Geração uniforme de *k-trees* para aprendizado de redes bayesianas

Tiago Madeira (Supervisor: Prof. Dr. Denis Deratani Mauá)

Trabalho de Formatura Supervisionado do Bacharelado em Ciência da Computação no Instituto de Matemática e Estatística (IME), Universidade de São Paulo (USP)

Por que estudar *k-trees*?

Muitos problemas NP-difíceis são resolvidos em tempo polinomial em *k-trees* e subgrafos de *k-trees*. Exemplos [1]:

- Encontrar tamanho máximo dos conjuntos independentes;
- Calcular número cromático;
- Determinar se tem um ciclo hamiltoniano.

Por que gerar *k-trees?*

Há muitas razões, como por exemplo para testar a eficácia de algoritmos aproximados. O problema que desperta nosso interesse é o **aprendizado de redes bayesianas**.

O que foi feito?

- Implementação do algoritmo de Caminiti *et al.* (2010) [2] para **codificar *k*-trees de forma bijetiva em tempo linear**.
- Implementação de algoritmo para **amostrar *k*-trees uniformemente** e testes para comprovar seu funcionamento.
- Estudo sobre **aprendizado de redes bayesianas com treewidth limitado** por meio da amostragem uniforme de *k-trees* conforme artigo de Nie *et al.* (2014) [6].
- **Comparação deste método** para aprender redes bayesianas com o estado da arte.

O que são *k-trees*?

Uma ***k*-tree** é definida da seguinte forma recursiva [3]:

- Um grafo completo com k vértices é uma *k-tree*.
- Se $T'_k = (V, E)$ é uma *k-tree*, $K \subseteq V$ é um k-clique e $v \notin V$, então $T_k = (V \cup \{v\}, E \cup \{(v, x) \mid x \in K\})$ é uma *k-tree*.

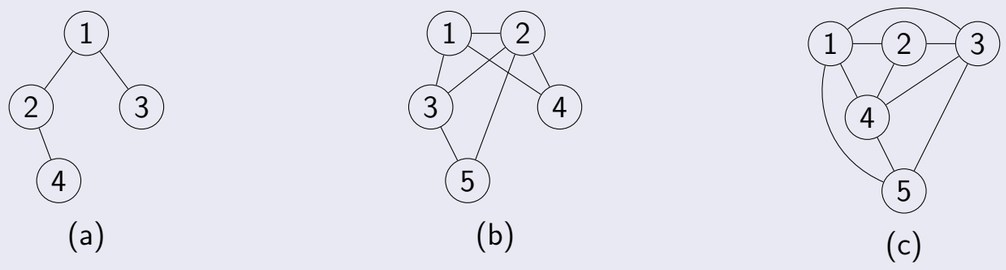

Figura: **(a)** Uma 1-*tree* (ou seja, uma árvore comum) com 4 vértices. **(b)** Uma 2-*tree* com 5 vértices. **(c)** Uma 3-*tree* com 5 vértices.

Relação entre geração e codificação

O problema de **gerar** *k-trees* está intimamente relacionado ao problema de **codificá-las e decodificá-las**: Se há uma codificação bijetiva que associa *k-trees* a *strings*, basta gerar *strings* uniformemente aleatórias para gerar *k-trees* uniformemente aleatórias.

Codificação de *k-trees*

Apenas **em 2008 surgiu um código bijetivo para *k*-trees com algoritmos lineares de codificação e decodificação**. Esses algoritmos, propostos por Caminiti *et al.*, foram implementados neste trabalho. O código é formado por uma permutação de tamanho k e uma generalização do *Dandelion Code* [8]. A codificação das *k-trees* associa elementos em $\mathcal{T}_k^n$ (conjunto das *k-trees* com n vértices) com elementos em:

$$\mathcal{A}_k^n = \binom{[1,n]}{k} \times (\{(0,\varepsilon)\} \cup ([1,n-k] \times [1,k]))^{n-k-2}$$

O processo consiste numa série de transformações:

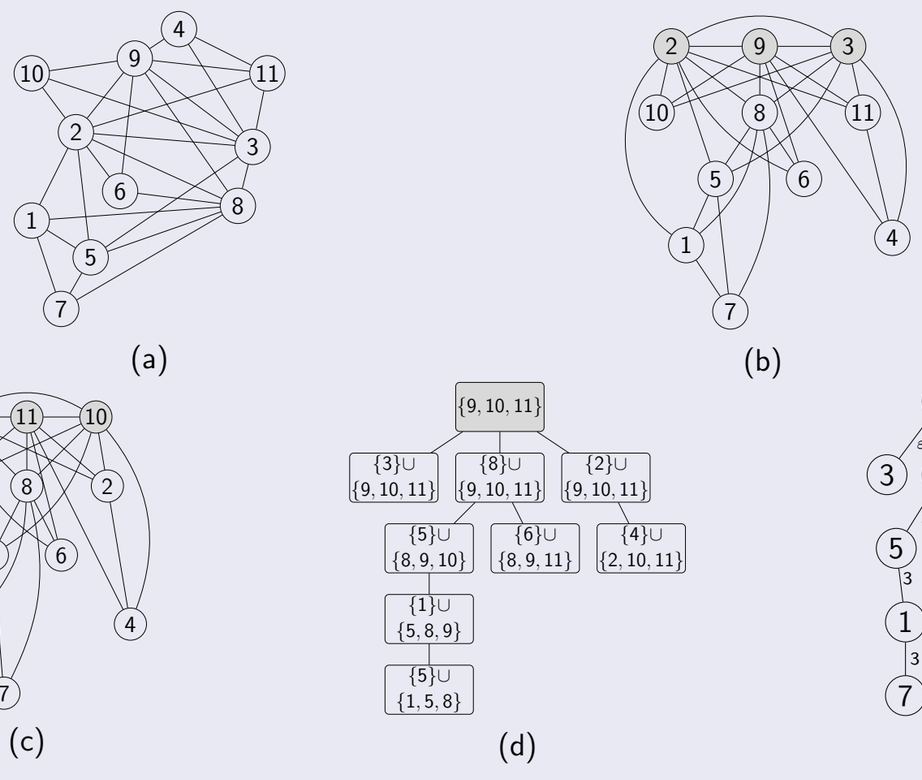

Figura: **(a)** Uma 3-*tree* T_3 com 11 vértices. **(b)** A mesma 3-*tree* (T_3) enraizada no 3-clique $\{2, 3, 9\}$. **(c)** R_3, 3-*tree* de Rényi gerada por meio da re-rotulação de T_3. **(d)** O esqueleto de R_3. **(e)** A árvore característica de R_3.

Dandelion Code generalizado correspondente a T_3, computado a partir da árvore característica de R_3: $\{2, 3, 9\}, [(0, \varepsilon), (2, 0), (8, 2), (8, 1), (1, 2), (5, 2)]$.

Geração uniforme de *k-trees*

Geramos *k-trees* uniformemente por meio da geração uniforme de *strings* em $\mathcal{A}_k^n$. A biblioteca que desenvolvemos foi implementada em *Go* e tem três utilitários que rodam na linha de comando:

- `code_ktree` (para codificar *k-trees*)
- `decode_ktree` (para decodificar *Dandelion Codes*)
- `generate_ktree` (para gerar uma *k-tree* uniformemente)

Aprendizado de redes bayesianas

- Aprender uma rede bayesiana se refere ao processo de **produzir a sua estrutura** (i.e., seu DAG) a partir de dados.
- **Inferência em rede bayesiana é NP-difícil até mesmo aproximadamente** e todos os algoritmos conhecidos têm complexidade no pior caso exponencial no *treewidth*.
- Resultados empíricos sugerem que **limitar o treewidth pode melhorar a performance** dos modelos e não causa perdas significativas na sua expressividade.
- Por isso estamos interessados em fixar k e aprender redes bayesianas cujo **DAG tem treewidth limitado a k**.
- **Função de score** $s(G)$ atribui pontuação para cada DAG G e pode ser escrita como soma de funções de *score* locais: $s(G) = \sum_{i \in N} s_i(X_{\pi_i})$.
- Para cada variável, sua **pontuação só depende do seu conjunto de pais**. Portanto, nosso problema é encontrar G^* tal que $G^* = \arg\max_{G \in \mathcal{G}_{n,k}} \sum_{i \in N} s_i(\pi_i)$ onde $\mathcal{G}_{n,k}$ é o conjunto de todos os DAGs de *treewidth* não maior que k. Esse problema é NP-difícil [4].

Aprendizado por amostragem de *k-trees*

A ideia para aprender um DAG por meio da amostragem de *k-trees* baseia-se em, para cada *k-tree* T_k amostrada, construir uma ordem parcial σ dos vértices e fazer com que o DAG G seja consistente com ela e com T_k. Construímos a ordem parcial σ a partir do enraizamento da *k-tree* num k-clique qualquer. Em particular, podemos escolher usar a raiz da *k-tree* de Rényi que aparece durante a decodificação de um *Dandelion Code* em uma *k-tree*.

Algoritmo para aprender estrutura

Entrada: n, k e função de *score* s_i para cada $i \in [0, n)$
Saída: um DAG G^{melhor}

1. Inicializar G^{melhor} como um grafo com $s(G^{\text{melhor}}) = -\infty$.
2. Repetir até atingir um determinado número de iterações:
   1. Gerar $(Q, S) \in \mathcal{A}_k^n$ e decodificar na árvore característica T;
   2. Sortear ordem pros vértices do k-clique raiz e usar função de *score* para calcular os melhores pais para cada um deles;
   3. Percorrer T a partir dos vértices ligados ao k-clique raiz: para cada v, é sorteado um lugar para ele em σ e selecionado seu melhor conjunto de pais dentre os vértices predecessores adjacentes, assim como são atualizados os melhores pais dos vértices sucessores adjacentes;
   4. Se $\left(\sum_{i \in [0,n)} s_i(\pi_i^G)\right) = s(G) > s(G^{\text{melhor}})$, atualiza $G^{\text{melhor}} = G$.

