Relatório
MCMC em Programas Lógicos Probabilísticos

Rodrigo Azevedo

Junho 2018

1 Introdução

O seguinte trabalho aborda o problema de inferência e aprendizado de estruturas em programas lógicos probabilísticos, extendendo o cenário comum determinístico. O aprendizado de estruturas em um cenário probabilístico busca aprender regras através de fatos e exemplos que são probabilísticos, ou seja, possuem probabilidades ou incertezas associadas. O primeiro problema consiste em realizar a inferência de forma eficiente no cenário probabilístico, em que consultas possuem uma certa probabilidade de serem provadas. O segundo utiliza a inferência probabilística e a técnica de MCMC para aprender regras através de fatos probabilísticos.

2 Programas lógicos probabilísticos

Um programa lógico probabilístico é constituído por um conjunto de fatos rotulados com probabilidades $p_i :: c_i$ e um conjunto de cláusulas definidas. Cada instância ground de um fato c_i, em outras palavras uma instância que não possui uma variável, pode ser verdadeira com uma probabilidade p_i. Portanto, cada instância corresponde a uma variável aleatória. Assume-se que são variáveis mutuamente independentes. O conjunto de cláusulas definidas (regras) constituem o *background knowledge*.

Dado um programa $T = \{p_1 :: c_1, ..., p_n :: c_n\} \cup BK$ e um conjunto de possíveis substituições $\{\theta_{i1}, ..., \theta_{ji_j}\}$ para cada fato, seja L_T o conjunto máximo de fatos lógicos obtidos com as substituições e assumindo que as variáveis aleatórias que correspondem aos fatos são mutuamente independente, a probabilidade de um $L \subseteq L_T$ é dada por :

$$P(L|T) = \prod_{c_i\theta_j \in L} p_i \prod_{c_i\theta_j \in L_T \setminus L} (1 - p_i)$$

Temos, portanto, a probabilidade de um subprograma L (com fatos verdadeiros ou falsos) dado um programa lógico probabilístico T.
Em um programa lógico determinístico uma consulta q pode ser provada ou não

dado um programa, no seu caso probabilístico, a consulta pode ser provada com uma certa probabilidade $P_s(q|T)$. Portanto, pode-se perguntar a probabilidade de existir um caminho do nó c ao d como mostrado no exemplo da Figura 1.
A probabilidade de sucesso $P_s(q|T)$ de uma consulta q em um programa lógico probabilístico T é dado por:

$$P_s(q|T) = \sum_{L \subseteq L_T} P(q|L) \cdot P(L|T)$$

onde $P(q|L) = 1$ se há um θ tal que $L \cup BK \models q\theta$, e $P(q|L) = 0$ caso contrário. A probabilidade de sucesso da consulta q é a probabilidade dela ser provada em uma amostra aleatória de um programa lógico probabilístico T. De um subprograma L amostrado aleatóriamente de T, apenas os que provam q de acordo com a função indicadora $P(q|L)$ contribuem para a probabilidade de sucesso da consulta. Listar todos os possíveis subprogramas é inviável tendo em vista que o número de subprogramas utilizados na prova cresce exponencialmente com o número de fatos probabilísticos.

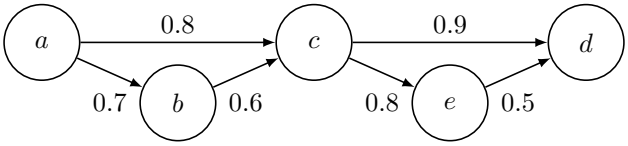

Figura 1: Exemplo de grafo probabilístico apresentado no artigo *On the implementation of the probabilistic logic programming language ProbLog* [2]. Cada rótulo na aresta indica a probabilidade dela fazer parte do grafo. O programa probabilístico é representado por $0.8 :: edge(a, c)$. e assim por diante, enquanto o BK é composto pelas regras $path(X, Y) : -edge(X, Y)$. e $path(X, Y) : -edge(X, Z), path(Z, Y)$.. Para uma consulta $path(c, d)$ tem-se 64 possíveis subprogramas em que apenas 40 provam a consulta. A probabilidade da consulta é a soma de todas as probabilidades desses subprogramas que provam a consulta, portanto $P(path(c, d)|T) = 0.94$.

