A Comparison of Bottom-Up Approaches to Grounding for Templated Markov Random Fields

Eriq Augustine
UC Santa Cruz
eaugusti@ucsc.edu

Lise Getoor
UC Santa Cruz
getoor@soe.ucsc.edu

1 Introduction

Markov Random Fields (MRFs) have been shown to be a flexible and powerful tool in modeling complex problems [5]. Templating languages like *Markov Logic Networks* (MLN) [2] and *Probabilistic Soft Logic* (PSL) [1] have emerged to help construct MRFs. They are particularly well-suited to constructing MRFs over richly structured domains, because they provide a logical formalism for describing entities and their relationships, and a compact mechanism for describing the parameters of the Markov Random Field. These languages define weighted *rules* using a first order logic-like syntax. The rules act as templates for real-valued feature functions called *potentials*. These templates are then combined with data to form *ground rules* in a process called *grounding*. Together, the ground rules define the probability distribution of an MRF. More formally:

Definition 1.1. Let $\boldsymbol{x} = (x_1, ..., x_m)$ be a vector of known variables, $\boldsymbol{y} = (y_1, ..., y_n)$ be a vector of unknown random variables, $\mathcal{R} = (R_1, ..., R_l)$ be a set of rules, $\boldsymbol{w} = (w_1, ..., w_l)$ be a set of real-valued weights each corresponding to a rule, and $\boldsymbol{\phi} = (\phi_1, ..., \phi_l)$ be a set of potentials where each potential corresponds to a rule and $\phi_R(\boldsymbol{x}_r, \boldsymbol{y}_r)$ assigns the variables of this ground instance r of rule R a real-valued score. Then, a templated Markov Random Field is a probability distribution of the form:

$$P(\boldsymbol{y}|\boldsymbol{x}) \propto exp\Big(\sum_{R \in \mathcal{R}} \sum_{r \in R} w_R \phi_R(\boldsymbol{x}_r, \boldsymbol{y}_r)\Big)$$

2 Grounding Templated MRFs

Grounding, the process of instantiating each rule in the templated MRF, is a key step in performing inference. Grounding is a difficult problem and can be the limiting factor on inference in templated MRFs [6, 9–11].

Consider a simple link prediction rule: Link(R1,R2) ∧Sim(R2,R3) → Link(R1,R3) or a simple transitive equality rule from the entity resolution domain: SameRef(R1,R2) ∧ SameRef(R2,R3) → SameRef(R1,R3). These rules seem simple, but the number of groundings is potentially cubic in number of entity references in the input data. Rules like these allow our MRF to be expressive, but present a significant systems challenge to efficiently ground.

Grounding is generally approached in two ways: top-down and bottom-up. Top-down grounding starts with the rules and employs nested loops to perform replacements over all the variables. It is simple and easy to implement, but slow. Alternatively, bottom-up grounding expresses the grounding for each rule as a database query. As a result, grounding leverages the huge amount of query optimization work done by the RDBMS community. Bottom-up grounding has shown significant improvements over top-down grounding, but comes at the cost of more complex systems [9].

There are several key design decisions that need to be made when building a bottom-up grounding system.

Defining Inference Targets & Scoping: Inference in MRFs involves assigning values to unknown random variables (*inference targets*), conditioned on known variables (*evidence*). There are two common ways of defining inference targets: implicitly through variable types and populations [6, 9] and explicitly [1]. Implicit definition first requires that types are assigned to each variable used in the rules. Populations (*domains*) are built up for each type using the provided data. Inference targets are then defined by the cross product of the involved type populations. Some systems further provide *scoping rules* to help define targets and avoid costly cross products. MLN-based systems such as Tuffy [9] and Alchemy [4] implicitly define inference targets. Explicitly defining targets force the user to pre-define all targets. This pushes additional work onto the modeler, but also allows the user to specify exactly what needs to be inferred. PSL uses explicitly defined inference targets.

Trivial Potential Removal: During grounding, it is possible for potentials to be generated that are already fully satisfied or dissatisfied. That is, observed values in the data cause some potentials to have a known value regardless of the value assigned to any unobserved random variables; i.e. $\forall_{y_r} \phi_R(x_r, y_r) = c$. We refer to these fixed-value potentials as *trivial*. To reduce the size of the MRF and make downstream inference faster, trivial potentials should be removed in the grounding process. The two common places to remove trivial potentials are at the database level as part of the grounding query, or in the application level right after the grounding query. Removing trivial potentials at the database level reduces the number of potentials that are materialized out of the database. However, constructing a grounding query that removes trivial potentials typically involves a disjunctive query. The disjunction is over variables that have a non-trivial fixed value **or** variables that are inference targets. Disjunctions are notoriously difficult for query optimization and can severely degrade performance [3]. In contrast, removing trivial potentials at the application layer requires materializing larger result sets from the database, since trivial potentials are included. However the grounding query does not require any disjunctions, resulting in a faster query. In addition, checking for trivial potentials can be done in parallel at the application level.

