no outro lado.

Destacamos brevemente a razão pela qual o MPSM é um caso particular do Problema do Máximo Emparelhamento Bipartido Consecutivo. Duas strings $A$ e $B$ de qualquer instância de MPSM podem ser transformadas em conjuntos de duos ordenados $D^A$ e $D^B$ (por exemplo, se $A = abc$ e $B = bac$, então $D^A = ((ab), (bc))$, e $D^B = ((ba), (ac))$).

Considere o grafo bipartido $G(I)$ construído da seguinte maneira:

- cada vértice do lado esquerdo representa um duo do conjunto $D^A$, e cada vértice do lado direito representa um duo de $D^B$;
- arestas existem entre dois vértices se e somente se eles representam o mesmo duo (mesmo par de caracteres na mesma ordem).

Em [2], é mostrado que qualquer solução viável $M$ para o Problema do Máximo Emparelhamento Bipartido Consecutivo no grafo $G(I)$ produz um mapeamento $\pi$ entre as strings $A$ e $B$ que preserva pelo menos $|M|$ duos (e exatamente $|M|$ duos, caso $M$ seja maximal em termos de inclusão).

De fato, tal emparelhamento pode ser visto como um mapeamento parcial, e o número de arestas no emparelhamento é igual ao número de duos que o mapeamento preserva. O mapeamento parcial pode, então, ser completado de forma arbitrária, uma vez que o conjunto de caracteres não mapeados em $A$ é uma permutação de vértices não mapeados em $B$.

A partir deste ponto, os autores se referem apenas ao Problema do Máximo Emparelhamento Bipartido Consecutivo, tendo em mente que qualquer resultado de aproximação válido para o Problema do Máximo Emparelhamento Bipartido Consecutivo também é válido para o MPSM.

Chamamos duas arestas de *conflitantes* se elas não puderem ser ambas parte da mesma solução, seja porque compartilham um extremo comum ou porque seus extremos são consecutivos em um lado do grafo, mas não no outro.

### B.5.2 Algoritmo de busca local

Nesta seção, os autores apresentam um algoritmo que produz um mapeamento parcial com base na técnica de busca local.

Algoritmos de busca local produzem soluções que são definidas como ótimos locais. Um ótimo local de um problema de otimização é uma solução ótima (maximal ou minimal) dentro de um conjunto vizinhança de soluções candidatas. Partindo de qualquer solução viável, o algoritmo busca uma uma solução aprimorada no conjunto vizinhança e repetidamente se move para uma solução de vizinha aprimorada, desde que tal solução exista. Quando nenhuma solução vizinha aprimorada pode ser encontrada, a solução atual é, por definição,

um ótimo local. A qualidade do ótimo local obviamente depende da definição do conjunto vizinhança.

Desenvolvemos um algoritmo de busca local denominado `LOCAL`, que é baseado em uma estrutura de vizinhança $\mathcal{N}$. Dado um emparelhamento $M$ que é uma solução viável para o problema, a vizinhança de $M$, denotada por $\mathcal{N}(M)$, contém todas as soluções viáveis $M'$ tais que $|M \setminus M'| \leq 1$. Em outras palavras, $M'$ deve conter todas as arestas de $M$, exceto possivelmente uma.

Enquanto procura por uma solução aprimorada no conjunto vizinhança, o algoritmo `LOCAL` primeiro tentará aprimorar a solução sem remover nenhuma aresta da solução atual $M$. Por um lado, se $M$ não é maximal em termos de inclusão, então há uma aresta que pode ser adicionada à solução atual $M$ sem ter que remover nenhuma aresta dela. Por outro lado, se $M$ é maximal em termos de inclusão, então o algoritmo olha para cada emparelhamento $M \setminus \{v\}$, para cada $v \in M$, e verifica se pelo menos duas arestas podem ser adicionadas a um desses emparelhamentos.

