Tractable Probabilistic Reasoning Through Effective Grounding

Eriq Augustine [1] **Theodoros Rekatsinas** [2] **Lise Getoor** [1]

Abstract

Templated Statistical Relational Learning languages, such as Markov Logic Networks (MLNs) and Probabilistic Soft Logic (PSL), offer much of the expressivity of probabilistic graphical models in a compact form that is intuitive to both experienced modelers and domain experts. However, these languages have historically suffered from tractability issues stemming from the large size of the instantiated models and the complex joint inference performed over these models. Although much research has gone into improving the tractability of these languages using approximate or lifted inference, a relatively small amount of research has gone into improving tractability through efficient instantiation of these large models. In this position paper, we will draw attention to open research areas around efficiently instantiating templated probabilistic models.

1. Introduction

Templated Statistical Relational Learning languages, such as Markov Logic Networks (MLNs) (Richardson & Domingos, 2006) and Probabilistic Soft Logic (PSL) (Bach et al., 2017), offer much of the expressivity of probabilistic graphical models in a compact form that is intuitive to both experienced modelers and domain experts. However, these languages have historically suffered from tractability issues. These tractability issues stem from two causes: the large size of the fully instantiated model when compared to the compact template representation, and the complex joint inference performed over these models.

Efforts to make these languages tractable fall into three general categories. The first category of efforts is approximation. This involves either approximating the logical problem with a continuous relaxation as is done in PSL, or using approximate inference as is done in MLNs (Poon & Domingos, 2006; Geier & Biundo, 2011; Sarkhel et al., 2016). The second category of efforts is lifted inference (Nath & Domingos, 2010; Broeck et al., 2011; Kersting, 2012; Kimmig et al., 2015; Kazemi & Poole, 2016; Srinivasan et al., 2019). Lifted inference detects symmetries and common substructures in the data and then uses those structures to avoid redundant computations. Finally, the third category of efforts into tractable templated SRL languages move away from inference and instead focus on efficient ground model instantiation. The process of full instantiating a ground graphical model from its compact template representation is called *grounding*.

In this position paper, we will draw attention to the least researched of the three tractability efforts, grounding. Throughout this paper, we will use PSL as a context in which to discuss grounding. However, all strategies and research areas discussed will be applicable to any templated SRL language. Section 2 will cover some background on templated SRL languages, PSL, and grounding. Section 3 will then look into specific opportunities for research in efficient grounding for templated SRL languages. Lastly, Section 4 will end with some concluding remarks.

2. Background

2.1. SRL Templating Languages

SRL techniques combine the power of statistical inference with relational data to produce rich models with intricate constraints and dependencies (Getoor & Taskar, 2007). Modeling relational data is inherently complicated by the large number of interconnected and overlapping structural dependencies that are typically present. To make modeling relational data easier, many SRL frameworks use familiar first order logic as a compact representation of the model (Raedt et al., 2007; Riedel, 2008; Kok et al., 2009; Niu et al., 2011; Noessner et al., 2013; Magliacane et al., 2015; Poole & Mackworth, 2017; Bach et al., 2017; Alberti et al., 2017). For example, the following logical clause can be used to express the concept of transitive similarity:

$$Similar(A, B) \wedge Similar(B, C) \rightarrow Similar(A, C)$$

[1]Department of Computer Science, University of California, Santa Cruz, California, USA [2]Department of Computer Sciences, University of Wisconsin, Madison, Wisconsin, USA. Correspondence to: Eriq Augustine <eaugusti@ucsc.edu>.

3rd Tractable Probabilistic Modeling Workshop, 36^{th} International Conference on Machine Learning, Long Beach, California, 2019.

