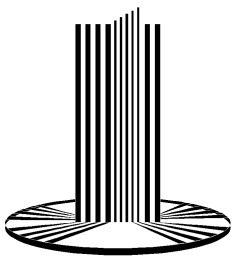
ALGORITMOS PARA ALINHAMENTO DE REDES METABÓLICAS

Diego Padilha Rubert

Dissertação de Mestrado

Orientação: Prof. Dr. Fábio Henrique Viduani Martinez

Área de Concentração: Teoria da Computação



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
Junho/2012

ALGORITMOS PARA ALINHAMENTO DE REDES METABÓLICAS

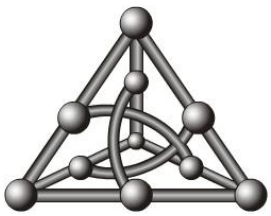
Diego Padilha Rubert

Dissertação de Mestrado

Orientação: Prof. Dr. Fábio Henrique Viduani Martinez

Área de Concentração: Teoria da Computação

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, Curso de Pós-Graduação em Ciência da Computação, Faculdade de Computação da Fundação Universidade Federal de Mato Grosso do Sul.



Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
Junho/2012

Agradecimentos

Agradeço primeiramente ao meu orientador Fábio, por sua paciência, compreensão, bom ânimo e por me conduzir tão bem no desenvolvimento deste trabalho.

À minha esposa Evelyn, pelo seu amparo em todos os momentos e por compreender as inúmeras noites em que a deixei sozinha.

À minha família e amigos, em especial à minha mãe Lisete e ao meu irmão Cássio, por me apoiarem diante a tantas ausências necessárias ao estudo no mestrado.

Enfim, agradeço a Deus pela oportunidade de viver e ter essas pessoas ao meu lado.

Resumo

O alinhamento de redes metabólicas é um tópico habitual no contexto da biologia computacional. Por ele podemos, por exemplo, aferir relações de evolução, parentesco ou funcionais entre espécies. O problema pode ser modelado nas mais variadas formas e trabalhado utilizando diferentes metodologias.

Neste trabalho buscamos descrever uma visão geral sobre o assunto, todavia, dedicando algumas páginas ao estudo aprofundado de recentes trabalhos de relevância sobre o tema à luz da teoria dos grafos. Inicialmente apresentamos definições básicas necessárias ao estudo do tema, da biologia à teoria da computação, seguidas pela exposição das modelagens mais comuns de redes metabólicas. Passamos ao alinhamento de sequências, vias e redes metabólicas, sendo os dois últimos o foco do trabalho, estudando detalhadamente alguns algoritmos. Apresentamos ainda uma síntese acerca de trabalhos relacionados ao trabalho corrente.

Palavras-Chave: algoritmos, alinhamento, vias metabólicas, redes metabólicas

Abstract

The alignment of metabolic networks is an usual topic in the context of computational biology. Studying it we can, for example, measure evolutionary, lineage or functional relationships between species. The problem can be modeled in many ways and handled using different methodologies.

In this work we want to achieve a survey about the subject, devoting a few pages to the study of recent work of relevance to the topic using graph theory. First we present basic definitions necessary to the study of subject, from biology to theoretical computer science, followed by exposure of the most common ways of modeling metabolic networks. Then we discuss the alignment of sequences, pathways and metabolic networks, the last two being the focus of this work, studying some algorithms in detail. We also present an overview of related work.

Keywords: algorithms, alignment, metabolic pathways, metabolic networks

Sumário

Capítulo 1

Introdução

Redes metabólicas são conjuntos de reações químicas que ocorrem dentro das células de organismos vivos e constituem a base da vida. O metabolismo vem sendo estudado há séculos, indo do estudo de animais como um todo ao exame de reações individuais do metabolismo nos tempos atuais. No passado remoto, os mecanismos dos processos metabólicos não haviam sido identificados e se pensava que os tecidos vivos eram animados por uma força vital. Com o passar do tempo, o estudo das células já era realizado, mas como um todo. Apenas após o descobrimento das enzimas no século XX por Eduard Buchner separou-se o estudo de reações químicas no metabolismo do estudo biológico das células. O conhecimento bioquímico cresceu então rapidamente, incluindo técnicas que permitiram a descoberta e análise detalhada das vias metabólicas, as quais compõem uma rede metabólica de um organismo.

Um grande volume de informações tem sido produzido pelo mapeamento de redes metabólicas de diversos organismos, disponibilizadas em bases de dados públicas como KEGG, BioCyc, EcoCyc, MetaCyc, ERGO, ENZYME, metaTIGER, BRENDA, entre outras, fazendo-se necessária a criação de métodos para análise e processamento desses dados. Um tipo de análise muito comum na bioinformática é a comparação de redes metabólicas pelo chamado "alinhamento". Nesse contexto, alinhamento refere-se a comparar e identificar semelhanças entre padrões apresentados, podendo representar relações funcionais, estruturais ou evolucionárias.

Diversos trabalhos estão sendo desenvolvidos no intuito de realizar o alinhamento de redes metabólicas como um todo ou o alinhamento de vias metabólicas. Uma leitura introdutória sobre o tema pode ser vista em [31]. Os trabalhos em [9, 8, 7] descrevem um apanhado geral sobre o alinhamento de redes metabólicas, cobrindo modelagens, diversos tipos de alinhamentos e trabalhos pretéritos e relacionados.

Chen e Hofestädt [6] implementaram uma ferramenta que constrói por um método simples o alinhamento entre vias extraindo informações de bancos de dados na internet. Em [9], Cheng *et al.* utilizaram a ideia chamada de homomorfismo, juntamente com programação dinâmica, para alinhar uma via metabólica a uma rede, estendendo este trabalho em [8] para reduzir o tempo de processamento e remover algumas limitações encontradas no trabalho anterior. Podemos ver em [29] uma formulação para, diferentemente da maioria das referências, alinhar redes metabólicas utilizando o método de programação linear inteira. Já Berg e Lässig apresentaram em [2] o alinhamento de duas redes através de um método estatístico teórico. Em [52], Zaslavskiy *et al.* avaliaram os algoritmos para realizar o alinhamento de grafos que são o estado da arte e propuseram novos métodos que produzem resultados com qualidade superior. Flannick *et al.*

[19] apresentaram o algoritmo GRÆMLIN, que realiza o alinhamento múltiplo escalonável de redes e que permite a generalização de esquemas de pontuação de alinhamento existentes e a localização de topologias de rede conservadas. Podemos ver em [13] o algoritmo C3PART-M para o alinhamento de redes baseado em [4], sendo mais eficiente no caso de múltiplas redes e com resultados similares aos do algoritmo baseado em heurísticas [25]. Tohsato *et al.* [45] propuseram o alinhamento múltiplo de vias metabólicas não ramificadas baseando-se em um algoritmo de alinhamento de duas sequências. Pinter *et al.* [39] se basearam no homeomorfismo de subárvores para alinhar uma via com outra via ou uma rede metabólica. Yang e Sze [51] apresentaram um algoritmo para alinhar uma via não ramificada com uma rede e um algoritmo para alinhar uma via ou parte de uma rede com uma rede metabólica, devolvendo as melhores soluções encontradas até um limite pré-estabelecido. O foco do nosso trabalho é justamente as últimas três publicações, as duas primeiras por serem referências de grande importância no contexto aqui estudado e a última por ser uma publicação recente que apresenta ideias diferentes das anteriores e permite o alinhamento de redes metabólicas com uma complexidade de tempo aceitável.

No trabalho corrente, dedicamos o Capítulo 2 exclusivamente às definições da biologia e da teoria dos grafos e aos métodos de modelagens de redes metabólicas em grafos. Procuramos cobrir as modelagens mais comuns e suas variantes, mesmo várias delas não possuindo aqui utilidade prática, apresentando também alguns problemas relacionados e possíveis soluções.

A seguir, no Capítulo 3, tratamos do problema de alinhamento de sequências e suas variantes, problema muito utilizado para aferir semelhanças entre sequências de caracteres na computação e sequências de DNA na biologia. O problema ganha ainda maior relevância por servir como base para questões tratadas posteriormente. No mesmo capítulo apresentamos a classificação de enzimas e sua hierarquia, juntamente de uma visão geral sobre o alinhamento de vias e redes metabólicas, tema tratado em capítulos específicos por serem o foco deste estudo.

O Capítulo 4 sobre o alinhamento de vias e redes metabólicas apresenta algoritmos para: (i) realizar o alinhamento múltiplo de vias metabólicas não ramificadas, (ii) alinhar uma via com uma via ou uma rede metabólica, (iii) alinhar uma via não ramificada com uma rede e (iv) alinhar uma via ou parte de uma rede com uma rede metabólica. Os algoritmos são estudados minuciosamente, apresentando todas definições e informações necessárias para o entendimento. Apresentamos ainda descrições na forma de algoritmos para uma melhor compreensão, sendo estas mais detalhadas ou inexistentes nos trabalhos de referência.

O Capítulo 5 enumera alguns trabalhos relacionados de destaque dentre a abrangente literatura, apresenta a seguir as conclusões deste trabalho e encerra-se relacionando ainda algumas sugestões de trabalhos futuros.

Capítulo 2

Redes metabólicas

Redes metabólicas são conjuntos de reações químicas que ocorrem dentro das células de organismos vivos. Estas reações são responsáveis pelos processos de síntese e degradação dos nutrientes na célula e constituem a base da vida, permitindo o crescimento e reprodução das células, mantendo as suas estruturas e adequando respostas aos seus ambientes. Contudo, para o estudo e processamento computacional das redes metabólicas é necessário que sejam modeladas de alguma forma.

O presente capítulo é iniciado por uma seção dedicada a definições necessárias para a compreensão do assunto. As definições são divididas em definições da biologia e da teoria dos grafos. A seguir apresentamos os métodos mais comuns de modelagem computacional de redes metabólicas, expondo considerações sobre variações, restrições, vantagens, desvantagens e possíveis simplificações.

2.1 Definições

Temos na seção corrente duas subseções dedicadas a definições, sendo elas responsáveis por apresentar definições da biologia e da teoria dos grafos, respectivamente. Referente a este último assunto, sugerimos como referência [3, 18, 20] para uma compreensão mais abrangente.

2.1.1 Biologia

Metabolismo é o conjunto de reações químicas intracelulares que ocorrem em organismos vivos, sendo elas responsáveis pelo fornecimento de energia para a manutenção da vida. Uma **reação química** é uma transformação da matéria na qual ocorrem mudanças qualitativas na composição química de uma ou mais substâncias, resultando em uma ou mais substâncias. Tais reações químicas intracelulares podem ser chamadas **reações bioquímicas**.

Reações bioquímicas envolvem enzimas e compostos químicos. **Enzimas** são substâncias, normalmente formadas de proteínas, responsáveis por acelerar ou permitir que reações bioquímicas se realizem. Um **composto químico**, também chamado **metabólito**, é uma substância formada por dois ou mais elementos químicos, ligados em uma proporção fixa e definida. Como exemplo, podemos citar a água (H_2O), que é um composto formado por hidrogênio e oxigênio na proporção de dois para um.

Algumas reações podem ocorrer naturalmente, mas a maioria é catalisada por enzimas, consumindo alguns compostos, chamados **substratos**, e produzindo outros, chamados **produtos**. Apresentamos na Figura 2.1 um diagrama UML simplificado dos objetos envolvidos em reações químicas.

Reações químicas podem ser escritas da seguinte forma: se uma dada reação consome metabólitos A e B produzindo C, escrevemos A + B → C. Podemos também atribuir rótulos às reações, escrevendo R1: A + B → C, assim atribuindo o rótulo R1 à reação. As reações podem ocorrer também no sentido inverso, sendo chamadas então de **inversíveis**. Por exemplo, uma reação inversível A + B → C poderá também ocorrer de forma inversa, isto é, C → A + B. Portanto, escrevemos tal reação como A + B ↔ C. Reações que não são inversíveis são definidas como **irreversíveis**.

O conjunto de reações bioquímicas em sequência é chamado **via metabólica**. Nesse caso, os produtos de uma reação serão os substratos da subsequente. Uma **rede metabólica** é um conjunto de vias metabólicas de um organismo, sobrepostas ou não.

Uma visão detalhada sobre reações químicas no metabolismo foge ao escopo deste trabalho, mas o leitor interessado em mais detalhes pode verificar a referência [32].

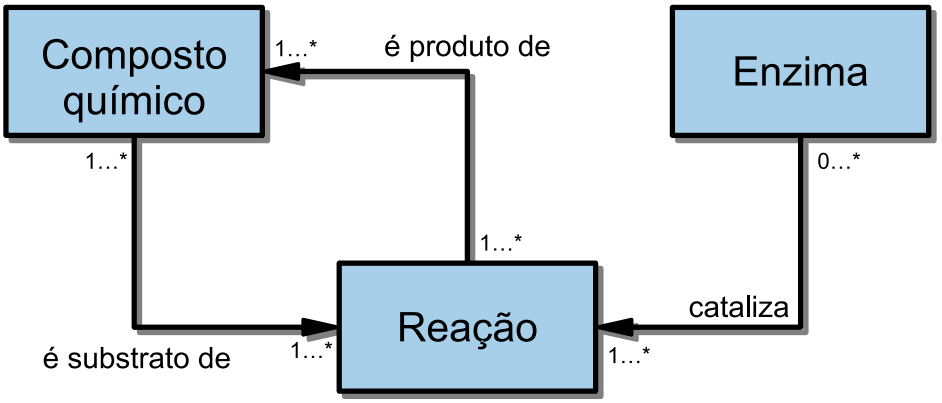


Figura 2.1: Diagrama UML simplificado dos objetos envolvidos em reações químicas de uma rede metabólica.

2.1.2 Teoria dos grafos

Um **grafo** é um par ordenado de conjuntos disjuntos (V, E), tal que E é um subconjunto do conjunto $V^{(2)}$ de pares não ordenados de V. V e E são os conjuntos de **vértices** e **arestas** (ou **arcos**), respectivamente. Se $G = (V, E)$ é um grafo, então $V = V(G)$ é o conjunto de vértices e $E = E(G)$ é o conjunto de arestas de G. Se $\{x, y\}$ é uma aresta, dizemos que ela **incide** nos (ou **liga** os) vértices x e y, e é denotada por xy. Assim, xy e yx são a mesma aresta e x e y são os **extremos** ou **pontas** da aresta.

Quando $xy \in E(G)$, x e y são **vértices vizinhos** ou **vértices adjacentes** de G. O **grau** de um vértice x representa a quantidade de arestas incidentes nele e é denotado por $grau(x)$. Temos duas ou mais **arestas adjacentes** quando essas arestas incidem em um mesmo vértice. Para um

exemplo de um grafo, veja a Figura 2.2a. Um **emparelhamento** em um grafo é um conjunto de arestas não adjacentes entre si. Dizemos que cada aresta desse conjunto **cobre** os vértices nos quais incide. Temos um **emparelhamento máximo** se ele possuir a maior quantidade possível de arestas e um **emparelhamento perfeito** se todos os vértices do grafo forem adjacentes a alguma aresta desse conjunto. Um grafo é **rotulado** em vértices ou arestas quando a cada vértice ou aresta, respectivamente, estiver associado um rótulo. Um grafo **ponderado** possui um valor numérico associado a cada aresta, chamado **peso**. O peso de uma aresta e é denotado por $w_E(e)$. É possível também haver pesos associados aos vértices, denotados como $w_V(v)$ para um vértice v.

Dizemos que $G'(V', E')$ é um **subgrafo** de $G(V, E)$ se $V' \subseteq V$ e $E' \subseteq E$. O subgrafo de G **induzido** por um subconjunto V' de $V(G)$ é o grafo $G = (V', E')$ em que E' é o conjunto de todas as arestas de G que incidem apenas em vértices que estejam em V'. Esse subgrafo é denotado por $G[V']$.

Um **passeio** em um grafo é uma sequência alternada de vértices e arestas de G, na forma $x_0, e_1, x_1, e_2, \ldots, e_t, x_t$ com $e_i = x_{i-1}x_i$ para $0 < i \leq t$. De forma simplificada, podemos expressar tal passeio como $x_0, x_1, \ldots, x_t$, onde dizemos que há um passeio de x_0 até x_t e que seu **tamanho** é t. Um passeio é chamado de **trilha** se suas arestas são distintas e é chamado **caminho** se seus vértices são distintos. A **distância** entre dois vértices é o tamanho do menor caminho entre eles. Um **ciclo** é um tipo de caminho em particular, onde $x_0 = x_t$, ou seja, apenas o primeiro vértice do caminho se repete uma única vez, sendo o último do caminho. Um grafo **acíclico** é aquele que não possui ciclo. Dizemos que um grafo é **conexo** se para cada par de vértices x e y distintos existe um caminho de x para y. Um **componente** de um grafo é um subgrafo conexo maximal. Um grafo **não conexo** é aquele que possui dois ou mais componentes. Uma **aresta de corte** ou **ponte** é uma aresta cuja remoção em um grafo aumenta o número de componentes conexos deste. Da mesma forma, um **vértice de corte** é um vértice tal que sua remoção provoca um aumento no número de componentes conexos.

Dois grafos $G = (V, E)$ e $G' = (V', E')$ são **isomorfos** se existe uma correspondência entre seus conjuntos de vértice que preserve adjacências, ou seja, se existe um bijeção $\phi : V \to V'$ de forma que uma aresta $xy \in E$ se e somente se $\phi(x)\phi(y) \in E'$.

Grafo **bipartido** é aquele no qual $V = V_1 \cup V_2$ e $V_1 \cap V_2 = \emptyset$, tal que para toda aresta $xy \in E$, $x \in V_1$ e $y \in V_2$, ou vice-versa. Um grafo bipartido pode ser denotado como $G = (V_1 \cup V_2, E)$ e dizemos que G tem uma bipartição (V_1, V_2). Para um exemplo de um grafo bipartido, veja a Figura 2.2c.

Um grafo $G = (V, E)$ é dito **direcionado** ou **orientado** quando E é um conjunto de pares ordenados. Portanto, existe uma direção associada à cada aresta. Nesse caso, a aresta xy tem x como **origem** e y como **destino** e yx não se refere à mesma aresta. Além disso, cada vértice x possui um **grau de entrada**, denotado por $grau^-(x)$, que representa o número de arestas em que é destino e um **grau de saída**, denotado por $grau^+(x)$, que representa o número de arestas em que é origem. Um vértice x é chamado de **fonte** se $grau^-(x) = 0$ e **sorvedouro** se $grau^+(x) = 0$. Para um exemplo de um grafo orientado, veja a Figura 2.2b.

Um **hipergrafo** é uma generalização de um grafo, onde uma aresta pode incidir em vários vértices. Formalmente, um hipergrafo H é um par ordenado (V, E), onde V é o conjunto de vértices e E o conjunto de hiperarestas. **Hiperaresta** ou **hiperarco** é um subconjunto não vazio de V. Como em um grafo, podemos ter um **hipergrafo orientado** onde cada hiperaresta $e \in E$ é um par ordenado $e = (S, T)$, sendo $S \subseteq V$ o conjunto de vértices fonte e $T \subseteq V$ o

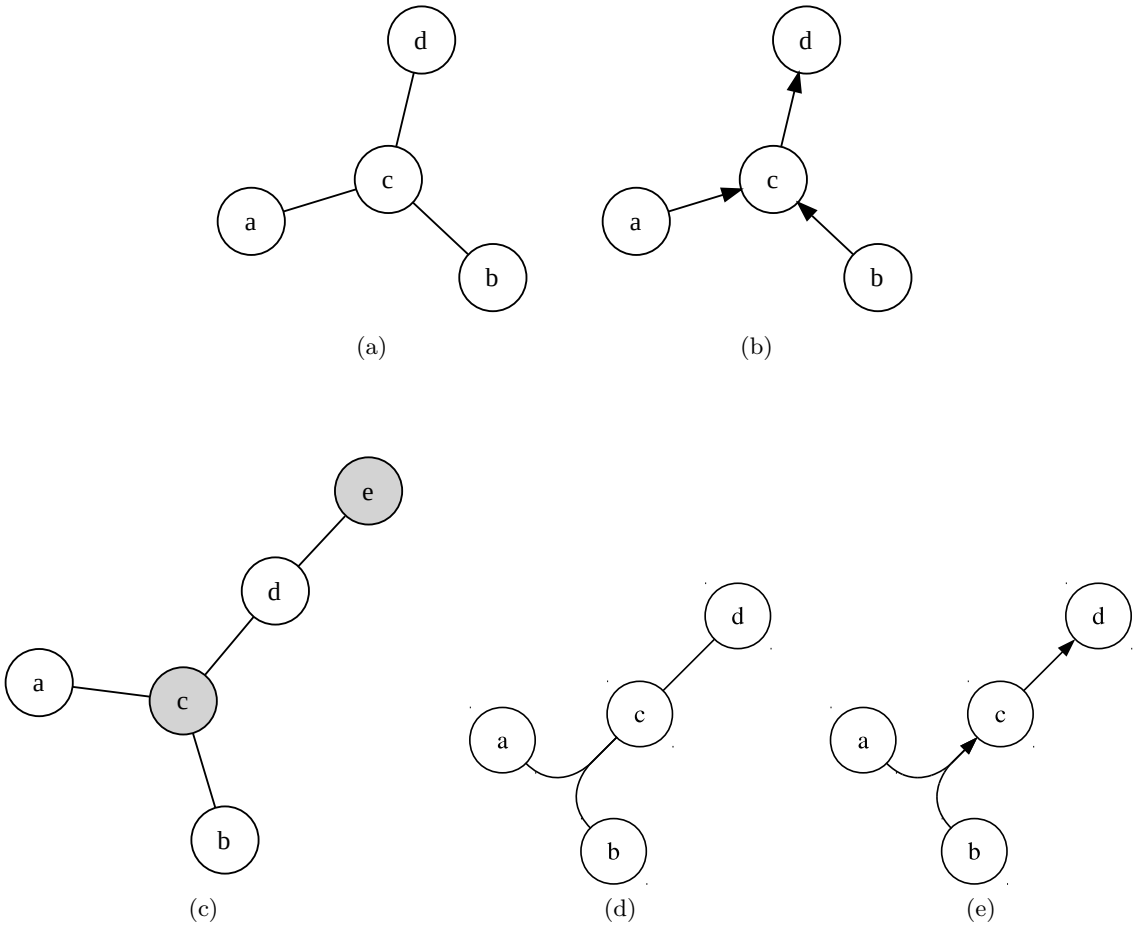


Figura 2.2: Representações gráficas de (a) grafo, com $V = \{a, b, c, d\}$ e $E = \{ac, bc, cd\}$; (b) grafo orientado, com $V = \{a, b, c, d\}$ e $E = \{ac, bc, cd\}$; (c) grafo bipartido, com $V = \{a, b, c, d, e\}$, $V_1 = \{a, b, d\}$, $V_2 = \{c, e\}$ e $E = \{ac, bc, cd, de\}$; (d) hipergrafo, com $V = \{a, b, c, d\}$ e $E = \{\{a, b, c\}, \{c, d\}\}$; (e) hipergrafo orientado, com $V = \{a, b, c, d\}$ e $E = \{(\{a, b\}, \{c\}), (\{c\}, \{d\})\}$.

conjunto de vértices alvo. Para exemplos ilustrados de um hipergrafo e um hipergrafo orientado, observe as Figuras 2.2d e 2.2e.

Uma **árvore** é um grafo não orientado conexo e acíclico. Em uma árvore, vértices de grau 1 são chamados **folhas** e os restantes de vértices **internos**. Uma **subárvore** é um subgrafo conexo de uma árvore. Uma **árvore com raiz** é aquela onde há um vértice especial chamado raiz. Considere um grafo G com vértice raiz p_0 e um caminho $p_0, p_1, \ldots, p_{n-1}, p_n$ em G. O vértice p_{n-1} é chamado **pai** de p_n e este é chamado **filho** daquele. Além disso, qualquer vértice $p_0, \ldots, p_{n-1}$ é chamado **ascendente** de p_n e qualquer vértice $p_1, \ldots, p_n$ é chamado **descendente** de p_0. O número de filhos de um vértice x é denotado por $c(x)$.

Podemos ainda ter outros tipos de árvores, diferentes daquela definida no parágrafo anterior. Destacamos uma em especial, definida da seguinte forma: consideremos uma árvore onde adicionamos uma direção a cada aresta. O grafo resultante dessa operação é chamado de **árvore multifontes**. Nela, os vértices origem e destino de uma aresta são pai e filho, respectivamente. Exemplos das definições relacionadas a árvores podem ser vistos na Figura 2.3.