### B.5.3 Análise de complexidade

Provaremos que o algoritmo é executado, de fato, em tempo polinomial. Em primeiro lugar, mesmo partindo de uma solução vazia, o algoritmo incrementará o valor de sua solução em pelo menos um, a cada passo, de forma que será concluído após no máximo $|\text{SOL}| \leq n$ passos.

Em cada passo, o algoritmo primeiro examina todas as arestas que não estão em SOL e verifica se uma delas não está em conflito com qualquer aresta de SOL. Isso é feito em tempo $O(n^2)$. Se tal aresta for encontrada, o passo atual estará concluído. Caso contrário, para cada aresta $u$ de SOL, o algoritmo considera todos os conjuntos de no máximo 6 arestas que não estão na solução e estão em conflito com $u$, e verifica se elas podem ser adicionadas ao emparelhamento $\text{SOL} \setminus \{u\}$ sem gerar nenhum conflito. Isso é feito em tempo $O(n^6)$ para cada aresta $u$ da solução atual: cada aresta da solução entra em conflito com $O(n)$ arestas que não estão na solução, de modo que há $O(n^6)$ combinações candidatas de no máximo 6 arestas que não estão na solução. Considerando que, a cada passo, a solução atual possui $O(n)$ arestas, a complexidade de um único passo é $O(n^7)$.

Ao todo, o algoritmo termina após no máximo $n$ passos, cada passo sendo executado em tempo $O(n^7)$, de forma que a complexidade geral seja $O(n^8)$.

A complexidade de `LOCAL` pode ser reduzida a $O(n^4)$ graças à seguinte observação. Se pode ser feita uma melhoria que incrementa a cardinalidade da solução em pelo menos um em algum passo (removendo uma aresta $u$ de SOL e adicionando um conjunto $X$ de pelo menos duas arestas não conflitantes a SOL), então uma melhoria que incrementa este valor em exatamente um também é possível (removendo a mesma aresta $u$ de SOL e adicionando

exatamente qualquer par de arestas do conjunto $X$). Assim, em vez de examinar todos os conjuntos de no máximo 6 arestas que não estão na solução e estão em conflito com cada aresta $u$, basta que o algoritmo examine apenas cada par de arestas que não estão na solução e que são conflitantes com $u$. Se nenhum par aprimorado for encontrado, então nenhuma melhoria pode ser feita, e a solução atual é um ótimo local.

### B.5.4 Análise da aproximação

Agora provaremos que, de fato, `LOCAL` melhora a 4-aproximação, que é o melhor algoritmo conhecido para o Problema do Máximo Emparelhamento Bipartido Consecutivo:

**Teorema 20.** *O algoritmo `LOCAL` produz uma* $3,5$*-aproximação para o Problema do Máximo Emparelhamento Bipartido Consecutivo.*

*Demonstração.* Considere que o algoritmo `LOCAL` roda sobre uma instância $I$ do Problema do Máximo Emparelhamento Bipartido Consecutivo e produz uma solução SOL.

A prova é baseada na contagem dos conflitos entre as arestas de SOL e as arestas de um emparelhamento ótimo desconhecido OPT. Denotamos esse número de conflitos por $C$.

Por um lado, uma única aresta de SOL não pode estar em conflito com mais de 6 arestas de OPT. De fato, qualquer aresta $u$ pode estar em conflito apenas com arestas que compartilham um extremo com $u$, ou que possuem um extremo que é consecutivo a um extremo de $u$ (imediatamente antes ou imediatamente depois), o que resulta em não mais do que 6 extremos possíveis para arestas em conflito com $u$ (os dois extremos de $u$ e os quatro vértices consecutivos). Por outro lado, qualquer solução viável, incluindo a ótima, pode escolher no máximo uma aresta por vértice do grafo. Isso nos dá o seguinte limite superior no valor de $C$:

$$C \leq 6|\text{SOL}|. \tag{5}$$

Lembramos que, por definição, não há solução SOL$'$ na vizinhança $\mathcal{N}(\text{SOL})$ que tenha mais arestas que SOL. Portanto, dada qualquer aresta $v$ de SOL, o seguinte fato é válido:

*Afirmação* 3. Seja $v$ uma aresta da solução SOL gerada por `LOCAL`, e seja OPT uma solução ótima para o problema. Existe no máximo uma aresta $u$ de OPT que entra em conflito apenas com $v$ em SOL.