We refer to the logical clauses that define models in these languages as *rules*. A rule is composed of a collection of *atoms* joined by logical operators. An atom consists of a predicate with constants or variables as arguments. A *ground atom* is an atom where all the arguments are constants. Ground atoms with a fixed value act as observed variables, while ground atoms with an unknown value are random variables. A *ground rule* is a rule instance that only consists of ground atoms. These ground rules act as cliques in a graphical model, typically a Markov random field (MRF). During inference these cliques are transformed into functions that assign a numeric value to the variable assignment, i.e. a potential function, the exact form of this function is chosen by the specific SRL language. Each rule is also assigned a weight that affects the penalty for not satisfying the rule. Most templated SRL languages come with the ability to either set weights manually, or learn them from data (Kok & Domingos, 2005; Singla & Domingos, 2005; Lowd & Domingos, 2007; Huynh & Mooney, 2009; 2010; Bach et al., 2013; Chou et al., 2016; Sarkhel et al., 2016; Das et al., 2016).

2.2. Probabilistic Soft Logic

Probabilistic Soft Logic (PSL) is one of the aforementioned templating SRL languages. As its underlying graphical model, PSL uses hinge-loss Markov random fields (HL-MRF), a special class of the undirected graphical model given by Definition 1. HL-MRFs are conditional distributions over real-valued atom assignments in $[0, 1]$. Unlike discrete MRFs where ground logical clauses are either satisfied or violated, a potential in HL-MRFs returns a continuous value that represents the potential's distance to satisfaction. More formally:

Definition 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:*

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z} \exp\Big(-\sum_{R \in \mathcal{R}} \sum_{r \in R} w_R \cdot \phi_R(\boldsymbol{x}_r, \boldsymbol{y}_r)\Big)$$

where

$$Z = \sum_{\boldsymbol{x}} \exp\Big(-\sum_{R \in \mathcal{R}} \sum_{r \in R} w_R \cdot \phi_R(\boldsymbol{x}_r, \boldsymbol{y}_r)\Big)$$

Given this definition, MAP inference is achieved by maximizing the sum of weighted potentials in the model:

$$\arg\max_{\boldsymbol{y}} \sum_{R \in \mathcal{R}} \sum_{r \in R} w_R \cdot \phi_R(\boldsymbol{x}_r, \boldsymbol{y}_r)$$

Furthermore, in an HL-MRF the potential function, ϕ, is always a linear combination of the variables involved in the potential. This means that each potential is not just continuous, but also convex. These convex potentials makes MAP inference on an HL-MRF a convex optimization problem. Framing inference as a convex optimization problem allows PSL to provide an exact solution to continuous problems and an approximate answer to discrete (MAX SAT) problems with rounding guarantees (Bach et al., 2015).

The continuous nature of the HL-MRF allows PSL to scale beyond what was previously feasible for SRL frameworks. Solving MAP inference as a convex optimization problem makes inference in PSL linear time with respect to the number of ground rules. This fast inference has allowed PSL to solve problems with tens of millions of ground rules in minutes (Kouki et al., 2018). PSL has also been shown to outperform industrial interior point methods (Bach et al., 2017) as well as existing MLN implementations (Pujara et al., 2013; Augustine & Getoor, 2018).

The expressive modeling and tractability of PSL has lead it to be used in many types of problems such as image classification (Gridach et al., 2017; Aditya et al., 2018a), scene understanding (Aditya et al., 2018b), activity recognition (London et al., 2013), natural language processing (Beltagy et al., 2014; Deng & Wiebe, 2015; Ebrahimi et al., 2016; Wang & Ku, 2016), bioinformatics (Sridhar et al., 2016), recommender systems (Kouki et al., 2015; Lalithsena et al., 2017), diagnosis of physical systems (Chen et al., 2014), knowledge bases (Pujara et al., 2015; Pujara & Getoor, 2016; Embar et al., 2018), and information retrieval (Alshukaili et al., 2016; Platanios et al., 2017).