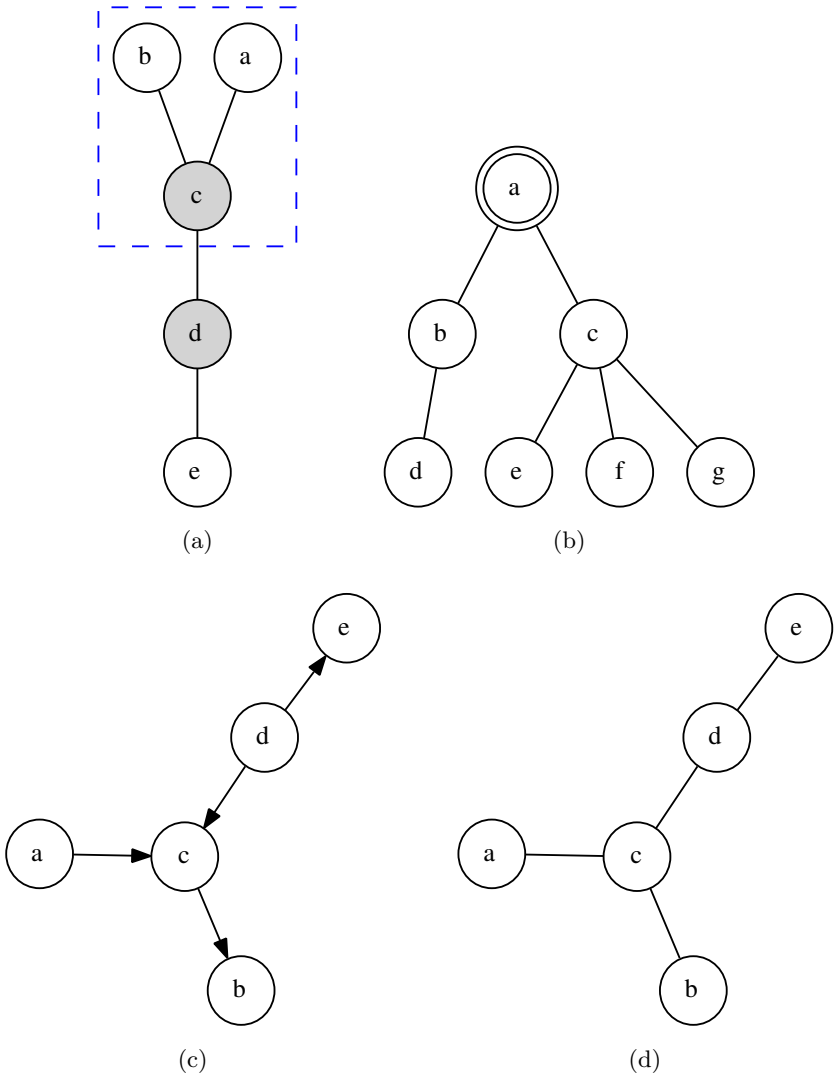


Figura 2.3: Representações gráficas de (a) árvore com vértices internos preenchidos e folhas não preenchidas, sendo a região delimitada por uma linha tracejada uma subárvore; (b) árvore com raiz, onde o vértice raiz está circulado; (c) árvore multifontes; (d) grafo do item anterior sem orientação, onde vemos uma árvore.

2.2 Modelagem de redes metabólicas

É possível modelar uma rede metabólica de várias maneiras, sendo as mais comuns utilizando grafos, modelos baseados em restrições e equações diferenciais. Considerando o objeto desse trabalho, apresentamos modelos baseados em grafos – a modelagem utilizada nesse caso. Para um estudo mais abrangente com outros métodos de modelagem, veja [31, 14].

Nas modelagens, consideramos uma rede metabólica como sendo formada por enzimas, metabólitos ou reações. Portanto, ao modelar uma rede, associamos ou definimos uma correspondência entre elementos da rede e elementos do grafo. Iniciaremos expondo três modelos simples e, então, dois outros modelos mais elaborados, que suprem deficiências e possuem van-

tagens em relação aos primeiros. Para as definições a seguir, consideremos G ou H um grafo ou hipergrafo que representa uma rede metabólica.

Grafo de compostos Seja V o conjunto de vértices do grafo G correspondendo aos compostos da rede e x e y dois compostos tais que $x, y \in V$. Existe aresta $xy \in E$ se, e somente se, existe uma reação onde x é substrato e y é produto. Esse modelo é claramente deficiente ao apresentar certas informações sobre a rede, visto que não carrega dados sobre enzimas e sobre reações em específico. Além disso, a representação é ambígua, visto que o mesmo grafo pode representar conjuntos distintos de reações, como mostra exemplo da Figura 2.4.

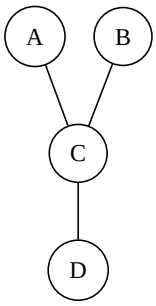


Figura 2.4: Ambiguidade no grafo de compostos. O mesmo grafo é gerado para os conjuntos de reações A $\leftrightarrow$ C, B $\leftrightarrow$ C, C $\leftrightarrow$ D e A + B $\leftrightarrow$ C, C $\leftrightarrow$ D.

Grafo de reações Seja V o conjunto de vértices do grafo G correspondendo às reações da rede e x e y duas reações tais que $x, y \in V$. Existe aresta $xy \in E$ se, e somente se, existe produto de x que é substrato de y. Da mesma forma que o anterior, essa abordagem pode levar a ambiguidades, como mostra a Figura 2.5. Ainda, deixa de apresentar informações, pois não prevê enzimas e reações em específico. Todavia, tanto no grafo de compostos quanto no de reações, adicionando enzimas como rótulos das arestas ou vértices, respectivamente, é possível preservar dados referentes à função das enzimas na rede.

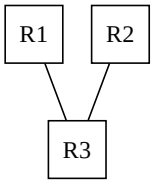


Figura 2.5: Ambiguidade no grafo de reações. O mesmo grafo é gerado para os conjuntos de reações R1: A $\leftrightarrow$ B, R2: C $\leftrightarrow$ D, R3: B + D $\leftrightarrow$ E e R1: A $\leftrightarrow$ B, R2: C $\leftrightarrow$ B, R3: B $\leftrightarrow$ E.

Grafo de enzimas Seja V o conjunto de vértices do grafo G correspondendo às enzimas da rede e x e y duas enzimas tais que $x, y \in V$. Existe aresta $x, y \in E$ se, e somente se, x e y catalisam reações que envolvem um metabólito em comum. Pode ser usado em casos onde se deve colocar em evidência as enzimas e as relações entre elas. Caso contrário, pode gerar um grafo contendo informações pouco relevantes, como a aproximação de compostos distantes ou sem relação na rede, mas que compartilham uma mesma enzima catalisadora nas reações em que são metabólitos, como mostra a Figura 2.6.

Grafo bipartido Seja V o conjunto de vértices do grafo bipartido G, seja (V_1, V_2) a bipartição de G, de modo que V_1 corresponde aos compostos e V_2 às reações, e seja x um composto

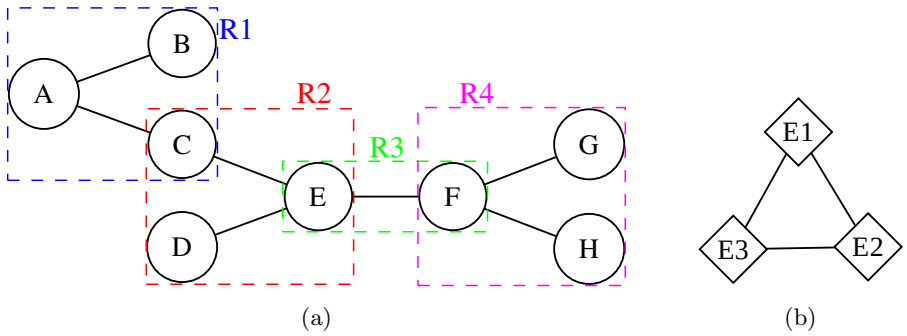


Figura 2.6: Representações gráficas do conjunto de reações R1: A ↔ B + C, R2: C + D ↔ E, R3: E ↔ F, R4: F ↔ G + H (a) usando grafo de compostos e (b) usando grafo de enzimas. Para o conjunto de reações apresentado, E1, E2 e E3 são enzimas, onde E1 catalisa R1 e R4, E2 catalisa R2 e E3 catalisa R3.

e y uma reação tais que $x \in V_1$ e $y \in V_2$. Existe aresta $xy \in E$ se, e somente se, x é substrato ou produto de y. Esse modelo amplia a capacidade de representação da rede em relação aos anteriores, evitando ambiguidades e suprindo algumas deficiências, como mostra a Figura 2.7.

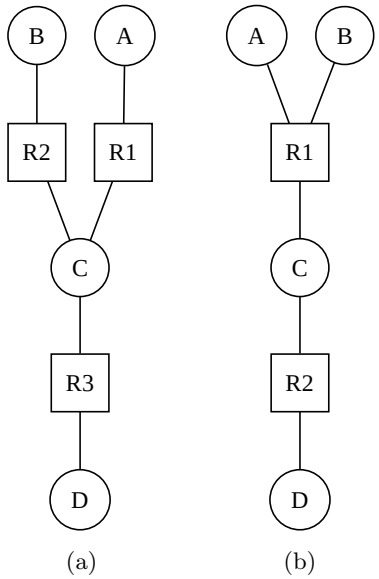


Figura 2.7: Grafos bipartidos para os conjuntos de reações (a) R1: A ↔ C, R2: B ↔ C, R3: C ↔ D e (b) R1: A + B ↔ C, R2: C ↔ D. Note que para esses conjuntos de reações, o mesmo grafo de compostos é gerado, como mostra a Figura 2.4.

Hipergrafo Seja V o conjunto de vértices do hipergrafo H correspondendo aos compostos da rede, E o conjunto $\{e_1, e_2, \cdots, e_m\}$ de hiperarestas correspondendo às reações e x um composto. O vértice $x \in e_i$ se, e somente se, x é metabólito da reação representada por e_i. Esse modelo representa de forma aprimorada uma rede da mesma forma que o anterior. Veja a Figura 2.8.

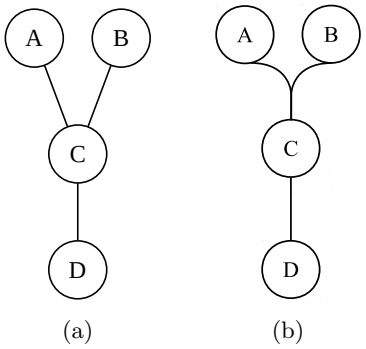


Figura 2.8: Hipergrafos para os conjuntos de reações (a) A ↔ C, B ↔ C, C ↔ D e (b) A + B ↔ C, C ↔ D. Note que para esses conjuntos de reações, o mesmo grafo de compostos é gerado, como mostra a Figura 2.4.

As modelagens vistas até agora utilizam grafos não orientados, a abordagem mais natural para uma rede onde todas reações são reversíveis. Porém, quando houver reações irreversíveis, é possível modelar a rede utilizando grafos orientados, onde é possível representar a direção de uma reação. Quando apenas algumas reações forem irreversíveis (ou reversíveis), pode-se utilizar um grafo misto, com algumas arestas orientadas e outras não. Em várias publicações [17], assume-se que todas reações são reversíveis. De fato, é possível argumentar que na presença de excesso de um produto, mesmo reações que têm uma direção favorecida podem ocorrer na direção oposta.

Podemos verificar que a modelagem de reações reversíveis com arestas não direcionadas pode ainda gerar um grafo ambíguo, mesmo usando grafos bipartidos. É o que acontece ao modelar o conjunto de reações A ↔ B + C, C ↔ D, pois ao comparar o grafo gerado por esse conjunto com o da Figura 2.7b, que foi obtido a partir do conjunto A + B ↔ C, C ↔ D, verificamos que os dois são iguais. Isso acontece porque não há indícios no grafo de quais compostos estão de que lado da reação, ou seja, quais são substratos e quais são produtos.

Considerando que em uma modelagem usando grafos bipartidos as arestas ligam uma reação a um metabólito, o problema da ambiguidade pode ser resolvido usando **rótulos** nas arestas, ou seja, indicando em que tipo de metabólito uma determinada aresta incide: um substrato ou um produto (Figura 2.9). Ainda, o uso de rótulos evita que sejam considerados caminhos sem significância biológica no grafo, como ligações entre substratos ou produtos de uma mesma reação. Desse modo, a modelagem usando grafos bipartidos será equivalente àquela usando hipergrafos em termos de informações armazenadas.

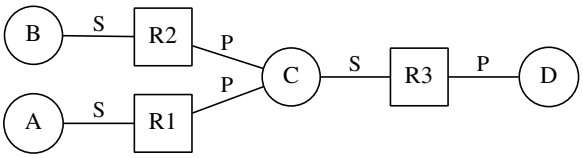


Figura 2.9: Grafo bipartido com rótulos para os conjuntos de reações R1: A ↔ C, R2: B ↔ C, R3: C ↔ D, onde o rótulo S indica que a aresta liga uma reação a um substrato e P a um produto.

Nos modelos apresentados, todos compostos e reações são igualmente importantes. Na prática, para uma reação, alguns compostos podem ser considerados primários e outros auxiliares. Por exemplo, o ATP[1] e o NADH[2] estão envolvidos em muitas reações. Se os considerarmos como compostos comuns, de mesma importância dos restantes para a modelagem, o resultado será um grafo com um grande número de reações muito próximas, mesmo que, na prática, distantes. Um exemplo pode ser visto na Figura 2.10.

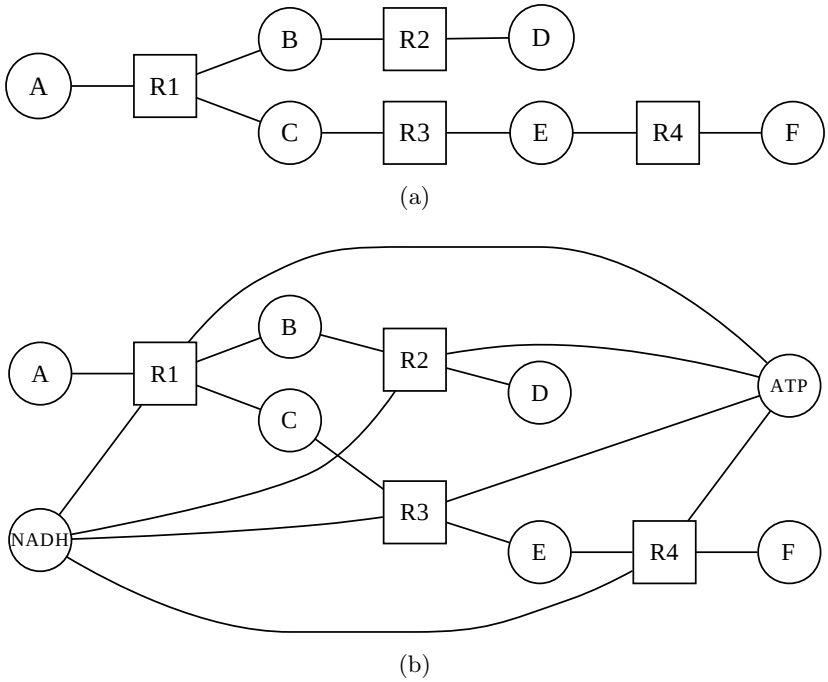


Figura 2.10: Representações gráficas usando grafos bipartidos do conjunto de reações A + NADH ↔ B + C + ATP, B + NADH ↔ D + ATP, C + NADH ↔ E + ATP, E + NADH ↔ F + ATP, (a) removendo os compostos ATP e NADH da rede; (b) não removendo os compostos ATP e NADH da rede.

Uma forma de resolver esse problema é remover da rede todos compostos que participam de uma grande quantidade de reações [23], [33], [34]. Esse método possui algumas restrições: (a) é preciso definir através de alguma heurística um limitante, para determinar quais metabólitos serão removidos; (b) alguns metabólitos altamente conectado, como piruvato ou frutose, são comumente considerados como parte principal do metabolismo; (c) mesmo compostos como o ATP não podem ser eliminados das reações envolvidas em sua própria síntese.

Outra alternativa é remover apenas a participação dos compostos em reações onde são auxiliares, ou seja, remover arestas ao invés de vértices do grafo. A ideia é manter o mais importante ou relevante da rede e eliminar atalhos. Vale notar que a distinção entre compostos primários e auxiliares para cada reação nem sempre está previamente disponível. Portanto, não aplicar nenhum tratamento ou filtragem a compostos altamente conectados pode levar a conclusões errôneas quando avaliamos relações na rede, como distância entre compostos ou reações.

[1] Trifosfato de adenosina, nucleotídeo responsável pelo armazenamento de energia em suas ligações químicas.
[2] Nicotinamida adenina dinucleotídeo, possui papel preponderante na produção de energia.

Capítulo 3

Alinhamentos

Problemas de alinhamento são muito comuns em bioinformática. Nesse contexto, alinhamento refere-se a comparar e identificar semelhanças entre padrões apresentados, tornando clara a correspondência entre eles. Tais semelhanças podem representar relações funcionais, estruturais ou evolucionárias. O conceito de alinhamento difere-se de similaridade que, de modo geral, é uma medida que representa o quão semelhantes dois padrões são. Neste capítulo, teremos inicialmente seções dedicadas ao alinhamento de sequências, exposição necessária para alguns dos assuntos posteriores, seguida de uma visão geral sobre o alinhamento de redes metabólicas. Outras informações e variações não cobertas neste trabalho sobre o alinhamento de sequências podem ser encontradas em [43, 22].

3.1 Alinhamento de duas sequências

Uma **sequência de caracteres** é uma sequência finita de caracteres ou símbolos, escolhidos a partir de um conjunto finito pré-determinado, chamado **alfabeto**. Por exemplo, considerando um alfabeto Σ tal que $\Sigma = \{\texttt{A}, \texttt{B}, \texttt{C}\}$, podemos formar uma sequência de caracteres `ABCB`. Nesse caso, dizemos que `A` é o primeiro caractere da sequência, `B` o segundo e assim sucessivamente. Uma **subsequência de caracteres** de uma sequência s é uma sequência formada por um ou mais símbolos consecutivos de s, por exemplo, `CB` é uma subsequência de caracteres de `ABCB` mas `AC` não é. Nesse caso, denotamos por $\boldsymbol{s[i..j]}$ uma subsequência de caracteres que contém do i-ésimo ao j-ésimo caractere de s. O **tamanho** ou **comprimento** de uma sequência de caracteres s, denotado por $|s|$, representa a quantidade de caracteres dela. Uma **sequência de caracteres vazia** não possui nenhum caractere e é denotada por ε. Usaremos os termos **sequência** e **subsequência** para nos referir a sequência de caracteres e subsequência de caracteres, respectivamente.

O **alinhamento de duas sequências** é uma operação primitiva na Biologia Computacional, servindo como base para outras mais complexas. Resumidamente, consiste em comparar duas sequências, encontrando partes em que se assemelham e partes em que se diferem. Tais sequências podem ser, por exemplo, de DNA, RNA ou proteínas.

Na prática, existem diversas variações do problema de alinhamento de sequências, como por exemplo: (a) o objetivo é alinhar as sequências inteiras, efetuando-se comparações globais, como mostra a Figura 3.1a; (b) o objetivo é alinhar apenas partes delas ou uma sequência a uma parte de outra, onde são feitas comparações locais, como mostra a Figura 3.1b. Para obter um melhor

alinhamento, podemos adicionar espaços, representados por `-`, a qualquer uma das sequências alinhadas, de acordo com critérios que serão apresentados mais à frente.

Podemos ver na Figura 3.1 a diferença entre os alinhamentos global e o local. Enquanto o alinhamento global tenta alinhar as duas sequências como um todo, o alinhamento local tenta encontrar regiões de similaridade, ou seja, subsequências similares entre as duas sequências, resultando em um alinhamento com uma maior quantidade de caracteres seguidos sem espaços.

```
ACTGCATCTTTG
A---CA-C---G
```

(a) Alinhamento global.

```
CATC
CA-C
```

(b) Alinhamento local.

Figura 3.1: Exemplos de alinhamento das sequências `ACTGCATCTTTG` e `ACACG`.

Iremos focar as atenções nos problemas de comparações globais e comparações locais, usados como base para algumas operações no alinhamento de redes metabólicas.

3.1.1 Alinhamento global de sequências

Tomemos como exemplo duas sequências de DNA, `GTACAC` e `GTTACAA`. Percebemos que as duas sequências são muito semelhantes, de modo que podemos alinhá-las como mostra a Figura 3.2. As diferenças entre elas são o último caractere `C` e `A` e um caractere adicional `T` na segunda. Foi adicionado um espaço após a segunda posição ou coluna da primeira sequência, de modo que as duas possam ter o mesmo tamanho e se alinharem corretamente. É comum duas sequências terem tamanhos diferentes, como no exemplo.

```
GT-ACAC
GTTACAA
```

Figura 3.2: Exemplo de alinhamento global das sequências `GTACAC` e `GTTACAA`.

O **alinhamento entre duas sequências de caracteres** é a inserção de espaços em locais arbitrários das sequências, incluindo o início e o final delas, de modo que possam ser posicionadas uma acima da outra, criando um **pareamento** entre caracteres/espaços da primeira e caracteres/espaços da segunda. Cada par de caracteres alinhados dessas sequências é chamado de **coluna**. Além disso, não pode haver dois espaços em uma mesma coluna.

Dado um alinhamento entre duas sequências, é possível atribuir a ele uma **pontuação**, determinada da seguinte forma: cada coluna do alinhamento recebe um valor dependendo dos caracteres das duas sequências naquela posição. A pontuação do alinhamento é a soma dos valores das suas colunas. O **melhor alinhamento** é aquele, dentre todos os possíveis, com a maior pontuação. Convém notar que, entre duas sequências, podem existir vários alinhamentos com melhor pontuação. Dado um esquema de pontuação que penalize a inserção de espaços e atribua valores maiores para pareamentos de maior semelhança entre símbolos, a melhor pontuação é chamada de **similaridade** entre as duas sequências. Para duas sequências s e t, a similaridade entre as duas é denotada por $\text{sim}(s, t)$.

Ainda é preciso definir quais são os valores atribuídos às colunas do alinhamento. Se uma coluna possui dois caracteres iguais, ocorre uma **correspondência** e seu valor é $+1$. Se os caracteres forem diferentes, ocorre uma **diferença** e seu valor é -1. Caso seja inserido um espaço na coluna seu valor é -2. Tais valores são usados frequentemente na prática tendo como objetivo recompensar correspondências e penalizar diferenças e espaços. Utilizando as definições anteriores, é possível contabilizar a pontuação do alinhamento apresentado na Figura 3.2, que apresenta 5 correspondências, 1 diferença e 1 inserção de espaço, resultando em um valor total igual a 2 como podemos ver a seguir:

$$5 \times 1 + 1 \times (-1) + 1 \times (-2) = 2.$$

Uma possível solução para encontrar a similaridade entre duas sequências é gerar e computar a pontuação para todos alinhamentos possíveis, de forma a encontrar a melhor pontuação. Considere duas sequências s e t de tamanhos m e n respectivamente, onde a segunda é a menor das duas, ou seja, $n \leq m$. Considere ainda que dois possíveis alinhamentos são equivalentes se eles alinham os mesmos índices de s e t. Por exemplo, os dois alinhamentos a seguir são equivalentes:

```
A-CGTA-G        -ACGT-AG
-AC-T-CG   e    A-C-TC-G   ,
```

pois ambos alinham $s[2]$ com $t[2]$, $s[4]$ com $t[3]$ e $s[6]$ com $t[5]$. Dois alinhamentos são distintos se não são equivalentes.

Seja C_k a quantidade de alinhamentos distintos onde há a correspondência de k símbolos de s com k símbolos de t. Então, a quantidade total de alinhamentos distintos é $\sum_{k=1}^{n} C_k$. O valor de C_k pode ser computado como a quantidade de maneiras possíveis de escolher k símbolos de s e k símbolos de t. Logo,

$$C_k = \binom{n}{k}\binom{m}{k} .$$

Então, a quantidade total de alinhamentos distintos é:

$$\begin{aligned}\sum_{k=1}^{n} \binom{n}{k}\binom{m}{k} &\geq \sum_{k=1}^{n} \binom{n}{k}\binom{n}{k}, \text{ pois } n \leq m \\ &\geq \sum_{k=1}^{n} \binom{n}{k} \\ &= 2^n .\end{aligned}$$

Portanto o tempo para gerar todos os alinhamentos possíveis é $\Omega(2^n)$. Tal abordagem revela-se ineficiente do ponto de vista prático.

É possível utilizar o algoritmo de Needleman-Wunsch [35], muito mais eficiente por utilizar a técnica de programação dinâmica [11]. O algoritmo consiste em, como todos aqueles baseados em programação dinâmica, computar a solução ótima de uma instância utilizando resultados já obtidos para instâncias menores do mesmo problema, onde tais instâncias sobrepostas compõem o problema original. Considerando duas sequências s e t, em vez de determinar a similaridade

entre elas levando em conta toda a extensão das duas sequências, o algoritmo obtém soluções determinando as similaridades entre prefixos arbitrários das mesmas. Ele começa trabalhando com pequenos prefixos e usa os resultados já obtidos para alcançar resultados para prefixos maiores.

Sejam m e n os tamanhos de s e t, respectivamente. Existem $m+1$ prefixos possíveis para s, sendo eles: $\varepsilon, s[1], s[1..2], \ldots, s[1..m]$. Da mesma forma, existem $n+1$ prefixos possíveis para t. Assim, é possível organizar os cálculos de similaridades em uma matriz de dimensões $(m+1) \times (n+1)$, onde a posição (i, j) contém a similaridade entre $s[1..i]$ e $t[1..j]$. No caso de i ou j serem iguais à 0, considere s ou t, respectivamente, como uma sequência vazia ε.

O ponto chave do algoritmo é o fato de haver apenas três maneiras de se obter uma alinhamento entre $s[1..i]$ e $t[1..j]$ utilizando-se dos resultados já obtidos para instâncias menores:

- Alinhar $s[1..i]$ com $t[1..j-1]$ e parear $t[j]$ com um espaço;
- Alinhar $s[1..i-1]$ com $t[1..j-1]$ e parear $s[i]$ com $t[j]$;
- Alinhar $s[1..i-1]$ e $t[1..j]$ e parear $s[i]$ com um espaço.

