O problema da subsequência comum máxima sem repetições

Carlos E. Ferreira[1], **Christian Tjandraatmadja**[1]

[1]Instituto de Matemática e Estatística, Universidade de São Paulo, Brasil

Abstract. *We study the following problem: given two sequences X and Y over a finite alphabet, find a longest common subsequence of X and Y without repeated symbols. This NP-hard problem appears in the context of genome rearrangement when gene duplication is considered. We describe an algorithm that solves the problem optimally based on the branch-and-cut technique. The algorithm takes advantage of good heuristics and of a simple but effective technique that eliminates variables during the algorithm. Finally, we describe an algorithm based on dynamic programming that takes linear time and space when the size of the alphabet is constant.*
Keywords: *Repetition-free longest common subsequence, longest common subsequence, integer programming, branch-and-cut, heuristics.*

Resumo. *Estudamos o seguinte problema: dadas duas sequências X e Y sobre um alfabeto finito, encontre uma subsequência comum máxima de X e Y sem símbolos repetidos. Este problema NP-difícil surge no contexto de rearranjo de genomas quando duplicação de genes é considerada. Descrevemos um algoritmo que resolve o problema de forma ótima baseado na técnica branch-and-cut. O algoritmo se aproveita de boas heurísticas e de uma técnica simples mas eficaz que elimina variáveis do problema no decorrer do algoritmo. Finalmente, descrevemos um algoritmo baseado em programação dinâmica que consome tempo e espaço linear quando o tamanho do alfabeto é constante.*
Palavras-chave: *Subsequência comum máxima sem repetições, subsequência comum máxima, programação inteira, branch-and-cut, heurísticas.*

1. Introdução

No contexto de análise de rearranjo de genomas, uma medida de similaridade útil entre dois genomas é o comprimento de uma subsequência comum máxima entre os genomas. Porém, esta medida não considera o evento de duplicação de genes por ser infrequente em genomas simples e tornar o problema computacionalmente mais difícil. Para levá-lo em consideração, Sankoff [Sankoff and Kruskal 1983] propôs o conceito de distância exemplar, que considera um representante para cada família de genes duplicados.

Neste trabalho, consideramos uma medida de similaridade proposta por Adi et al. [Adi et al. 2010] também baseada nesse conceito. A medida é o comprimento de uma subsequência comum máxima com a restrição de que existe no máximo um representante de cada família de genes duplicados na sequência.

Denotamos o comprimento de uma sequência S por $|S|$, o i-ésimo símbolo de S por s_i e a sequência $s_i s_{i+1} \dots s_j$ por $S[i..j]$. Uma *subsequência* de S é uma sequência obtida removendo zero ou mais símbolos de S (não necessariamente contígua em relação a S). Uma *subsequência comum* de S e T é uma subsequência de tanto S como T. Ela é

sem repetições se cada símbolo aparece no máximo uma vez. Uma *subsequência comum máxima*, ou LCS (do inglês *longest common subsequence*), é uma subsequência comum de comprimento máximo. Uma *subsequência comum máxima sem repetições*, ou RFLCS (do inglês *repetition-free longest common subsequence*) é uma subsequência comum sem repetições de comprimento máximo. Por exemplo, *cad* é um RFLCS de *caadbd* e *abacdad*.

Estamos interessados em resolver o problema do RFLCS, que consiste em encontrar uma subsequência comum máxima sem repetições de duas sequências finitas X e Y sobre um alfabeto finito Σ. Foi provado em [Adi et al. 2010] que o problema do RFLCS é APX-difícil mesmo quando cada símbolo ocorre no máximo duas vezes em cada sequência X e Y. Isto também implica que o problema é NP-difícil.

Na seção seguinte, apresentamos uma formulação de programação inteira para o problema. A seguir, discutimos uma abordagem *branch-and-cut* e técnicas usadas. Finalmente, descrevemos um algoritmo de programação dinâmica para casos de alfabeto pequeno. Estes tópicos são abordados de forma bastante superficial. O leitor interessado em detalhes pode consultar a dissertação resumida por este texto [Tjandraatmadja 2010].