2.3. Grounding

Grounding is the process of instantiating each ground rule given a rule template and the data. Grounding is generally approached in one of two ways: top-down or bottom-up. Top-down grounding starts with the rules and employs nested loops to perform replacements over all the variables (Kok et al., 2009). It is simple and easy to implement, but slow. Alternatively, bottom-up grounding works in two phases (Niu et al., 2011). First, a database query is issued that finds all the assignments of constants to variables for a specific rule. As an example, variable assignments for the transitive similarity rule from Section 2.1 can be computed by the following SQL query:

```
SELECT
    S1.argument1 AS A,
    S1.argument2 AS B,
    S2.argument2 AS C
FROM
    Similar S1,
    Similar S2,
    Similar S3
WHERE
    S1.argument1 = S3.argument1
    AND S1.argument2 = S2.argument1
    AND S2.argument2 = S3.argument2
```

We will refer to these queries for variable assignment as *grounding queries*. After executing a grounding query, bottom-up grounding instantiates each returned variable assignment into a ground rule. As a result of finding variable assignments using a relational database, grounding leverages the huge amount of work done by the RDBMS community in query optimization, fast relational joins, and memory management. Bottom-up grounding has shown significant improvements over top-down grounding, but comes at the cost of more complex systems (Niu et al., 2011; Augustine & Getoor, 2018). For the remainder of this paper, we will only be discussing bottom-up grounding.

3. Opportunities for Improving Grounding

All of the templated SRL frameworks discussed thus far are severely impacted by grounding time. Grounding is difficult because the number of ground rules can quickly become intractable. Seemingly innocuous rule templates can hide a polynomial number of instantiations. Once again, consider the case of transitive similarity:

$$Similar(A, B) \wedge Similar(B, C) \rightarrow Similar(A, C)$$

This pattern is simple and is seen in many practical problem domains. However, the number of ground rules instantiated by this rule is cubic in the size of the input data. When applying this rule in a social networking context where each variable represents a user, even a modest social network of 1,000 users generates a ground model of a billion ground rules. Rules like these allow for expressive and interdependent models, but present a significant systems challenge to efficiently ground.

To illustrate the impact of different grounding decisions, we have rerun the "End-to-End Performance" experiment from (Augustine & Getoor, 2018). The experiment is a simple SRL model implemented in Tuffy and PSL on increasingly larger instances of a synthetic dataset. In addition to Tuffy and the same version of PSL, we have included a version of PSL with naive implementations of the containing query discussed in Section 3.3 and grounding query sharing discussed in Section 3.4. Figure 3 shows the result of this experiment. Grounding in Tuffy quickly becomes intractable for this

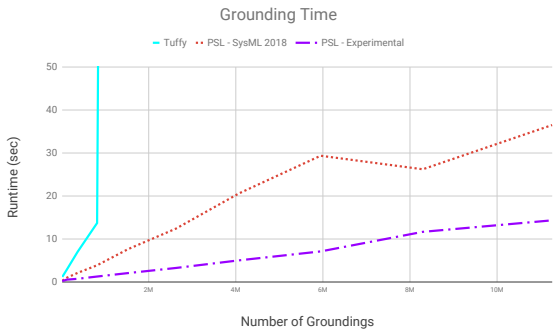

Figure 1. Runtime time across different grounding implementations: Tuffy, PSL, and PSL with experimental grounding additions.

model once the number of ground rules exceeds one million. Both versions of PSL scale with the number of ground rules approximately linearly. However, the version of PSL with the grounding improvements runs about twice as fast as the unmodified PSL. As illustrated by this experiment, there is significant room for improvement in the space of grounding. In the rest of this section, we will cover several different opportunities for improving grounding for templated SRL languages.

3.1. Exposing Sparse Structure in Data

As previously discussed, the number of ground rules generated from grounding can quickly become prohibitively large. *Blocking* (Papadakis et al., 2016) can be used to limit the size of the ground model. Blocks work by using problem specific heuristics to eliminate infeasible ground rules. In templated SRL languages, these blocks can be included in the model by incorporating them as data and including them in the rules as predicates. Then during grounding, potentials with components outside of the block will not be included in the ground model. Injecting these sparse structures into the data can limit the overall size of the model.