3 Inferência

Obter a proabilidade de sucesso de uma determinada consulta calculando a probabilidade de cada possível subprograma é inviável pelo crescimento exponencial dado pelo número de fatos. Ademais, obter a probabilidade de uma consulta pela fórmula DNF obtida pela resolução-SDL é um problema #P-completo conhecido como *disjoint-sum-problem* [7].
Considerando cada fato probabilístico como uma variável aleatória booleana, um método alternativo é o Monte Carlo que utiliza amostragens aleatórias massivas de uma determinada distribuição. O método foi implementado da seguinte forma:

1. Amostra aleatoriamente N subprogramas de T

2. Checa a existência de uma prova para uma consulta q em cada subprograma amostrado (1 caso exista, 0 caso contrário)

3. Estima a probabilidade P da consulta q pela média amostral (fração de amostras em que a consulta q é provada)

A convergência é estimada computando o intervalo de confiança de 95% em m amostras utilizando a aproximação da distribuição binomial pela normal [1]:

$$\delta \approx 2 \times \sqrt{\frac{P \cdot (1 - P)}{N}}$$

O algoritmo utiliza então como critério de parada um intervalo de confiança máximo δ_p e gera m amostras por iteração até o intervalo de confiança for alcançado [2].

A primeira abordagem apresentada amostra N subprogramas aleatoriamente e checa a existência de uma prova para cada amostra. Embora funcione muito bem para pequenos bancos de fatos, o mesmo não acontece para casos com mais fatos disponíveis. Gerar uma amostra de um subprograma significa amostrar variáveis aleatórias booleanas para milhares ou milhões de fatos probabilísticos. Porém, muitos dos fatos probabilísticos não são usados na prova das consultas. Muitas das consultas necessitam apenas de uma pequena parcela dos fatos para a prova, o que indica um gasto desnecessário de amostragem de variáveis aleatórias booleanas não utilizadas na prova. Uma segunda abordagem aproveita a resolução-SLD para verificar se o fato está na amostra, ou seja, amostrar a variável aleatória booleana do fato, apenas quando for necessário o seu uso na prova. Dessa forma, os subprogramas são amostrados diretamente na resolução-SLD da seguinte forma:

1. Inicia a resolução-SLD de uma determinada consulta q

2. Realiza a amostra da variável aleatória booleana respectiva a um fato probabilístico assim que for requisitado na resolução-SLD

3. Estima a probabilidade P da consulta q pela média amostral (fração de amostras em que a consulta q é provada)

O trabalho manteve o uso da segunda abordagem nos experimentos depois de avaliar um ganho significativo do desempenho.

[1] Usando a aproximação normal, a probabilidade é estimada com $p \pm z \times \sqrt{\frac{p(1-p)}{n}}$. Para uma confiança de 95% temos $\alpha = 1 - 0.95 = 0.05$, $1 - \alpha/2 = 0.975$ e $z = 1.96$.

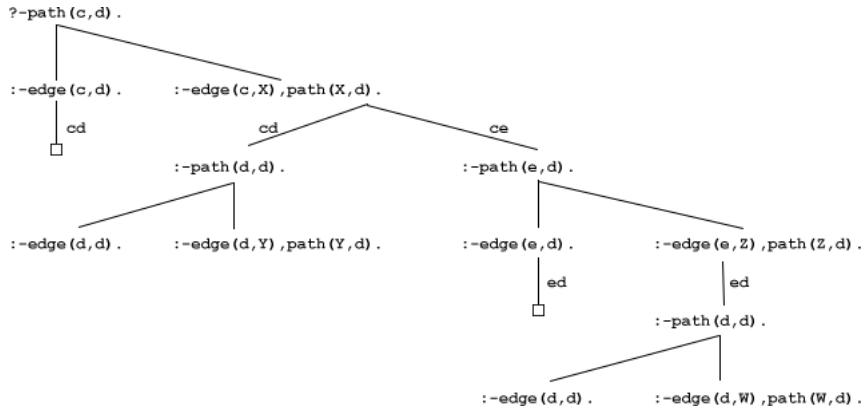

Figura 2: Resolução-SLD para a consulta $path(c, d)$ do exemplo do grafo probabilístico da Figura 1. A segunda abordagem de amostragem de subprogramas amostra as variáveis aleatórias booleanas dos fatos no decorrer da resolução-SLD.