Desse modo, para computar o valor da posição (i, j) da matriz, basta analisar as posições $(i-1, j)$, $(i-1, j-1)$ e $(i, j-1)$. Portanto, a fórmula de recorrência que define a similaridade para $s[1..i]$ e $t[1..j]$ pode ser escrita da seguinte forma:

$$\text{sim}(s[1..i], t[1..j]) = \max \begin{cases} \text{sim}(s[1..i], t[1..j-1]) - 2 \\ \text{sim}(s[1..i-1], t[1..j-1]) + p(i,j) \\ \text{sim}(s[1..i-1], t[1..j]) - 2 \end{cases} \tag{3.1}$$

onde $p(i, j)$ é a pontuação dada ao pareamento de $s[i]$ e $t[j]$:

$$p(i,j) = \begin{cases} +1 & \text{, se } s[i] = t[j] \\ -1 & \text{, se } s[i] \neq t[j] \ . \end{cases} \tag{3.2}$$

Se definirmos a como a matriz que armazena os valores computados das similaridades entre prefixos de s e t e g como a penalidade de inserção de espaço, a Equação 3.1 pode ser reescrita de maneira simplificada, de modo a refletir diretamente tais valores:

$$a[i,j] = \max \begin{cases} a[i, j-1] + g \\ a[i-1, j-1] + p(i,j) \\ a[i-1, j] + g \ . \end{cases} \tag{3.3}$$

As posições de a devem ser computadas de forma conveniente para que, no momento da computação da similaridade para algum (i, j), os valores das posições que representam prefixos menores já tenham sido obtidos. Isso pode ser feito, por exemplo, calculando as posições de a linha por linha (i de 1 até m), da esquerda para a direita em cada linha (j de 1 até n).

Convém notar que a primeira linha e a primeira coluna ($a[0][0..n]$ e $a[0..m][0]$) são inicializados com múltiplos do valor de penalidade de inserção de espaço, que definimos como $g < 0$. Isso

acontece porque há apenas um alinhamento possível se uma das sequências é vazia: a inserção de um espaço para parear cada caractere da outra sequência. Portanto:

$$a[0][j] = g \times j, \text{ para } j = 0, \ldots, n$$

e

$$a[i][0] = g \times i, \text{ para } i = 0, \ldots, m\ .$$

Logo, a pontuação de tal alinhamento é $g \times k$, onde k é o tamanho da sequência não vazia. Podemos ver o Algoritmo 3.1, chamado de SimilaridadeGlobal, que calcula a similaridade entre duas sequências de caracteres de acordo com o apresentado anteriormente, preenchendo a matriz a. Ele depende do parâmetro g que especifica a penalidade de inserção de espaço, geralmente menor que zero, e uma função de pontuação p.

Algoritmo 3.1 SimilaridadeGlobal(s, t, g): Alinha duas sequências de caracteres

Entrada: sequências s e t e penalidade de inserção de espaço g
Saída: similaridade entre s e t e matriz a

```
1:  m ← |s|
2:  n ← |t|
3:  para i ← 0 até m faça
4:      a[i, 0] ← i × g
5:  para j ← 0 até n faça
6:      a[0, j] ← j × g
7:  para i ← 1 até m faça
8:      para j ← 1 até n faça
9:          a[i, j] ← max(a[i − 1, j] + g,
                          a[i − 1, j − 1] + p(i, j),
                          a[i, j − 1] + g)
10: devolva a[m, n], a
```

Podemos ver na Figura 3.3 a matriz a preenchida pelo Algoritmo 3.1 para as sequências `AAAC` e `AGC`, com $g = -2$ e p conforme apresentado na Equação 3.2. As setas mostram de que posição da matriz cada valor máximo foi obtido, de acordo com a Equação 3.3. Note que pode haver mais de uma seta por posição se houver dois ou mais valores máximos entre as opções a serem escolhidas na equação.

Construção do alinhamento global ótimo

Vimos até agora como computar a similaridade entre duas sequências. Agora veremos como, a partir da matriz a, obter o alinhamento global ótimo entre elas, usando o exemplo da Figura 3.3.

A sequência s é posicionada em a verticalmente e a sequência t horizontalmente. Iniciamos na posição (m, n) de a e seguimos as setas até alcançarmos a posição $(0, 0)$, obtendo a cada passo uma coluna do alinhamento da seguinte forma, de acordo com a seta que seguimos:

- uma seta horizontal corresponde a uma coluna com espaço em s pareado com $t[j]$;
- uma seta vertical corresponde a uma coluna com $s[i]$ pareado com um espaço em t;
- uma seta diagonal corresponde a uma coluna com $s[i]$ pareado com $t[j]$.

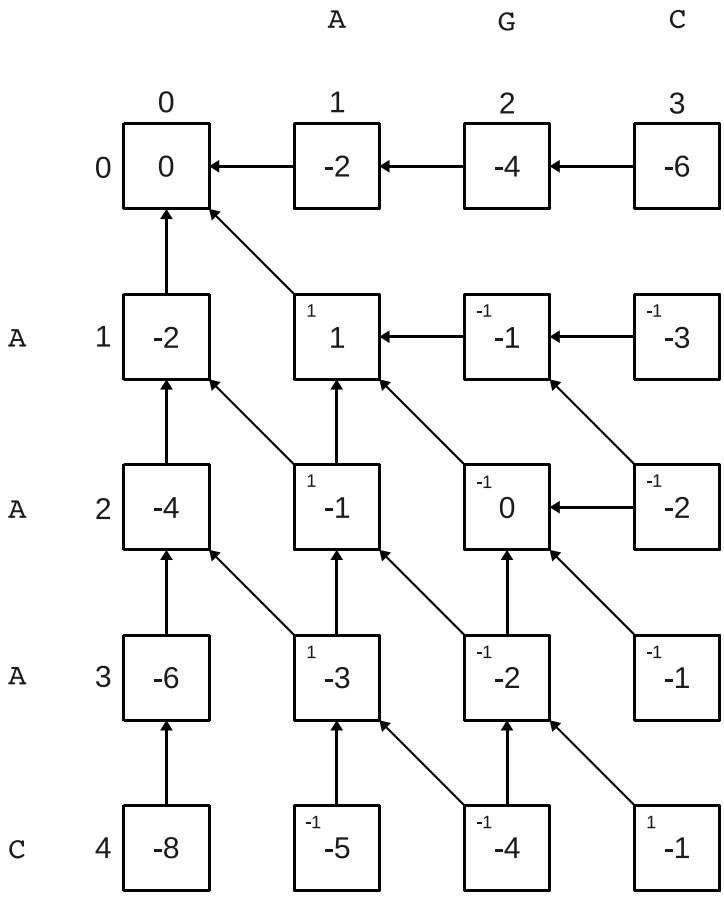


Figura 3.3: Matriz a obtida no alinhamento global das sequências `AAAC` e `AGC`. Representamos no canto superior esquerdo de cada posição (i, j) o valor de $p(i, j)$.

O caminho tomado na matriz, representado pelas setas, não precisa estar armazenado, sendo possível descobri-lo realizando o seguinte teste:

- se $a[i, j] = a[i - 1, j] + g$, então existe seta vertical;
- se $a[i, j] = a[i, j - 1] + g$, então existe seta horizontal;
- se $a[i, j] = a[i - 1, j - 1] + p(i, j)$, então existe seta diagonal.

O Algoritmo 3.2, mostra uma forma recursiva de determinar um alinhamento global ótimo. Ele recebe inicialmente a matriz a e os valores $|s|$ e $|t|$ como índices i e j, armazenando nas posições $1, \ldots, c$ dos vetores *alin-s* e *alin-t* os símbolos das colunas $1, \ldots, c$ do alinhamento, sendo c o comprimento deste. Tais símbolos podem ser espaços ou ainda símbolos pertencentes ao alfabeto das sequências. Além disso, é fácil verificar que $\max(|s|, |t|) \leq c \leq m + n$.

Note que é possível obter vários caminhos, de acordo com a seta escolhida e isso ocorre porque podem existir vários alinhamentos ótimos. O Algoritmo 3.2 devolve um entre os alinhamentos

Algoritmo 3.2 AlinhamentoGlobal(a, i, j): Constrói o alinhamento global ótimo

Entrada: matriz a dada pelo algoritmo SimilaridadeGlobal, índices i e j e sequências s e t
Saída: comprimento c do alinhamento e vetores *alin-s* e *alin-t*

```
1: se i = 0 e j = 0 então
2:     c ← 0
3: senão se i > 0 e a[i, j] = a[i − 1, j] + g então
4:     c ← AlinhamentoGlobal(a, i − 1, j) + 1
5:     alin-s[c] ← s[i]
6:     alin-t[c] ← -
7: senão se i > 0 e j > 0 e a[i, j] = a[i − 1, j − 1] + p(i, j) então
8:     c ← AlinhamentoGlobal(a, i − 1, j − 1) + 1
9:     alin-s[c] ← s[i]
10:    alin-t[c] ← t[j]
11: senão          ▷ este é o último caso, onde j > 0 e a[i, j] = a[i, j − 1] + g
12:    c ← AlinhamentoGlobal(a, i, j − 1) + 1
13:    alin-s[c] ← -
14:    alin-t[c] ← t[j]
15: devolva c, alin-s, alin-t
```

possíveis, dando preferência às setas verticais, então às diagonais e finalmente às horizontais. Essa escolha pode ser traduzida como a preferência do algoritmo por colunas com:

1. símbolo em s e espaço em t;
2. dois símbolos;
3. espaço em s e símbolo em t.

Por exemplo, ao alinhar $s =$ `ATAT` com $t =$ `TATA` obtemos:

```
-ATAT
TATA-
```

ao invés de:

```
ATAT-
-TATA
```

que é outro alinhamento ótimo para as duas sequências. Da mesma forma, ao alinhar $s =$ `AA` com $t =$ `AAAA` obtemos:

```
--AA
AAAA
```

apesar de existirem cinco outros alinhamentos ótimos. Esse alinhamento é chamado de **mais alto** por usar mais as setas para cima na matriz. Para inverter a ordem de preferência, basta inverter a ordem das expressões condicionais **se** no algoritmo. Nesse caso, o alinhamento obtido

é chamado de **mais baixo**. Uma coluna que apareça nos dois tipos de alinhamento estará presente em todos os alinhamentos ótimos possíveis entre as duas sequências consideradas.

É possível modificar o algoritmo para produzir todos os alinhamentos ótimos, memorizando os locais onde há mais de uma opção de caminho para percorrer e explorando todas as possibilidades de se alcançar a posição $(0, 0)$. Convém notar, porém, que pode haver uma quantidade muito grande de alinhamentos ótimos, dependendo dos valores dos parâmetros do problema, como os valores das pontuações de correspondência, diferença e de g.

3.1.2 Alinhamento local de sequências

Um alinhamento local entre s e t é um alinhamento entre uma subsequência de s e outra de t. Desse modo, queremos encontrar os alinhamentos locais com maior pontuação entre duas sequências. Isso pode ser feito com uma modificação do Algoritmo 3.1, que é conhecido como algoritmo de Smith-Waterman [44].

Usamos, da mesma forma que no alinhamento global, uma matriz a de dimensões $(m+1) \times (n+1)$. Contudo, a interpretação dos valores em a é diferente: cada posição (i, j) armazena a maior pontuação de um alinhamento entre um sufixo de $s[1..i]$ e um sufixo de $t[1..j]$. As posições da primeira linha e a primeira coluna de a ($a[0][0..n]$ e $a[0..m][0]$) são inicializadas com 0, pois essa é a melhor pontuação entre um alinhamento de um sufixo de ε e:

- ε;
- um sufixo de $s[1..m]$;
- um sufixo de $t[1..n]$.

De modo geral, para uma posição (i, j), sempre haverá o alinhamento entre sufixos vazios de $s[1..i]$ e $t[1..j]$, que possui pontuação 0. Logo, a matriz a terá sempre valores iguais ou maiores que 0 em todas as posições.

A matriz a é então preenchida como visto anteriormente, com a possibilidade adicional de se atribuir 0 a cada posição, resultando na seguinte equação:

$$a[i,j] = \max \begin{cases} a[i, j-1] + g \\ a[i-1, j-1] + p(i,j) \\ a[i-1, j] + g \\ 0\ . \end{cases} \tag{3.4}$$

Ao final, basta encontrar o maior valor em a e esse será a pontuação do alinhamento ótimo local. O Algoritmo 3.3 mostra o alinhamento local de sequências.

Construção do alinhamento local ótimo

Para iniciar a construção do alinhamento local ótimo, usamos a mesma ideia do Algoritmo 3.2. A posição inicial da construção pode ser qualquer posição de a que possua o valor da pontuação do alinhamento ótimo, ou seja, qualquer posição em a que possua o maior valor entre

as posições da matriz. Então o alinhamento é obtido de forma similar à vista no Algoritmo 3.2, mas parando ao encontrar uma posição sem nenhuma seta partindo dela, ou seja, quando encontramos uma posição com valor 0. Podemos ver mais à frente o Algoritmo 3.4 conforme descrito acima, chamado de ALINHAMENTOLOCAL.

Algoritmo 3.3 SIMILARIDADELOCAL(s, t, g): Alinha duas sequências de caracteres

Entrada: sequências s e t e penalidade de inserção de espaço g
Saída: similaridade entre s e t e matriz a

$m \leftarrow |s|$
$n \leftarrow |t|$
para $i \leftarrow 0$ **até** m **faça**
    $a[i, 0] \leftarrow 0$
para $j \leftarrow 0$ **até** n **faça**
    $a[0, j] \leftarrow 0$
para $i \leftarrow 1$ **até** m **faça**
    para $j \leftarrow 1$ **até** n **faça**
        $a[i, j] \leftarrow \max(a[i-1, j] + g,$
            $a[i-1, j-1] + p(i, j),$
            $a[i, j-1] + g,$
            $0)$
devolva $\max(a[1..m, 1..n]), a$

Algoritmo 3.4 ALINHAMENTOLOCAL(a, i, j): Constrói o alinhamento local ótimo

Entrada: matriz a dada pelo algoritmo SIMILARIDADELOCAL, índices i e j e sequências s e t
Saída: comprimento c do alinhamento e vetores *alin-s* e *alin-t*

se $a[i, j] = 0$ **então**
    $c \leftarrow 0$
senão se $a[i, j] = a[i-1, j] + g$ **então**
    $c \leftarrow$ ALINHAMENTOLOCAL$(a, i-1, j) + 1$
    alin-s$[c] \leftarrow s[i]$
    alin-t$[c] \leftarrow$ -
senão se $a[i, j] = a[i-1, j-1] + p(i, j)$ **então**
    $c \leftarrow$ ALINHAMENTOLOCAL$(a, i-1, j-1) + 1$
    alin-s$[c] \leftarrow s[i]$
    alin-t$[c] \leftarrow t[j]$
senão ▷ este é o último caso, onde $a[i, j] = a[i, j-1] + g$
    $c \leftarrow$ ALINHAMENTOLOCAL$(a, i, j-1) + 1$
    alin-s$[c] \leftarrow$ -
    alin-t$[c] \leftarrow t[j]$
devolva c, *alin-s*, *alin-t*

É possível que, como no alinhamento global, o alinhamento local ótimo tenha o tamanho da maior das sequências, contudo nem sempre é o que ocorre. Isso acontece porque o algoritmo tenta encontrar um alinhamento de maior pontuação possível entre alguma subsequência de s e outra t, sem necessariamente alinhar as sequências inteiras. Por exemplo, considere $s =$ `ABCDXXXXXEF` e $t =$`AGXXYXXHIJK`. Podemos ver que as duas sequências possuem subsequências quase idênticas. Além disso, o restante possui poucos símbolos em comum. É fácil verificar que

a pontuação de um alinhamento local contendo as subsequências quase idênticas certamente será maior do que a pontuação obtida ao alinharmos as sequências inteiras.

3.1.3 Complexidade dos algoritmos

O Algoritmo 3.1 possui quatro laços. Os dois primeiros são responsáveis pela inicialização da matriz a e consomem tempo $O(m)$ e $O(n)$, respectivamente. Os outros dois laços estão aninhados e preenchem o restante da matriz, consumindo portanto tempo $O(mn)$. O espaço consumido também é proporcional ao tamanho da matriz. Então, se as duas sequências têm aproximadamente o mesmo tamanho, ou seja, $m = O(n)$, temos complexidade quadrática $O(n^2)$ tanto no tempo quanto no espaço usados. O mesmo ocorre para o Algoritmo 3.3, que computa a similaridade para o alinhamento local de sequências.

O Algoritmo 3.2 por sua vez constrói o alinhamento global ótimo, percorrendo um caminho entre as posições de a através de setas. Visto que as setas sempre se aproximam da posição $(0, 0)$, a maior sucessão de setas possível tem $m + n$ setas, portanto o algoritmo consome tempo $O(m+n)$. Da mesma forma, os vetores que armazenam o alinhamento no Algoritmo 3.2 possuem tamanho máximo $m + n$, logo o espaço consumido é proporcional a $2 \times (m + n) = O(m + n)$. O mesmo vale para o Algoritmo 3.4, que constrói o alinhamento local ótimo.

O alinhamento de duas sequências é, sem dúvida, um tópico de grande relevância na biologia computacional. Mas existem diversas questões que não podem ser modeladas dessa forma, onde há a necessidade do alinhamento de várias sequências. Esse problema, muito mais complexo que o anterior, pertence à classe de problemas chamada NP-difícil [11], não sendo conhecido até o presente um método exato de encontrar a solução ótima em tempo polinomial. Podemos citar os algoritmos em [21] e [1] que não encontram uma solução ótima mas oferecem a garantia de uma solução aproximada computável eficientemente.

3.2 Alinhamento múltiplo de sequências

O **alinhamento múltiplo de sequências**, ou **AMS**, está entre os problemas mais relevantes da bioinformática por se relacionar com diversos outros problemas da biologia neste contexto. É amplamente empregado em algumas áreas, onde podemos citar a análise de sequências de DNA e proteínas. Generalização do problema de alinhamento de duas sequências, consiste em alinhar três ou mais sequências, resultando em dados que possibilitam mensurar variados tipos de relações de acordo com a técnica utilizada. Um exemplo é o estudo da estrutura, função e evolução dos genes. Durante bilhões de anos o código genético de organismos têm sofrido mutações constantes, como inserções, substituições, remoções, recombinações, entre outras. Estudar essas mutações através da análise de sequências pode ser útil não apenas para obter relações evolucionárias entre sequências, mas também a fim de identificar limitações estruturais ou funcionais nestas, sejam de DNA, RNA ou proteínas. No contexto de relações evolucionárias, um alinhamento é um modelo hipotético das mutações que ocorreram durante a evolução das sequências. O melhor alinhamento será aquele que representa o cenário mais provável e geralmente não é estabelecido sem ambiguidades. Isso se deve (i) pela complexidade computacional do problema, já que algoritmos que podem ser utilizados na prática não garantem encontrar a melhor solução, e (ii) mesmo que se disponha de um algoritmo ideal, não se pode garantir que o alinhamento encontrado seja aquele que melhor representa a solução do problema estudado,

pois o conhecimento atual sobre a probabilidade de ocorrência de diferentes tipos de mutações, como substituições, inserções e remoções, é ainda limitado [15].

Dessa forma, o AMS permite elucidar uma grande variedade de questões da biologia, como por exemplo [36, 15]:

- Análise filogenética, aferindo relações evolucionárias entre organismos variados;
- Identificação de motifs e domínios conservados pela evolução, permitindo identificar quais têm um papel importante na estrutura ou função de um grupo de proteínas relacionadas;
- Predição de estruturas, ajudando descobrir a função de um resíduo na estrutura de uma proteína ou mutações correlacionadas;
- Demonstração de homologia entre sequências, encontrando sequências que compartilham ancestrais anteriores a eventos de especialização ou duplicação;
- Localização de similaridades fracas porém significantes em bases de dados de sequências, apontando membros distantes de uma mesma família;
- Identificação de genes relacionados.

A pontuação de um alinhamento múltiplo deve refletir a semelhança entre as sequências alinhadas. Contudo, há diferentes formas de medir a pontuação de um alinhamento. Vejamos uma definição formal para o problema de AMS baseada em [15]. Uma sequência é um conjunto ordenado de símbolos de um alfabeto Σ. Um alinhamento de n sequências $S_1, \ldots, S_n$ é denotado por $a(S_1, \ldots, S_n)$ e resulta em uma matriz $\mathcal{A}$, onde cada posição $\mathcal{A}[i][j]$ é um símbolo de Σ ou o símbolo de espaço, denotado por `-`. A linha i de $\mathcal{A}$ é a sequência S_i, possivelmente com espaços inseridos em algumas posições. No modelo mais simples, a pontuação de um alinhamento de n sequências é definida como a soma das pontuações de todas colunas, portanto a contribuição de cada coluna para a pontuação do alinhamento é independente da pontuação das restantes. Em um modelo mais realístico, uma inserção de diversos espaços é interpretada como um único evento de mutação e associada a um custo que depende do seu tamanho. Em tais modelos, o alinhamento de um par de sequências é definido como a soma das penalidades de diferenças e inserções de espaços. Então, uma forma de definir a pontuação de um alinhamento múltiplo, conhecida como soma dos pares ou SP, consiste em calcular a pontuação a partir da pontuação dos pares. A SP é definida como a soma das pontuações das $n(n-1)/2$ combinações de alinhamentos de pares, descartando-se colunas que tenham `-` em ambas sequências, como mostra o exemplo da Figura 3.4.

Os principais métodos utilizados para a resolução do AMS são baseados na técnica chamada "alinhamento progressivo" e datam do final da década de 80. Não obstante, muitas melhorias foram obtidas em respeito ao tempo de resolução, precisão e capacidade de lidar com grandes números de sequências. De acordo com [47], até a metade da década de 80 os alinhamentos múltiplos eram realizados manualmente, devido ao tempo excessivamente lento para realizá-los com generalizações do algoritmo de programação dinâmica para duas sequências. Diversas técnicas então surgiram. No entanto eram lentas ou requeriam que as sequências comparadas fossem muito similares. Recentemente, métodos iterativos têm se mostrado bem sucedidos, sozinhos ou em combinação com outros métodos.

Como visto acima, existiam algoritmos que resolviam o problema utilizando programação dinâmica, mas que não apresentavam um tempo de execução razoável ou bons resultados. Esse

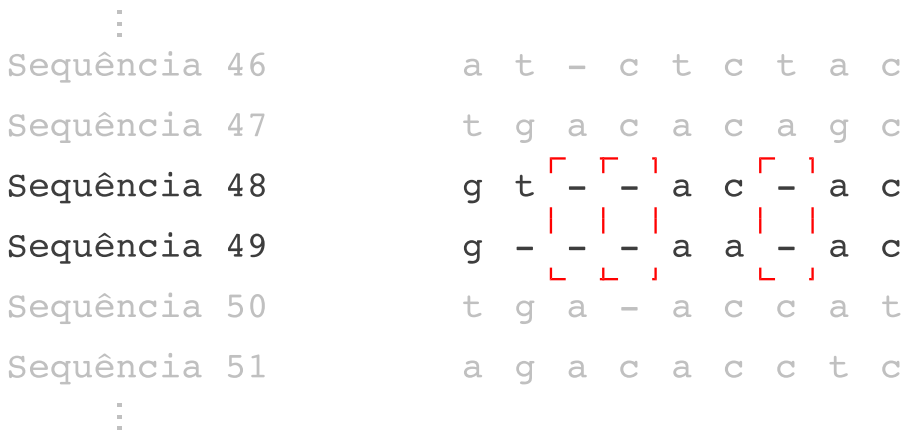


Figura 3.4: Exemplo de alinhamento múltiplo de sequências utilizando para o cálculo da pontuação o método de soma dos pares (SP). As colunas marcadas serão ignoradas no cálculo da pontuação dos pares em questão, as sequências 48 e 49. Note que quando há espaço em apenas uma sequência, a coluna permanece inalterada.

fato encaminhou naturalmente os pesquisadores ao desenvolvimento de algoritmos heurísticos, que possuíam tempos de execução aceitáveis e encontravam boas soluções na prática.

Apesar de todos os métodos existentes para resolver o problema do AMS, é comum a dúvida se realmente existiria algum algoritmo que o resolva de forma rápida, ou seja, polinomial, ou ainda qual seria o tempo mínimo necessário. Contudo, ainda não se conhece um algoritmo exato que resolva o problema AMS encontrando a solução ótima de forma rápida. De fato, é provado que o problema é NP-difícil [48, 24]. Isso significa que o problema está na classe de problemas mais difíceis de serem resolvidos, onde ainda não se sabe se é possível resolvê-los em tempo polinomial. Outras elucidações acerca do tema podem ser encontradas em [5].

Neste ponto, pode ser natural deduzir que, se existem técnicas para resolver o AMS, podemos então empregá-las para o alinhamento múltiplo de vias ou redes metabólicas. Embora esta pudesse, em uma primeira impressão, ser uma aplicação direta do que já existe, nenhum dos algoritmos de alinhamento de vias ou redes metabólicas estudados no curso de produção deste trabalho utilizam o AMS para a resolução do problema. Apesar disso, um dos algoritmos que será visto à frente, mais especificamente na seção 4.1, utiliza a ideia do alinhamento de duas sequências para realizar o alinhamento múltiplo de vias metabólicas não ramificadas, fato pelo qual o alinhamento de pares de sequências foi estudado detalhadamente em seções anteriores.

3.3 Visão geral sobre o alinhamento de redes metabólicas

De modo geral, ao se falar de alinhamento no contexto de redes metabólicas, podemos realizar o alinhamento (i) entre vias, (ii) entre uma via e alguma via similar encontrada em uma rede e (iii) entre redes. Diversos trabalhos têm sido desenvolvidos no intuito de realizar alinhamentos de redes metabólicas como um todo, a fim de, por exemplo, aferir semelhanças, evoluções ou parentescos entre espécies de seres vivos. Contudo, para o estudo desses alinhamentos é preciso abordar, como pré-requisito, o alinhamento de vias metabólicas, as quais compõem uma rede metabólica.