O fato é bastante simples: suponha que existam duas arestas $u$ e $t$ em uma solução OPT que conflitam com uma única aresta $v$ em SOL. A solução $\text{SOL} \setminus \{v\} \cup \{u, t\}$ é um emparelhamento admissível na vizinhança de SOL e contém mais arestas. Portanto, `LOCAL` deveria tê-lo escolhido, em vez de SOL.

Vamos denotar por $k_1$ o número de arestas em OPT que conflitam com apenas uma aresta de SOL. A Afirmação 3 produz naturalmente o seguinte limite:

$$k_1 \leq |\text{SOL}|. \tag{6}$$

Em OPT, as $|\text{OPT}| - k_1$ arestas restantes conflitam com pelo menos 2 arestas de SOL, o que nos dá o seguinte limite inferior no número de conflitos $C$:

$$C \geq k_1 + 2(|\text{OPT}| - k_1) \geq 2|\text{OPT}| - k_1 \geq 2|\text{OPT}| - |\text{SOL}|. \tag{7}$$

Combinando as equações (6) e (7), podemos facilmente obter o seguinte limite para o fator de aproximação de `LOCAL`, que conclui a prova:

$$\frac{\text{OPT}}{\text{SOL}} \leq \frac{7}{2}.$$

□

## B.6 Brubach (2016)

Neste artigo [4], o autor propõe uma 13/4-aproximação para o MPSM, usando uma nova estratégia de emparelhamento de trios. Isso aprimora o melhor fator de aproximação anterior, proposto em [3], igual a $3,5$.

Primeiro, relembraremos algumas definições sobre o MPSM. Denotamos por $a_i$ e $b_j$ os $i$-ésimo e $j$-ésimo caracteres de suas respectivas strings. Um mapeamento *próprio* $\pi$ de $A$ para $B$ é um mapeamento um-para-um com $a_i = b_{\pi(i)}$ para todo $i = 1, \ldots, n$. Um *duo* denota dois caracteres consecutivos da mesma string. Dizemos que um duo $(a_i, a_{i+1})$ é *preservado* se $a_i$ é mapeado para algum $b_j$ e $a_{i+1}$ é mapeado para $b_{j+1}$. O objetivo do MPSM é retornar um mapeamento próprio das letras de $A$ para as letras de $B$ que preserve o número máximo de duos.

Assim, a principal contribuição do artigo é descrito pelo seguinte teorema.

**Teorema 21.** *Para quaisquer duas strings $A$ e $B$, tal que $B$ é uma permutação de $A$, existe um algoritmo que encontra um mapeamento próprio de $A$ para $B$ que preserva pelo menos $4/13$ dos duos que um mapeamento ótimo preserva.*

Sejam $A = a_1 a_2 \ldots a_n$ e $B = b_1 b_2 \ldots b_n$ duas strings de comprimento $n$, com $a_i$ e $b_i$ sendo os $i$-ésimos caracteres de suas respectivas strings. Um *duo* $D_i^A = (a_i, a_{i+1})$ corresponde ao par de caracteres consecutivos $a_i$ e $a_{i+1}$ na string $A$. Usamos $D^A = (D_1^A, \ldots, D_{n-1}^A)$ e $D^B = (D_1^B, \ldots, D_{n-1}^B)$ para denotar os conjuntos de duos de $A$ e $B$, respectivamente. Da

mesma forma, definimos um *trio* $T_i^A = (a_i, a_{i+1}, a_{i+2})$ como um conjunto de três caracteres consecutivos $a_i$, $a_{i+1}$, e $a_{i+2}$ na string $A$, e $T^A = (T_1^A, \ldots, T_{n-2}^A)$ e $T^B = (T_1^B, \ldots, T_{n-2}^B)$ como os conjuntos de trios das strings $A$ e $B$, respectivamente. Observe que os duos $D_i^A$ e $D_{i+1}^A$ correspondem aos dois primeiros e dois últimos caracteres, respectivamente, do trio $T_i^A$. Nos referimos aos duos $D_i^A$ e $D_{i+1}^A$ como subconjuntos do trio $T_i^A$.

