Aprendizado Descritivo

Aula 07 – Mineração de grafos

Professor Renato Vimieiro

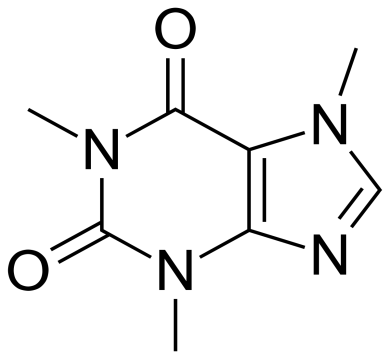
DCC/ICEx/UFMG

Introdução

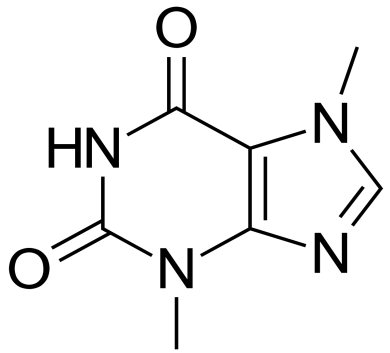
- Nessa aula, vamos discutir o problema de mineração de subgrafos frequentes em bases de dados de grafos
- A mineração de subgrafos frequentes tem ganhado destaque nos últimos anos com aplicações em diversas áreas:
  - Quimioinformática: descoberta de princípios ativos de fármacos
  - Bioinformática: padrões de interação em redes de proteínas
  - Engenharia de Software: análise do fluxo de controle de aplicações

Introdução

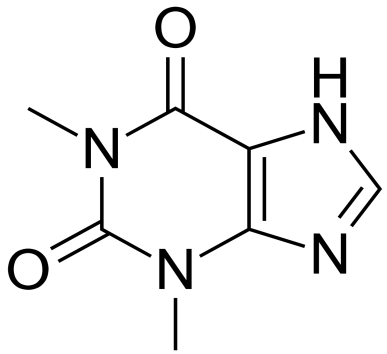
- Considerando o contexto de quimioinformática, temos o seguinte exemplo:



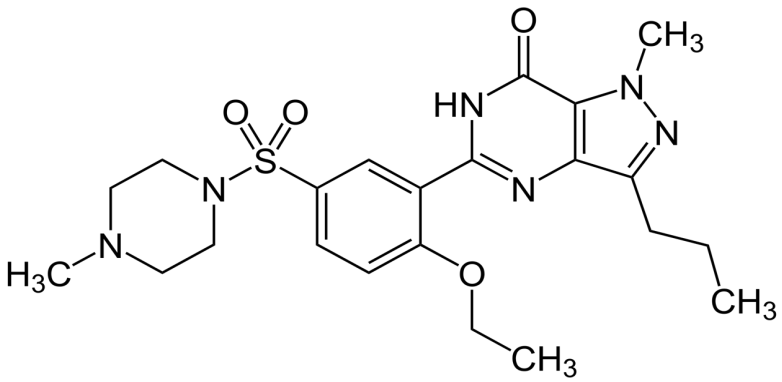
Cafeína



Teobromina



Teofilina



Sildenafil (Viagra)

Definições

- Um grafo é uma quadrupla (V, E, l, L) em que:
  - V é um conjunto de vértices;
  - E é um conjunto de arestas;
  - $l: V \rightarrow \Sigma_{\mathrm{V}}$ é uma função que determina o rótulo dos vértices
  - $L: E \rightarrow \Sigma_E$ é uma função que determina o rótulo das arestas

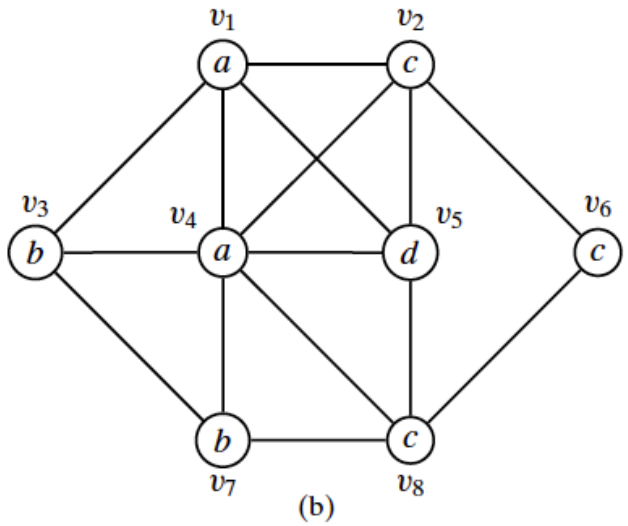
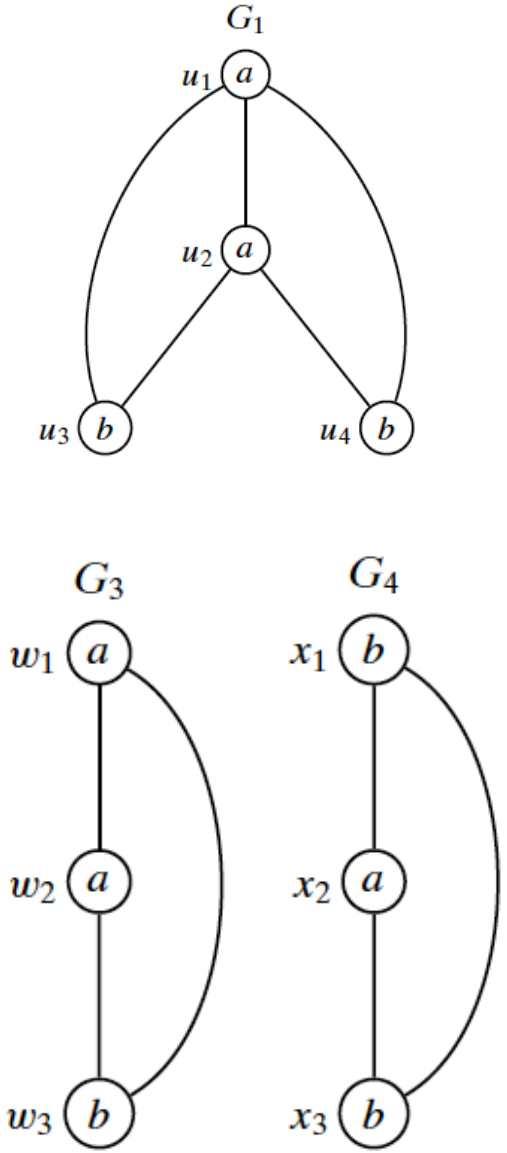