O alinhamento de vias metabólicas possui muitas semelhanças com o alinhamento de sequências, mas difere principalmente em dois aspectos: (a) não alinhamos mais símbolos, mas reações, e (b) as estruturas não são lineares, mas grafos ou hipergrafos.

Para alinhar reações, é necessário definir uma forma de medir a distância entre elas. Geralmente a distância entre reações é funcional e pode não refletir uma relação de evolução. Isso não é problema se o objetivo é realizar uma comparação estrutural entre redes metabólicas, mas é um ponto chave ao se procurar uma relação de evolução.

Podemos utilizar um método simples para aferir a distância entre duas reações, baseado nos números na classificação enzimática [50] das enzimas que as catalisam [45]. O **número na classificação enzimática** ou **número EC** é um sistema de nomenclatura de enzimas, onde cada enzima possui um número EC associado. Tais números codificam reações catalisadas por enzimas, logo, enzimas diferentes que catalisam a mesma reação recebem o mesmo número. Por formar uma estrutura hierárquica, os números EC podem ser representados através de uma árvore representando a **hierarquia de enzimas**, onde reações similares se encontram próximas na árvore. Desse modo, a similaridade entre eles pode ser definida em função da distância dos mesmos na árvore. Os números EC são expressados como uma sequência de quatro números, separados por pontos e entre colchetes. Por exemplo, considere o número EC `[1.2.3.4]`. Ele descreve uma reação que pertence à classe `[1]`, à subclasse `[1.2]`, à sub-subclasse `[1.2.3]` e representa a enzima `[1.2.3.4]` em particular. Podemos ver na Figura 3.5 um exemplo de uma árvore representando a hierarquia de enzimas, com uma classe adicional `[*]`, que representa a raiz da árvore.

Cada elemento da hierarquia de enzimas é genericamente chamado de **classe de enzima**. Dadas duas enzimas e_i e e_j, o primeiro ascendente em comum entre as duas, ou seja, o de nível mais baixo na árvore, é chamado de **classe superior em comum** e é representado por h_{ij}. Por exemplo, `[*]` é a classe superior em comum entre `[2.3.4]` e `[3.1.2]`, e `[5.1]` é a classe superior em comum entre `[5.1.2]` e `[5.1.3.4]`. Referências a um número EC com menos de quatro números indicam vértices internos da árvore e reações específicas se encontram nas folhas. O número EC pode ser utilizado, portanto, para definir uma pontuação para o alinhamento de duas reações. Desse modo, é possível alinhar duas vias metabólicas sem ramificações, utilizando a mesma ideia do alinhamento de sequências, conforme proposto em [45].

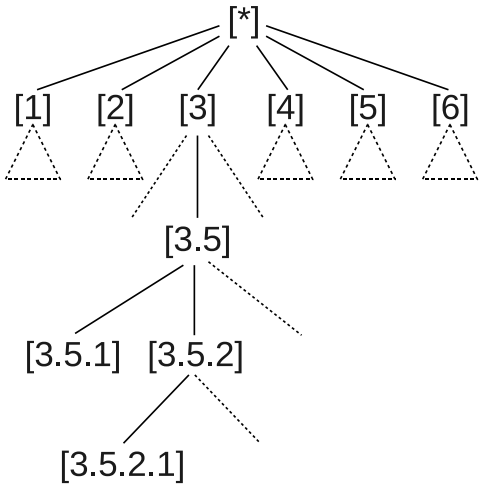


Figura 3.5: Árvore representando a hierarquia de enzimas pelo número EC.

Uma generalização desse método foi proposta por Pinter *et al.* [39], para o alinhamento de vias metabólicas com ramificações. O algoritmo, contudo, por questões de eficiência, não processa quaisquer grafos, sendo necessário quebrar heuristicamente ciclos que porventura façam parte dos mesmos, transformando-os assim em árvores, que podem ser vistas como vias metabólicas ramificadas.

Para o contexto específico de redes de iterações proteicas, Kelley *et al.* [27] propõem o alinhamento dos grafos decompondo-os em caminhos, técnica que pode ser usada para redes metabólicas. Contudo, os resultados dos alinhamentos dos caminhos devem ser agrupados posteriormente, o que demanda muito tempo na execução do algoritmo.

Outras medidas de distância mais simples podem ser utilizadas, ao exemplo de [46], onde a distância entre duas redes é definida como a distância de Hamming entre as matrizes de adjacências dos grafos. Uma outra alternativa é ignorar completamente a topologia, considerando as redes a serem comparadas apenas como conjuntos de enzimas e metabólitos [10], ou como vários índices estruturais como coeficiente de agrupamento, intermediação de centralidade, comprimento médio de caminhos, diâmetro e concentração de subgrafos [53, 49, 37]. Todavia, tais simplificações podem ser simplistas em muitos casos.

Fica claro que, nos trabalhos estudados, do alinhamento de vias ao alinhamento de redes metabólicas há um aumento grande na complexidade. Apesar disso, algoritmos eficientes podem ser desenvolvidos ao levarmos em conta as particularidades do contexto explorado. Outras medidas de distância mais complexas podem ser consideradas, levando em conta informações que tenham relevância no contexto.

Veremos no próximo capítulo alguns algoritmos para realizar o alinhamento de vias e redes metabólicas.

Capítulo 4

Alinhamento de vias e redes metabólicas

O alinhamento de vias metabólicas assemelha-se em alguns casos ao alinhamento de sequências. Apesar disso, vias podem possuir ramificações, ao contrário das estruturas sempre lineares das sequências. Podemos ainda almejar computar um alinhamento de várias vias e não apenas de um par delas. Vale ressaltar que apesar do alinhamento de duas vias lineares ser muito similar ao alinhamento de sequências, isso não ocorre no alinhamento múltiplo. Visando cobrir alguns pontos desse assunto nada sucinto, as seções a seguir apresentam um resumo dos algoritmos de alinhamento de vias metabólicas apresentados por Tohsato *et al.* [45], Pinter *et al.* [39] e Yang e Sze [51], superficialmente abordados na seção 3.3. O primeiro realiza o alinhamento múltiplo de vias metabólicas não ramificadas, o segundo alinha uma via com uma via ou uma rede metabólica, e o último trabalho apresenta algoritmos para alinhar uma via não ramificada com uma rede e para alinhar uma via ou parte de uma rede com uma rede metabólica.

4.1 Algoritmo de Tohsato *et al.*

O algoritmo de Tohsato *et al.* propõe o alinhamento múltiplo de vias metabólicas não ramificadas, baseado na similaridade entre as reações. A similaridade entre reações pode ser expressada pela similaridade entre os números EC das enzimas que as catalisam. Para a compreensão do algoritmo, é preciso considerar algumas definições, expostas nos próximos parágrafos.

Para uma classe de enzima h, denotamos por $E(h)$ o conjunto de todas as classes de enzima abaixo de h na hierarquia de enzimas e $C(h)$ a quantidade de classes de enzima em $E(h)$, como mostra a Figura 4.1.

Para um conjunto de vias $S = \{s_1, \ldots, s_n\}$, com $1 \leq i \leq n$, $N(s_i)$ é o número de enzimas em s_i (distintas ou não) e $o(e, s_i)$ o número de vezes que a enzima e aparece em s_i. Então,

$$p(e) = \frac{\sum_{i=1}^{n} o(e, s_i)}{\sum_{i=1}^{n} N(s_i)} \tag{4.1}$$

é a **probabilidade de ocorrência** de e em S, onde sempre $0 \leq p(e) \leq 1$.

Dado um conjunto de vias S e uma classe de enzima h, $p(h)$ é a **probabilidade de classe** h em S e representa a soma da probabilidade de ocorrência de todas as enzimas em $E(h)$.

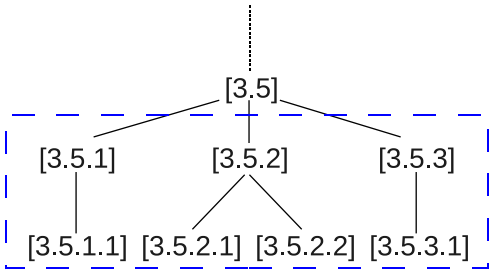


Figura 4.1: No exemplo, se h é a classe de enzima `[3.5]`, $E(h)$ corresponde às enzimas no interior do tracejado e $C(h) = 7$.

Para uma classe de enzima h, $I(h)$ é o **volume de informação** de h, sendo calculado assim:

$$I(h) = \log_2 \frac{1}{C(h)} - \log_2 p(h) \ . \tag{4.2}$$

Para duas enzimas e_i e e_j, se a classe superior em comum é h_{ij}, então $I(h_{ij})$ representa a similaridade entre elas. Quanto maior a similaridade entre e_i e e_j, maior o valor de $I(h_{ij})$. Na Equação 4.2, o primeiro termo cresce inversamente aos valores de $C(h)$ e $p(h)$, o que significa que teremos um valor maior para classes de enzima mais próximas às folhas na hierarquia de enzimas. Note que quanto mais próximas duas enzimas estiverem na árvore, menos distante delas (e mais próxima às folhas) estará sua classe superior em comum. O segundo termo cresce inversamente à probabilidade de ocorrência de classe de h, o que significa que se duas enzimas comparadas pertencerem à uma subárvore onde a ocorrência de suas enzimas é muito comum, a similaridade entre elas é menos relevante.

Convém notar que $I(\texttt{[*]})$ é constante, com o valor $\log_2(\frac{1}{3705}) - \log_2(1) = -11,85$ quando [45] foi publicado, visto que havia 3705 números EC na época e $p(\texttt{[*]}) = 1$. Além disso, $I(h) \leq I(h_i)$ para $i = 1, \ldots, n$ onde $h_1, \ldots, h_n$ são subclasses de h.

Tendo como base as definições vistas acima, é possível realizar o alinhamento de duas vias metabólicas não ramificadas da mesma forma que o alinhamento de sequências. A penalidade de inserção de espaço g é definida como -15, de forma que tenha uma pontuação pior que o volume de informação de `[*]`. Para duas vias metabólicas v_1 e v_2, se definirmos a como a matriz que armazena os valores computados das similaridades entre prefixos de v_1 e v_2, a posição $a[i, j]$ pode ser calculada como:

$$a[i,j] = \max \begin{cases} a[i, j-1] + g \ , \\ a[i-1, j-1] + I(h_{ij}) \ , \\ a[i-1, j] + g \ . \end{cases} \tag{4.3}$$

Do alinhamento resulta um **padrão**, determinado de acordo com as setas escolhidas no alinhamento, que representam o maior valor na Equação 4.3. Por exemplo, se para duas vias `[2.4.2.4][3.1.3.5][2.7.4.9]` e `[2.4.2.3][3.5.4.5][3.1.3.5][2.7.4.14]` o melhor alinhamento é:

```
[2.4.2.4]     -     [3.1.3.5][2.7.4.9]
[2.4.2.3][3.5.4.5][3.1.3.5][2.7.4.1]
```

então o padrão obtido no alinhamento é `[2.4.2]-[3.1.3.5][2.7.4]`. A inserção de espaço expressa qualquer enzima e é representada pelo símbolo "`-`". Vale ressaltar que o alinhamento de qualquer classe de enzima ou espaço com "`-`" resulta em "`-`" e nesse caso a pontuação do alinhamento é -15. O **volume de informação** $I(p)$ de um padrão p é a pontuação do alinhamento. O tempo para o alinhamento de duas vias é o mesmo do alinhamento de sequências: $O(n^2)$, onde n é o comprimento da maior das duas vias. O Algoritmo 4.1 mostra os passos para a construção do padrão.

Algoritmo 4.1 AlinhamentoVias(a, i, j): Constrói o alinhamento global ótimo de duas vias

Entrada: matriz a de similaridades e índices i e j
Saída: comprimento c do alinhamento e o vetor *padrão* com o padrão obtido

1: **se** $i = 0$ **e** $j = 0$ **então**
2: $\quad c \leftarrow 0$
3: **senão se** $i > 0$ **e** $a[i, j] = a[i-1, j] + g$ **então**
4: $\quad c \leftarrow$ AlinhamentoVias$(a, i-1, j) + 1$
5: $\quad padrão[c] \leftarrow$ `-`
6: **senão se** $j > 0$ **e** $a[i, j] = a[i, j-1] + g$ **então**
7: $\quad c \leftarrow$ AlinhamentoVias$(a, i, j-1) + 1$
8: $\quad padrão[c] \leftarrow$ `-`
9: **senão** $\quad \triangleright\ i > 0$ **e** $j > 0$ **e** $a[i, j] = a[i-1, j-1] + I(h_{ij})$
10: $\quad c \leftarrow$ AlinhamentoVias$(a, i-1, j-1) + 1$
11: $\quad padrão[c] \leftarrow h_{ij}$
12: **devolva** c, *padrão*

Dado um conjunto de várias vias, o Algoritmo 4.2 realiza o alinhamento entre todos os pares possíveis de vias, substituindo um par que possuir a maior pontuação pelo padrão obtido, diminuindo em um o tamanho do conjunto. A operação é repetida até que reste apenas um padrão ou não seja possível melhorar a pontuação do volume de informação do conjunto, obtendo assim um alinhamento múltiplo ótimo. Dessa forma, o algoritmo visa maximizar o volume de informação do conjunto de padrões.

Para determinar o volume de informação de um conjunto de padrões, não convém apenas somar o volume de informação de cada um dos padrões do conjunto. Dessa forma, o volume de informação do conjunto diminuiria a cada repetição do algoritmo por conta da redução do número de padrões pelo alinhamento de vias. Ainda, para cada alinhamento, as classes de enzimas do padrão obtido tendem a estar um nível acima na hierarquia de enzimas comparado ao par alinhado. Por esses motivos, consideramos que a diferença média do volume de informação entre classes de enzimas que diferem em um nível é w e adicionamos w ao volume de informação do padrão obtido em cada alinhamento. Em [45], os autores verificaram que o volume de informação diferia em 5 para classes de enzimas com um nível de diferença, então definiram $w = 5$.

Dado um conjunto S de vias, o volume de informação $I(P)$ de um conjunto P de vias e padrões resultante da execução do algoritmo é:

$$I(P) = w\bar{l}(n-k) + \frac{1}{k}\sum_{p \in P} I(p)\ , \tag{4.4}$$

onde $n = |S|$, $k = |P|$, p um padrão obtido pelo alinhamento múltiplo, $\bar{l}$ é uma constante definida como o comprimento médio das vias de S e w é a constante que representa a diferença

do volume de informação entre as classes de enzima que diferem em um nível. Dessa forma, há um equilíbrio no volume de informação após cada novo conjunto P, pois a menor quantidade de padrões no conjunto é compensada pela forma com que k é utilizado na fórmula. Um exemplo de execução do algoritmo pode ser visto nas Figuras 4.2 e 4.3.

Algoritmo 4.2 IMA(S): Realiza o alinhamento múltiplo de vias metabólicas

Entrada: conjunto S de vias
Saída: conjunto P de padrões

```
1:  n ← |S|
2:  P ← S
3:  P_0 ← S
4:  max ← I(P)
5:  para k ← n − 1 descendo até 1 faça
6:      P_{n−k} ← ∅
7:      para cada p_i ∈ P faça
8:          para cada p_j ∈ P − {p_i} faça
9:              p ← AlinhamentoVias(p_i, p_j)
10:             P' ← (P − {p_i, p_j}) ∪ {p}
11:             se max < I(P') então
12:                 P_{n−k} ← P'
13:                 max ← I(P')
14:     se P_{n−k} = ∅ ou I(P_{n−k}) < I(P_{n−k−1}) então
15:         devolva P
16:     P ← P_{n−k}
17: devolva P
```

Vejamos o tempo de execução do alinhamento, considerando que nas análises assintóticas desse trabalho usamos a notação X em vez de $|X|$ para tamanho de conjuntos. No Algoritmo 4.2 são realizadas, a cada iteração do laço principal, $P_k \times (P_k - 1)$ alinhamentos de vias, onde P_k é o conjunto de padrões na iteração k do laço. Portanto, são realizados no máximo $(n-1) \times P_k \times (P_k - 1)$ alinhamentos, onde $n = |S|$. Já vimos que o tempo para o alinhamento de um par de vias é $O(c^2)$, onde c é o comprimento da maior das duas vias, então o tempo de qualquer alinhamento de vias é no máximo $c_{\max}^2$ onde $c_{\max}$ é o comprimento da maior via em S. Considerando que $P_k \leq n$ para qualquer iteração k, o tempo de execução do algoritmo é limitado a $(n-1) \times P_k \times (P_k - 1) \times c_{\max}^2 \leq (n-1) \times n \times n \times c_{\max}^2$. Logo, o algoritmo tem tempo $O(n^3 c_{\max}^2)$.

4.2 Algoritmo de Pinter *et al.*

Veremos o algoritmo usado na construção da ferramenta `MetaPathwayHunter`, concebida para realizar e exibir o alinhamento de padrões em grafos rotulados. As vias metabólicas são modeladas utilizando grafos de reações, onde vértices representam reações, rotulando-os com as enzimas que catalisam essas reações. Dado um modelo de pontuação, são levadas em conta a similaridade entre as enzimas, representadas pelos vértices e a semelhança topológica entre os grafos de forma biologicamente significativa. Inicialmente veremos como o algoritmo funciona para árvores com raiz, estendendo-o então para árvores multifontes, que têm arestas orientadas e não têm raiz. Os autores verificaram que a topologia da maioria das redes metabólicas pode

S

1:	[5.3.1.5]	[2.1.2.1]	[2.7.1.9]	[4.1.2.7]	
2:	[5.3.1.1]	[2.1.2.3]	[2.7.1.9]	[4.1.2.2]	$n = 5$
3:	[5.3.1.6]	[2.7.1.9]	[4.1.2.9]		$\bar{l} = 3{,}25$
4:	[5.3.1.9]	[2.7.1.9]	[4.1.2.3]		$w = 7$
5:	[5.3.1.8]	[2.7.1.9]	[4.1.2.3]		$g = -22$

(a) Dados de entrada: conjunto S com com vias a serem alinhadas, tamanho n de S, comprimento médio $\bar{l}$ das vias de S, diferença w do volume de informação entre as classes de enzima que diferem em um nível e penalidade g. Inicialmente $P = S$.

P

1:	[5.3.1.5]	[2.1.2.1]	[2.7.1.9]	[4.1.2.7]	
2:	[5.3.1.1]	[2.1.2.3]	[2.7.1.9]	[4.1.2.2]	
3:	[5.3.1.6]	[2.7.1.9]	[4.1.2.9]		
6:	[5.3.1]	[2.7.1.9]	[4.1.2.3]		$I(P) \approx 33{,}71$

(b) Resultado após a primeira iteração do algoritmo. Foram removidos de P os itens 4 e 5 e adicionado o item 6, resultado do alinhamento deles, e que resultou no conjunto P de maior pontuação possível $\approx 33{,}71$.

Figura 4.2: Exemplo de execução do algoritmo de Tohsato *et al.*. Para a hierarquia de enzimas, utilizamos a base do ano de 2012 que possui 5744 enzimas, resultando em uma árvore de hierarquia com 6097 vértices e $I(\texttt{[*]}) \approx -12{,}57$. Por conta disso, os valores de w e g são diferentes do artigo original.

P

1:	[5.3.1.5]	[2.1.2.1]	[2.7.1.9]	[4.1.2.7]	
2:	[5.3.1.1]	[2.1.2.3]	[2.7.1.9]	[4.1.2.2]	
7:	[5.3.1]	[2.7.1.9]	[4.1.2]		$I(P) \approx 55{,}13$

(a) Resultado após a segunda iteração do algoritmo. Foram removidos de P os itens 3 e 6 e adicionado o item 7, resultando um volume de informação $\approx 55{,}13$.

P

8:	[5.3.1]	[2.1.2]	[2.7.1.9]	[4.1.2]	
7:	[5.3.1]	[2.7.1.9]	[4.1.2]		$I(P) \approx 65{,}55$

(b) Resultado após a terceira iteração do algoritmo. Foram removidos de P os itens 1 e 2 e adicionado o item 8, resultando um volume de informação $\approx 65{,}55$.

P

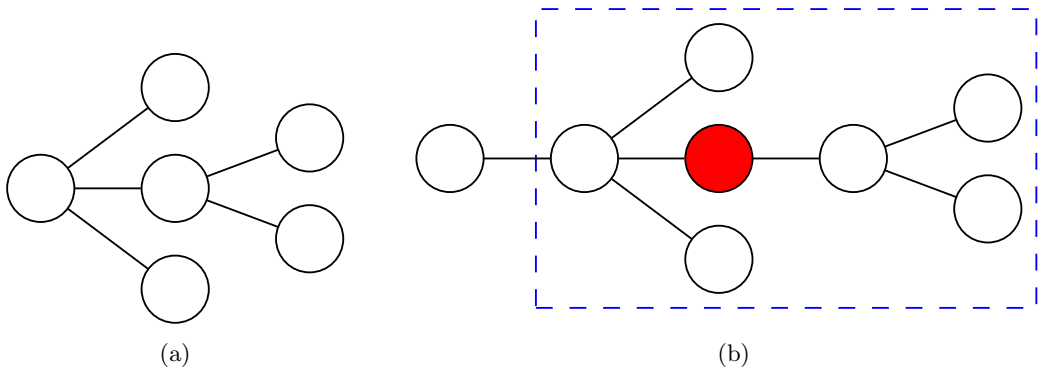
9:	[5.3.1]	[-]	[2.7.1.9]	[4.1.2]	$I(P) \approx 67{,}76$

(c) Resultado após a quarta e última iteração do algoritmo. Foram removidos de P os itens 7 e 8 e adicionado o item 9, resultando um volume de informação $\approx 67{,}76$. Caso essa pontuação fosse menor que 65,55, o algoritmo teria parado na iteração anterior com dois padrões em P.

Figura 4.3: Exemplo de execução do algoritmo de Tohsato para a entrada da Figura 4.2, contemplando da segunda iteração à final.

ser moldada ou transformada em árvores multifontes sem perda de generalidade, visto que ciclos são raros nesse contexto.

Para a comparação dos grafos, é usada uma variação do problema de isomorfismo de subárvores chamada **homeomorfismo de subárvores**, que consiste em dadas duas árvores T e P chamadas texto e padrão, respectivamente, descobrir se T tem uma subárvore t isomorfa a P. Nesse problema, t pode ser obtida também pela remoção de vértices de grau 2, ligando por uma aresta os vizinhos do vértice removido, ao exemplo da Figura 4.4.



(a) (b)

Figura 4.4: (a) árvore P; (b) árvore T, onde há uma subárvore delimitada pela linha tracejada, isomorfa a P ao remover o vértice de grau 2 preenchido.

Os autores argumentam que a remoção desse tipo de vértices é perfeitamente aceitável do ponto de vista biológico e torna o problema mais fácil de ser resolvido do ponto de vista computacional. Biologicamente, uma única enzima pode substituir algumas enzimas atuantes em uma via. A substituição pode ocorrer se a enzima substituta é multifuncional e portanto pode catalizar diversas reações consecutivas, ou se ela utiliza uma catálise alternativa que produz os produtos finais diretamente dos substratos iniciais da sequência de reações original. Ainda, enzimas que catalizam apenas uma reação são mais suscetíveis de serem substituídas do que aquelas que catalizam várias reações, por razões bioquímicas e de parcimônia. Traduzir essa descrição biológica para grafos implica na possibilidade de vértices de grau 2 serem removidos do grafo, conceito perfeitamente capturado pelo homeomorfismo de subárvores.

O algoritmo realiza um alinhamento aproximado, visto que dois vértices podem ser pareados de acordo com sua similaridade, não necessitando que sejam idênticos. Tomemos como entrada uma tabela S de similaridade entre as enzimas das vias a serem alinhadas, como mostra a Tabela 4.5. A similaridade entre um par de enzimas pode ser definida manualmente ou por algum processo, como por exemplo, o cálculo utilizando a Equação 4.2. É necessário também um valor g pré-definido para a penalidade de remoção de vértices.

	a	b	d
a	$+2$	$+1$	-2
c	-3	-2	-2

Figura 4.5: Exemplo de tabela de similaridade entre enzimas. Tal tabela é necessária caso dois grafos T e P com $V(T) = \{a, b, d\}$ e $V(P) = \{c, a\}$ sejam comparados.

Consideremos duas árvores rotuladas T_1 e T_2, onde T_2 é homeomorfa à T_1, ou seja, T_1 e T_2 são isomorfas ao realizarmos em T_2 algumas operações de remoção de vértices de grau 2. $\mathcal{M}[T_1, T_2]$ é um **mapeamento de vértices um-para-um de homeomorfismo** de T_1 para T_2 que preserva

as relações de ascendência/descendência entre os vértices. A pontuação de homeomorfismo de subárvores rotuladas, denotada por $HSR(\mathcal{M}[T_1, T_2])$ é:

$$HSR(\mathcal{M}[T_1, T_2]) = g(|T_2| - |T_1|) + \sum_{\forall(u,v)\in\mathcal{M}} S[u, v] \ . \tag{4.5}$$

A partir das definições anteriores, o problema de **homeomorfismo de subárvores rotuladas aproximado** pode ser definido desta forma: dadas duas árvores rotuladas P e T, uma tabela de similaridade S e uma penalidade de inserção de espaço g, precisamos encontrar um mapeamento $\mathcal{M}[P, t]$ de P para alguma subárvore t de T tal que $HSR(\mathcal{M}[P, t])$ é maximal.