**Nota importante:** No primeiro passo do nosso algoritmo, acrescentamos um caractere especial '&' no início e no final de cada string (índices 0 e $n+1$). Definimos este caractere de modo que não seja igual a nenhum outro caractere incluindo ele mesmo (o que significa $\& \neq \&$). Isso garante que cada duo possa ser um subconjunto de exatamente dois trios.

Um mapeamento próprio $\pi$ de $A$ para $B$ é um mapeamento um-para-um das letras de $A$ para as letras de $B$, com $a_i = b_{\pi(i)}$ para todo $i = 1, \ldots, n$. Lembre-se de que um duo $(a_i, a_{i+1})$ é preservado se e somente se $a_i$ é mapeado para algum $b_j$ e $a_{i+1}$ é mapeado para $b_{j+1}$. Chamamos um par de duos $(D_i^A, D_j^B)$ de *preservável* se e somente se $a_i = b_j$ e $a_{i+1} = b_{j+1}$.

Para consistência, definimos o conceito de pares de duos conflitantes usando a terminologia de [2], com uma pequena modificação, para acomodar nossa análise particular. Dois pares de duos preserváveis $(D_i^A, D_j^B)$ e $(D_h^A, D_\ell^B)$ são considerados *conflitantes* se nenhum mapeamento próprio puder preservar ambos. Esses conflitos podem ser de dois tipos: Tipo 1 e Tipo 2.

**Tipo 1:** Ou $(i = h) \wedge (j \neq \ell)$, ou $(i \neq h) \wedge (j = \ell)$.

**Tipo 2:** Ou $(i = h + 1) \wedge (j \neq \ell + 1)$, ou $(i \neq h + 1) \wedge (j = \ell + 1)$.

**Exceção:** Em nossa análise, também consideramos dois pares de duos preserváveis consecutivos $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ e um terceiro par de duos $(D_h^A, D_\ell^B)$ que entra em conflito com um ou ambos, potencialmente criando conflitos do Tipo 1 e do Tipo 2. No entanto, classificamos esses conflitos simplesmente como conflitos do Tipo 1.

### B.6.1 Descrição do algoritmo de emparelhamento de trios

Nesta seção, é apresentado e analisado o algoritmo de emparelhamento de trios. Começamos encontrando um emparelhamento ponderado de trios que limita a solução ótima, traduzindo isso em uma emparelhamento fracionário de duos e arredondando a solução fracionária para um mapeamento $S$, que preserva um número de duos que é pelo menos 4/13 o peso do emparelhamento de trios.

**Etapa 1: Construa um grafo bipartido ponderado $G_T$ com os trios.**

Primeiramente, anexamos o caractere especial '&' ao início e ao final de cada string, conforme citado anteriormente. Lembre-se de que $\& \neq \&$. Isso garante que cada duo possa ser um subconjunto de exatamente dois trios.