For example, consider the transitive similarity pattern applied to friendship prediction in a social network. We can include a block in this rule which enforces the requirement that for people to be friends they must have been members of the same social group:

$$\begin{aligned} InGroup(A, G) \wedge InGroup(B, G) \wedge InGroup(C, G) \\ \wedge Friends(A, B) \wedge Friends(B, C) \\ \rightarrow Friends(A, C) \end{aligned}$$

Augustine & Getoor (2018) have shown large reductions in grounding time by having the modeler explicitly mark blocks in the data. However, there are two obvious places for improvement here. The first is to achieve the grounding performance of explicitly marking blocks without requiring

the modeler to actually mark them. The second, and more interesting, would be to discover sparse components in the data without the user needing to introduce them. Automatically discovering sparse components would allow the user to focus on modeling tasks without needing to worry about performance tweaks.

3.2. Improving the Performance of Ground Rule Validation Functions

In the grounding process, each configuration of variables returned from the database is checked to see if it constitutes a valid ground rule. A ground rule is valid if it is not trivial and it conforms to the semantics of the specific SRL framework. Where a trivial ground rule is one whose potential value is fixed and unchangeable regardless of the value assigned to any random variables:

$$\forall_{y_r} \phi_R(x_r, y_r) = c$$

Removing trivial ground rules at the grounding phase improves performance in the later inference phase by not wasting resources on potentials with a fixed value.

The function that validates a ground rule may be as simple as just checking if the ground rule is trivial, as is the case in the Alchemy framework (Kok et al., 2009). Tuffy, which performs partial groundings of the MRF, has a more sophisticated validation function that checks to make sure only the appropriate portions of the MRF are grounded (Domingos & Lowd, 2009). PSL enforces more language semantics in the validation function; such as requiring that every generated inference target has been explicitly defined in the input data and ensuring that legal values are returned from user-defined functional predicates (predicates where the user can write custom code to evaluate ground atoms) (Bach et al., 2017). Extensions to the different frameworks' validation functions have been proposed to improve performance by reducing the size of the set of ground rules which pass validation. Glass & Barker (2012) and Beltagy et al. (2014) modified the Alchemy and PSL validation functions, respectively, by introducing a "relevance" score for each ground rule and only validating the ground rules with the highest score. Beltagy & Mooney (2014) used a custom validation function to enforce a stricter closed-world assumption.

These validation functions have been shown to be critical to scaling (Singla & Domingos, 2006), but there has been little research in how to evaluate them most effectively. Augustine & Getoor (2018) showed that there was a substantial trade-off between validating ground rules in the database (via additions to the `WHERE` clause in the grounding query) or in the application layer (via iteration through the results returned from the grounding query). In Augustine & Getoor's experiments, datasets with less than 85% trivial ground rules benefited from executing these validation functions in the application layer rather than in the database. This threshold raised to 95% when execution of the validation functions was parallelized over more than one core. Potential areas for research around effective use of these validation queries include: hybrid validation functions that are split between the database and application layers, domain-independent relevance scores (generalizations of (Glass & Barker, 2012) and (Beltagy et al., 2014)), and parallel execution of validation functions.

3.3. Exploring the Database-Memory Trade-off

Early grounding techniques for SRL were primarily centered around top-down grounding. These techniques were simple and fully resided in memory. As the need for more performant grounding rose, bottom-up grounding came into favor. More and more computation was pushed into the database. However, there is more room for research in the trade-off between grounding time spent in the database and grounding time spent in memory. Tuffy has already achieved much success through a hybrid memory/database inference infrastructure (Niu et al., 2011). A similar approach to grounding could prove beneficial.

One way to exploit this trade-off would be to use simpler grounding queries that may return larger result sets in exchange for faster query execution. As previously discussed, the results of the grounding query are filtered through a validation function which removes spurious variable configurations. Because of this validation function, it is possible to substitute the grounding query with one that returns a superset of results. As long as the optimized query is safe (i.e., every variable appears in a positive atom) (Garcia-Molina et al., 2008), then the validation function will invalidate any spurious variable configurations returned by the grounding query. So instead of executing the base query, any *containing query* can be invoked. A query q_1 contains another query q_2 if q_2 produces only answers that q_1 also produces (Garcia-Molina et al., 2008). Because the validation functions make it possible to use queries that return larger result sets than the base grounding query, we can explore the trade-off between query size and total grounding time. More query results means more variable configurations that need to be validated, however the executed query can be simpler and contain fewer joins.