O problema do **emparelhamento máximo em grafos bipartidos com pesos nas arestas** se resume em encontrar um emparelhamento de cardinalidade máxima onde a soma dos pesos das arestas é máxima. O algoritmo exposto nesta seção é baseado nesse problema e utiliza a técnica de programação dinâmica, computando os alinhamentos ótimos entre P e qualquer subárvore homeomorfa t de T, maximizando $HSR(\mathcal{M}[P, t])$. Inicialmente, resolve o problema para árvores com raiz, sendo então estendido para árvores multifontes.

Tomemos como entrada do problema uma árvore texto $T^r = (V_{T^r}, E_{T^r})$, onde r é o vértice raiz de T^r, e uma árvore padrão $P^{r'} = (V_{P^{r'}}, E_{P^{r'}})$, onde r' é raiz de $P^{r'}$. Sejam $p_u^{r'}$ uma subárvore de $P^{r'}$ com raiz no vértice $u \in V_{P^{r'}}$, t_v^r uma subárvore de T^r com raiz no vértice $v \in V_T$, $y_1, \ldots, y_{c(v)}$ os filhos de v e $x_1, \ldots, x_{c(u)}$ os filhos de u.

Considerando uma tabela PA de pontuação de alinhamentos, cada linha representa um vértice de $P^{r'}$ e cada coluna representa um vértice de T^r. A posição $PA[u, v]$ armazena, para dois vértices $v \in V_T$ e $u \in V_P$, o HSR maximal entre qualquer subárvore $p_u^{r'}$ de $P^{r'}$ e uma subárvore homeomorfa t_v^r de T^r correspondente, se existir, ou $-\infty$ caso contrário.

Algoritmo 4.3 HSRVias(r', v, S, PA): Realiza o alinhamento de árvores padrão e texto

Entrada: Vértice r' raiz de $P^{r'}$, vértice $v \in V_T$ e tabelas S e PA
Saída: Tabela PA preenchida

```
1: para cada filho y_i de v faça
2:     PA ← HSRVias(r', y_i, S, PA)
3: PA ← HSRPadrão(r', v, S, PA)
4: devolva PA
```

Algoritmo 4.4 HSRPadrão(u, v, S, PA): Calcula o HSR máximo entre v e vértices de V_P

Entrada: Dois vértices $u \in V_P$ e $v \in V_T$ e tabelas S e PA
Saída: Tabela PA com a coluna de v preenchida

```
1: para cada filho x_i de u faça
2:     PA ← HSRPadrão(x_i, v, S, PA)
3: PA[u, v] ← CalculaPA(u, v, S, PA)
4: devolva PA
```

A construção dessa tabela corresponde ao fluxo principal do Algoritmo 4.3, que chamamos de HSRVias, feita recursivamente passando por cada vértice da árvore T^r. Os parâmetros iniciais desse algoritmo são r', r, a tabela S e a tabela PA vazia, ou seja, chamamos inicialmente HSRVias(r', r, S, PA). O Algoritmo 4.4, chamado HSRPadrão, é uma função auxiliar que, dado um vértice v da árvore texto, calcula recursivamente o HSR máximo entre v e todos vértices da árvore padrão.

São preenchidas as posições da tabela para todos os pares de vértices utilizando os valores previamente computados. Consideremos dois vértices $v \in V_T$ e $u \in V_P$ e seus respectivos filhos $x_1, \ldots, x_{c(u)}$ e $y_1, \ldots, y_{c(v)}$. Construímos um grafo bipartido G com bipartição (X, Y), onde os vértices em X são os filhos de u e os vértices de Y são os filhos de v. Adicionamos em G uma aresta $x_i y_j$ para cada par de vértices x_i e y_j, com $w_E(x_i y_j) = PA[x_i, y_j]$. O próximo passo é encontrar o emparelhamento de cardinalidade máxima M que tenha a maior soma dos pesos das arestas, chamando de $EMP(G)$ o valor da soma das arestas de M.

Então o valor de $PA[u, v]$ é calculado usando o Algoritmo 4.5, chamado CalculaPA, que escolhe o maior entre: (i) o valor de $S[u, v]$ somado a $EMP(G)$ e (ii) o maior valor de $PA[u, y_j]$, chamado $MelhorFilho(u, v)$, somado à penalidade de remoção de v. Note que o primeiro item só é levado em conta e seu emparelhamento computado se $c(u) \leq c(v)$, visto que não é permitido deleções na árvore padrão, ou seja, um alinhamento deve conter todos os vértices dela. A Figura 4.6 mostra um exemplo de alinhamento de vias utilizando esse algoritmo.

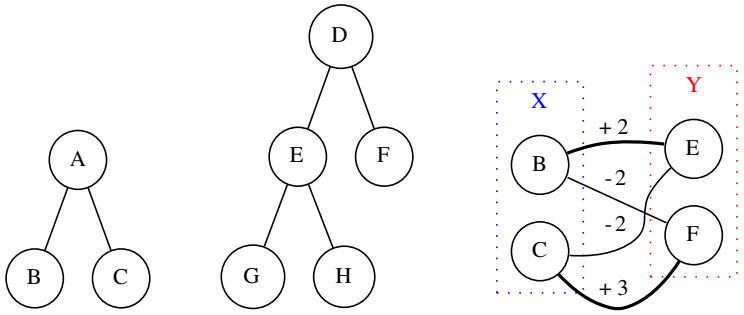
Algoritmo 4.5 CalculaPA(u, v, S, PA): Calcula a pontuação do alinhamento de u e v

Entrada: vértices $u \in V_P$ e $v \in V_T$ e tabelas S e PA
Saída: pontuação do alinhamento dos vértices u e v

se $c(u) > c(v)$ **então**
    $EMP(G) \leftarrow -\infty$
senão
    $X \leftarrow$ filhos $x_1, \ldots, x_{c(u)}$ de u
    $Y \leftarrow$ filhos $y_1, \ldots, y_{c(v)}$ de v
    $E \leftarrow \emptyset$
    para cada par de vértices $x_i \in X$ e $y_j \in Y$ **faça**
        $E \leftarrow E \cup \{x_i y_j\}$
        $w_E(x_i y_j) \leftarrow PA[x_i, y_j]$
    $G \leftarrow (X \cup Y, E)$
    $M \leftarrow$ emparelhamento de cardinalidade máxima de maior peso em G
    $EMP(G) \leftarrow \sum_{x_i y_j \in M} w_E(x_i y_j)$
se $c(v) = 0$ **então**
    $MelhorFilho(u, v) \leftarrow -\infty$
senão
    $MelhorFilho(u, v) \leftarrow \max_{j=1,\ldots,c(v)} PA[u, y_j]$
devolva $\max\{S[u, v] + EMP(G), MelhorFilho(u, v) + g\}$

O algoritmo, que realiza o alinhamento entre árvores com raiz, pode ser estendido para árvores em geral da seguinte maneira: dadas duas árvores $P = (V_P, E_P)$ e $T = (V_T, E_T)$, selecionamos um vértice r qualquer de T como raiz e para cada vértice $u \in V_P$ computamos HSRVias(u, r, S, PA). Assim estaremos calculando o alinhamento de todos pares de subárvores $(p_u^{r'}, t_v^r)$ para cada $u \in V_P$ e $v \in V_T$. Uma variação mais sofisticada desse método pode ser encontrada em [38].

O algoritmo pode ainda ser estendido para realizar o alinhamento de árvores multifontes. Para tal, é necessário que o alinhamento das subárvores de P e T leve em conta a orientação das arestas. Então consideramos algumas restrições adicionais no alinhamento de dois vértices no Algoritmo 4.5. Para os vértices u e v, o emparelhamento M só é computado se $grau^-(u) \leq grau^-(v)$ e $grau^+(u) \leq grau^+(v)$, caso contrário o valor de $EMP(G)$ é $-\infty$. Além disso, considerando o



(a) (b) (c)

	D	E	F	G	H
A	$+1$	$+1$	-3	-2	$+2$
B	-1	$+2$	-2	$+1$	$+1$
C	$+1$	-3	$+3$	-4	-1

(d)

	D	E	F	G	H
A	$+6$	0	$-\infty$	$-\infty$	$-\infty$
B	$+1$	$+2$	-2	$+1$	$+1$
C	$+2$	-2	$+3$	-4	-2

(e)

Figura 4.6: (a) árvore padrão P^A; (b) árvore texto T^D; (c) $G = (X \cup Y, E)$ usado no pareamento de A e D, sendo o emparelhamento de cardinalidade máxima de maior peso formado pelas arestas em destaque; (d) tabela de similaridade S; (e) tabela PA preenchida pelo Algoritmo HSRVias considerando $g = -1$.

grafo $G = (X \cup Y, E)$ utilizado para calcular esse emparelhamento, uma aresta (x_i, y_j) é incluída em E somente se a orientação da aresta que liga x_i a u for a mesma da aresta que liga y_j a v.

Os Algoritmos 4.3 e 4.4 preenchem a tabela PA, de dimensões $V_P \times V_T$, utilizando o Algoritmo 4.5. Ele por sua vez, tem na linha 11 sua operação mais custosa, o cálculo do emparelhamento máximo, que pode ser feito utilizando o Algoritmo de Kuhn [30] com tempo $c(u) \times c(v) \times E$. Devemos lembrar ainda que X e Y são apenas subconjuntos de V_P e V_T e têm tamanho $c(u)$ e $c(v)$, respectivamente. Considerando que $E < V_P V_T$, o tempo máximo de execução dessa operação pode ser calculado pelo seguinte somatório:

$$\begin{aligned}
\sum_{u \in V_P} \sum_{v \in V_T} c(u)c(v)E &< \sum_{u \in V_P} \sum_{v \in V_T} c(u)c(v)V_P V_T \\
&= V_P V_T \sum_{u \in V_P} \sum_{v \in V_T} c(u)c(v) \\
&= V_P V_T \sum_{u \in V_P} c(u) \sum_{v \in V_T} c(v) \\
&= V_P V_T \sum_{u \in V_P} c(u) V_T \\
&= V_P {V_T}^2 \sum_{u \in V_P} c(u) \\
&= {V_P}^2 {V_T}^2 \ .
\end{aligned}$$

Partindo dos cálculos acima, que mostram que o tempo de execução da operação mais custosa do algoritmo será menor que ${V_P}^2{V_T}^2$, podemos estabelecer sua complexidade assintótica como $O({V_P}^2{V_T}^2)$. Este por sua vez é executado V_P vezes para realizar o alinhamento de árvores em geral, resultando em uma complexidade $O({V_P}^3{V_T}^2)$. Já o algoritmo para obter o alinhamento de árvores multifontes não realiza nenhuma operação adicional que implique um aumento de sua complexidade, resultando também em $O({V_P}^3{V_T}^2)$.

4.3 Algoritmos de Yang e Sze

Yang e Sze [51] desenvolveram algoritmos para resolver os seguintes problemas: (i) dados um caminho ou via sem ramificações p e um grafo G, encontrar um caminho p' similar a p em G e (ii) dados dois grafos G_0 e G, encontrar um grafo G'_0 similar a G_0 em G. Nos dois casos, p e G_0 representam uma subestrutura de interesse a um biólogo, como uma via ou uma parte de uma rede metabólica, e G representa uma grande rede metabólica na qual se deseja encontrar uma subestrutura relacionada às citadas. Ambos algoritmos permitem remoção de vértices no grafo G.

4.3.1 Algoritmo para alinhar uma via a uma rede

Iremos focar inicialmente o primeiro algoritmo, que se resume a resolver o problema de encontrar um caminho de peso máximo em um grafo auxiliar direcionado com pesos nas arestas e vértices, obtido a partir de G, e então mostrar que o problema de encontrar os k melhores caminhos nesta estrutura pode ser revolvido em tempo polinomial.

Considere um caminho $p = p_1, p_2, \ldots, p_n$ com n vértices e um grafo $G = (V, E)$. Um conjunto de correspondências $V_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,t_i}\}$, com t_i elementos, define vértices em V que podem ser associados a p_i. Este pode ser visto como um subconjunto dos vértices em V que possuem alguma semelhança biologicamente significativa com p_i. Para cada V_i, um conjunto de similaridade $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,t_i}\}$ é formado por valores numéricos em $\mathbb{Q}_{\geq 0}$ que representam a semelhança de p_i com os elementos de V_i, ou seja, a similaridade de p_i com algum $v_{i,j}$ é representada pelo valor $s_{i,j}$. Partindo das definições acima, o problema pode ser formulado da seguinte forma: dado um caminho p de tamanho n, um grafo G, um valor de penalidade g, um limite m e conjuntos de correspondências V_i e similaridades S_i para cada p_i em p, queremos descobrir um caminho p' em G mais similar a p, onde p' é um caminho formado por elementos de $V_1, V_2, \ldots, V_n$. Deste ponto em diante, para um conjunto qualquer X, usaremos a notação $X_{i..j}$ para indicar os conjuntos $X_i, X_{i+1}, \ldots, X_{j-1}, X_j$. Veremos mais à frente os papéis de g e m no algoritmo. Convém destacar que os índices dos elementos nos conjuntos de correspondências não guardam relação com os vértices referenciados em V. Assim, por exemplo, para $i \neq j$, os elementos $v_{i,5}$ e $v_{j,8}$ de V_i e V_j podem se referir ao mesmo vértice de V, enquanto $v_{i,2}$ e $v_{j,2}$ podem se referir a vértices distintos de V.

Um alinhamento entre caminhos p e p' é definido de forma similar ao alinhamento de duas sequências. Em uma coluna, se há um pareamento com vértices que podem ser associados, ou seja, p_i com um elemento $v_{i,j}$ de V_i, então temos uma correspondência com pontuação $s_{i,j}$, dada como entrada em S_i. Se há um pareamento de p_i com um elemento não pertencente a V_i, ocorre uma diferença, penalizada com g. No caso de uma inserção de espaço, o pareamento é igualmente penalizado com o valor g. Uma remoção em G ou p' pode ser vista como uma inserção de espaço em p. No intuito de evitar um número demasiadamente grande de diferenças ou espaços, Yang e Sze seguem a abordagem de Kelley *et al.* [26], adotando um limite m de ocorrências desses casos entre duas colunas com correspondências em um alinhamento. Dessa forma, é possível ter no máximo m diferenças ou espaços após cada correspondência, exemplificado na Figura 4.7.

Para obter o alinhamento é preciso construir o grafo auxiliar direcionado $G' = (V', E')$, com $V' = \bigcup_{i=1}^{n} V_i \cup \{x, y\}$, ou seja, V' é formado pela união de todos os conjuntos V_i mais os vértices x e y, fonte e sorvedouro respectivamente. Nesse caso, embora elementos de conjuntos de correspondências diferentes possam referenciar o mesmo vértice em G, cada elemento $v_{i,j}$ é

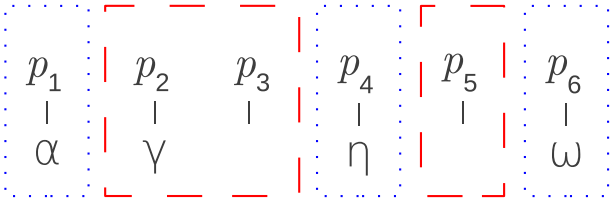


Figura 4.7: Exemplo de alinhamento com $m = 2$ e $V_1 = \{\alpha, \beta, \delta\}$, $V_2 = \{\beta, \delta, \varepsilon\}$, $V_3 = \{\varepsilon\}$, $V_4 = \{\eta, \lambda\}$, $V_5 = \{\}$ e $V_6 = \{\alpha, \gamma, \pi, \omega\}$. Nas colunas pontilhadas em azul ocorrem correspondências, uma vez que $\alpha \in V_1$, $\eta \in V_4$ e $\omega \in V_6$. Na coluna 2 ocorre uma diferença, já que $\gamma \notin V_2$, e nas restantes inserção de espaço, sendo em ambos casos as colunas tracejadas em vermelho. Como $m = 2$, não há mais de duas colunas tracejadas vermelhas seguidas após uma pontilhada azul.

tratado como um vértice distinto de peso $s_{i,j}$. Os vértices x e y têm peso 0.

O conjunto E' é construído da seguinte forma: para cada par de vértices $v_{i,j}$ e $v_{i+d,l}$ tal que $0 < d \leq m + 1$, compute d', que representa o tamanho do menor caminho em G do vértice referenciado por $v_{i,j}$ ao vértice referenciado por $v_{i+d,l}$. Se $d' \leq m+1$, adicione em E' uma aresta orientada de $v_{i,j}$ à $v_{i+d,l}$ com peso $(\max(d, d') - 1)g$. Adicione também, para cada $v_{i,j}$, arestas orientadas de x à $v_{i,j}$ com peso $(i-1)g$ e de $v_{i,j}$ à y com peso $(n-i)g$. É importante ressaltar que as fórmulas acima para o cálculo dos pesos das arestas foram concebidas para serem utilizadas em conjunto com um valor de $g \leq 0$. A construção de G' pode ser vista no Algoritmo 4.6, que por sua vez necessita de algum algoritmo ComputaDistâncias que receba um grafo e devolva uma tabela com o tamanho do menor caminho entre todos os pares de vértices.

O grafo G' pode ser intuitivamente visto como um grafo com níveis, semelhante a uma árvore com raiz. No nível 0 está o vértice x, no nível $n + 1$ está o vértice y e em cada nível i estão os vértices de V_i que podem ser associados à p_i. Um exemplo pode ser visto na Figura 4.8.

Na construção de G' acima, G pode ser uma grafo orientado ou não, enquanto G' é sempre orientado. Dessa forma, o problema de alinhamento se reduz a encontrar um caminho p' de x a y em G' com a soma máxima de pesos de vértices e arestas. Cada vértice $v_{i,j}$ no caminho p' significa que o vértice em G representado por $v_{i,j}$ foi associado à p_i. Não é difícil perceber que no máximo um vértice de cada nível em G' estará em p', uma vez que não há arestas ligando vértices de um mesmo nível. Cada aresta $(v_{i,j}, v_{i+d,l})$ representa que é possível encontrar um caminho entre os vértices representados por $v_{i,j}$ e $v_{i+d,l}$ em G de modo que haja no máximo m diferenças ou espaços entre as correspondências p_i com $v_{i,j}$ e p_{i+d} com $v_{i+d,l}$, conforme exemplificado na Figura 4.7. Como visto anteriormente, diferenças e inserções de espaço são tratadas e penalizadas da mesma forma no algoritmo, levando em conta que, quando há em p' uma aresta $(v_{i,j}, v_{i+d,l})$ com $d > 1$, os vértices $p_{i+1\ldots i+d-1}$ não são associados a vértice algum de G', e, como consequência, de $V_{i+1\ldots i+d-1}$, o que indica que ocorreu uma diferença ou inserção de espaço, ambas penalizadas com o mesmo valor. Assim, uma diferença ou espaço pode ser visto como um "pulo" de um nível em G'. É importante ressaltar que na soma dos pesos de p' as correspondências pontuam nos vértices, uma vez que estes têm peso de acordo com suas similaridades com os vértices relacionados em p. Já as diferenças ou espaços pontuam nas arestas, pois é fácil verificar que, quanto maior a distância em G ou G' entre dois vértices alinhados de p', mais negativos serão seus pesos. Caso os vértices alinhados sejam vizinhos em G e G', a aresta em p' terá peso igual a 0.

Para que o problema seja mais fácil de resolver, uma restrição importante é tratar diferenças e inserções de espaço exatamente da mesma maneira, incluindo a relação com a definição de

Algoritmo 4.6 YangSze$G'(n, m, g, t_{1...n}, V_{1...n}, S_{1...n}, G)$: Constrói o grafo auxiliar G'

Entrada: tamanhos n e $t_{1...n}$, limite m, penalidade g, conjuntos $V_{1...n}$ e $S_{1...n}$ e grafo G
Saída: grafo G' utilizado para o alinhamento

```
1:  V' ← {x, y}
2:  w_V(x) ← w_V(y) ← 0
3:  para i ← 1 até n faça
4:      para j ← 1 até t_i faça
5:          V ← V ∪ {v_{i,j}}
6:          w_V(v_ij) ← s_{i,j}
7:  Distâncias ← ComputaDistâncias(G)
8:  E' ← ∅
9:  para i ← 1 até n faça
10:     para j ← 1 até t_i faça
11:         para d ← 1 até m + 1 faça
12:             para l ← 1 até t_{i+d} faça
13:                 d' ← Distâncias[v_{i,j}, v_{i+d,l}]
14:                 se d' ≤ m + 1 então
15:                     E' ← E' ∪ {v_{i,j} v_{i+d,l}}
16:                     w_E(v_{i,j} v_{i+d,l}) ← (max(d, d') − 1)g
17:         E' ← E' ∪ {x v_{i,j}}
18:         w_E(x v_{i,j}) ← (i − 1)g
19:         E' ← E' ∪ {v_{i,j} y}
20:         w_E(v_{i,j} y) ← (n − i)g
21: devolva G' = (V', E')
```

m e o valor da penalidade g, como visto acima. Os autores não estão seguros se o problema ainda pode ser resolvido em tempo polinomial se forem usados esquemas de pontuação mais gerais, como por exemplo quando penalidades distintas são aplicadas de acordo com a diferença ocorrida na coluna pareada ou quando a penalidade de diferença é diferente da penalidade de inserção de espaço.

O Algoritmo 4.6 requer um algoritmo ComputaDistâncias que compute a distância entre todos os pares de vértices de um grafo sem pesos e preencha uma tabela com seus tamanhos. Isto pode ser feito de duas formas: com um algoritmo que calcule o menor caminho para cada par de vértices, que pode ser feito em tempo $O(E + V \log V)$ por par se o algoritmo de Dijkstra com heaps de Fibonacci [28] for usado, ou com um que calcule diretamente o menor caminho entre todos os pares de vértices, realizado em tempo $O(V^3)$ com o algoritmo de Floyd-Warshall [11].

Com G' em mãos, para obter o caminho p' em G mais similar a p, ou seja, o de maior soma de pesos das arestas e vértices, basta utilizar um algoritmo para encontrar um caminho de peso mínimo alterado para encontrar o caminho de peso máximo de x a y em G'. Como G' possui arestas de peso negativo mas não possui ciclos, o caminho pode ser computado utilizando o algoritmo de Dijkstra com heaps de Fibonacci em tempo $O(E + V \log V)$. Para que os pesos dos vértices sejam levados em conta, podemos, por exemplo, somar ao peso de cada aresta o peso de seu vértice destino.

Embora já tenhamos uma resposta para o problema, o cenário costuma ser um pouco diferente na prática. Os biólogos comumente desejam não apenas um caminho em G, mas os k

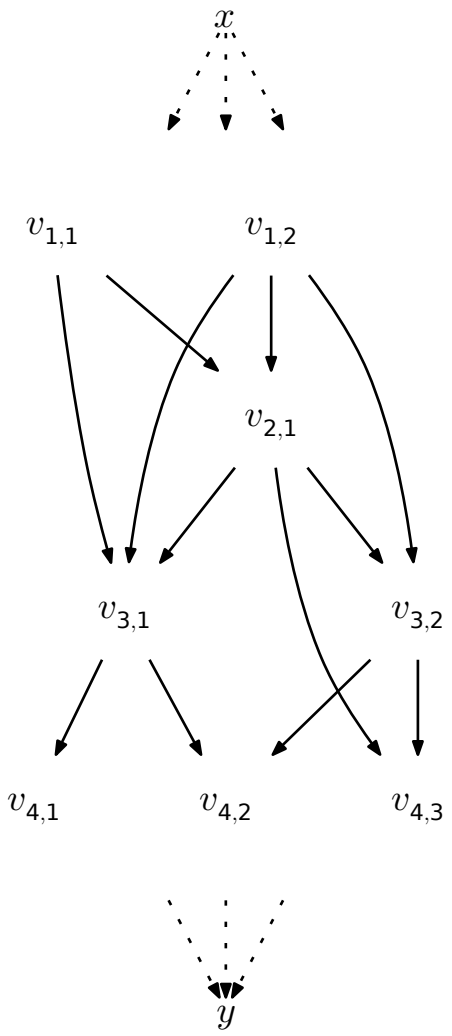
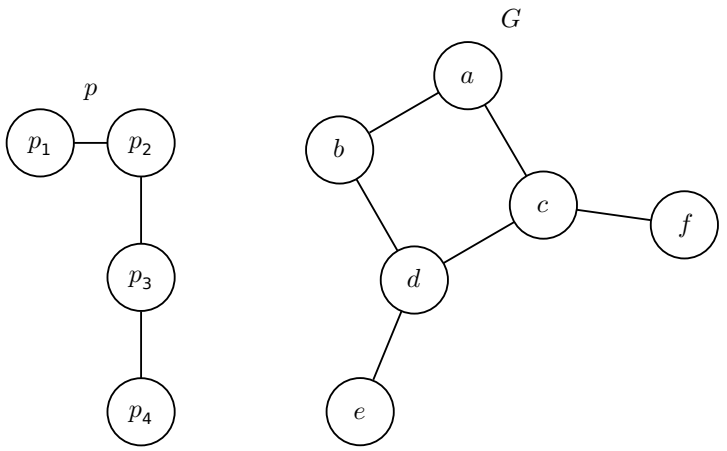


Figura 4.8: Exemplo de um grafo G' construído pelo algoritmo YangSzeG'. Por questões de clareza, todas arestas que têm x como origem ou y como destino e todos pesos de arestas e vértices foram omitidos. Cada vértice $v_{i,j}$ representa um vértice em G. No exemplo definimos $m = 1$ e como consequência não há arestas que ligam vértices com uma diferença maior que dois níveis, exceto aquelas que têm x como origem ou y como destino.

caminhos em G mais semelhantes a p. Assim, é possível realizar uma análise mais criteriosa dos resultados, levando em conta particularidades biológicas sob o crivo do conhecimento humano já acumulado de cada contexto de aplicação do algoritmo. Dessa forma, os autores do algoritmo encontram os k caminhos mais similares à p utilizando a solução o problema de encontrar os k caminhos de x a y de maior pontuação em G'. Para tal, é necessário que apenas os pesos das arestas sejam levados em conta, logo, o peso de cada vértice é adicionado aos pesos das arestas para os quais é origem. Além disso, os pesos do grafo têm os sinais trocados, ou seja, multiplicados por -1. Podemos ver as operações acima como: para cada aresta aresta uv, faça $w_E(uv) = -(w_E(uv) + w_V(u))$ e $w_V(u) = 0$. Note que o o grafo resultante pode ter arestas de peso negativo, mas não ciclos. Vemos na Figura 4.9 um exemplo de execução do algoritmo e nas Figuras 4.10 e 4.11 algumas soluções.