Figura retirada de Zaki e Meira: Figura 11.1b

Definições

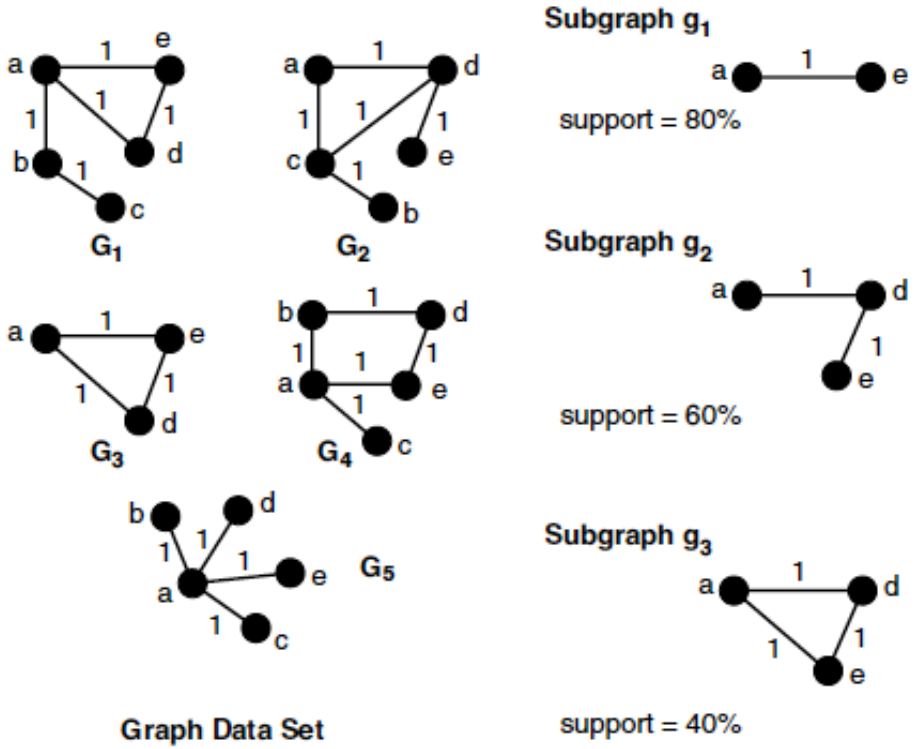
- Sejam G_1 e G_2 dois grafos definidos rotulados com os mesmos alfabetos Σ_V e Σ_E
- Dizemos que G_1 é **subgrafo isomorfo** a G_2, representado por $G_1 \subseteq G_2$, se existe uma função injetora $\varphi: V_1 \rightarrow V_2$ tal que:
  - $(u, v) \in E_1 \leftrightarrow \big(\varphi(u), \varphi(v)\big) \in E_2$
  - $\forall u \in V_1 [l(u) = l\big(\varphi(u)\big)]$
  - $\forall (u, v) \in E_1 \, [L(u, v) = L\big(\varphi(u), \varphi(v)\big)]$



Retirado de Zaki e Meira, Figura 11.3

Definições

- Uma base de dados $D = \{G_1, G_2, \ldots, G_n\}$ é um conjunto de grafos rotulados
- O suporte de um grafo (padrão) G é o tamanho do conjunto de grafos de D em que G está contido (é subgrafo isomorfo):
  - $\sup(G) = |\{G_i \in D | \, G \subseteq G_i\}|$
- Um padrão G é frequente se $\sup(G) \geq minsup$



Retirado de Figura 7.10 de Tan et al.