Em seguida, construímos um grafo bipartido ponderado $G_T = (T^A \cup T^B, E)$ com cada partição sendo o conjunto de trios de uma string. Adicionamos três tipos de arestas a este

gráfico: arestas *completas*, arestas de *primeira metade* e arestas de *segunda metade*. Para um determinado par de trios, $(T_i^A, T_j^B)$, das duas diferentes strings, podemos adicionar no máximo um tipo de aresta. Uma aresta completa é adicionada se $(a_i = b_j) \wedge (a_{i+1} = b_{j+1}) \wedge (a_{i+2} = b_{j+2})$. Uma aresta de primeira metade é adicionada se $(a_i = b_j) \wedge (a_{i+1} = b_{j+1}) \wedge (a_{i+2} \neq b_{j+2})$. Da mesma forma, uma aresta de segunda metade é adicionada se $(a_i \neq b_j) \wedge (a_{i+1} = b_{j+1}) \wedge (a_{i+2} = b_{j+2})$. As arestas completas têm peso 1 e arestas de primeira e segunda metade têm peso 1/2.

Em outras palavras, se os trios são uma correspondência perfeita, o peso da aresta é 1. Caso contrário, se apenas os dois primeiros ou os dois últimos caracteres corresponderem, o peso será 1/2. Finalmente, se as condições anteriores não forem atendidas, não adicionamos aresta entre esses trios.

**Etapa 2: Encontre um emparelhamento de trios $M_T$ de peso máximo em $G_T$.**

Encontramos um emparelhamento $M_T$ de peso máximo no grafo $G_T$. Provaremos mais adiante que o peso desse emparelhamento é um limite superior válido para a solução ótima do MPSM.

**Etapa 3: Transfira o emparelhamento para um grafo bipartido ponderado $G_D$ com os duos.**

Agora, construímos um grafo bipartido $G_D = (D^A \cup D^B, E)$ com os duos, usando as arestas do emparelhamento $M_T$ encontrado para $G_T$. Para toda aresta $(T_i^A, T_j^B) \in M_T$, adicionamos uma ou duas arestas a $G_D$. Cada aresta adicionada a $G_D$ tem peso 1/2. Uma vez que cada duo da string original está contido em dois trios separados, pode acontecer que obtenhamos duas cópias da aresta $(D_i^A, D_j^B)$. Nesse caso, simplesmente as combinamos em uma única aresta com peso 1. As arestas são adicionadas de acordo com as seguintes regras simples:

- Se $(T_i^A, T_j^B)$ é uma aresta completa, adicionamos as arestas $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ a $G_D$.
- Se $(T_i^A, T_j^B)$ é uma aresta de primeira metade, adicionamos a aresta $(D_i^A, D_j^B)$ a $G_D$.
- Se $(T_i^A, T_j^B)$ é uma aresta de segunda metade, adicionamos a aresta $(D_{i+1}^A, D_{j+1}^B)$ a $G_D$.

Lembre-se de que $T_i^A$ e $D_i^A$ referem-se ao trio e ao duo, respectivamente, começando na letra $a_i$ da string $A$, e os duos $D_i^A$ e $D_{i+1}^A$ são ambos subconjuntos do trio $T_i^A$. Se a aresta de trio $(T_i^A, T_j^B)$ produz as arestas de duo $(D_i^A, D_j^B)$ ou $(D_{i+1}^A, D_{j+1}^B)$, dizemos que os trios *suportam* as arestas de duo. Os caracteres extras '&' são descartados nesta etapa, pois, por definição, eles não podem fazer parte de nenhum par de duos emparelhados.

**Etapa 4: Use $G_D$ para encontrar um mapeamento da string $A$ para a string $B$.**

Nesta etapa, selecionamos um subconjunto das arestas em $G_D$ para serem os duos

preservadas em nosso mapeamento final $S$. Essa etapa acontece em três fases, podendo cada uma incluir várias iterações. Cada iteração de uma fase remove arestas de $G_D$ correspondentes a um ou dois pares de duos preservados, bem como quaisquer arestas conflitantes. Cada uma das duas primeiras fases remove todas as instâncias de uma estrutura específica do grafo, enquanto a terceira fase tenta preservar o máximo de duos possível a partir do grafo restante.

**Fase 1.** Para cada aresta $(D_i^A, D_j^B) \in G_D$ com peso 1, removemos $(D_i^A, D_j^B)$ de $G_D$ e mapeamos $a_i$ e $a_{i+1}$, respectivamente, para $b_i$ e $b_{i+1}$ em $S$. Também removemos quaisquer arestas conflitantes de $G_D$.