4 Aprendizado de estruturas

Um problema clássico em ILP é o aprendizado de regras, que busca cláusulas definidas que provem os exemplos dado fatos disponíveis. Dois algoritmos clássicos como Progol [3] e FOIL [5] abordam este problema. Uma abordagem probabilística foca em um cenário probabilístico dos programas lógicos e relações de modo que fatos e exemplos sejam probabilísticos, isto é, atomos nos fatos e exemplos do predicado alvo possuam uma probabilidade. O algoritmo de aprendizado de regras probabilísticas ProbFOIL+ aborda um cenário similar e combina princípios do algoritmo FOIL com o sistema ProbLog [1].
Para o aprendizado de estruturas a partir de fatos probabilísticos é proposto o uso do método MCMC para efetuar a busca no espaço de cláusulas definidas que melhor expliquem um predicado alvo. A cadeia é modela da seguinte forma:

1. Um estado representa o corpo de uma claúsula definida (regra).
2. Uma transição adiciona um literal no final da cláusula, altera o último literal ou remove o último literal e altera o penúltimo de acordo com os modes definidos para a busca.
3. A transição pode alterar o último literal da cláusula para ele mesmo (self-loop).
4. O estado inicial é *True*. Este estado consta na lista de possíveis estados de um único literal e prova todas as consultas obtidas nos exemplos.
5. O algoritmo pode receber como parâmetro um limite máximo de literais numa cláusula definida, restringindo dessa forma os estados a não adicionarem literais quando não for permitido.

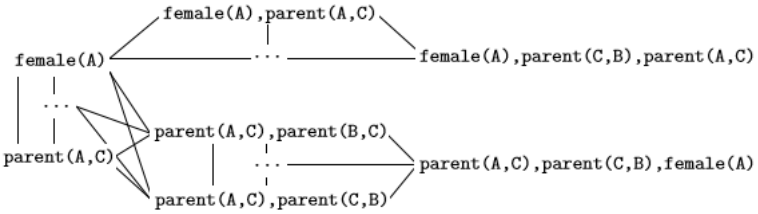

Figura 3: Pequena porção de exemplo da Cadeia de Markov para o dataset *Family* [1] que possui os predicados $female(person)$ e $parent(person, person)$. O predicado alvo no aprendizado é o $grandmother(A, B)$.

Como frequente nos algoritmos de aprendizado de estrutura, o algoritmo apresentado utiliza uma declaração baseada em *mode declarations* para restringir o espaço de busca de cláusulas [3], bem similar ao apresentado no algoritmo ProbFOIL. Na declaração dos predicados são utilizados os modes que adicionam literais nas cláusulas na forma definida que podem ser definidas da seguinte forma:

```
+: a variável nesta posição deve existir na cláusula
-: a variável nesta posição ainda não existe
c: uma constante deve ser adicionar nesta posição
```

Para o trabalho apresentado, foi definido para predicados de aridade 1 a declaração $(+)$ em que apenas variáveis já presentes na cláusula podem ser utilizadas. Casos de predicados com aridade 2, foram utilizados as declaraçãos $(+,+)$, $(+,-)$ e $(-,+)$. Nos experimentos apresentados, os datasets possuíam apenas predicados de aridade no máximo 2. A negação de literais não foi adicionado às possibilidades de transições entre claúsulas (estados) para diminuir o espaço de buscas nos experimentos.

O objetivo é encontrar um estado $\sigma \in \Sigma$ que minimize uma função custo $f(\sigma)$, em que f é uma função custo dada definida no conjunto Σ de nós da Cadeia de Markov. Considerando um parâmetro $T > 0$, chamado de temperatura e aplicando a distribuição de Boltzman, temos a probabilidade associada a um estado definido por:

$$\pi_\sigma = d(\sigma)\frac{e^{\frac{-f(\sigma)}{T}}}{Z}$$

Em que $d(\sigma)$ é o grau do nó σ e $Z = \sum_{\sigma\in\Sigma} d(\sigma)e^{\frac{-f(\sigma)}{T}}$ uma constante de normalização. Utilizando a ideia de Metropolis-Hasting obtemos uma matriz de transição P da seguinte forma:

$$P(i,j) = \frac{1}{d(i)}\min\left\{1, e^{\frac{f(i)-f(j)}{T}}\right\}$$

A heurística de otimização utilizada foi a Simulated Annealing para obter uma sequência da Cadeia de Markov utilizando pares de temperatura e número

de iterações (T_i, N_i), com valores utilizados $N_i = 1$ para todo i e a temperatura T descrescendo com a função *S-curve* dada da seguinte forma:

$$T(i) = A \cdot \frac{1}{1 + e^{i/L}}$$

Onde A e L são valores fixos no decorrer da amostragem e representam respectivamente a amplitude e largura da curva para determinar o decaimento da temperatura no decorrer das iterações i.