Definições

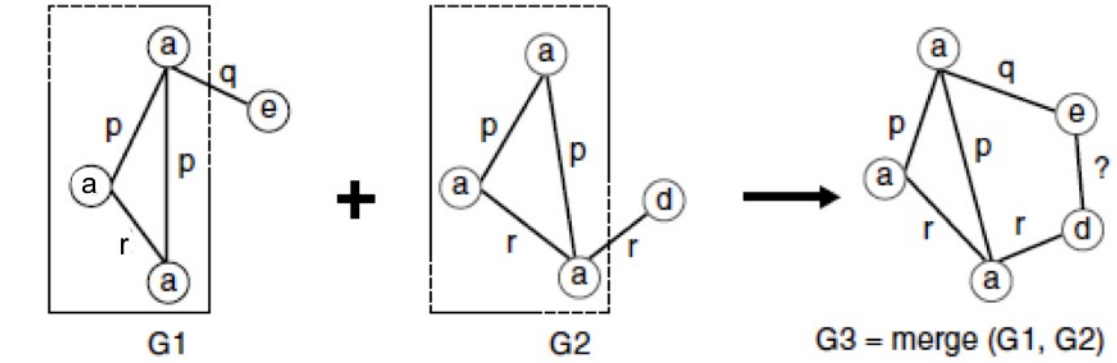
- O problema de mineração de subgrafos frequentes consiste em encontrar todos os **subgrafos conexos** que satisfaçam o suporte mínimo definido pelo usuário
- Comparado aos problemas vistos até agora, esse é o mais difícil em termos computacionais
  - O problema de isomorfismo é NP-difícil
- Um grafo com m vértices possui $O(m^2)$ arestas
- O número de subgrafos com m vértices é portanto $2^{O(m^2)}$
- Considerando grafos rotulados com $|\Sigma_V| = |\Sigma_E| = s$, s^m rotulações diferentes de vértices e s^{m^2} rotulações de arestas
- Logo, o espaço de busca do problema é $2^{O(m^2)}\left(s^{O(m)}\right)\left(s^{O(m^2)}\right)$

Mineração de subgrafos frequentes

- Assim como aconteceu com os problemas anteriores, existem duas categorias de algoritmos:
  - Baseados no ‘Apriori’ (geração de candidatos)
  - Baseados no ‘FP-Growth’ (crescimento de padrões)
- Os algoritmos baseados no Apriori se subdividem em mais grupos:
  - Os baseados no aumento do número de vértices; e
  - Os baseados no aumento do número de arestas.
- Os baseados em crescimento de padrões frequentemente utilizam extensões de arestas para crescê-los.

Geração de candidatos (AGM)

- A geração de candidatos com aumento do número de vértices é usada no algoritmo AGM (Apriori-based Graph Mining) proposto por Inokuchi et al. em 2000
- A ideia é juntar dois grafos com k vértices que compartilhem um núcleo com k-1 vértices para formar um candidato com k+1 vértices

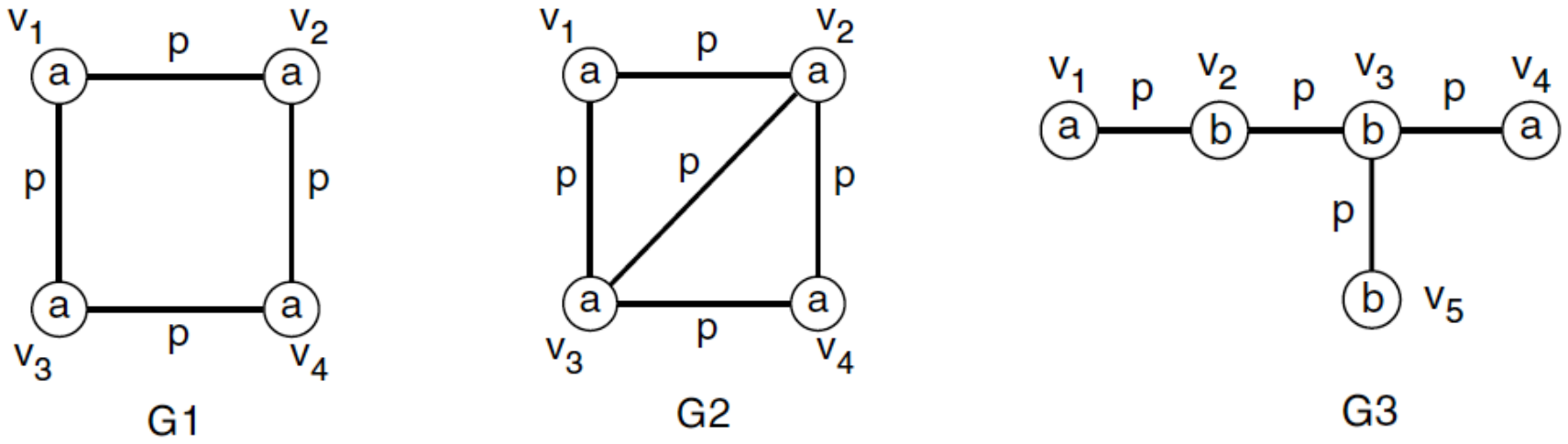


Geração de candidatos (FSG)