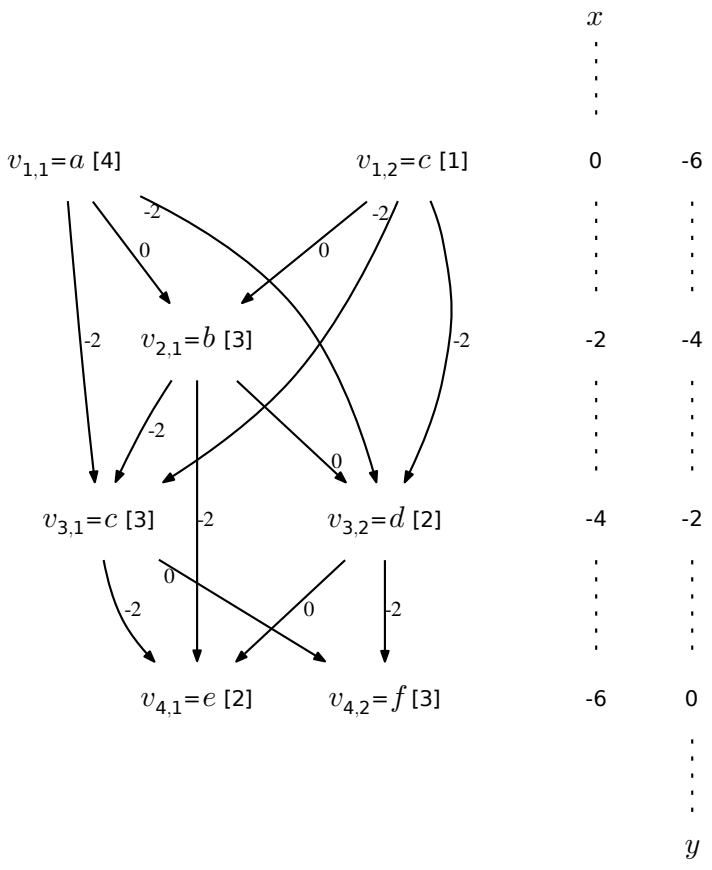


$$
\begin{array}{lcl}
n & = & 4 \\
V_1 & = & \{a, c\} \\
S_1 & = & \{4, 1\} \\
V_2 & = & \{b\} \\
S_2 & = & \{3\} \\
V_3 & = & \{c, d\} \\
S_3 & = & \{3, 2\} \\
V_4 & = & \{e, f\} \\
S_4 & = & \{2, 3\} \\
t_1, t_2, t_3, t_4 & = & 2, 1, 2, 2 \\
g & = & -2 \\
m & = & 1
\end{array}
$$

(a) Dados de entrada: caminho p, grafo G, tamanho n de p, conjuntos de correspondências e similaridades V_i e S_i com seus tamanhos t_i, penalidade g e limite m.

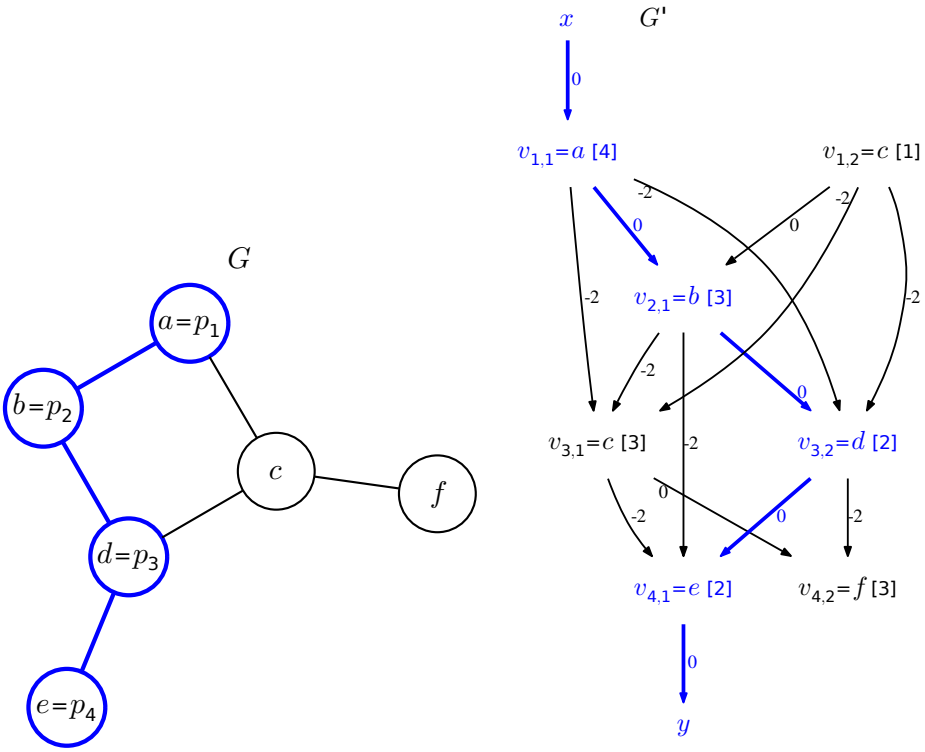
	a	b	c	d	e	f
a	0	1	1	2	3	2
b	1	0	2	1	2	3
c	1	2	0	1	2	1
d	2	1	1	0	1	2
e	3	2	2	1	0	3
f	2	3	1	2	3	0

(b) Tabela devolvida após a chamada ComputaDistâncias, utilizada em YangSzeG'.

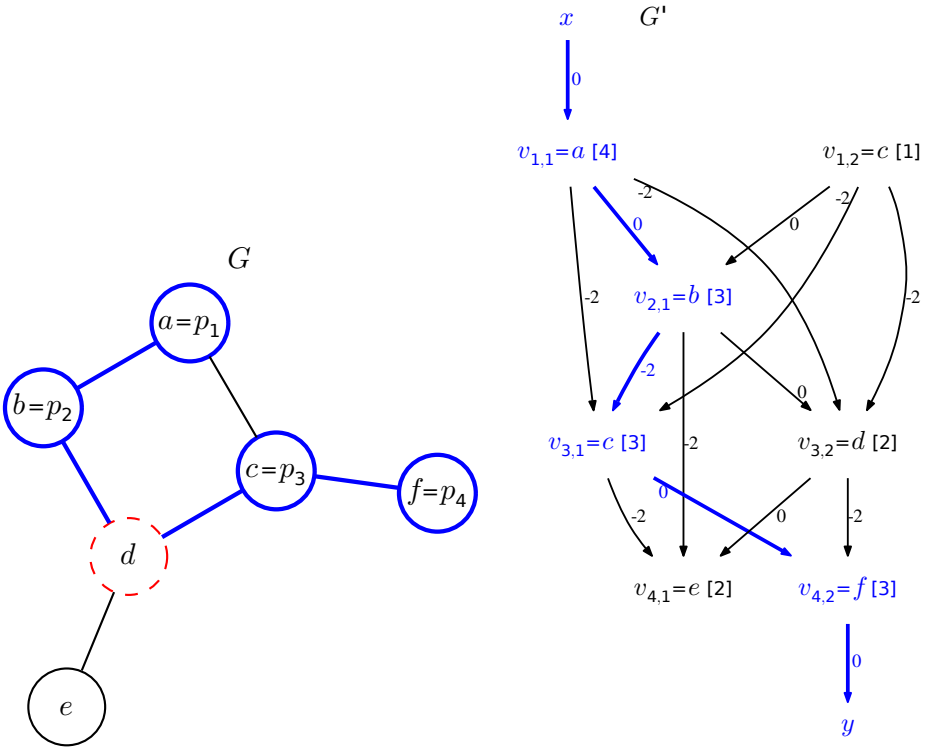


(c) Grafo G' construído pelo algoritmo YangSzeG'. No rótulo de cada vértice $v_{i,j}$ é apontado qual vértice este representa em G. Os números entre colchetes junto aos vértices e ao lado das arestas indicam os pesos dos vértices e arestas, respectivamente. À direita é mostrada a pontuação das arestas do vértice x para os vértices de cada nível e das arestas deles para o vértice y, aqui omitidas.

Figura 4.9: Exemplo de execução do algoritmo YangSzeG'.

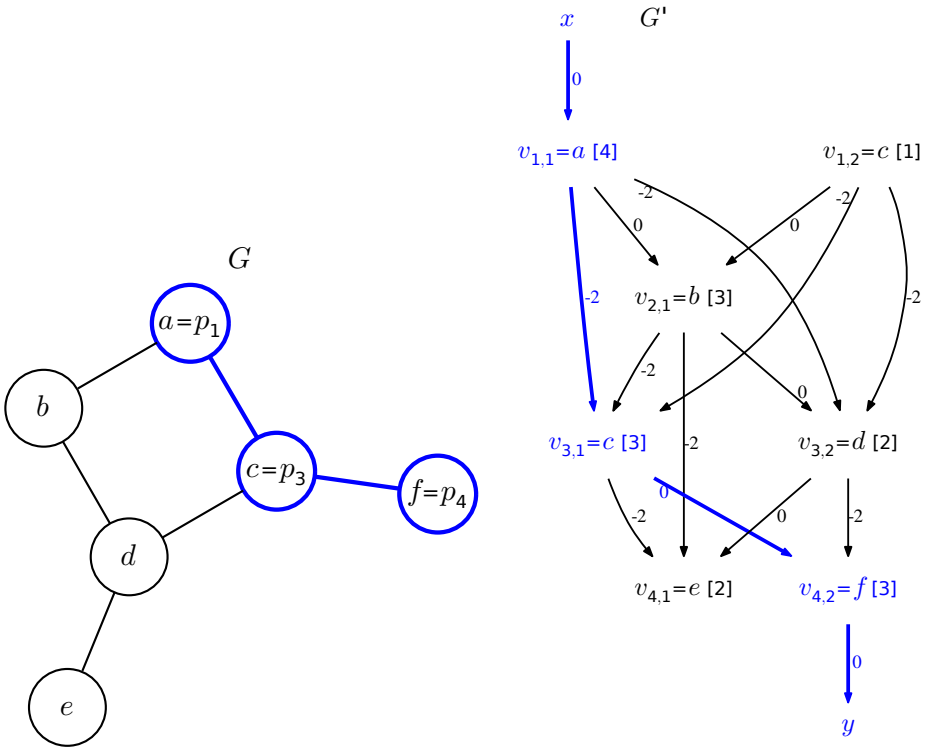


(a) Solução de pontuação 11.

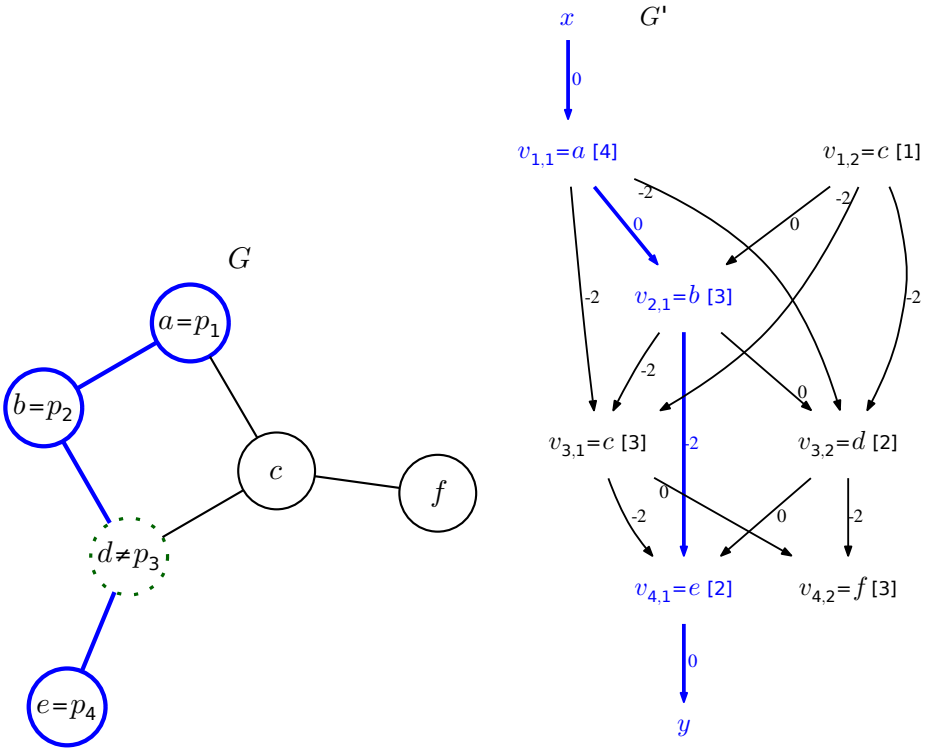


(b) Uma solução de pontuação 11 com remoção em G.

Figura 4.10: Soluções com as melhores pontuações do algoritmo de alinhamento. Os vértices e arestas em negrito e azul representam aqueles associados à p em G e o caminho de x a y em G'. Vértices tracejados em vermelho representam remoções.



(a) Uma solução de pontuação 8 com inserção de espaço em G.



(b) Uma solução de pontuação 7 com uma ocorrência de diferença.

Figura 4.11: Outras possíveis soluções do algoritmo de alinhamento. Os vértices e arestas em negrito e azul representam aqueles associados à p em G e o caminho de x a y em G'. Vértices pontilhados em verde representam diferenças.

Os autores apontam que para encontrar os k caminhos mínimos, Eppstein [16] elaborou dois algoritmos: um algoritmo básico que utiliza tempo $O(E + V \log V + k \log k)$ e é simples o bastante para ser implementado, e um algoritmo mais elaborado com tempo melhorado para $O(E+V \log V+k)$, mas que pode ser difícil de ser implementado. Realizaram então uma implementação do algoritmo básico, modificada para permitir arestas de peso negativo substituindo o algoritmo de Dijkstra, que foi utilizado para computar uma árvore de caminhos mínimos em um passo intermediário, por um algoritmo recursivo. Apesar da substituição reduzir o tempo de execução desse passo de $O(E+V \log V)$ para $O(E)$, o tempo total do algoritmo básico não é reduzido. Embora não tenham implementado o algoritmo completo, a substituição acima reduz sua complexidade de tempo para $O(E + V + k)$, tornando-o a melhor opção, se for ignorado o tempo de devolver os k caminhos $O(kV)$. Em suma, lembrando que o algoritmo ComputaDistâncias utilizado para a resolução do problema é polinomial, ambos algoritmos podem ser utilizados para resolver o problema em tempo polinomial.

4.3.2 Algoritmo para alinhar uma subrede a uma rede

Passemos agora ao problema de, dados dois grafos G_0 e G, encontrar um grafo G_0' similar a G_0 em G. Este se resume a encontrar os subgrafos de maior pontuação em G, sendo que Yang e Sze apresentam um algoritmo exato para resolver o problema quando G_0 tem um tamanho moderado, contendo em torno de 20 vértices, o suficiente para representar a maioria dos módulos funcionais. Módulos funcionais são subconjuntos de uma rede metabólica com funcionalidades relativamente independentes.

Considere os grafos $G_0 = (V_0, E_0)$ e $G = (V, E)$. Estes grafos podem ser orientados ou não, sendo que os autores consideram um grafo orientado como conexo se tal grafo é conexo ao desconsiderarmos a orientação das arestas. Um conjunto de correspondências $V_i = \{v_{i,1}, v_{i,2}, \ldots, v_{i,t_i}\}$, com t_i elementos, define vértices em V que podem ser associados a $v_i \in V_0$. De forma semelhante ao algoritmo anterior, cada conjunto V_i pode ser visto como um subconjunto dos vértices em V que possuem alguma semelhança biologicamente significativa com v_i. Para cada V_i, um conjunto de similaridade $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,t_i}\}$ é formado por valores numéricos em $\mathbb{Q}_{\geq 0}$ que representam a semelhança de v_i com os elementos de V_i, ou seja, a similaridade de v_i com algum $v_{i,j}$ é representada pelo valor $s_{i,j}$.

Partindo das definições acima, o problema pode ser formulado da seguinte forma: dados grafos G_0 e G, valores de penalidade Δ_0 e Δ_1, um limite m e conjuntos de correspondências V_i e similaridades S_i para cada v_i em V_0, queremos descobrir um grafo $G_0{}' = (V_0{}', E_0{}')$ em G mais similar a G_0. Note que $V_0{}'$ é formado por elementos de $V_{1..n}$. Veremos mais à frente os papéis de Δ_0, Δ_1 e m no algoritmo. Convém destacar que os índices dos elementos nos conjuntos de correspondências não guardam relação com os vértices referenciados em V. Assim, por exemplo, para $i \neq j$, os elementos $v_{i,5}$ e $v_{j,8}$ de V_i e V_j podem se referir ao mesmo vértice de V, enquanto $v_{i,2}$ e $v_{j,2}$ podem se referir a vértices distintos de V.

Um alinhamento entre G_0 e $G_0{}'$ é definido estabelecendo-se uma partição (V_0^+, V_0^-) de V_0, onde V_0^+ contém vértices v_i de V_0 associados a algum $v_{i,j}$ de $G_0{}'$, e V_0^- contém os vértices restantes de V_0. De modo a preservar a estrutura de G_0 em $G_0{}'$, é requisito levar em conta o valor de m, que representa o limite de inserções contíguas de espaços em $G_0{}'$. Portanto, para vértices v_i e v_k vizinhos em G_0 que estejam associados a $v_{i,j}$ e $v_{k,l}$ em $G_0{}'$, respectivamente, a distância entre os vértices representados por $v_{i,j}$ e $v_{k,l}$ em G deve ser no máximo $m + 1$. Outro requisito é que o subgrafo induzido por V_0^+ em G_0 seja conexo, de forma que $G_0{}'$ também o

seja. Um exemplo de alinhamento e seus requisitos pode ser visto na Figura 4.12. Para calcular a **pontuação de um alinhamento**, basta somar os valores de $s_{i,j}$ para cada $v_{i,j}$ associado a um v_i de V_0^+, subtraindo para cada vértice em V_0^- o valor de penalidade Δ_0 e para cada espaço em $G_0{}'$ o valor de penalidade de inserção de espaço Δ_1. Essa formulação ignora diferenças por questões de simplicidade.

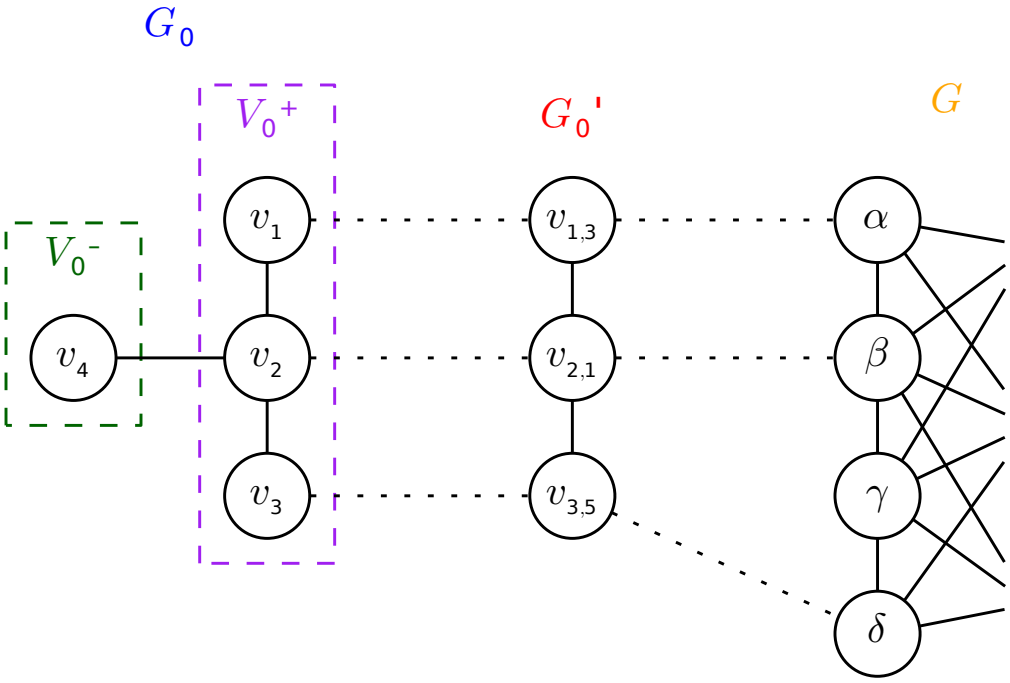


Figura 4.12: Exemplo de alinhamento entre G_0 e $G_0{}'$ para $m = 1$. As arestas tracejadas mostram as associações de vértices entre os grafos. Note que, para os vértices vizinhos v_2 e v_3 em G_0, a distância entre os vértices representados em G pelos seus respectivos vértices associados $v_{2,1}$ e $v_{3,5}$ em $G_0{}'$ não ultrapassa $m + 1 = 2$. Em outras palavras, há no máximo uma inserção de espaço em G entre vértices vizinhos em $G_0{}'$, já que temos o limite $m = 1$. Veja ainda que o subgrafo induzido por V_0^+ em G_0 é conexo.

Para resolver o problema de obter o melhor alinhamento, são enumerados todos subgrafos conexos possíveis induzidos em G_0, cada um representando uma forma de obter V_0^+ de V_0. Isso é possível pois G_0 representa um pequeno módulo funcional e é muito menor que G, a rede metabólica completa de um organismo. Sendo assim, iniciamos com um conjunto de vértices W inicialmente vazio e adicionamos um vértice $v \in V_0 \setminus W$ por vez, de forma que o subgrafo induzido em G_0 por $W \cup \{v\}$ seja conexo.

A fim de evitar a geração de cada subgrafo induzido de G_0 repetidas vezes, uma estrutura de árvore com raiz T é utilizada para gravar aqueles que já foram encontrados. Cada $G_0[V_0^+]$ com $|W|$ vértices é representado em T como um caminho de tamanho $|W|$ partindo da raiz com seus vértices ordenados pelos seus índices, marcando com o símbolo $*$ o último vértice do caminho, como pode ser observado na Figura 4.13. Em outras palavras, durante a execução do

algoritmo que gera os subgrafos induzidos, se um subgrafo induzido gerado é conexo, ordenamos seus vértices pelos índices e inserimos à árvore na sequência, adicionando os vértices necessários caso ainda não existam, e marcamos o último com $*$. Na geração dos subgrafos induzidos em G_0, quando um novo subgrafo é gerado, verificamos se ele já está representado na árvore T, o que pode ser feito em tempo $O(V_0)$. Se estiver, não é necessário continuar gerando subgrafos induzidos adicionando vértices ao subgrafo atual.

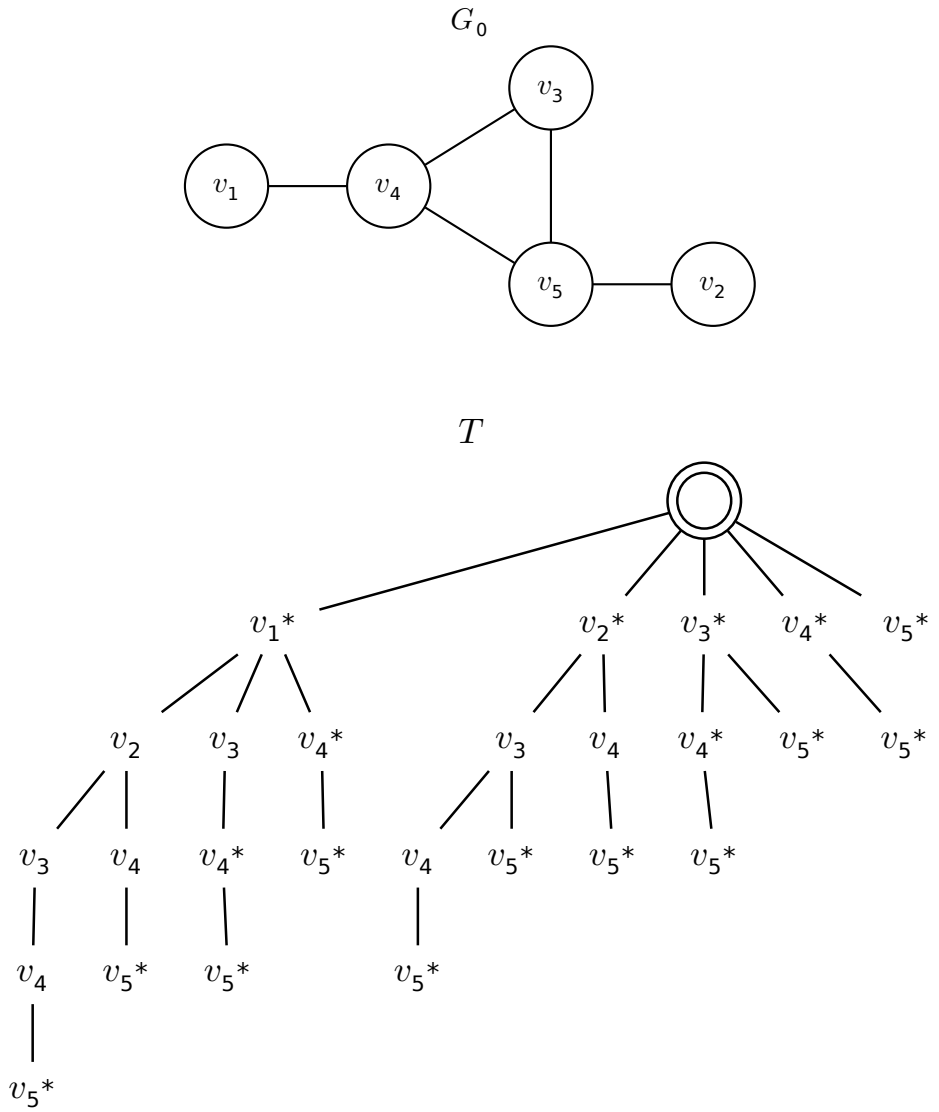


Figura 4.13: Exemplo de um grafo G_0 e da árvore T que armazena os subgrafos conexos induzidos por V_0^+ em $G_0 = (V_0, E_0)$. Veja que o subgrafo em G_0 com conjunto de vértices $W = \{v_4, v_5, v_2\}$ é conexo, por isso há um caminho $v_2, v_4, {v_5}^*$ partindo da raiz na árvore, terminado por um vértice marcado com $*$. Podemos observar que como o subgrafo com vértices $W = \{v_4, v_2\}$ não é conexo, embora exista um caminho v_2, v_4 em T partindo da raiz, o mesmo não termina com a marcação $*$.