3.4. Reuse of Grounding Results

For large and/or complex models, the grounding query itself can take much more time than instantiating the results into ground rules. In addition, models often have rules that share many atoms. Once again, consider the transitive similarity example:

$$Similar(A, B) \wedge Similar(B, C) \rightarrow Similar(A, C)$$

A natural complement to this rule would be to negate several of the atoms:

$$Similar(A, B) \wedge !Similar(B, C) \rightarrow !Similar(A, C)$$

These rules are semantically quite different, but almost identical in the eyes of the first phase in bottom-up grounding where the task it to collect the combinations of constants that map to the variables in the rule. These rules share all the same variables and atoms. In cases like these, it could be possible to reuse the results of a grounding query to ground multiple rules.

When explored together with the modified grounding queries discussed in Section 3.3, the opportunities to expand grounding queries becomes much more available. Each rule can have multiple possible grounding queries (any query that contains the original query). When the collection of all rules in a model are considered together, it is possible to choose a combination of grounding queries where each individual query is not the fastest available, but the collection of queries maximize the number of rules grounded per query. For example, consider the following two rules from our social networking example:

$$\begin{aligned} InGroup(A, G) \wedge InGroup(B, G) \wedge InGroup(C, G) \\ \wedge Friends(A, B) \wedge Friends(B, C) \\ \rightarrow Friends(A, C) \end{aligned}$$

$$\begin{aligned} InGroup(A, G) \wedge InGroup(B, G) \wedge InGroup(C, G) \\ \wedge Similar(A, B) \wedge Similar(B, C) \\ \rightarrow Similar(A, C) \end{aligned}$$

Both rules could be grounding by using a grounding query that covers these atoms:

$$InGroup(A, G) \wedge InGroup(B, G) \wedge InGroup(C, G)$$

4. Conclusion

Templated SRL languages have proven to be expressive and powerful tools for creating complex probabilistic models. To improve tractability, much research has gone into inference and lifting, but there are many research opportunities left in the area of grounding. In this paper, we discussed several potential areas of research pertaining to grounding for templated SRL languages. We have also illustrated through a simple experiment the performance gains a more intelligent grounding strategy can offer. Although our examples have been made concrete using PSL, the grounding tactics discussed here are general and applicable to any templated SRL language. Addressing the bottleneck of grounding breaks one of the barriers keeping templated SRL languages from being tractable.

5. Acknowledgements

This work was partially supported by the National Science Foundation grants CCF-1740850 and IIS-1703331 and by AFRL and the Defense Advanced Research Projects Agency.

References

Aditya, S., Yang, Y., and Baral, C. Explicit reasoning over end-to-end neural architectures for visual question answering. In *AAAI*, 2018a.

Aditya, S., Yang, Y., Baral, C., and Aloimonos, Y. Combining knowledge and reasoning through probabilistic soft logic for image puzzle solving. In *UAI*, pp. 238–248, 2018b.

Alberti, M., Bellodi, E., Cota, G., Riguzzi, F., and Zese, R. cplint on swish: Probabilistic logical inference with a web browser. *Intelligenza Artificiale*, 11:47–64, 2017.

Alshukaili, D., Fernandes, A. A. A., and Paton, N. W. Structuring linked data search results using probabilistic soft logic. In *ISWC*, pp. 3–19, 2016. doi: 10.1007/978-3-319-46523-4_1.

Augustine, E. and Getoor, L. A comparison of bottom-up approaches to grounding for templated markov random fields. In *SysML*, 2018.

Bach, S., Huang, B., and Getoor, L. Unifying local consistency and max sat relaxations for scalable inference with rounding guarantees. In *AIStats*, pp. 46–55, 2015.