- A geração de candidatos com aumento no número de arestas é usada pelo algoritmo FSG, proposto por Kuramochi e Karypis em 2001.
- Dois padrões com k arestas são juntados se eles compartilham um núcleo (possuem um mesmo subgrafo) com k-1 arestas.
  - Ou seja, G_1 e G_2 são juntados se existe uma aresta em G_1 e uma aresta em G_2 tais que a remoção das duas torna os subgrafos isomorfos entre si
  - Consequentemente, o candidato terá as arestas de G_1 mais essa aresta de G_2
- Ao contrário do que ocorre com o método baseado em vértices, aqui o candidato resultante pode não ter um número maior de vértices que os padrões do qual foi gerado

Geração de candidatos (FSG)

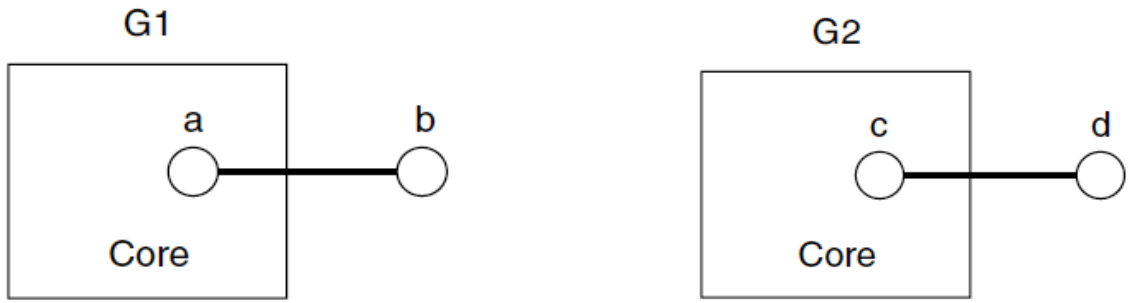
- Um conceito importante para a geração de candidatos é o de vértices topologicamente equivalentes
- Considere os grafos abaixo



Retirado da Figura 7.15 Tan et al.

Geração de candidatos (FSG)

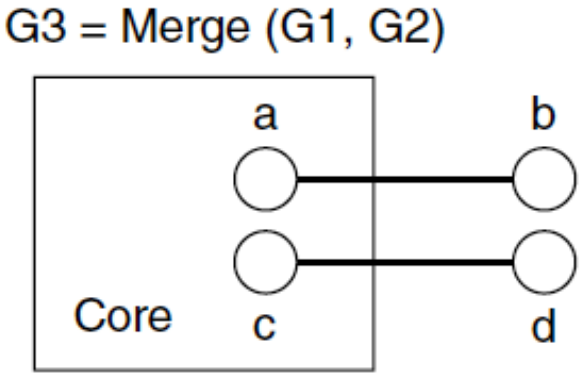
- Para entender o processo de geração de candidatos, vamos considerar dois grafos genéricos G1 e G2 com k arestas que compartilham um núcleo comum com k-1 arestas
  - Na figura, o núcleo é omitido e somente as arestas não compartilhadas são ilustradas; os vértices dentro da caixa fazem parte do núcleo.
  - Se a é topologicamente equivalente a c, dizemos que a=c
  - Se b e d possuem o mesmo rótulo, dizemos que b=d
- Temos 4 possíveis situações



Retirado da Figura 7.16 Tan et al.

Geração de candidatos (FSG)

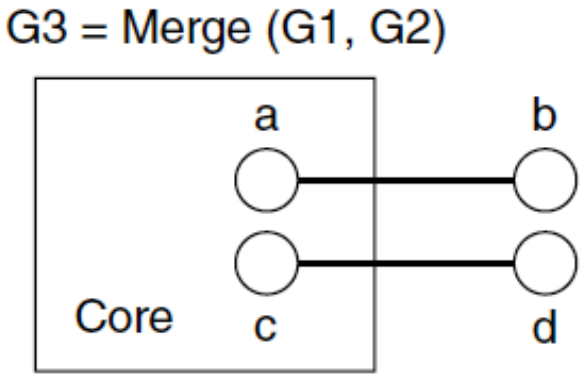
- Situação 1 [a≠c e b≠d]: Nesse caso, somente um candidato pode ser gerado, já que a ‘nova’ aresta incide em vértices distintos

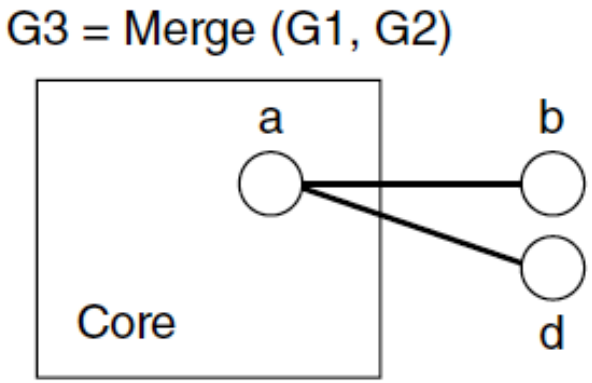


Retirado da Figura 7.17 Tan et al.