**Fase 2.** Defina um *par de arestas paralelas consecutivas* como arestas $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ em $G_D$ tais que a aresta do trio $(T_i^A, T_j^B)$ foi escolhida em $M_T$. Começando no início da string $A$, escolhemos o primeiro par de arestas paralelas consecutivas $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ em $G_D$. Em outras palavras, encontramos o menor $i$ tal que $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ são um par de arestas paralelas consecutivas. Mapeamos $a_i$, $a_{i+1}$, e $a_{i+2}$, respectivamente, para $b_i$, $b_{i+1}$ e $b_{i+2}$ em $S$. Em seguida, removemos as arestas $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ de $G_D$, bem como quaisquer arestas conflitantes. Continuamos esse processo até chegar ao final da string $A$, e não restar nenhum par de arestas paralelas consecutivas em $G_D$.

**Fase 3.** Começando no início da string $A$, adicionamos a $S$ os duos da primeira aresta que encontramos, e removemos quaisquer arestas conflitantes. Repetimos esta fase até chegar ao final de $A$ e nenhuma aresta restar em $G_D$.

### B.6.2 Demonstração da 13/4-aproximação

Mostraremos primeiro que o peso emparelhamento de trios de peso máximo $M_T$ encontrado na Etapa 2 (e, por construção, o peso total de $G_D$) é um limite superior do número máximo de duos preservados. Depois, mostraremos que o número de duos preservados adicionados a $S$ em cada iteração da Etapa 4 é pelo menos 4/13 do peso total das arestas removidas de $G_D$ naquela iteração. Finalmente, mostraremos que no final da Fase 3 da Etapa 4, nenhuma aresta permanece em $G_D$.

**Lema 22.** *Os pesos do emparelhamento de trios de peso máximo $M_T$ e o grafo de duos correspondente $G_D$ são um limite superior do número máximo de duos preservados.*

*Demonstração.* Mostraremos que qualquer mapeamento próprio $\pi$ de $A$ para $B$ que preserva $\Delta$ duos implica em um emparelhamento $M_T$ de peso pelo menos $\Delta$ no grafo de trios correspondente $G_T$.

Para cada duo preservado $(D_i^A, D_j^B)$ em $\pi$, adicionamos as arestas dos trios $(T_{i-1}^A, T_{j-1}^B)$ e $(T_i^A, T_j^B)$ a $M_T$, caso elas já não tiverem sido adicionadas. Note que na construção de $G_T$, $(D_i^A, D_j^B)$ foi responsável por adicionar $1/2$ aos pesos de ambos $(T_{i-1}^A, T_{j-1}^B)$ e $(T_i^A, T_j^B)$, como uma contribuição total de 1. Portanto, se pudermos garantir que ambas as arestas de trios são adicionadas ao emparelhamento para cada duo preservado, isso garante que $M_T$ tenha peso pelo menos $\Delta$.

Suponha, para uma contradição, que encontramos algum duo preservado $(D_i^A, D_j^B)$ e pelo menos uma das arestas de trios correspondente a $(D_i^A, D_j^B)$ não pode ser adicionado. Sem perda de generalidade, assuma que a aresta de trio que não pode ser adicionada é $(T_i^A, T_j^B)$ e está bloqueada por alguma outra aresta $(T_i^A, T_\ell^B)$, com $j \neq \ell$. A aresta $(T_i^A, T_\ell^B)$ deve ter sido adicionada pelo duo preservado $(D_i^A, D_\ell^B)$ ou $(D_{i+1}^A, D_{\ell+1}^B)$. No entanto, ambos os duos estão em conflito com $(D_i^A, D_j^B)$ e, portanto, não poderiam existir no mapeamento $\pi$, levando a uma contradição. Segue que ambas as arestas de trios são adicionadas ao emparelhamento para cada duo preservado em $\pi$. □

