XEGGORA: Exploiting Immune-to-Evidence Symmetries with Full Aggregation in Statistical Relational Models (Extended Abstract)*

Mohammad Mahdi Amirian[1] and **Saeed Shiry Ghidary**[2]
[1]Department of Computer Engineering, Amirkabir University of Technology, Tehran, Iran
[2]Department of Math & Computer Science, Amirkabir University of Technology, Tehran, Iran
{m.amirian, shiry}@aut.ac.ir

Abstract

We present improvements in maximum a-posteriori inference for Markov Logic, a widely used SRL formalism. Several approaches, including Cutting Plane Aggregation (CPA), perform inference through translation to Integer Linear Programs. Aggregation exploits context-specific symmetries independently of evidence and reduces the size of the program. We illustrate much more symmetries occurring in long ground clauses that are ignored by CPA and can be exploited by higher-order aggregations. We propose Full-Constraint-Aggregation, a superior algorithm to CPA which exploits the ignored symmetries via a lifted translation method and some constraint relaxations. RDBMS and heuristic techniques are involved to improve the overall performance. We introduce XEGGORA as an evolutionary extension of ROCKIT, the query engine that uses CPA. XEGGORA evaluation on real-world benchmarks shows progress in efficiency compared to ROCKIT especially for models with long formulas.

1 Introduction

MAP inference is a concerning need in Markov Logic [Richardson and Domingos, 2006] because of its high complexity. Several *lifting* approaches [Kimmig et al., 2015] including Cutting Plane Aggregation (CPA) [Noessner et al., 2013] are introduced to speed up inference by detecting symmetries and avoiding repetitive computations. CPA improves compilation from a ground network to an Integer Linear Program (ILP) by decreasing the size of the program and better exposing its symmetries. The ILP is further passed to traditional solvers and the solution is mapped to a MAP state of the original MLN.

There still exist further symmetries that CPA is unable to handle. In this work [Amirian and Shiry Ghidary, 2019], we first illustrate some types of these symmetries by introducing *order of aggregation* and showing that CPA is able to perform only aggregations of order one, whereas higher-order aggregations are required to exploit the ignored symmetries. Secondly, a superior algorithm, namely *Full Constraint Aggregation* (FCA) is introduced, empowered with any possible order of aggregation with respect to the model. FCA performs a novel translation to efficient ILPs by relaxing some constraints. We then propose complementary techniques, including heuristics and RDBMS leverage to choose efficiently among multiple candidates the one in which FCA fits the best. The proposed methods are implemented within an evolutionary extension of ROCKIT, namely XEGGORA. Finally, we show that XEGGORA's time-preserving techniques outperform ROCKIT while applied to models with long formulas.

1.1 MAP Inference in Markov Logic

Markov Logic allows softening a first-order formula *f*, by attaching a real-valued weight *w* to it. A positive (negative) weight makes the formula support (penalize) worlds in which it is satisfied. Hard formulas are regular first-order formulas which are expressed with infinite weights and have to be fulfilled by every possible world. The probability of a possible world *x* in the presence of evidence *e* is defined as:

$$P(X = x|e) = \frac{1}{Z_e} exp\left(\sum_{i:f_i \in F} w_i n_i(x, e)\right)$$

where Z_e is a normalization constant with respect to e, and $n_i(x, e)$ is the number of true groundings of f_i in x that satisfy e. MAP inference corresponds to inferring the most likely possible world, and reduces to finding the interpretation that maximizes the sum of the weights of the satisfied clauses.:

$$argmax_x P(x|e) = argmax_x \frac{1}{Z_e} exp\left(\sum_{i:f_i \in F} w_i n_i(x, e)\right)$$
$$= argmax_x \sum_{i:f_i \in F} w_i n_i(x, e).$$

This can be left as an ILP formulation [Wolsey, 1998] for efficient optimizers to be solved. Huynh and Mooney [2009]

* This paper is an extended abstract of an article in the Journal of Artificial Intelligence Research [Amirian and Shiry Ghidary, 2019]

proposed a translation of MAP queries to ILPs and Noessner et al. [2013] extended the approach by applying constraint aggregation. We improve the result by applying higher-order aggregations and introduce Full Constraint Aggregation. The method is powered with RDBMS and heuristic techniques to choose efficiently among candidate clauses to be aggregated.

1.2 Preface

In the next section we describe the ILP formulation and constraint aggregation methods. Section 3 is composed of the FCA algorithm description. Section 4 describes add-on techniques that leverage RDBMS and related heuristics to improve the overall performance. The reader is referred to the original paper for details, examples, soundness proofs, ILP solving analytics, and the empirical evaluation of FCA on six benchmark MLNs.

2 MAP ILP Formulation Techniques

In all parts of the process, formulas are supposed to be composed of disjunctive clauses. The traditional ILP formulation, and the enhancements of constraint aggregation are introduced here. We then continue by introducing symmetries that CPA fails to exploit.