Geração de candidatos (FSG)

- Situação 2 [a=c e b≠d]: Nesse caso, temos duas possibilidades:
  - A ‘nova’ aresta incide sobre vértices distintos como na situação 1
  - A ‘nova’ aresta incide sobre o vértice equivalente a

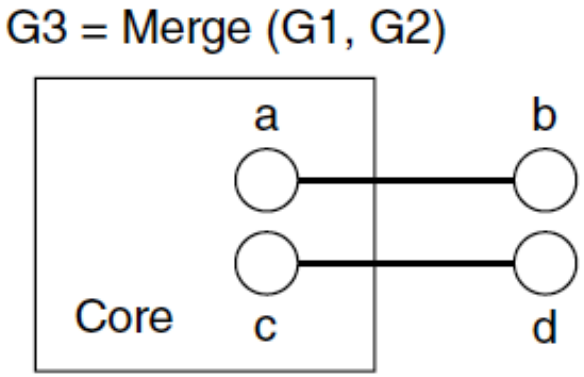


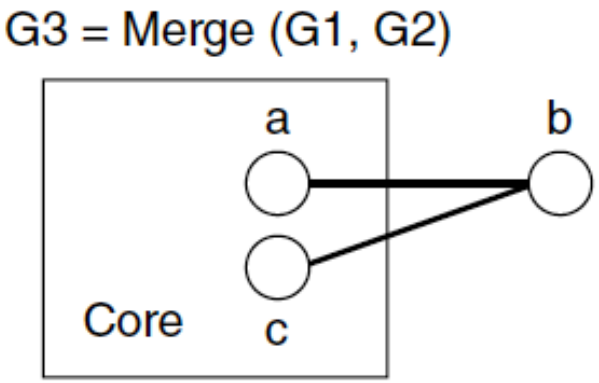


Retirado da Figura 7.17 Tan et al.

Geração de candidatos (FSG)

- Situação 3 [a≠c e b=d]: novamente temos duas possibilidades:
  - A ‘nova’ aresta incide sobre vértices distintos como na situação 1
  - A ‘nova’ aresta incide sobre o vértice equivalente b

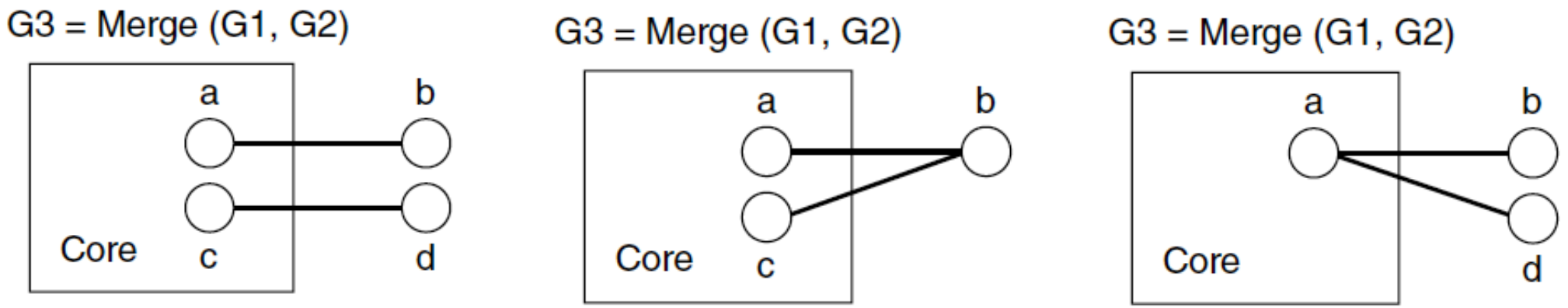




Retirado da Figura 7.17 Tan et al.

Geração de candidatos (FSG)