Bach, S. H., Huang, B., London, B., and Getoor, L. Hinge-loss markov random fields: Convex inference for structured prediction. In *UAI*, 2013.

Bach, S. H., Broecheler, M., Huang, B., and Getoor, L. Hinge-loss markov random fields and probabilistic soft logic. *JMLR*, 18:1–67, 2017.

Beltagy, I. and Mooney, R. Efficient markov logic inference for natural language semantics. In *StarAI*, 2014.

Beltagy, I., Erk, K., and Mooney, R. Probabilistic soft logic for semantic textual similarity. In *ACL*, pp. 1210–1219. Association for Computational Linguistics, 2014. doi: 10.3115/v1/P14-1114.

Broeck, G. V. D., Taghipour, N., Meert, W., Davis, J., and Raedt, L. D. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*, pp. 2178–2185. AAAI Press, 2011. doi: 10.5591/978-1-57735-516-8/IJCAI11-363.

Chen, P.-T., Chen, F., and Qian, Z. Road traffic congestion monitoring in social media with hinge-loss markov random fields. In *ICDM*, pp. 80–89. IEEE Computer Society, 2014. doi: 10.1109/ICDM.2014.139.

Chou, L., Sarkhel, S., Ruozzi, N., and Gogate, V. On parameter tying by quantization. In *AAAI*, pp. 3241–3247, 2016.

Das, M., Wu, Y., Khot, T., Kersting, K., and Natarajan, S. Scaling lifted probabilistic inference and learning via graph databases. In *SIAM*, pp. 738–746, 2016.

Deng, L. and Wiebe, J. Joint prediction for entity/event-level sentiment analysis using probabilistic soft logic models. In *EMNLP*, pp. 179–189. Association for Computational Linguistics, 2015. doi: 10.18653/v1/D15-1018.

Domingos, P. and Lowd, D. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan and Claypool Publishers, 1st edition, 2009.

Ebrahimi, J., Dou, D., and Lowd, D. Weakly supervised tweet stance classification by relational bootstrapping. In *EMNLP*, pp. 1012–1017. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1105.

Embar, V., Farnadi, G., Pujara, J., and Getoor, L. Aligning product categories using anchor products. In *KBC*, 2018.

Garcia-Molina, H., Ullman, J. D., and Widom, J. *Database Systems: The Complete Book*. Prentice Hall Press, 2 edition, 2008.

Geier, T. and Biundo, S. Approximate online inference for dynamic markov logic networks. In *ICTAI*, pp. 764–768, 2011.

Getoor, L. and Taskar, B. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

Glass, M. R. and Barker, K. Focused grounding for markov logic networks. In *FLAIRS*, 2012.

Gridach, M., Haddad, H., and Mulki, H. Churn identification in microblogs using convolutional neural networks with structured logical knowledge. In *ACL NUT*, pp. 21–30, 2017.

Huynh, T. N. and Mooney, R. J. Max-margin weight learning for markov logic networks. In *KDD*, pp. 564–579, 2009.

Huynh, T. N. and Mooney, R. J. Online max-margin weight learning with markov logic networks. In *StarAI*, pp. 32–37, 2010.

Kazemi, S. M. and Poole, D. Knowledge compilation for lifted probabilistic inference: Compiling to a low-level language. In *KR*, pp. 561–564. AAAI Press, 2016.

Kersting, K. Lifted probabilistic inference. In *ECAI*, pp. 33–38. IOS Press, 2012. doi: 10.3233/978-1-61499-098-7-33.

Kimmig, A., Mihalkova, L., and Getoor, L. Lifted graphical models: a survey. *Machine Learning*, 99:1–45, 2015. doi: 10.1007/s10994-014-5443-2.

Kok, S. and Domingos, P. Learning the structure of markov logic networks. In *ICML*, pp. 441–448, 2005.

Kok, S., Sumner, M., Richardson, M., Singla, P., Poon, H., Lowd, D., Wang, J., and Domingos, P. The alchemy system for statistical relational ai. Technical report, Department of Computer Science and Engineering, University of Washington, 2009.