A função custo f foi definida como o erro médio quadrático (MSE) entre as probabilidades de sucesso estimadas para as cláusulas definidas e as probabilidades reais apresentadas nos exemplos:

$$MSE(\sigma) = \frac{1}{n} \sum_{i=1}^{n} (p_i - p_i')^2$$

Cada exemplo contribui como um exemplo positivo com p_i e como exemplo negativo com $1 - p_i$, generalizando para um cenário determinístico onde $p_i = 1$ para exemplos positivos e $p_i = 0$ para exemplos negativos. O estimador elaborado com algoritmo Monte Carlo consegue estimar probabilidades de sucesso com um intervalo de confiança δ_p definido, não sendo necessário mais do que uma amostra para um cenário puramente determinístico. Como o cálculo da probabilidade de sucesso de uma cláusula é um processo custoso, levando em conta o número de fatos e exemplos disponíveis no dataset, o algoritmo armazena a função custo de um determinado estado evitando que seja calculado novamente caso esse estado seja revisitado.

5 Experimentos

Com o intutito de testar a abordagem MCMC para aprendizado de regras foram utilizados três datasets reais: Family, pequeno dataset contendo relações familiares dos autores do algoritmo ProbFOIL; IMDB; dataset determinístico contendo relações entre atores, diretores e filmes; NELL (Never-Ending Language Learning system) que extrai informações online de bilhões de páginas web.

Foi extraído fatos de todos os predicados relacionados ao domínio de esporte da iteração 850 da base do NELL [2]. Para o dataset IMDB, Foi extraído um único fold com exemplos balanceados [3]. O dataset Family foi utilizado removendo predicados redundantes [4].

O número de fatos e exemplos por dataset e predicado alvo estão listados na Tabela 1. Cada fato possui uma probabilidade associada (por exemplo, 0.9375 :: $teamplaysagainstteam(mighty\_ducks\_of\_anaheim, northeastern\_huskies)$. No dataset determinístico IMDB os fatos possuem probabilidades associadas como

[2] Da página: http://rtw.ml.cmu.edu/rtw/resources

[3] Do repositório: https://github.com/friguzzi/cplint_datasets

[4] Do repositório: https://bitbucket.org/antondries/prob2foil

1.0 ou 0.0 (por exemplo, 1.0 :: $movie(aoceanstwelve, adontiffany)$.) e o uso da abordagem Monte Carlo para inferência não é necessária. Todo o algoritmo e datasets utilizados podem ser encontrados no repositório GitHub do projeto [5].

Dataset	Predicados	Número de fatos	Número de exemplos
Family	parent(person,person) grandmother(person,person) female(person)	32	20 (grandmother)
IMDB	actor(person) director(person) female(person) workedunder(person,person) movie(movie,person) genre(person,genre)	159	232 (workedunder)
NELL	athleteplaysinleague(athlete,sportsleague) teamplaysinleague(sportsteam,sportsleague) athleteplaysforteam(athlete,sportsteam) teamalsoknownas(sportsteam,sportsteam) athleteledsportsteam(athlete,sportsteam) teamplaysagainstteam(sportsteam,sportsteam) teamplayssport(sportsteam,sport) athleteplayssport(athlete,hobby)	8032	94 (athleteplaysforteam) 91 (athleteplaysinleague)

Tabela 1: Configuração dos diferentes experimentos.

A Tabela 2 contém os parâmetros utilizados para cada execução dos experimentos. As iterações são referentes ao Simmulated Annealing e MCMC no aprendizado de estruturas, o número de iteração no uso do método Monte Carlo para estimar a inferência fica a cargo do algoritmo que verifica o intervalo de confiança máximo. Os parâmetros de amplitude e largura da função temperatura foram estipulados de forma diferente para cada experimento.