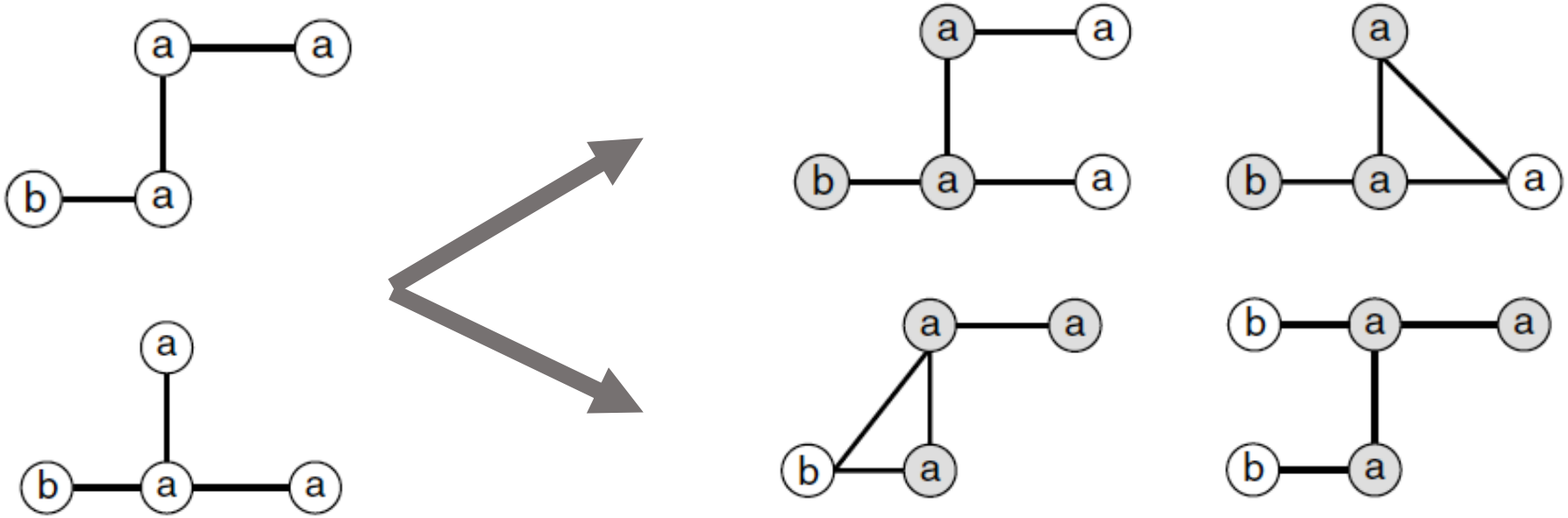
- Situação 4 [a=c e b=d]: Nesse caso, qualquer das possibilidades anteriores pode ocorrer; ou seja, pelo menos 3 candidatos são gerados.



Retirado da Figura 7.17 Tan et al.

Geração de candidatos (FSG)

- Além dessas situações, dois grafos podem compartilhar múltiplos núcleos. Cada um deles pode gerar candidatos distintos.
  - Podem existir até k-1 núcleos distintos para dois padrões de tamanho k



Situação 3

Retirado da Figura 7.18 Tan et al.

Geração de candidatos (FSG)

- Naturalmente, muitos desses candidatos serão isomorfos entre si, devendo, portanto, serem descartados exceto uma cópia
- O controle de isomorfismo é feito atribuindo-se um código (string) a partir de cada matriz de adjacências
- Grafos com mesmo código são isomorfos
- Os detalhes são omitidos, mas podem ser consultados no artigo original

Crescimento de padrões (gSpan)

- De maneira similar ao que acontece com itemsets, os algoritmos baseados em geração de candidatos possuem desempenho inferior aos baseados em crescimento de padrões, já que muitos candidatos são podados
- Em 2002, Yan e Han (FP-Growth) propuseram o algoritmo gSpan (Graph-based Substructure Pattern Mining)
- O algoritmo estende os padrões acrescentando arestas, uma por vez, de forma similar ao comportamento do FP-Growth
- Os autores também propuseram representar grafos por sequências de strings
  - Isso, de certa forma, mapeia o problema de mineração de grafos para mineração de sequências; levando ao aproveitamento de ideias propostas na última
- Cada aresta do grafo é representada pela string:
  - $\langle v_i, v_j, l(v_i), l(v_j), L(v_i, v_j) \rangle$

Crescimento de padrões (gSpan)

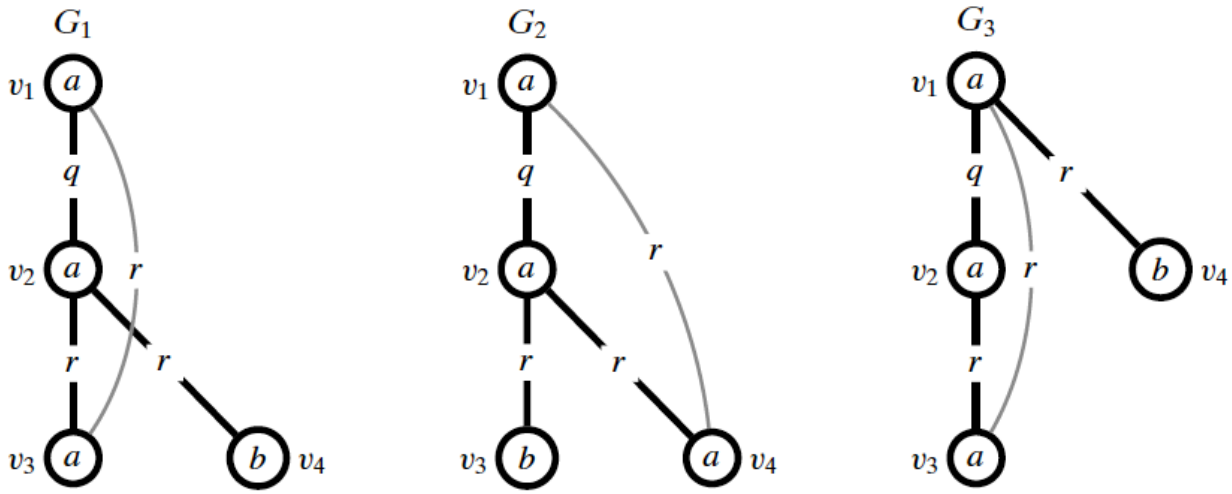
- A extensão dos padrões proposta por Yan e Han se baseia na árvore geradora da busca em profundidade
- Os vértices do grafo são ordenados conforme a ordem de visitação de uma busca em profundidade no grafo
  - Vértices visitados no início são menores que os visitados no fim
  - Subscritos dos vértices indicam a ordem de visitação
- O caminho mais curto entre o primeiro vértice e o último é chamado de **caminho mais à direita (rightmost path)**
- Considerando a busca em profundidade, as arestas são divididas em dois conjuntos:
  - **Fordward edges:** arestas que fazem parte da árvore geradora da DFS
  - **Backward edges:** demais arestas