Kouki, P., Fakhraei, S., Foulds, J., Eirinaki, M., and Getoor, L. Hyper: A flexible and extensible probabilistic framework for hybrid recommender systems. In *RecSys*. ACM, 2015.

Kouki, P., Pujara, J., Marcum, C., Koehly, L., and Getoor, L. Collective entity resolution in multi-relational familial networks. *KAIS*, pp. 1–35, 2018.

Lalithsena, S., Perera, S., Kapanipathi, P., and Sheth, A. P. Domain-specific hierarchical subgraph extraction: A recommendation use case. *Big Data*, pp. 666–675, 2017. doi: 10.1109/BigData.2017.8257982.

London, B., Khamis, S., Bach, S. H., Huang, B., Getoor, L., and Davis, L. Collective activity detection using hinge-loss markov random fields. In *CVPR SPTLI*, 2013.

Lowd, D. and Domingos, P. Efficient weight learning for markov logic networks. In *KDD*, pp. 200–211, 2007.

Magliacane, S., Stutz, P., Groth, P., and Bernstein, A. foxpsl: A fast, optimized and extended psl implementation. *IJAR*, 67:111 – 121, 2015. doi: https://doi.org/10.1016/j.ijar.2015.05.012.

Nath, A. and Domingos, P. Efficient lifting for online probabilistic inference. In *StarAI*, 2010.

Niu, F., Ré, C., Doan, A., and Shavlik, J. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *VLDB*, 4:373–384, 2011.

Noessner, J., Niepert, M., and Stuckenschmidt, H. Rockit: Exploiting parallelism and symmetry for map inference in statistical relational models. In *StarAI*, pp. 37–42. AAAI Press, 2013.

Papadakis, G., Svirsky, J., Gal, A., and Palpanas, T. Comparative analysis of approximate blocking techniques for entity resolution. *VLDB*, 9:684–695, 2016.

Platanios, E., Poon, H., Mitchell, T. M., and Horvitz, E. J. Estimating accuracy from unlabeled data: A probabilistic logic approach. In *NIPS*, pp. 4361–4370. Curran Associates, Inc., 2017.

Poole, D. and Mackworth, A. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2nd edition, 2017.

Poon, H. and Domingos, P. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, volume 6, pp. 458–463, 2006.

Pujara, J. and Getoor, L. Generic statistical relational entity resolution in knowledge graphs. In *StarAI*. IJCAI 2016, 2016. On arXiv: https://arxiv.org/abs/1607.00992.

Pujara, J., Miao, H., Getoor, L., and Cohen, W. Knowledge graph identification. In *ISWC*, pp. 542–557, 2013.

Pujara, J., London, B., Getoor, L., and Cohen, W. Online inference for knowledge graph construction. In *StarAI*, 2015.

Raedt, L. D., Kimmig, A., and Toivonen, H. Problog: A probabilistic prolog and its application in link discovery. In *IJCAI*, pp. 2468–2473, 2007.

Richardson, M. and Domingos, P. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

Riedel, S. Improving the accuracy and efficiency of map inference for markov logic. In *UAI*, pp. 468–475, 2008.

Sarkhel, S., Venugopal, D., Pham, T., Singla, P., and Gogate, V. Scalable training of markov logic networks using approximate counting. In *AAAI*, 2016.

Singla, P. and Domingos, P. Discriminative training of markov logic networks. In *AAAI*, pp. 868–873, 2005.

Singla, P. and Domingos, P. Memory-efficient inference in relational domains. In *AAAI*, pp. 488–493, 2006.

Sridhar, D., Fakhraei, S., and Getoor, L. A probabilistic approach for collective similarity-based drug-drug interaction prediction. *Bioinformatics*, 2016.

Srinivasan, S., Babaki, B., Farnadi, G., and Getoor, L. Lifted hinge-loss markov random fields. In *AAAI*, 2019.

Wang, W.-C. and Ku, L.-W. Identifying chinese lexical inference using probabilistic soft logic. In *ASONAM*, pp. 737–743. IEEE Press, 2016.