**Lema 23.** *O número de duos preservados adicionados a $S$ em cada iteração da Fase 1 da Etapa 4 é pelo menos $1/3$ do peso total das arestas removidas de $G_D$ naquela iteração.*

*Demonstração.* Suponha que alguma aresta $(D_i^A, D_j^B)$ tenha peso 1 em $G_D$. Então, ambas as arestas de trio $(T_{i-1}^A, T_{j-1}^B)$ e $(T_i^A, T_j^B)$ contendo $(D_i^A, D_j^B)$ devem ter sido escolhidas no emparelhamento $M_T$. Portanto, não pode haver conflitos do Tipo 1.

Observe que a aresta $(D_i^A, D_j^B)$ pode ter no máximo quatro conflitos do Tipo 2, decorrentes dos duos vizinhos $D_{i-1}^A$, $D_{i+1}^A$, $D_{j-1}^B$ e $D_{j+1}^B$. Cada um desses conflitos potenciais é simétrico. Então, sem perda de generalidade, focaremos no conflito com $D_{i-1}^A$ e mostraremos que há no máximo uma aresta $(D_{i-1}^A, D_\ell^B)$, com $\ell \neq j-1$, e esta aresta só pode ter peso $1/2$.

Por construção, a aresta $(D_{i-1}^A, D_\ell^B)$ só pode ser adicionada pelas arestas de trios $(T_{i-2}^A, T_{\ell-1}^B)$ ou $(T_{i-1}^A, T_\ell^B)$. No entanto, a aresta de trios $(T_{i-1}^A, T_\ell^B)$ não poderia existir em $M_T$, uma vez que assumimos que $M_T$ contém a aresta $(T_{i-1}^A, T_{j-1}^B)$, e $M_T$ é um emparelhamento. Portanto, existe no máximo uma dessas arestas $(D_{i-1}^A, D_\ell^B)$ com peso $1/2$ que deve ter sido produzida por uma aresta de trios $(T_{i-2}^A, T_{\ell-1}^B)$ escolhida no emparelhamento $M_T$.

Então, a soma dos pesos das arestas removidas é no máximo o peso de $(D_i^A, D_j^B)$ somado ao peso das quatro arestas conflitantes de Tipo 2, cada uma com peso $1/2$:

$$1 + 4(1/2) = 3.$$

Segue que a razão entre o número de duos preservados adicionados a $S$ e o peso das arestas removidas de $G_D$ é pelo menos $1/3$. □

Note que após a Fase 1 da Etapa 4, todas as arestas restantes em $G_D$ têm peso 1/2, pois todas as arestas com peso 1 foram removidas.

**Lema 24.** *O número de duos preservados adicionados a $S$ em cada iteração da Fase 2 da Etapa 4 é pelo menos 4/13 do peso total das arestas removidas de $G_D$ naquela iteração.*

*Demonstração.* Suponha que selecionamos as arestas $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ na Fase 2. Podemos limitar o número de arestas removidas identificando todos os trios que poderiam suportar arestas de duos conflitantes e limitando o número dessas arestas que eles poderiam ter suportado. Lembre-se de que um trio suporta uma aresta de duos se ela pertence a uma aresta de trios em $M_T$ e, portanto, fez com que a aresta de duos fosse adicionada a $G_D$.

Em primeiro lugar, existem quatro trios a uma distância dois de $i$ e $j$ que podem, cada um, suportar no máximo uma aresta conflitante. São os trios $T_{i-2}^A$, $T_{i+2}^A$, $T_{j-2}^B$ e $T_{j+2}^B$. Em segundo lugar, existem quatro trios a uma distância um. Três deles, $T_{i+1}^A$, $T_{j-1}^B$, e $T_{j+1}^B$, podem suportar duas arestas conflitantes. No entanto, o quarto trio, $T_{i-1}^A$, pode suportar no máximo uma aresta conflitante, uma vez que escolhemos o menor $i$ tal que $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$ são pares de arestas paralelas consecutivas.