Crescimento de padrões (gSpan)

- As arestas podem ser ordenadas dentro dos respectivos conjuntos da seguinte forma:
  - Forward edges: $e_1 = (v_{i_1}, v_{j_1}), e_2 = (v_{i_2}, v_{j_2}),\ e_1 \preccurlyeq_F e_2 \leftrightarrow j_1 < j_2$
  - Backward edges: $e_1 \preccurlyeq_B e_2 \leftrightarrow [i_1 < i_2 \vee (i_1 = i_2 \wedge j_1 < j_2)]$
- Adicionalmente, comparamos arestas dos conjuntos da seguinte forma:
  - $e_1 \preccurlyeq_{BF} e_2$ sse:
    - e_1 é uma aresta de retorno; e_2 é uma aresta de árvore; e $i_1 < j_2$
    - e_1 é uma aresta de árvore; e_2 é uma aresta de retorno; e $j_1 \leq i_2$
- Combinando a ordem dos vértices, com a ordem sobre as arestas e a ordem lexicográfica definida sobre os rótulos (de vértices e arestas), definimos uma ordem total sobre as arestas (códigos)
- Dessa forma, os grafos podem ser representados como sequências de strings (códigos) referentes às suas arestas

Crescimento de padrões (gSpan)

- Exemplo:



$t_{11} = \langle v_1, v_2, a, a, q \rangle$
$t_{12} = \langle v_2, v_3, a, a, r \rangle$
$t_{13} = \langle v_3, v_1, a, a, r \rangle$
$t_{14} = \langle v_2, v_4, a, b, r \rangle$

DFScode(G_1)

$t_{21} = \langle v_1, v_2, a, a, q \rangle$
$t_{22} = \langle v_2, v_3, a, b, r \rangle$
$t_{23} = \langle v_2, v_4, a, a, r \rangle$
$t_{24} = \langle v_4, v_1, a, a, r \rangle$

DFScode(G_2)

$t_{31} = \langle v_1, v_2, a, a, q \rangle$
$t_{32} = \langle v_2, v_3, a, a, r \rangle$
$t_{33} = \langle v_3, v_1, a, a, r \rangle$
$t_{34} = \langle v_1, v_4, a, b, r \rangle$

DFScode(G_3)

Retirado da Figura 11.6 Zaki e Meira

Crescimento de padrões (gSpan)

- Note que diferentes buscas geram diferentes representações
- Podemos estender a ordem lexicográfica das arestas para as representações (grafos)
- Chamamos de **representação canônica** a menor dentre todas as representações de um grafo
- Dois grafos são isomorfos se possuem a mesma representação canônica
- Assim, o problema de encontrar subgrafos frequentes é equivalente a enumerar suas representações canônicas
  - Uma vez que as representações são sequências, o problema é essencialmente um problema de mineração de sequências

Crescimento de padrões (gSpan)

- O gSpan explora em profundidade o espaço de busca estendendo prefixos (subgrafos) com arestas no caminho mais à direita
  - O artigo original demonstra que representações canônicas só podem ser geradas por arestas seguindo o caminho mais à direita
- Ele inicia com o prefixo vazio e o estende com cada uma das arestas em ordem
  - As possíveis extensões são determinadas a partir dos caminhos mais à direita dos grafos em que o prefixo ocorre
- O suporte das extensões são computados e, caso sejam frequentes e canônicas, são exploradas recursivamente

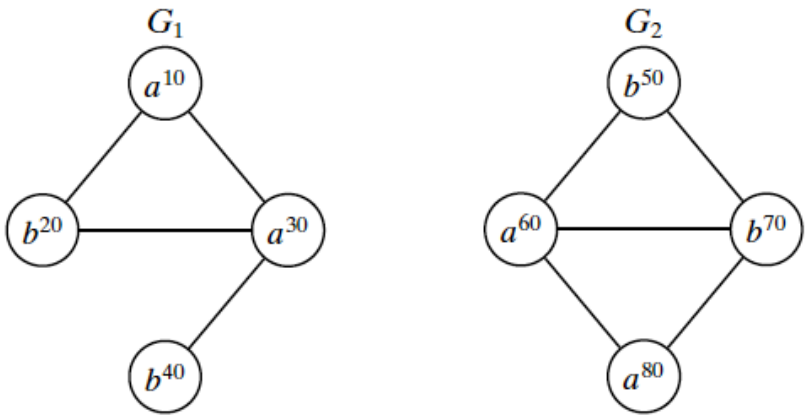
Crescimento de padrões (gSpan)