2. Formulação

Um par ordenado (i, j) é um *casamento* sobre X e Y (de símbolo σ) se $x_i = y_j$ $(= \sigma)$. Seja $\mathcal{C}$ o conjunto de todos os casamentos sobre X e Y. Dizemos que dois casamentos (i, j) e (k, l) se *cruzam* se $i \leq k$ e $j \geq l$, ou $i \geq k$ e $j \leq l$. Como representado visualmente na Figura 1, eles se cruzam se os segmentos que o representam se intersectam graficamente, incluindo o caso em que eles têm um extremo em comum. Dizemos que dois casamentos (i, j) e (k, l) são *conflitantes* se eles se cruzam ou são do mesmo símbolo.

Seja C um conjunto de casamentos não-conflitantes dois a dois. É possível ordenar os casamentos de C em ordem crescente e tomar seus símbolos nessa ordem para obter S, uma subsequência comum sem repetições de X e Y. Dizemos que C *define* S. Por exemplo, se $X =$ *caadbd* e $Y =$ *abacdad*, o conjunto de casamentos $\{(1, 4), (2, 6), (6, 7)\}$ define *cad*. Ademais, qualquer subsequência comum sem repetições é definida por algum conjunto desse tipo. Portanto, o problema do RFLCS pode ser visto como encontrar um conjunto de casamentos não-conflitantes dois a dois de cardinalidade máxima.

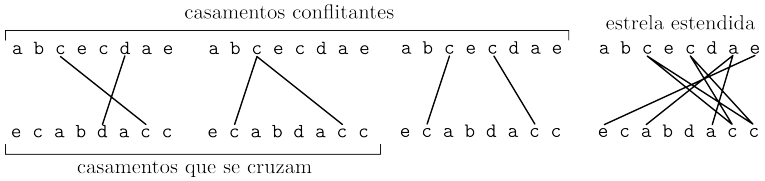

Figura 1. Casamentos que se cruzam, conflitantes e estrela estendida

Uma *estrela estendida* S é um conjunto de casamentos conflitantes dois a dois. Ela é *maximal* se não existe casamento em $\mathcal{C} \setminus S$ conflitante com todo casamento em S.

Construiremos uma formulação de programação linear inteira. Uma *formulação de programação linear* descreve um problema de otimização através de uma função linear que se deseja maximizar ou minimizar (*função objetivo*) e um conjunto de inequações lineares sobre um conjunto de variáveis. Se restringirmos variáveis a serem inteiras, ela é

uma *formulação de programação inteira*. Uma *solução viável* é um conjunto de valores para as variáveis que satisfaz as restrições, e ela é *ótima* se ela otimiza a função objetivo.

A formulação a seguir modela o problema do RFLCS, pois cada uma de suas soluções viáveis corresponde a um conjunto de casamentos não-conflitantes dois a dois de $\mathcal{C}$ e vice-versa. Seja z_{ij} uma variável binária para um casamento (i, j) em $\mathcal{C}$.

$$\begin{aligned} \max \quad & \sum_{(i,j)\in\mathcal{C}} z_{ij} & & \\ \text{s. a} \quad & \sum_{(i,j)\in K} z_{ij} \;\leq\; 1 & & \text{para cada estrela estendida maximal } K \subseteq \mathcal{C}, \\ & z_{ij} \;\in\; \{0,1\} & & \text{para cada } (i,j) \in \mathcal{C}. \end{aligned}$$

Provamos em [Fernandes et al. 2008] que as inequações acima definem facetas, ou seja, são necessárias na descrição do poliedro dado pela formulação.