Blocking: As previously mentioned, the number of potentials in a MRF can quickly grow prohibitively large. To limit the number of potentials, *blocks* [8] or *canopies* [7] are often constructed. Blocks and canopies use problem specific heuristics to eliminate infeasible groundings. In templated MRF languages, blocking structures can be constructed by treating block definitions as data and including them in the rules. In this case, potentials with components outside of the block are trivial and removed during the grounding phase. Another approach is to explicitly define blocking structures within

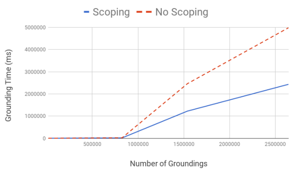

Figure 1: Scoping

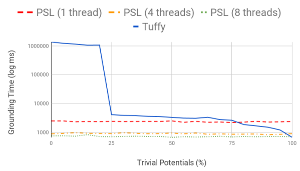

Figure 2: Trivial Potential Removal

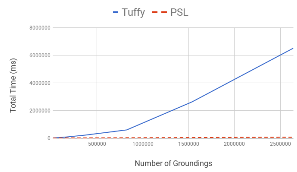

Figure 3: End-to-End Performance

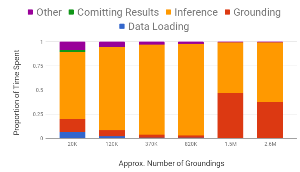

Figure 4: Tuffy Timings

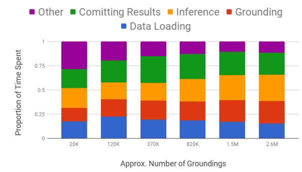

Figure 5: PSL Timings

	Similarity			Symmetry			Transitivity		
	Result Size	Max Node Size	Query Time (ms)	Result Size	Max Node Size	Query Time (ms)	Result Size	Max Node Size	Query Time (ms)
No Blocking	999,000	999,000	211	999,000	999,000	385	997,002,000	997,002,000	266,313
Implicit Blocking	100,204	999,000	599	100,204	999,000	709	10,064,832	11,063,313	8,385
Explicit Blocking	100,204	100,204	49	100,204	100,204	50	10,064,832	10,064,832	2,809

Table 1: Execution results of queries using different blocking methods.

the language. If blocking information is explicitly known, then grounding queries can be written to reflect the blocking structure. This has the potential to greatly reduce the size of intermittent operations performed by the database as well as the number of joins required by the database for grounding queries.

3 Experiments[1]

3.1 Defining Inference Targets & Scoping

The work of defining explicit targets is done outside of the grounding system. Therefore, it would be unfair to compare the time to generate targets between implicit and explicit target definition system. Instead, we look at how scoping can effect grounding time in Tuffy, a system that implicitly defines targets. Scoping is a cost-efficient way for methods that implicitly define targets to limit the target set. Figure 1 shows the grounding time for the same program with and without scoping. Using scoping reduced the grounding time by on average more than 30%.

3.2 Trivial Potential Removal

Next we investigate the impact of where trivial potentials are removed. We ground a MRF with a single rule:

$$\textsc{Similar}(P1, P2) \longrightarrow \textsc{Friends}(P1, P2)$$

on datasets which vary the percentage of trivial potentials from 0% to 100%.

In all cases, the full size of the MRF (including trivial potentials) is around 160K potentials. We compare Tuffy, which removes trivial potentials at the database level, with PSL, which removes trivial potentials at the application level. Figure 2 shows the grounding time of Tuffy and PSL with a varying number of threads. When there are fewer trivially satisfied rules, the database removal performance is very poor. It becomes more competitive as more potentials require removal. However, it is only at the 80% trivial mark that we see a crossover where database removal outperforms single-threaded application removal. At the 100% trivial mark, database removal outperforms all shown methods[2]. Because application removal requires testing all materialized potentials, it runs in constant time regardless of the percentage of trivial potentials.

3.3 Blocking

Next, we investigate the impact of blocking for three rules:

- Similarity – $\textsc{Similar}(P1, P2) \rightarrow \textsc{Friends}(P1, P2)$
- Symmetry – $\textsc{Friends}(P1, P2) \rightarrow \textsc{Friends}(P2, P1)$
- Transitivity – $\textsc{Friends}(P1, P2) \wedge \textsc{Friends}(P2, P3) \longrightarrow \textsc{Friends}(P1, P3)$

We compare three different blocking methods:

- No Blocking – Results in a larger result set, but there is no blocking overhead.
- Implicit Blocking – Blocking structures are defined as data. This generates a reduced result set, but suffers overhead from a larger grounding query.
- Explicit Blocking – System is given explicit knowledge about the desired blocking structures. An optimal query is constructed that minimizes the result set and number of joins.