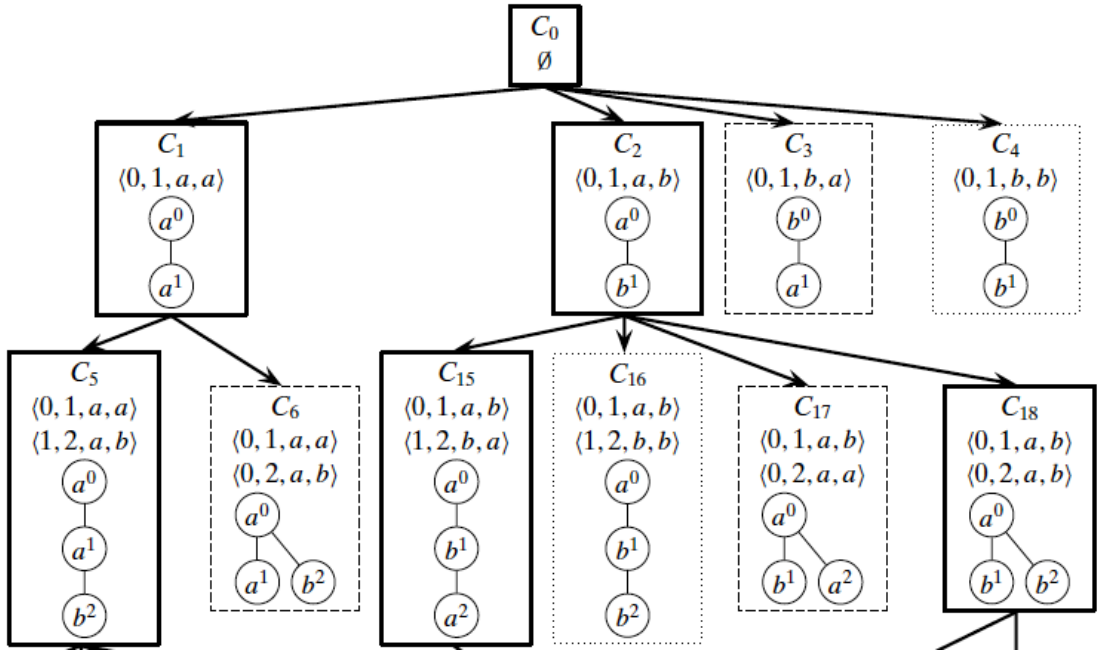
ALGORITHM 11.1. Algorithm gSpan

```
   // Initial Call: C ← ∅
   gSpan (C, D, minsup):
1  E ← RightMostPath-Extensions(C, D) // extensions and
       supports
2  foreach (t, sup(t)) ∈ E do
3      C' ← C ∪ t // extend the code with extended edge tuple t
4      sup(C') ← sup(t) // record the support of new extension
       // recursively call gSpan if code is frequent and
          canonical
5      if sup(C') ≥ minsup and IsCanonical (C') then
6          gSpan (C', D, minsup)
```

Crescimento de padrões (gSpan)

- Exemplo: D= {G1, G2}, minsup=2





Extensões do problema

- Assim como ocorre com sequências e itemsets, o número de subgrafos frequentes pode ser bastante elevado em uma base
  - Isso acaba dificultando a análise dos resultados
- Existem abordagens, como o CloseGraph proposto por Yan e Han, que mineram padrões fechados
- Pode-se também buscar padrões contrastantes: com suporte mínimo em um subconjunto e máximo em outro
  - Exemplo: subgrafos que ocorrem com frequência no conjunto de moléculas que potencializam um efeito desejado; e que sejam infrequentes no conjunto de moléculas que apresentem efeitos adversos
  - Essa abordagem foi sugerida por Borgelt e Berthold no algoritmo chamado MoFa
  - Suporte mínimo é usado para podar o espaço de busca, enquanto o máximo usado para filtrar padrões indesejados
- Na linha de algoritmos, existem abordagens heurísticas para o problema, dado seu alto custo computacional
- Outras linhas de pesquisa incluem uso de abordagens distribuídas como o Fractal proposto por Dias et al. em 2019

Leitura

- Capítulo 11 Zaki e Meira
- Seção 7.5 Tan et al.
- Capítulo 5 Mining Graph Data, Diane Cook e Lawrence Holder
- Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. 2000. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD '00). Springer-Verlag, Berlin, Heidelberg, 13–23.
- Michihiro Kuramochi and George Karypis. 2001. Frequent Subgraph Discovery. In Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM '01). IEEE Computer Society, USA, 313–320.
- Xifeng Yan and Jiawei Han. 2002. GSpan: Graph-Based Substructure Pattern Mining. In Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM '02). IEEE Computer Society, USA, 721.
- *Xifeng Yan and Jiawei Han. 2003. CloseGraph: mining closed frequent graph patterns. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '03). Association for Computing Machinery, New York, NY, USA, 286–295. https://doi.org/10.1145/956750.956784*
- Christian Borgelt and Michael R. Berthold. 2002. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. In <i>Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM '02). IEEE Computer Society, USA, 51.

Aprendizado Descritivo

Aula 07 – Mineração de grafos

Professor Renato Vimieiro

DCC/ICEx/UFMG