Um passo inicial do algoritmo principal é gerar o grafo $G' = (V', E')$, que contém a enumeração de todas as soluções possíveis, em outras palavras, todos os grafos ${G_0}'$ possíveis. Esse grafo é necessário para, quando for gerada uma possível solução no fluxo principal do algoritmo, possamos verificar se a mesma é válida ou não, como veremos mais à frente. Outra função é armazenar a estrutura das soluções, já que o algoritmo obtém as melhores soluções apenas com os vértices. O Algoritmo 4.7, chamado EnumeraSoluções, mostra como a construção de G' pode ser feita. Este por sua vez, necessita de algum algoritmo ComputaDistâncias, que receba um grafo e devolva uma tabela com a distância entre todos os pares de vértices. Note que a linha 9 do algoritmo leva em conta a limitação imposta pelo valor de m, representada na Figura 4.12.

Algoritmo 4.7 EnumeraSoluções$(G, G_0, V_{1...n}, m)$: Constrói o grafo de soluções G' que enumera todas soluções possíveis

Entrada: grafos $G = (V, E)$ e $G_0 = (V_0, E_0)$, conjuntos de correspondências $V_{1...n}$ e limite m
Saída: grafo $G' = (V', E')$ que enumera todas as soluções possíveis

1: $V' \leftarrow \emptyset$
2: **para** $i \leftarrow 1$ **até** $|V_0|$ **faça**
3: $\quad V' \leftarrow V' \cup V_i$
4: *Distâncias* $\leftarrow$ ComputaDistâncias(G)
5: $E' \leftarrow \emptyset$
6: **para** $i \leftarrow 1$ **até** $|V_0|$ **faça**
7: $\quad$ **para** $j \leftarrow 1$ **até** $|V_0|$ **tal que** $j \neq i$ **faça**
8: $\quad\quad$ **para cada** par de vértices $v_{i,k} \in V_i$ e $v_{j,l} \in V_j$ **faça**
9: $\quad\quad\quad$ **se** $v_i v_j \in E_0$ **e** *Distâncias*$[v_{i,k}, v_{j,l}] \leq m + 1$ **então**
10: $\quad\quad\quad\quad E' \leftarrow E' \cup \{(v_{i,k}, v_{j,l})\}$
11: **devolva** $G' = (V', E')$

Antes de abordar o fluxo principal do algoritmo que encontra a solução do problema de alinhamento, devemos compreender a definição do que é uma solução válida, que utiliza o conjunto de arestas de G'. Considere um conjunto qualquer $V_0^+ = \{v_{a_1}, v_{a_2}, \ldots, v_{a_s}\}$ de tamanho s, sendo $a_{1..s}$ os índices dos vértices de V_0^+, sem nenhum requisito de ordem. Dado um conjunto V_0^+, uma **solução válida** é um conjunto de vértices $\{v_{a_1,b_1}, v_{a_2,b_2}, \ldots, v_{a_s,b_s}\}$, um de cada conjunto de correspondências $V_{a_1..a_s}$ com índices quaisquer $b_{1..s}$, tal que se existe uma aresta (v_{a_i}, v_{a_j}) em E_0 para algum i e algum j, então deve haver uma aresta $(v_{a_i,b_i}, v_{a_j,b_j})$ em E'. Isso corresponde a definir um alinhamento da forma descrita no início da corrente seção e exemplificado na Figura 4.12.

Para resolver o problema do alinhamento de redes, resta encontrar os k melhores alinhamentos. O algoritmo principal integra a busca por soluções válidas e a enumeração dos conjuntos V_0^+, ou seja, a construção da árvore T. Dessa forma evita-se a repetição desnecessária de procedimentos. Já que a intenção é obter apenas as k melhores soluções, ao longo da busca por soluções o algoritmo armazena somente as k melhores, utilizando a k-ésima solução como limite inferior para o armazenamento de soluções. Para tal, podemos utilizar a conhecida estrutura chamada heap mínimo [11], utilizando como chaves as pontuações das soluções e armazenando apenas as k melhores. Assim, o tempo para acessar a pior solução é $O(1)$ e para remover a pior solução e então inserir uma outra melhor é $O(\log k)$.

O pseudocódigo dessas operações pode ser vista no Algoritmo 4.8, chamado AlinhaGrafos, sendo que este é invocado pelo fluxo principal, o Algoritmo 4.9, chamado YangSze, com os parâmetros iniciais G_0, G', $W = W' = \emptyset$, um heap mínimo S com k soluções vazias de pontuação $-\infty$ e uma árvore T apenas com a raiz. É necessário o algoritmo AtualizaHeapMin que recebe um heap mínimo, uma solução, as penalidades Δ_0 e Δ_1, os conjuntos $S_{1...n}$ e calcula a pontuação da solução. Por fim, se ela é melhor do que a pontuação da raiz do heap, remove a raiz, insere a solução e devolve o heap atualizada.

Algoritmo 4.8 AlinhaGrafos$(G_0, G', W, W', M, T, \Delta_0, \Delta_1, V_{1...n}, S_{1...n})$: Obtém recursivamente os melhores alinhamentos

Entrada: grafos G_0 e G', conjuntos W, W', $V_{1...n}$ e $S_{1...n}$, heap M, árvore T e penalidades Δ_0 e Δ_1
Saída: melhores soluções de alinhamento em um heap mínimo M

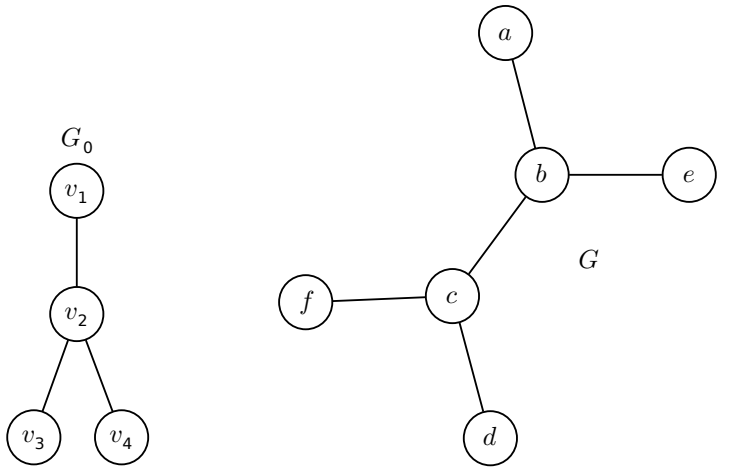
1: **para cada** $v_i \in V_0 \backslash W$ **faça**
2: **se** $G_0[W \cup \{v_i\}]$ é conexo **e** não está representado em T **então**
3: $W_{atual} \leftarrow W \cup \{v_i\}$
4: **para cada** $v_{ij} \in V_i$ **faça**
5: **se** $W' \cup \{v_{ij}\}$ é uma solução válida com $E_0(G_0)$, $E'(G')$ e $V_0^+ = W_{atual}$ **então**
6: $W'_{atual} \leftarrow W' \cup \{v_{ij}\}$
7: $M \leftarrow$ AtualizaHeapMin$(M, W'_{atual}, \Delta_0, \Delta_1, S_{1...n})$
8: $M \leftarrow$ AlinhaGrafos$(G_0, G', W_{atual}, W'_{atual}, M, T, \Delta_0, \Delta_1, V_{1...n}, S_{1...n})$
9: $T \leftarrow T \cup$ representação de $G_0[W_{atual}]$
10: **devolva** M

Algoritmo 4.9 YangSze$(G, G_0, V_{1...n}, S_{1...n}, \Delta_0, \Delta_1, k, m)$: Realiza o alinhamento entre redes metabólicas

Entrada: grafos $G = (V, E)$ e $G_0 = (V_0, E_0)$, conjuntos de correspondências $V_{1...n}$ e similaridades $S_{1...n}$, penalidades Δ_0 e Δ_1 e limites k e m
Saída: grafo $G' = (V', E')$ que enumera todas as soluções possíveis e heap mínimo M com as k melhores soluções (apenas vértices)

1: $M \leftarrow$ heap mínimo com k soluções vazias de pontuação $-\infty$
2: $T \leftarrow$ vértice raiz
3: $W \leftarrow W' \leftarrow \emptyset$
4: $G' \leftarrow$ EnumeraSoluções$(G, G_0, V_{1...n}, m)$
5: $M \leftarrow$ AlinhaGrafos$(G_0, G', W, W', M, T, \Delta_0, \Delta_1, V_{1...n}, S_{1...n})$
6: **devolva** G', M

Vemos na Figura 4.14 um exemplo de execução do algoritmo para o alinhar uma subrede a uma rede metabólica. Na sequência, podemos ver na Figura 4.15 a solução do exemplo com os k melhores alinhamentos e na Figura 4.16 outros exemplos de alinhamentos com pontuações menores que não fazem parte das melhores soluções.

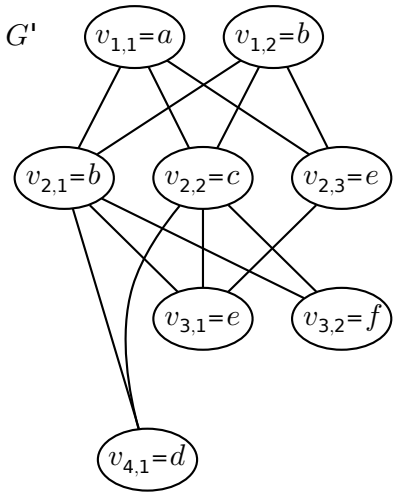


$$
\begin{array}{lcl}
n & = & 4 \\
V_1 & = & \{a, b\} \\
S_1 & = & \{3, 0\} \\
V_2 & = & \{b, c, e\} \\
S_2 & = & \{4, 1, 3\} \\
V_3 & = & \{e, f\} \\
S_3 & = & \{3, 2\} \\
V_4 & = & \{d\} \\
S_4 & = & \{1\} \\
t_1, t_2, t_3, t_4 & = & 2, 3, 2, 1 \\
\Delta_0 & = & -3 \\
\Delta_1 & = & -1 \\
k & = & 3 \\
m & = & 1
\end{array}
$$

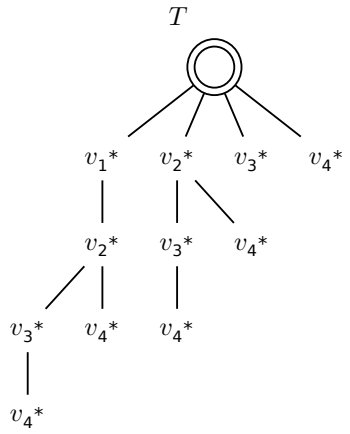
(a) Dados de entrada: grafos G_0 e G, tamanho n de G_0, conjuntos de correspondências e similaridades V_i e S_i com seus tamanhos t_i, penalidades Δ_0 e Δ_1 e limites k e m.

	a	b	c	d	e	f
a	0	1	2	3	2	3
b	1	0	1	2	1	2
c	2	1	0	1	2	1
d	3	2	1	0	3	2
e	2	1	2	3	0	3
f	3	2	1	2	3	0

(b) Tabela devolvida após a chamada de ComputaDistâncias, utilizada em EnumeraSoluções.

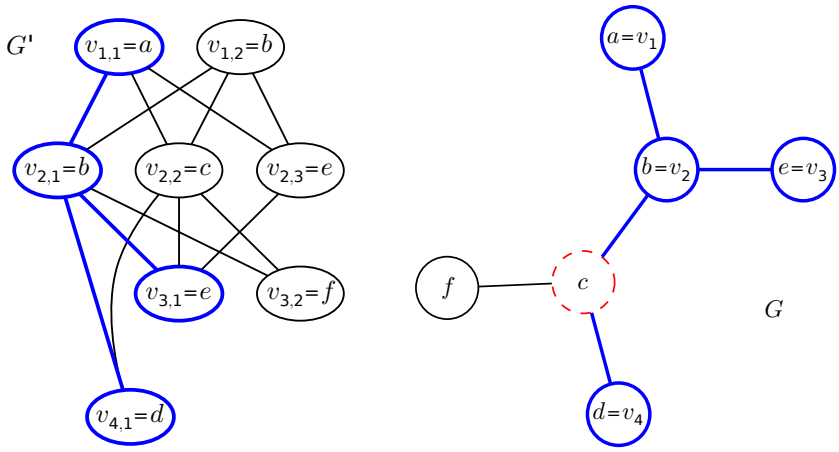


(c) Grafo G' construído por EnumeraSoluções. No rótulo de cada vértice $v_{i,j}$ é apontado qual vértice este representa em G.

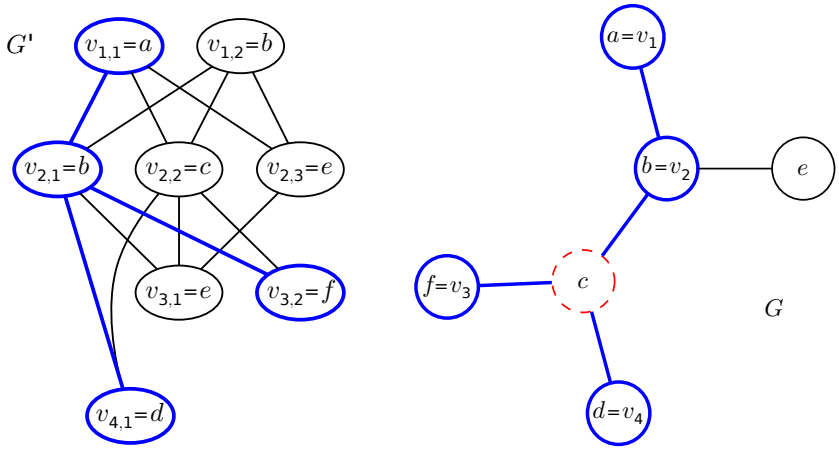


(d) Árvore T gerada por AlinhaGrafos.

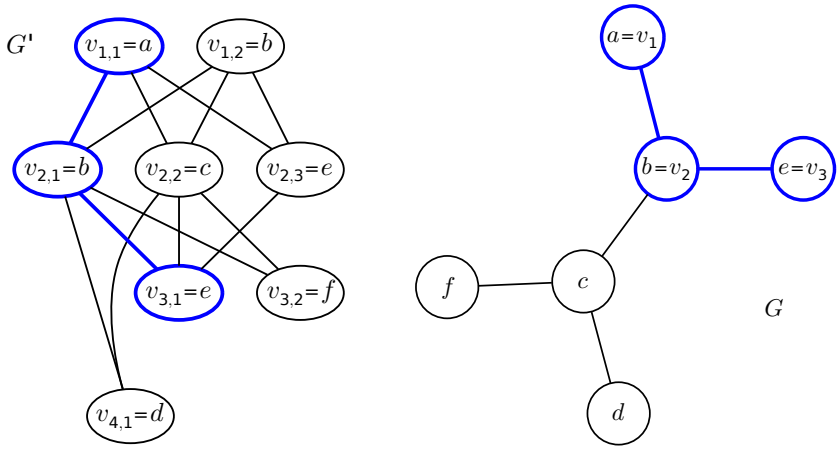
Figura 4.14: Exemplo de execução do algoritmo de alinhamento de redes.



(a) Solução com $V_0^+ = \{v_1, v_2, v_3, v_4\}$ associados a $\{a, b, e, d\}$ e com similaridades $\{3, 4, 3, 1\}$, respectivamente. Como $|V_0^-| = 0$ e há uma inserção de espaço em G', a pontuação é $3+4+3+1+(0\times-3)+(1\times-1) = 10$.

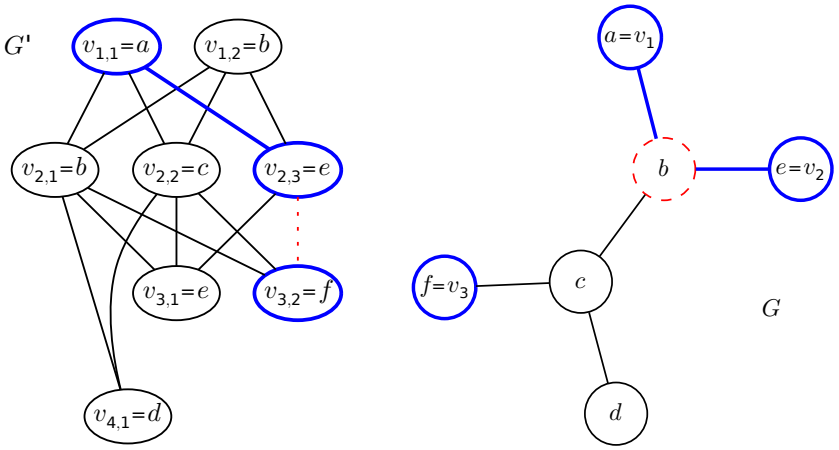


(b) Solução com $V_0^+ = \{v_1, v_2, v_3, v_4\}$ associados a $\{a, b, f, d\}$ e com similaridades $\{3, 4, 2, 1\}$, respectivamente. Como $|V_0^-| = 0$ e há duas inserções de espaço em G', nos locais das arestas $(v_{2,1}, v_{3,2})$ e $(v_{2,1}, v_{4,1})$, a pontuação é $3+4+2+1+(0\times-3)+(2\times-1) = 8$.

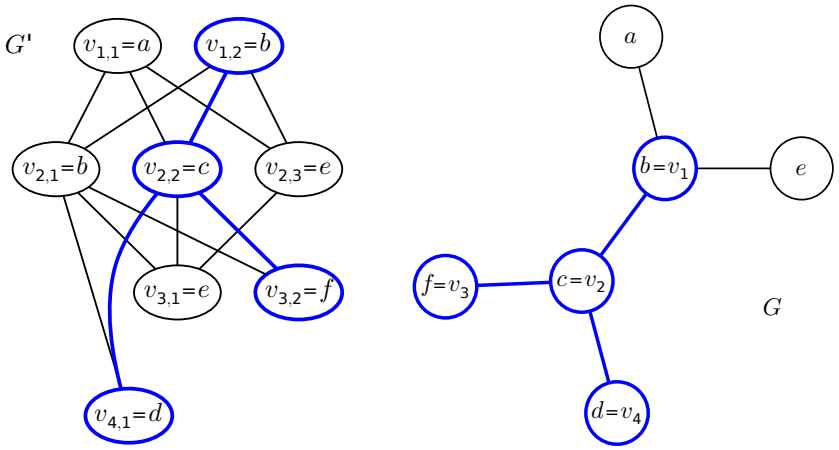


(c) Solução com $V_0^+ = \{v_1, v_2, v_3\}$ associados a $\{a, b, e\}$ e com similaridades $\{3, 4, 3\}$, respectivamente. Veja que d não faz parte da solução, logo $V_0^- = \{v_4\}$. Como $|V_0^-| = 1$ e não há nenhuma inserção de espaço em G', a pontuação é $3+4+3+(1\times-3)+(0\times-1) = 7$.

Figura 4.15: Os k melhores alinhamentos para o exemplo da Figura 4.14. Os vértices e arestas em negrito e azul representam a solução em G' e os vértices associados à G_0 em G. Vértices tracejados em vermelho representam inserções de espaço em G' (ou remoções em G).



(a) Uma solução que não é válida, com $V_0^+ = \{v_1, v_2, v_3\}$ associados a $\{a, e, f\}$, respectivamente, e com $V_0^- = \{v_4\}$. Há uma inserção de espaço em G' no local da aresta $(v_{1,1}, v_{2,3})$. Note que não há em G' aresta ligando $v_{2,3}$ a $v_{3,2}$, pois há uma distância maior que $m + 1 = 2$ entre os vértices que estes representam em G (e e f), e essa aresta deveria existir para que a solução fosse válida.



(b) Uma solução válida que não estará em M ao final da execução do algoritmo, com $V_0^+ = \{v_1, v_2, v_3, v_4\}$ associados a $\{b, c, f, d\}$ e com similaridades $\{0, 1, 2, 1\}$, respectivamente. Como $|V_0^-| = 0$ e não há nenhuma inserção de espaço em G', a pontuação é $0 + 1 + 2 + 1 + (0 \times -3) + (0 \times -1) = 4$.

Figura 4.16: Outros exemplos de alinhamentos para a entrada da Figura 4.14. Os vértices e arestas em negrito e azul representam a solução em G' e os vértices associados à G_0 em G. Vértices tracejados em vermelho representam inserções de espaço em G' (ou remoções em G).

Vejamos a complexidade do Algoritmo YangSze. É fácil perceber que o seu tempo de execução tem como operações mais custosas os algoritmos EnumeraSoluções e AlinhaGrafos, nas linhas 4 e 5 respectivamente, já que as operações restantes utilizam tempo $O(1)$ ou $O(k)$, sendo k uma constante. No primeiro, o laço da linha 2 utiliza tempo $O(V_0)$, a operação realizada por ComputaDistâncias na linha 4 pode ser feita em tempo $O(V^3)$ utilizando o algoritmo de Floyd-Warshall e os laços iniciados na linha 6 utilizam tempo $O({V_0}^2V^2)$. Como $V > V_0$, o tempo total de execução de EnumeraSoluções é proporcional a $V^3+{V_0}^2V^2 \leq V^3+V_0V^3 = (V_0+1)V^3$, ou seja, $O(V_0V^3)$.

Já no algoritmo AlinhaGrafos, sua complexidade é definida pelo tamanho de sua árvore de recursão. Consideremos que inicialmente $|W| = 0$, que a cada chamada recursiva do algoritmo esse tamanho aumenta em 1 até o limite de $|V_0|$ e que $T(0) = 1$. Para um determinado nó da árvore de recursões, realizamos uma chamada recursiva para cada combinação dos elementos v_i de $V_0 \backslash W$ e dos elementos de seus respectivos V_i. Então a cada nível da árvore de recursão, a quantidade de chamadas recursivas será no máximo a quantidade do nível anterior, multiplicada por $|V_0|-(|W|-1)$ e multiplicada pelo tamanho máximo de um conjunto V_i, ou seja, $|V|$. Assim, considerando $n = |W|$, $a = |V_0|$ e $b = |V|$, temos a seguinte recorrência:

$$\begin{aligned}
T(n) &= T(n-1)(a-n+1)b \\
&\leq T(n-1)(a-n)b \\
&\leq T(n-1)ab \\
&= T(n-2)(ab)^2 \\
&= T(n-3)(ab)^3 \\
&\vdots \\
&= T(0)(ab)^n \\
&= (ab)^n \\
&= |V_0|^{|V_0|}|V|^{|V_0|} \ .
\end{aligned}$$

Portanto, o tempo de execução de AlinhaGrafos no pior caso é $O(V_0^{V_0}V^{V_0})$. Podemos ainda verificar que:

$$\begin{aligned}
T(n) &= T(n-1)(a-n+1)b \\
&\geq T(n-1)b, \text{ pois } a > 0 \text{ e } 0 \leq n \leq a \\
&= T(n-2)b^2 \\
&= T(n-3)b^3 \\
&\vdots \\
&= T(0)b^n \\
&= b^n \\
&= |V|^{|V_0|} \ .
\end{aligned}$$

Obtemos dessa forma um limitante inferior $\Omega(V^{V_0})$. Essa complexidade é impraticável, considerando que o tempo de execução de AlinhaGrafos determina o tempo total de execução do Algoritmo YangSze.

Contudo, Yang e Sze apresentam ainda uma melhoria que, embora não seja possível definir com exatidão o ganho no tempo de execução, remove uma grande quantidade de ramificações na árvore de recursão do Algoritmo AlinhaGrafos, como exemplificado na Figura 4.17. A cada chamada, o laço da linha 4 realiza outra chamada recursiva, uma para cada possibilidade de associação entre o $v_i \in V_0$ atual e um $v_{ij} \in V_i$. Mas não convém associar v_i a v_{ij} adicionando v_i a W e v_{ij} a W', continuando a procurar soluções recursivamente a partir de W e W' se a pontuação deste não for melhor do que a pior das k melhores soluções, armazenada na raiz de M. Ainda há uma exceção: se para um v_i nenhum W' possui pontuação melhor que a raiz de M, mesmo assim deve ser realizada ao menos uma chamada recursiva do algoritmo para $W \cup \{v_i\}$ e $W' \cup \{v_{ij}\}$, já que adicionar v_i a W pode ser essencial para encontrar uma determinada boa solução posteriormente, por exemplo, se v_i for um vértice de corte em G_0. Uma forma de garantir isso é realizar a chamada recursiva para o $W' \cup \{v_{ij}\}$ com a melhor pontuação entre todos os elementos v_{ij} possíveis de V_i, como podemos ver a seguir no Algoritmo 4.10, chamado AlinhaGrafos2.