Neste trabalho, usamos a técnica *branch-and-cut* para resolver o problema. Existem algoritmos eficientes para resolver um problema de programação linear, mas a formulação acima apresenta dois obstáculos: restrições inteiras e um número exponencial de restrições lineares. Para lidar com o primeiro obstáculo, é construída uma árvore de subproblemas menos restritos, com apenas restrições lineares, em que cada ramificação fixa uma variável em 0 e em 1 (para o caso binário), com a ajuda de uma estratégia de poda. Para lidar com o segundo, as restrições são adicionadas por demanda: a cada solução ótima encontrada para um problema relaxado, procuramos uma restrição violada por tal solução, adicionamos-na, e reotimizamos o problema. Encontrar tal restrição é o *problema de separação*, discutido a seguir. Esta técnica funciona bem na prática.

3. Algoritmo de separação

O problema de separação para inequações de estrela estendida é o seguinte. Dadas duas sequências X e Y e uma solução viável z de um problema relaxado, encontre uma estrela estendida maximal K tal que $\sum_{(i,j)\in K} z_{ij} > 1$, ou informe se não houver tal K. Portanto, é suficiente encontrar uma estrela estendida maximal de peso máximo usando o valor z_{ij} como o peso de (i, j) para cada $(i, j) \in \mathcal{C}$ (denotada por *estrela estendida máxima*).

Realizaremos uma pequena transformação por conveniência. Se invertermos uma das sequências, digamos X, toda estrela estendida se torna um conjunto de casamentos no qual quaisquer dois casamentos de símbolos diferentes não são conflitantes. Denotamos este conjunto por *emparelhamento planar estendido* e desejamos encontrar tal conjunto.

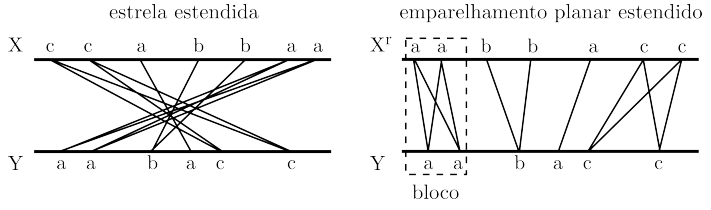

Figura 2. Estrela estendida e emparelhamento planar estendido correspondente.

Sejam (i', j') e (i, j) dois casamentos de um mesmo símbolo σ. Denote por *bloco* de ínicio (i', j') e fim (i, j) o conjunto de casamentos (k, l) de símbolo σ tal que $i' \leq k \leq i$

e $j' \leq l \leq j$ (veja Figura 2). Dizemos que dois blocos se cruzam se existem dois casamentos, um em cada bloco, que se cruzam. Podemos interpretar um emparelhamento planar estendido como um conjunto de blocos que não se cruzam dois a dois.

Seja $L_e(i, j)$ o peso de um emparelhamento planar estendido máximo de $X[1..i]$ e $Y[1..j]$. A recorrência a seguir é inspirada em uma recorrência para o problema do LCS.

$$L_e(i,j) = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0, \\ \max\{L_e(i,j-1), L_e(i-1,j), B_{max}(i,j)\} & \text{se } x_i^r = y_j, \\ \max\{L_e(i,j-1), L_e(i-1,j)\} & \text{se } x_i^r \neq y_j; \end{cases}$$

onde $B_{max}(i, j)$ é o peso de um emparelhamento planar estendido máximo de $X[1..i]$ e $Y[1..j]$ que contém o casamento (i, j).

A solução $L_e(|X|, |Y|)$ e o emparelhamento planar estendido associado podem ser computados em tempo $O(|X|^2|Y|^2)$ usando programação dinâmica, isto é, preenchemos uma tabela $|X| \times |Y|$ linha por linha de acordo com a recorrência acima. Uma modificação não-trivial para obtermos uma melhoria de tempo pode ser feita adaptando a ideia do algoritmo para o problema do LCS de Hunt e Szymanski [Hunt and Szymanski 1977].

4. Eliminação de variáveis

Dado um casamento (i, j), denote por $u_{pref}(i, j)$ e $u_{suf}(i, j)$ um limitante superior para o comprimento de um RFLCS de $X[1..i-1]$ e $Y[1..j-1]$, e de $X[i+1..|X|]$ e $Y[j+1..|Y|]$ respectivamente. Dizemos que a função $u(i, j) = u_{pref}(i, j) + u_{suf}(i, j) + 1$ é um *limitante de casamento* para (i, j).