Além dessas arestas conflitantes, também removemos as duas arestas $(D_i^A, D_j^B)$ e $(D_{i+1}^A, D_{j+1}^B)$, levando a um peso total removido de

$$4(1/2) + 7(1/2) + 2(1/2) = 6.5.$$

Segue que a razão entre o número de duos preservados adicionados a $S$ e o peso das arestas removidas de $G_D$ é pelo menos $2/6,5 = 4/13$. □

**Lema 25.** *O número de duos preservados adicionados a $S$ em cada iteração da Fase 3 da Etapa 4 é pelo menos 1/3 do peso total das arestas removidas de $G_D$ naquela iteração.*

*Demonstração.* Suponha que selecionamos a aresta de duo $(D_i^A, D_j^B)$ em alguma iteração da Fase 3. Podemos limitar o peso das arestas removidas de $G_D$ contando o número de trios que poderiam ter suportado arestas de duo que conflitam com $(D_i^A, D_j^B)$. Lembre-se de que um trio suporta uma aresta de duo se ela pertence a uma aresta de trio em $M_T$ e, portanto, fez com que a aresta de duo fosse adicionada a $G_D$. Como removemos todos os pares de arestas paralelas consecutivas na Fase 2, cada trio pode suportar no máximo uma aresta de duo restante em $G_D$.

Existem oito trios que podem potencialmente suportar uma aresta de duo conflitante: $T_{i-2}^A$, $T_{i-1}^A$, $T_i^A$, $T_{i+1}^A$, $T_{j-2}^B$, $T_{j-1}^B$, $T_j^B$, $T_{j+1}^B$. Observe que selecionamos as arestas começando no início de $A$ e avançamos em direção ao final. Portanto, qualquer aresta suportada pelo trio $T_{i-2}^A$ já teria sido selecionada ou removida antes da iteração atual. Além disso, observe

que dois desses trios devem suportar a aresta atualmente selecionada. Portanto, removemos a aresta de duo selecionada $(D_i^A, D_j^B)$ e no máximo cinco outras arestas de duo. Cada uma dessas arestas tem peso no máximo 1/2, uma vez que todas as arestas de peso 1 foram removidas na Fase 1. Então, a soma dos pesos das arestas removidas é no máximo

$$1/2 + 5(1/2) = 3.$$

Segue que a razão entre o número de duos preservados adicionados a $S$ e o peso das arestas removidas de $G_D$ é pelo menos 1/3. □

**Lema 26.** *Ao final da Etapa* 4*, não restam arestas em* $G_D$*.*

*Demonstração.* A Fase 3 itera através de cada aresta restante em $G_D$, removendo, assim, todas elas. □

### B.6.3 Conclusão interessante sobre a abordagem de emparelhamentos

Na conclusão do artigo, o autor faz o seguinte comentário importante sobre uma questão que surge quando comparamos a abordagem de emparelhamento de trios com a de emparelhamento de duos.

Dado que o emparelhamento de trios permite uma melhoria sobre o fator de aproximação alcançado pela abordagem de emparelhamento de duos, proposto em [2], uma questão natural é se um emparelhamento de 4-uplas poderia produzir resultados ainda melhores. No entanto, uma extensão direta do trabalho neste artigo para uma abordagem de emparelhamento de 4-uplas não é possível, porque o emparelhamento de 4-uplas não forneceria um limite superior para o MPSM. O problema com essa abordagem é que o primeiro duo e o último duo em uma 4-upla não têm potencial para serem conflitantes e provavelmente podem não ser agrupados juntos. Pelo lado positivo, conjecturamos que a abordagem de emparelhamento de trios pode ser levada mais longe, para atingir uma 3-aproximação. O problema neste artigo, que pode ser visto como um "gargalo", surge da Fase 2 da Etapa 4, mas esperamos que esse obstáculo possa ser evitado de alguma forma.