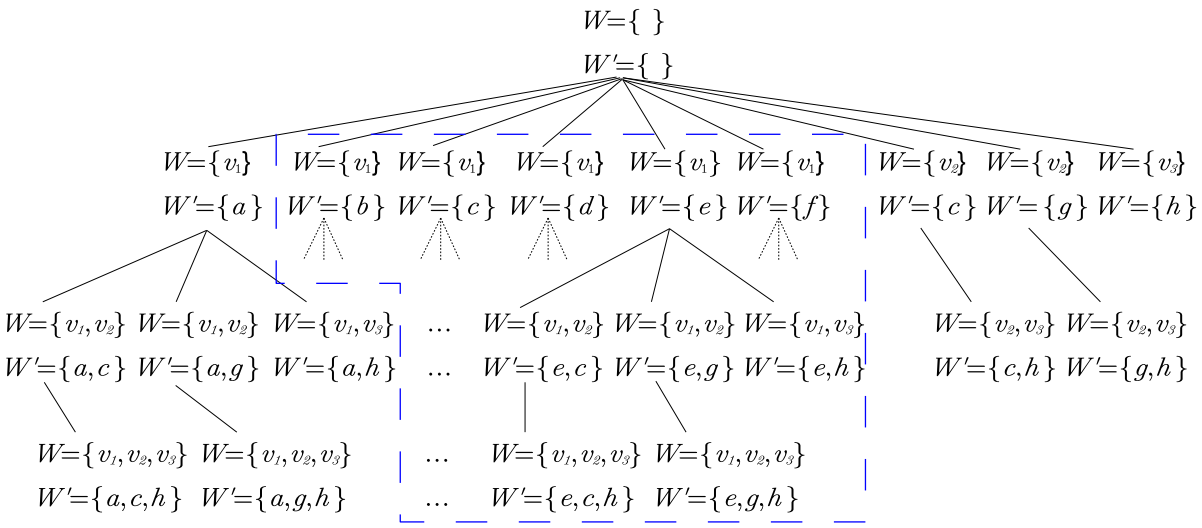


Figura 4.17: Exemplo de corte na árvore de recursão realizado pelo Algoritmo AlinhaGrafos2. Considere $V_0 = \{v_1, v_2, v_3\}$, $V_1 = \{a, b, c, d, e, f\}$, $V_2 = \{c, g\}$ e $V_3 = \{h\}$. Linhas pontilhadas representam múltiplas chamadas recursivas. Por exemplo, caso as soluções $(W = \{v_1\}, W' = \{a\})$, $(W = \{v_1, v_2\}, W' = \{a, c\})$ e $(W = \{v_1, v_3\}, W' = \{a, h\})$ tenham pontuação melhor que as soluções $(W = \{v_1\}, W' = \{b \dots f\})$, todas as chamadas na região tracejada em azul, que seriam executadas em AlinhaGrafos, não serão executadas em AlinhaGrafos2.

O tempo no pior caso de AlinhaGrafos2 ainda é o mesmo de AlinhaGrafos, mas o conjunto de entradas para que o pior caso ocorra é muito pequeno. É necessário que, para cada subgrafo induzido $W \cup \{v_i\}$ em G_0, a pontuação de $W' \cup \{v_{i,j}\}$ para cada $v_{ij} \in V_i$ seja menor que a pontuação de $W' \cup \{v_{i,j+1}\}$, dessa forma realizando sempre a chamada recursiva de AlinhaGrafos2. Ainda, para um subgrafo induzido $W \cup \{v_i\}$ em G_0, as pontuações de todos $W' \cup \{v_{i,j}\}$ possíveis devem ser piores que a pontuação de qualquer solução de um subgrafo induzido gerado posteriormente. Além disso, o tempo de execução do algoritmo é aceitável quando G_0 tem até 20 vértices, o suficiente para representar a maioria dos módulos funcionais.

Algoritmo 4.10 AlinhaGrafos2($G_0, G', W, W', M, T, \Delta_0, \Delta_1, V_{1...n}, S_{1...n}$): Obtém os melhores alinhamentos podando ramificações desnecessárias da árvore de recursões

Entrada: grafos G_0 e G', conjuntos W, W', $V_{1...n}$ e $S_{1...n}$, heap M, árvore T e penalidades Δ_0 e Δ_1

Saída: melhores soluções de alinhamento em um heap mínimo M

1: **para cada** $v_i \in V_0 \backslash W$ **faça**
2: **se** $G_0[W \cup \{v_i\}]$ é conexo **e** não está representado em T **então**
3: $W_{atual} \leftarrow W \cup \{v_i\}$
4: **para cada** $v_{ij} \in V_i$ **faça**
5: **se** $W' \cup \{v_{ij}\}$ é uma solução válida com $E_0(G_0)$, $E'(G')$ e $V_0^+ = W_{atual}$ **então**
6: $W'_{atual} \leftarrow W' \cup \{v_{ij}\}$
7: **se** pontuação de W'_{atual} > pontuação da raiz de M **então**
8: $M \leftarrow$ AtualizaHeapMin($M, W'_{atual}, \Delta_0, \Delta_1, S_{1...n}$)
9: $M \leftarrow$ AlinhaGrafos2($G_0, G', W_{atual}, W'_{atual}, M, T, \Delta_0, \Delta_1, V_{1...n}, S_{1...n}$)
10: **se** nenhuma chamada recursiva foi feita para W_{atual} **então**
11: $W'_{melhor} \leftarrow W' \cup \{v_{ij}\}$ de melhor pontuação entre os possíveis $v_{ij} \in V_i$
12: $M \leftarrow$ AlinhaGrafos2($G_0, G', W_{atual}, W'_{melhor}, M, T, \Delta_0, \Delta_1, V_{1...n}, S_{1...n}$)
13: $T \leftarrow T \cup$ representação de $G_0[W_{atual}]$
14: **devolva** M

Capítulo 5

Considerações finais

Pesquisas dedicadas ao alinhamento de vias e redes metabólicas são 1abundantes e o assunto constitui matéria de significativa relevância para o estudo de diversas questões da biologia computacional. Dispõe, desse modo, de vasta bibliografia. Durante o curso de estudos deste trabalho, pudemos verificar muitas outras abordagens e metodologias além daquelas até aqui apresentadas.

Chen e Hofestädt [6] implementaram uma ferramenta que extrai informações de metabolismo de bancos de dados na internet, exibe e manipula informações de metabolismo e constrói por um método simples o alinhamento entre vias.

Cheng *et al.* em [9] propuseram um mapeamento entre uma árvore multifontes P, representando uma via, e um subgrafo do grafo T, representando uma via ou uma rede, utilizando a ideia de homomorfismo, que permite mapear dois ou mais vértices do padrão para o mesmo vértice do texto. O algoritmo utiliza a técnica de programação dinâmica para encontrar o mapeamento de custo mínimo em tempo $O(V_T E_T + V_T^2 V_P)$ e permite a remoção de vértices no texto. Realizaram o alinhamento entre todas as vias metabólicas das espécies *E. coli*, *S. cerevisiae*, *B. subtilis* e *T. thermophilus*, encontrando então um conjunto grande de similaridades estatisticamente significativos.

Cheng *et al.* em [8] estenderam o algoritmo anterior, reduzindo seu tempo de execução para $O(V_P E_T + V_P V_T \log V_T)$, introduzindo a remoção de vértices tanto do texto quanto do padrão e permitindo no padrão a ocorrência de ciclos, tratando-os com um aumento modesto no tempo de execução.

Klau em [29] apresentaram uma formulação de emparelhamento estrutural máximo baseado em grafos para o alinhamento global de redes metabólicas. Utiliza o método de programação linear inteira, incorporando uma abordagem de relaxamento Lagrangiano em combinação com o método *branch-and-bound*, encontrando soluções próximas das ótimas com garantia de qualidade. Seus experimentos mostraram que sua formulação é razoavelmente rápida e vantajosa em relação ao uso de heurísticas.

Berg e Lässig [2] desenvolveram um método estatístico teórico para o alinhamento de duas redes metabólicas, envolvendo análise Bayesiana para a determinação de pontuações no alinhamento. O tempo de execução do alinhamento, para grafos G e H, é $O((V_G + V_H)^3)$.

Zaslavskiy *et al.* em [52] reformularam o problema de alinhamento de redes de interação proteína-proteína como o problema de alinhamento de grafos, avaliando como os algoritmos

que são o estado da arte podem resolvê-lo. Então apresentaram novos métodos que produzem resultados com qualidade superior aos dos outros autores.

Flannick *et al.* [19] buscaram encontrar módulos funcionais conservados em redes de espécies distintas, realizando o alinhamento múltiplo escalonável de redes por seu algoritmo denominado Græmlin. Seu modelo explícito de evolução funcional permite a generalização de esquemas de pontuação de alinhamento existentes e a localização de topologias de rede conservadas, não se limitando a vias metabólicas e complexos proteicos. Verificaram através de *benchmarks* quantitativos que o algoritmo obteve ganhos substanciais pela escalabilidade e uma melhor sensibilidade.

Deniélou *et al.* em [13] propuseram o algoritmo C3Part-M para o alinhamento de redes baseado em [4], sendo mais eficiente no caso de múltiplas redes. Alcançaram com seu algoritmo exato resultados similares aos do algoritmo baseado em heurísticas recém proposto em [25].

Já no trabalho corrente, não consideramos adequado iniciá-lo diretamente em discussões sobre seu tema específico, mas apresentar primeiramente no Capítulo 2 definições da biologia e da teoria dos grafos, imprescindíveis para a compreensão dos assuntos tratados nos capítulos posteriores. Aquelas exclusivamente utilizadas ou consideradas de maneira diferenciada em determinados algoritmos foram deixadas nas respectivas seções, de modo a não se confundirem com definições gerais de grafos. Vimos diversas modelagens de redes metabólicas, mas também que algumas não são interessantes em determinados contextos, seja pela demasiada complexidade ou simplicidade, pelas ambiguidades ou simplesmente por não serem adequadas aos dados analisados nos algoritmos.

No Capítulo 3 vimos que existem algoritmos muito eficientes para encontrar o alinhamento ótimo de um par de sequências, sendo que estes podem ser utilizados como base para o alinhamento de vias metabólicas. Já o alinhamento múltiplo de sequências introduz problemáticas referentes à complexidade e garantia de solução ótima, e embora permita elucidar uma grande variedade de questões da biologia, não foi possível encontrar algum trabalho que realize o alinhamento de vias ou redes metabólicas baseados nele.

Estudamos detalhadamente alguns algoritmos relevantes. O algoritmo de Tohsato *et al.* [45] propõem uma solução para encontrar um alinhamento múltiplo de vias metabólicas não ramificadas e o faz através de alinhamentos entre pares. Embora se baseie em um algoritmo de alinhamento de duas sequências, utiliza fórmulas que levam em consideração a probabilidade de ocorrência de enzimas para os cálculos das pontuações. Encontra a solução em tempo cúbico.

O algoritmo de Pinter *et al.* [39], baseado no homeomorfismo de subárvores, realiza o alinhamento de uma via com outra via ou uma rede metabólica levando em conta a similaridade entre as enzimas e a semelhança topológica, necessitando um modelo pré-definido de pontuação e devolvendo as melhores soluções até um limite definido. Inicialmente é apresentado um algoritmo restrito a árvores com raiz, estendendo-o então para um tipo de grafo chamado árvores multifontes, que permite uma topologia mais maleável que árvores com raiz. Dessa forma, a maioria das redes metabólicas pode ser modelada ou mapeada em árvores multifontes sem perda de generalidade. Seu tempo de execução é quadrático no tamanho da entrada.

No trabalho de Yang e Sze [51] pudemos encontrar algoritmos para alinhar uma via não ramificada com uma rede e alinhar uma via ou parte de uma rede com uma rede metabólica, sendo que ambos necessitam de antemão dos valores de similaridades entre os elementos das estruturas alinhadas e devolvem as melhores soluções encontradas até um limite k pré-estabelecido. O primeiro se resume a resolver o problema de encontrar um caminho de peso máximo em um grafo auxiliar direcionado com pesos nas arestas e vértices e então mostrar que o problema de encontrar

os k melhores caminhos nesta estrutura pode ser revolvido em tempo polinomial. Embora seja mais complexo que os anteriores em relação ao entendimento dos passos do algoritmo, sua complexidade total de tempo é polinomial no tamanho da entrada. O segundo é um algoritmo que se resume a encontrar os subgrafos de maior pontuação no grafo que representa a rede, viável apenas quando a via ou subrede tem um tamanho moderado, contendo em torno de 20 elementos, o suficiente para representar a maioria dos módulos funcionais. Essa limitação se deve ao fato do algoritmo gerar todas as possíveis soluções, resultando em uma complexidade exponencial no tamanho da entrada. Contudo há uma melhoria que, embora não seja possível definir com exatidão o ganho no tempo de execução, remove uma grande área no espaço de busca. O tempo no pior caso permanece o mesmo, mas o conjunto de entradas para que o pior caso ocorra é muito pequeno.

Uma extensão natural deste trabalho seria o estudo dos trabalhos de Cheng *et al.* em [9, 8], já citados anteriormente. Eles comentam diversos outros trabalhos, como por exemplo os aqui estudados Pinter *et al.* [39] e Yang e Sze [51], evidenciando algumas características consideradas por eles indesejáveis, como por exemplo: (i) restrições na topologia dos grafos alinhados, (ii) utilização de algoritmos de aproximação e heurísticas e (iii) uso das ideias de isomorfismo e homeomorfismo, pois estas não capturam o conceito de duplicação de genes que resultam em cópias de vértices. Consiste em um trabalho muito recente que promete realizar de maneira rápida o alinhamento de redes metabólicas sem as limitações das outras soluções consideradas.

Também seria conveniente o estudo do trabalho de Deniélou *et al.* [13] dedicado ao alinhamento múltiplo de redes metabólicas, tendo seus autores participado de diversas outras publicações nesse contexto [12, 31, 4, 42, 41, 40]. O documento em questão é recente e expõe um algoritmo exato que parece apresentar bons resultados.

Uma outra questão interessante a se considerar seria uma pesquisa bibliográfica acerca do uso de algoritmos de alinhamento múltiplo de sequências para o alinhamento múltiplo de vias metabólicas. Essa é uma ideia natural, uma vez que utilizamos o algoritmo de alinhamento global de duas sequências para o alinhamento de duas vias metabólicas. Contudo, nenhum dos autores estudados considerou essa possibilidade.

Existem inúmeras formulações dos problemas de alinhamento de duas ou várias vias ou redes metabólicas. A definição de alinhamento de duas vias não ramificadas de Tohsato *et al.* permitiu que se construísse um algoritmo polinomial baseado no alinhamento de duas sequências. Já pela definição de alinhamento de redes de Yang e Sze temos um algoritmo exponencial e não sabemos em que classe o problema se encontra. Dessa forma, a classificação dos problemas depende de responder "o que é um alinhamento?", e essa resposta não é única como no alinhamento de sequências, mas depende de diversos fatores que variam de acordo com o problema, como, por exemplo, a estrutura utilizada para representar as vias ou redes e se os alinhamentos podem conter remoções, espaços e diferenças. Uma pesquisa sobre essa questão seria um tópico a ser considerado em um trabalho futuro.

Lista de Algoritmos

Lista de Figuras

Referências Bibliográficas

[1] V. Bafna, E. L. Lawler, and P. A. Pevzner. Approximation algorithms for multiple sequence alignment. In *Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, CPM '94, pages 43–53, London, UK, 1994. Springer-Verlag. 3.1.3

[2] J. Berg and M. Lässig. Cross-species analysis of biological networks by bayesian alignment. *Proceedings of the National Academy of Sciences*, 103(29):10967–10972, 2006. 1, 5

[3] B. Bollobás. *Modern Graph Theory*. Springer, corrected edition, July 1998. 2.1

[4] F. Boyer, A. Morgat, L. Labarre, J. Pothier, and A. Viari. Syntons, metabolons and interactons: an exact graph-theoretical approach for exploring neighbourhood between genomic and functional data. *Bioinformatics*, 21(23):4209–4215, 2005. 1, 5

[5] R. T. de Brito. Alinhamento de sequências biológicas. Master's thesis, Universidade de São Paulo, 2003. Instituto de Matemática e Estatística. 3.2

[6] M. Chen and R. Hofestädt. Pathaligner: metabolic pathway retrieval and alignment. *Applied Bioinformatics*, 3(4):241–252, 2004. 1, 5

[7] Q. Cheng. *Metabolic Network Alignments and their Applications*. PhD thesis, Georgia State University, 2009. *Computer Science Dissertations*. Paper 44. http://digitalarchive.gsu.edu/cs_diss/44. 1

[8] Q. Cheng, P. Berman, R. Harrison, and A. Zelikovsky. Fast alignments of metabolic networks. In *Bioinformatics and Biomedicine, 2008. BIBM '08. IEEE International Conference on*, pages 147–152, nov. 2008. 1, 5

[9] Q. Cheng, R. Harrison, and A. Zelikovsky. Homomorphisms of multisource trees into networks with applications to metabolic pathways. In *Bioinformatics and Bioengineering, 2007. BIBE 2007. Proceedings of the 7th IEEE International Conference on*, pages 350–357, oct. 2007. 1, 5

[10] J. C. Clemente, K. Satou, and G. Valiente. Reconstruction of phylogenetic relationships from metabolic pathways based on the enzyme hierarchy and the gene ontology. *Genome Informatics*, 16:2005, 2005. 3.3

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009. 3.1.1, 3.1.3, 4.3.1, 4.3.2

[12] Y.-P. Denielou, M.-F. Sagot, F. Boyer, and A. Viari. Bacterial syntenies: an exact approach with gene quorum. *BMC Bioinformatics*, 12(1):193, 2011. 5

[13] Y.-P. Deniélou, F. Boyer, A. Viari, and M.-F. Sagot. Multiple alignment of biological networks: A flexible approach. In *Combinatorial Pattern Matching*, volume 5577 of *Lecture Notes in Computer Science*, pages 263–273. Springer Berlin / Heidelberg, 2009. 1, 5

[14] Y. Deville, D. Gilbert, J. v. Helden, and S. J. Wodak. An overview of data models for the analysis of biochemical pathways. *Briefings in Bioinformatics*, 4:246–259, 2003. 2.2

[15] L. Duret and S. Abdeddaim. Multiple alignment for structural functional or phylogenetic analyses of homologous sequences. In D. Higgins and W. Taylor, editors, *Bioinformatics: sequence, structure, and databanks*, Practical approach series. Oxford University Press, 2000. 3.2

[16] D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2):652–673, 1998. 4.3.1

[17] D. A. Fell and A. Wagner. The small world of metabolism. *Nature Biotechnology*, 18(11):1121–1122, 2000. 2.2

[18] P. Feofiloff, Y. Kohayakawa, and Y. Wakabayashi. Uma introdução sucinta à teoria dos grafos. SBM (Sociedade Brasileira de Matemática), 2004 (livreto). 2.1

[19] J. Flannick, A. Novak, B. S. Srinivasan, H. H. McAdams, and S. Batzoglou. Græmlin: General and robust alignment of multiple large interaction networks. *Genome Research*, 16(9):1169–1181, 2006. 1, 5

[20] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen. Directed hypergraphs and applications. *Discrete Appl. Math.*, 42(2-3):177–201, 1993. 2.1

[21] D. Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bulletin of Mathematical Biology*, 55(1):141–154, 1993. 3.1.3

[22] D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology*. Cambridge Univ. Press, January 1997. 3

[23] H. Jeong, B. Tombor, R. Albert, Z. N. Oltvai, and A.-L. Barabasi. The large-scale organization of metabolic networks. *Nature*, 407(6804):651–654, 2000. 2.2

[24] W. Just. Computational complexity of multiple sequence alignment with sp-score. *Journal of Computational Biology*, 8(6):615–623, 2001. 3.2

[25] M. Kalaev, V. Bafna, and R. Sharan. Fast and accurate alignment of multiple protein networks. In *Proceedings of the 12th annual international conference on Research in computational molecular biology*, RECOMB'08, pages 246–256, Berlin, Heidelberg, 2008. Springer-Verlag. 1, 5

[26] B. P. Kelley, R. Sharan, R. M. Karp, T. Sittler, D. E. Root, B. R. Stockwell, and T. Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network alignment. *Proceedings of the National Academy of Sciences*, 100(20):11394–11399, 2003. 4.3.1

[27] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwell, and T. Ideker. PathBLAST: a tool for alignment of protein interaction networks. *Nucleic Acids Research*, 32(suppl 2):W83–W88, 2004. 3.3

[28] J. Kingston. *Algorithms and data structures: design, correctness, analysis.* International computer science series. Addison-Wesley, 1997. 4.3.1

[29] G. Klau. A new graph-based method for pairwise global network alignment. *BMC Bioinformatics*, 10(Suppl 1):S59, 2009. 1, 5

[30] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955. 4.2

[31] V. Lacroix, L. Cottret, P. Thebault, and M.-F. Sagot. An introduction to metabolic networks and their structural analysis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5:594–617, 2008. 1, 2.2, 5

[32] A. Lehninger, D. L. Nelson, and M. M. Cox. *Lehninger Principles of Biochemistry.* W. H. Freeman, 5th edition, June 2008. 2.1.1

[33] S. Light, P. Kraulis, and A. Elofsson. Preferential attachment in the evolution of metabolic networks. *BMC Genomics*, 6(1):159, 2005. 2.2

[34] H. Ma and A.-P. Zeng. Reconstruction of metabolic networks from genome data and analysis of their global structure for various organisms. *Bioinformatics*, 19(2):270–277, 2003. 2.2

[35] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443 – 453, 1970. 3.1.1

[36] C. Notredame. Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, 3(1):131–144, 2002. 3.2

[37] M. Parter, N. Kashtan, and U. Alon. Environmental variability and modularity of bacterial metabolic networks. *BMC Evolutionary Biology*, 7(1):169, 2007. 3.3

[38] R. Y. Pinter, O. Rokhlenko, D. Tsur, and M. Ziv-ukelson. Approximate labelled subtree homeomorphism. In *In Proceedings of 15th Annual Symposium of Combinatorial Pattern Matching*, pages 59–73. Springer-Verlag, 2004. 4.2

[39] R. Y. Pinter, O. Rokhlenko, E. Yeger-Lotem, and M. Ziv-Ukelson. Alignment of metabolic pathways. *Bioinformatics*, 21:3401–3408, August 2005. 1, 3.3, 4, 5

[40] M. Sagot and A. Viari. A double combinatorial approach to discovering patterns in biological sequences. In D. Hirschberg and G. Myers, editors, *Combinatorial Pattern Matching*, volume 1075 of *Lecture Notes in Computer Science*, pages 186–208. Springer Berlin / Heidelberg, 1996. 5

[41] M.-F. Sagot and A. Viari. Flexible identification of structural objects in nucleic acid sequences: Palindromes, mirror repeats, pseudoknots and triple helices. In A. Apostolico and J. Hein, editors, *Combinatorial Pattern Matching*, volume 1264 of *Lecture Notes in Computer Science*, pages 224–246. Springer Berlin / Heidelberg, 1997. 5

[42] M.-F. Sagot, A. Viari, and H. Soldano. Multiple sequence comparison: A peptide matching approach. In Z. Galil and E. Ukkonen, editors, *Combinatorial Pattern Matching*, volume 937 of *Lecture Notes in Computer Science*, pages 366–385. Springer Berlin / Heidelberg, 1995. 5

[43] J. Setubal and J. Meidanis. *Introduction to computational molecular biology*. PWS Publishing Company, 1997. 3

[44] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195 – 197, 1981. 3.1.2

[45] Y. Tohsato, H. Matsuda, and A. Hashimoto. A multiple alignment algorithm for metabolic pathway analysis using enzyme hierarchy. In *Proceedings of the International Conference in Intelligent Systems for Molecular Biology*, volume 8, pages 376–383, 2000. 1, 3.3, 4, 4.1, 4.1, 5

[46] K. Tun, P. Dhar, M. Palumbo, and A. Giuliani. Metabolic pathways variability and sequence/networks comparisons. *BMC Bioinformatics*, 7(1):24, 2006. 3.3

[47] I. M. Wallace, G. Blackshields, and D. G. Higgins. Multiple sequence alignments. *Current Opinion in Structural Biology*, 15(3):261 – 266, 2005. 3.2

[48] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994. 3.2

[49] Z. Wang, X.-G. Zhu, Y. Chen, Y. Li, J. Hou, Y. Li, and L. Liu. Exploring photosynthesis evolution by comparative analysis of metabolic networks between chloroplasts and photosynthetic bacteria. *BMC Genomics*, 7(1):100, 2006. 3.3

[50] E. Webb. *Enzyme Nomenclature 1992*. Published for the International Union of Biochemistry and Molecular Biology by Academic Press, San Diego, 1992. 3.3

[51] Q. Yang and S.-H. Sze. Path matching and graph matching in biological networks. *Journal of Computational Biology*, 14(1):56–67, 2007. 1, 4, 4.3, 5

[52] M. Zaslavskiy, F. Bach, and J.-P. Vert. Global alignment of protein–protein interaction networks by graph matching methods. *Bioinformatics*, 25(12):i259–1267, 2009. 1, 5

[53] D. Zhu and Z. Qin. Structural comparison of metabolic networks in selected single cell organisms. *BMC Bioinformatics*, 6(1):8, 2005. 3.3