Pode-se provar que, dado um casamento (i, j), se existe uma subsequência comum sem repetições de X e Y de comprimento maior que $u(i, j)$, então (i, j) não pertence a um conjunto que define um RFLCS de X e Y. Este fato pode ser usado para fixar variáveis em zero conforme encontramos soluções viáveis. Como limitante superior, usamos o menor entre o comprimento de um LCS de X e Y e o número de símbolos distintos em X e Y.

5. Heurísticas

Heurísticas são usadas para obter limitantes primais para o algoritmo *branch-and-cut*, o que ajuda a obter uma solução ótima mais cedo e a eliminar variáveis através da técnica anterior. Descrevemos brevemente as heurísticas que mais contribuem ao algoritmo.

As heurísticas seguintes provêm ao algoritmo *branch-and-cut* um limitante inicial, mas também podem ser executadas independentemente dele.

LCS e remove repetições. Compute uma subsequência comum máxima das sequências e remova ocorrências repetidas de símbolos. Ele é eficiente e tem bons resultados.

Limitante de casamento. Esta heurística procura escolher recursivamente casamentos de alto limitante de casamento, que têm o potencial de pertencer a longas subsequências comuns sem repetições. Ela geralmente devolve a melhor solução entre as implementadas, embora tenha maior tempo de execução.

As heurísticas seguintes usam como parâmetro uma solução viável da relaxação e geralmente devolvem bons resultados quando ela está perto da ótima. Assim, elas podem ser executadas várias vezes durante o processo. Seja z uma solução viável da relaxação.

LCS com peso e remove repetições. Compute um conjunto de casamentos que não se cruzam dois a dois de peso máximo usando z como peso e, da sequência definida por este conjunto, remova ocorrências repetidas de símbolos.

Escolha simples. Escolha de forma gulosa casamentos (i, j) com z_{ij} o mais perto de 1, ignorando aqueles que conflitam com algum casamento já escolhido.

6. Resultados computacionais

O gráfico a seguir (Figura 3) compara o algoritmo *branch-and-cut* com e sem as técnicas desenvolvidas[1]. As sequências X e Y são sequências uniformemente aleatórias de comprimento 1024 e o tamanho do alfabeto é variado. A resolução de programas lineares e gerenciamento da árvore *branch-and-cut* é realizada pelo pacote GLPK.

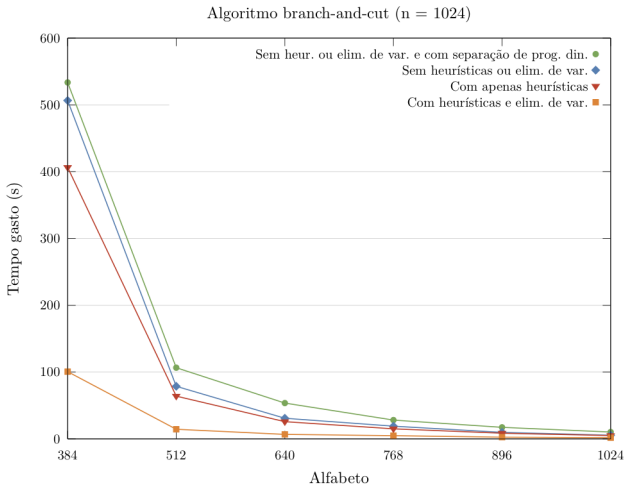

Figura 3. Gráfico ilustrando tempos de execução do algoritmo *branch-and-cut*.

O formato da curva é devido ao fato de que é mais difícil resolver instâncias com alfabeto pequeno usando *branch-and-cut*, pois há um número alto de repetições e variáveis. Os resultados mostram que usar a técnica de eliminação de variáveis e heurísticas primais em conjunto melhoram significativamente o tempo de execução do algoritmo. Para os resultados numéricos em si e outros testes, pode-se consultar a dissertação [Tjandraatmadja 2010].