The results are summarized in table 1. `Max Node Size` is the largest number of rows involved in a single operation in the query's execution plan. We see that for the `Similarity` and `Symmetry` queries, implicit blocking actually takes longer to run than without blocking. To understand this counterintuitive result, notice that the `Max Node Size` for implicit blocking and the `Result Size` for no blocking are the same. So, both queries examine the entire cross product. However in the more complex `Transitivity` query, we see implicit blocking outperforming no blocking by two orders of magnitude. In all queries, explicit blocking performs the fastest and generates execution plans with smaller nodes, while producing the same minimal result set as implicit blocking

3.4 End-to-End Performance

So far, we have focused on the decisions made when grounding and the impact of those decisions. However, it is important to also understand grounding in the context of the end-to-end inference over an MRF. Figure 3 shows the full run times of Tuffy and PSL for varying problem sizes. Figures 4 and 5 further show the breakdown of how much time is spent for each major task. In PSL, no single task dominates the run time. In Tuffy, however, we see that inference tends to dominate and as the result sets becomes large, the time spent grounding also jumps up.

4 Conclusion

From our experiments, we see several takeaways to consider when designing bottom-up grounding for templated MRFs. If using implicit target definition, be sure to include scoping. Removing trivial groundings at the database level is typically not worth the query overhead. In contrast, removing groundings in the application layer allows systems to exploit parallelism. Explicitly knowing the blocking structure can provide a huge boost to performance both in terms of memory usage and run time. However, there is still much more to be understood, when optimizing overall system performance. Acknowledgements: This work was supported by NSF CCF-1740850, NSF IIS-1703331, AFRL, and DARPA.

[1] All experiments were run on a machine with a 12 core (24 thread) 2.2 GHz CPU, 384 GB of RAM, and PostgreSQL v9.5.10. All data for experiments came from a synthetically generated dataset with variable size.

[2] Using all available threads on the machine (24), application removal performed best.

References

[1] Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2017. Hinge-Loss Markov Random Fields and Probabilistic Soft Logic. *Journal of Machine Learning Research (JMLR)* 18 (2017), 1–67.

[2] Pedro Domingos and Daniel Lowd. 2009. *Markov Logic: An Interface Layer for Artificial Intelligence* (1st ed.). Morgan and Claypool Publishers, San Rafael, CA, USA.

[3] Fisnik Kastrati and Guido Moerkotte. 2017. Optimization of Disjunctive Predicates for Main Memory Column Stores. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 731–744. https://doi.org/10.1145/3035918.3064022

[4] Stanley Kok, Parag Singla, Matthew Richardson, Pedro Domingos, Marc Sumner, and Hoifung Poon. 2010. The alchemy system for statistical relational ai: User manual. BIBLIOGRAPHY. (2010).

[5] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, Cambridge, MA, USA.

[6] Sara Magliacane, Philip Stutz, Paul Groth, and Abraham Bernstein. 2015. foxPSL: A Fast, Optimized and eXtended PSL implementation. *International Journal of Approximate Reasoning* 67, Supplement C (2015), 111 – 121. https://doi.org/10.1016/j.ijar.2015.05.012

[7] Andrew McCallum, Kamal Nigam, and Lyle H. Ungar. 2000. Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '00)*. ACM, New York, NY, USA, 169–178. https://doi.org/10.1145/347090.347123

[8] Howard B. Newcombe and James M. Kennedy. 1962. Record Linkage: Making Maximum Use of the Discriminating Power of Identifying Information. *Commun. ACM* 5, 11 (Nov. 1962), 563–566. https://doi.org/10.1145/368996.369026

[9] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. 2011. Tuffy: Scaling Up Statistical Inference in Markov Logic Networks Using an RDBMS. *PVLDB* 4, 6 (March 2011), 373–384. https://doi.org/10.14778/1978665.1978669

[10] Jay Pujara. 2016. *Probabilistic Models for Scalable Knowledge Graph Construction*. phd. University of Maryland, College Park.

[11] Sebastian Riedel. 2008. Improving the Accuracy and Efficiency of MAP Inference for Markov Logic. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence (UAI'08)*. AUAI Press, Arlington, Virginia, United States, 468–475. http://dl.acm.org/citation.cfm?id=3023476.3023532