Experimentos

- Algoritmo foi **implementado por João de Santana Brito Junior**, aluno de mestrado do Prof. Denis. Faz uso da biblioteca desenvolvida neste trabalho para amostrar *k-trees*.
- Usamos implementação para testar aprendizado por amostragem uniforme de *k-trees* com **cinco conjuntos de dados reais** contendo de 64 a 1556 variáveis.
- **Comparamos resultados** com os obtidos por Perez e Mauá [7] para o *Acyclic Selection Order-Based Search (ASOBS)* com *Best First-Based initialization heuristic (BFT)*, abordagem com resultados comparáveis ao estado da arte.

	$k = 4$		$k = 10$		Perez e Mauá	
Dataset	Máx.	Média	Máx.	Média	Máx.	Média
kdd ($n = 64$)	0,0755	0,0691 ± 0,0022	0,1037	0,1007 ± 0,0016	0,1468	0,1447 ± 0,0007
tretail ($n = 135$)	0,0188	0,0156 ± 0,0019	0,0289	0,0249 ± 0,0017	0,0447	0,0444 ± 0,0002
cr52 ($n = 889$)	0,0203	0,0171 ± 0,0011	0,0439	0,0333 ± 0,0028	0,1617	0,1598 ± 0,0010
bbc ($n = 1058$)	0,0059	0,0054 ± 0,0002	0,0128	0,0102 ± 0,0005	0,0777	0,0770 ± 0,0004
ad ($n = 1556$)	0,0448	0,0400 ± 0,0023	0,0905	0,0781 ± 0,0045	0,7114	0,7089 ± 0,0013

Tabela: Performance do aprendizado de redes bayesianas por amostragem de *k-trees* e comparação dele com os resultados obtidos por Perez e Mauá.

Considerações finais

- **Principal produto do trabalho:** biblioteca para codificação e geração uniforme de *k-trees* em tempo linear. Experimentos no aprendizado de redes bayesianas mostram como pode ser usada na prática. Método é eficiente e funciona com grandes domínios e *treewidth* alto.
- Muitos dos **conceitos estudados são recentes e pouco explorados**. Os dois principais artigos (Caminiti *et al.* e Nie *et al.*) foram publicados respec. em 2010 e 2014. Outros trabalhos intimamente relacionados foram publicados nos últimos anos.
- Extensão interessante deste trabalho seria estudar a **geração de DAGs de treewidth limitado de maneira uniforme**. A geração uniforme de *k-trees* não resolve esse problema.
- Há **outras amostragens (não uniformes)** que podem ser testadas e comparadas em trabalhos futuros. Outro artigo de Nie *et al.* (2015) [5] sugere um *Distance Preferable Sampling*.

Mais informações

Este pôster, a monografia e os códigos desenvolvidos estão disponíveis em `https://github.com/tmadeira/tcc/`

Referências

[1] Stefan Arnborg and Andrzej Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23:11–24, 1989.

[2] Saverio Caminiti, Emanuele G. Fusco, and Rossella Petreschi. Bijective linear time coding and decoding for *k*-trees. *Theory of Computing Systems*, 46:284–300, 2010.

[3] Frank Harary and Edgar M. Palmer. On acyclic simplicial complexes. *Mathematika*, 15:115–122, 1968.

[4] Janne H. Korhonen and Pekka Parviainen. Exact learning of bounded tree-width bayesian networks. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2013, Scottsdale, AZ, USA, April 29 - May 1, 2013*, volume 31 of *JMLR Workshop and Conference Proceedings*, pages 370–378. JMLR.org, 2013.

[5] Siqi Nie, Cassio P. de Campos, and Qiang Ji. *Learning Bounded Tree-Width Bayesian Networks via Sampling*, pages 387–396. Springer International Publishing, Cham, 2015.

[6] Siqi Nie, Denis Deratani Mauá, Cassio Polpo de Campos, and Qiang Ji. Advances in learning bayesian networks of bounded treewidth. *CoRR*, abs/1406.1411, 2014.

[7] Walter Perez and Denis Deratani Mauá. Improving acyclic selection order-based bayesian network structure learning. In *XIII Encontro Nacional de Inteligência Artificial e Computacional (ENIAC)*, pages 169–180, 2016.

[8] Ömer Eğecioğlu and J. B. Remmel. Bijections for cayley trees, spanning trees, and their q-analogues. *Journal of Combinatorial Theory*, 42:15–30, 1986.