Melhor MSE	Alvo	Máximo literais	δ_p	m	Iterações	Tempo de execução
2.5983e-06	grandmother	3	0.01	100	1000	00:08:58
0.0	workedunder	4	0.05	50	10000	00:00:58
0.3029	athleteplaysforteam	3	0.05	50	10000	62:36:36
0.3068	atheleteplaysinleague	3	0.1	50	5000	57:49:43

Tabela 2: Parâmetros dos diferentes experimentos.

A Tabela 3 apresenta as regras aprendidas (melhores nós visitados e avaliados com a fução custo) para cada predicado alvo e dataset. Foram selecionados do top 100 as cláusulas com melhores MSE e que não fossem as mesmas cláusulas apenas com literais trocados de posição ou literais adicionais que não trouxessem

[5]No repositório: https://github.com/rodrigoazs/MCMCPLP

muito significado. Uma boa solução para evitar este problema, seria a implementação da ideia do Relational Path Finding [6] [4] nas regras de transição, restrigindo cláusulas que apenas conectem a variável A ao B.

MSE	Predicado alvo		Cláusula
2.5983e-06	grandmother(A,B)	←	parent(A,C),female(A),parent(C,B)
0.0485	grandmother(A,B)	←	parent(C,B),parent(A,C),parent(D,B)
0.0489	grandmother(A,B)	←	parent(A,C),parent(C,B)
0.1231	grandmother(A,B)	←	parent(A,C),female(C),parent(C,B)
0.0	workedunder(A,B)	←	actor(A),director(B),movie(C,B),movie(C,A)
0.0	workedunder(A,B)	←	actor(A),genre(B,C),movie(D,A),movie(D,B)
0.0043	workedunder(A,B)	←	director(B),movie(C,A),movie(C,B)
0.0043	workedunder(A,B)	←	movie(C,A),genre(B,D),movie(C,B)
0.3029	athleteplaysforteam(A,B)	←	athleteledsportsteam(A,B)
0.3029	athleteplaysforteam(A,B)	←	teamplaysagainstteam(C,B),athleteledsportsteam(A,B)
0.3360	athleteplaysforteam(A,B)	←	athleteplaysinleague(A,C),teamplaysinleague(B,C)
0.3068	athleteplaysinleague(A,B)	←	teamplaysinleague(C,B),athleteledsportsteam(A,C), athleteplayssport(A,D)
0.3114	athleteplaysinleague(A,B)	←	athleteledsportsteam(A,C),teamplaysinleague(C,B)
0.3135	athleteplaysinleague(A,B)	←	athleteplaysforteam(A,C),teamplaysinleague(C,B)
0.4172	athleteplaysinleague(A,B)	←	athleteplaysforteam(A,C),teamalsoknownas(C,D), teamplaysinleague(D,B)

Tabela 3: Regras aprendidas para diferentes predicados alvos.

6 Conclusão

Foi apresentado uma abordagem Monte Carlo para a inferência de consultas em fatos probabilísticos de modo eficiente que pudesse estimar a probabilidade de sucesso de uma determinada consulta através de amostragens de subprogramas determinísticos de um programa probabilístico. Ademais, um cenário de aprendizado de estrutura utilizando o conceito MCMC mostrou-se eficiente em apresentar boas regras relacionais para um predicado alvo, evitando mínimos locais e fazendo uma vasta busca no espaço de cláusulas definidas possíveis. O resultado é uma extensão probabilística do cenário tradicional ILP de aprendizado de regras.

Referências

[1] Luc De Raedt, Anton Dries, Ingo Thon, Guy Van den Broeck, and Mathias Verbeke. Inducing probabilistic relational rules from probabilistic examples. In *Proceedings of 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 2015.

[2] Angelika Kimmig, Bart Demoen, Luc De Raedt, Vitor Santos Costa, and Ri-

cardo Rocha. On the implementation of the probabilistic logic programming language problog. *Theory Pract. Log. Program.*, 11(2-3):235–262, March 2011.

[3] Stephen Muggleton. Inverse entailment and progol, 1995.

[4] Irene M. Ong, Inês de Castro Dutra, David Page, and Vítor Santos Costa. Mode directed path finding. In *ECML*, volume 3720 of *Lecture Notes in Computer Science*, pages 673–681. Springer, 2005.

[5] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[6] Bradley L. Richards and Raymond J. Mooney. Learning relations by path-finding. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, pages 50–55, 1992.

[7] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.