7. Algoritmo de programação dinâmica

Considere a seguinte recorrência. Seja $R(i, j, S) = 1$ se existe uma subsequência comum sem repetições cujos símbolos são os de S, ou 0 caso contrário.

$$R(i,j,S) = \begin{cases} 0 & \text{se } i = 0 \text{ ou } j = 0 \text{ (e } S \neq \varnothing), \\ 1 & \text{se } S = \varnothing, \\ \max\{R(i,j-1,S), R(i-1,j,S)\} & \text{se } x_i \neq y_j \text{ ou } x_i = y_j \notin S, \\ R(i-1,j-1,S \setminus \{x_i\}) & \text{se } x_i = y_j \in S. \end{cases}$$

[1] A separação de programação dinâmica a que a primeira legenda se refere é a separação sem a melhoria.

Seja $R'(i, S)$ o menor j tal que $R(i, j, S) = 1$. Baseado em $R'(i, S)$, podemos desenvolver um algoritmo de programação dinâmica usando um vetor de tamanho $|X|$ para cada conjunto S. O algoritmo consome tempo $O(|\Sigma|n+m2^{|\Sigma|}|\Sigma|)$ e espaço $O(|\Sigma|n+(m+|\Sigma|)2^{|\Sigma|})$, supondo consumo de tempo e espaço $O(|S|)$ para acessar e armazenar os conjuntos S. Logo, se $|\Sigma|$ for constante, temos um algoritmo de tempo e espaço $O(n+m)$.

8. Conclusão

Neste projeto, exploramos o problema do RFLCS particularmente do ponto de vista de programação inteira, resultando em um algoritmo *branch-and-cut* e técnicas e heurísticas para melhorá-lo significativamente. Além disso, desenvolvemos um algoritmo de programação dinâmica eficiente na prática para alfabetos de tamanho baixo (menos de 20).

Notas sobre o trabalho

A parte central deste trabalho foi publicada e apresentada no ISCO 2010 (International Symposium on Combinatorial Optimization) [Ferreira and Tjandraatmadja 2010][2]. Uma versão do código-fonte do algoritmo *branch-and-cut* que usa CPLEX foi publicada em `http://www.ime.usp.br/~cef/rflcs/`.

Referências

Adi, S., Braga, M., Fernandes, C., Ferreira, C., Martinez, F., Sagot, M.-F., Stefanes, M., Tjandraatmadja, C., and Wakabayashi, Y. (2010). Repetition-free longest common subsequence. *Discrete Applied Mathematics*, 158(12):1315–1324. Traces from LAGOS'07 IV Latin American Algorithms, Graphs, and Optimization Symposium Puerto Varas - 2007.

Fernandes, C., Ferreira, C., Tjandraatmadja, C., and Wakabayashi, Y. (2008). A polyhedral investigation of the LCS problem and a repetition-free variant. In *Proc. of the 8th Latin American Symposium on Theoretical Informatics (LATIN'08)*, volume 4957 of *Lecture Notes in Computer Science*, pages 329–338. Springer.

Ferreira, C. and Tjandraatmadja, C. (2010). A branch-and-cut approach to the repetition-free longest common subsequence problem. *Electronic Notes in Discrete Mathematics*, 36:527–534. ISCO 2010 - International Symposium on Combinatorial Optimization.

Hunt, J. and Szymanski, T. (1977). A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353.

Sankoff, D. and Kruskal, J. (1983). *Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison*. Addison-Wesley.

Tjandraatmadja, C. (2010). O problema da subsequência comum máxima. Dissertação de mestrado.

[2] As duas outras referências citadas neste texto sobre o problema do RFLCS, para as quais o autor também contribuiu, foram desenvolvidas antes do projeto de mestrado. Elas têm foco em dificuldade e aproximabilidade do problema [Adi et al. 2010] e em características poliédricas de formulações para os problemas do LCS e RFLCS [Fernandes et al. 2008].