2.1 Traditional ILP Formulation

Given a set of ground clauses $\mathcal{G}$ as input, one binary ILP variable x_ℓ is associated with each ground atom ℓ occurring in the set, assigning it a value 1 provided ℓ is true and 0 otherwise. Each evidence atom can be represented as a linear constraint of the form $x_\ell \leq 0$ or $x_\ell \geq 1$. For each ground clause $g \in \mathcal{G}$ with weight $w_g \in \mathbb{R}$, a binary variable z_g with coefficient w_g is added to the objective. The objective totally becomes:

$$\textit{maximize} \sum_{g \in \mathcal{G}} w_g z_g.$$

It is straightforward to design constraints considering z_g to be assigned 1 provided its corresponding ground clause is satisfied in the MAP solution and 0 otherwise. For a clause with $w_g > 0$, the following constraint is added to the ILP:

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \geq z_g$$

where $L^+(g)$ and $L^-(g)$ refer to the sets of all ground atoms occurring unnegated and negated in g, respectively. A truth value assignment to any ground atom that fulfills the disjunctive clause g, makes the left-hand side of the constraint positive, thus, z_g can take its maximum possible value 1.
For hard clauses ($w_g = \infty$), no term is added to the objective and z_g in the constraint is simply replaced with 1:

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \geq 1.$$

For a clause with $w_g < 0$, the optimizer tries to lower z_g to its minimum possible value 0. The following constraint is added to permit this only if none of its constituting ground atoms satisfy g:

$$\sum_{\ell \in L^+(g)} x_\ell + \sum_{\ell \in L^-(g)} (1 - x_\ell) \leq (|L^+(g)| + |L^-(g)|)\, z_g.$$

Finally, for clauses with zero weights, no constraint is added to the ILP.

2.2 Cutting Plane Aggregation

Grounding formulas in MLN often results in symmetries in multiple ground clauses, and consequently, in the ILP constraints. Constraint aggregation proposes aggregating ground clauses that include symmetries, resulting in smaller constraint matrices and aiding symmetry detection algorithms of the ILP solver. The candidate sets of ground clauses to which aggregation can be applied are defined as follows:

Definition 1 *Let $G \subseteq \mathcal{G}$ be a set of n weighted ground clauses and let c be a ground clause. We say that G can be aggregated with respect to c if (a) all ground clauses in G have the same weight and (b) for every $g_i \in G, 1 \leq i \leq n$, we have that $g_i = \ell_i \vee c$ where ℓ_i is a positive or a negative literal.*

For such a set, all corresponding ILP constraints can be replaced by fewer constraints with the following rules:

Definition 2 (First-Order Aggregation Rules) *Let $G \subseteq \mathcal{G}$ be a set of n ground clauses with weight w_G that can be aggregated with respect to a ground clause c.*

I. *In case of a finite weight ($w_G \neq \infty$), replace the corresponding terms in the ILP objective with a new integer variable $z_G \in \{0, .., n\}$ with coefficient w_G:*

$$\textit{maximize } w_G z_G + \textit{rest of the objective.}$$

If $w_G > 0$, replace the corresponding ILP constraints with:

$$\sum_{p|p \vee c \in G} x_p + \sum_{q|\neg q \vee c \in G} (1 - x_q) + \sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) \geq z_G,$$

and if $w_G < 0$, with:

$$\sum_{p|p \vee c \in G} x_p + \sum_{q|\neg q \vee c \in G} (1 - x_q) \leq z_G,$$
$$n x_\ell \leq z_G \quad \textit{for every } \ell \in L^+(c),$$
$$n(1 - x_\ell) \leq z_G \quad \textit{for every } \ell \in L^-(c).$$

II. *In case of an infinite weight ($w_G = \infty$), only replace the corresponding ILP constraints with:*

$$\sum_{p|p \vee c \in G} x_p + \sum_{q|\neg q \vee c \in G} (1 - x_q) + \sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) \geq n.$$

According to the aggregation rules, z_G is considered to be assigned the number of satisfied ground clauses in G; it gains the maximum value *n* provided a solution satisfies the ground clause *c,* otherwise it is equal to the number of the literals (of the form p or $\neg q$) satisfied by the solution. The CPA algorithm chooses greedily among sets that can be aggregated to a more compact ILP and applies constraint aggregation.

2.3 CPA Failure

There still exist similarities between clauses in the following example, but in addition, diversity in more than one literal.

Example 1 *Consider an MLN with a single formula of weight 2.3 stating that each kid may be happy of having a kind parent who has fun with him (her):*
2.3 Child(k, p) ∧ Kind(p) ∧ HasFun(p, k) ⇒ Happy(k).

By grounding and applying the evidence:
Child(Mary, Jack), Child(Mary, Rose)
the essential ground clauses in normal form would be:
2.3 ¬Kind(Jack) ∨ ¬HasFun(Jack, Mary) ∨ Happy(Mary)
2.3 ¬Kind(Rose) ∨ ¬HasFun(Rose, Mary) ∨ Happy(Mary)

These symmetries are ignored by the CPA algorithm; it allows only one literal to be distinct while all others must be identical. This is the justification of naming the method *first-order* aggregation. This failure often happens in models containing long formulas. We introduce Full-Constraint-Aggregation that can exploit this type of symmetry.

3 Full Constraint Aggregation

We start by illustrating symmetry types that appear in ground clauses as in Example 1, based on a parameter we name *order of* a*ggregation*. Then we describe our superior algorithm which performs higher-order aggregations.

Definition 3 *Let $G \subseteq \mathcal{G}$ be a set of n weighted ground clauses and let c be a ground clause with non-zero length. We say that G can be aggregated of order k with respect to c, if (a) all ground clauses in G have the same weight and (b) for every $g_i \in G$, $1 \leq i \leq n$, we have that $g_i = L_i \vee c$ where L_i is a ground clause of maximum length k. We call k the aggregation order of G, if k is the minimum order of which G can be aggregated (with respect to any ground clause c).*

Due to Definition 3, the constraint aggregation method is only applicable to the sets of ground clauses with aggregation order of 1, performing *first-order aggregation*. For a set of a higher aggregation order, we need to associate disjunctions of literals with binary terms in the optimization program, e.g. $a \vee b \equiv x_a + x_b - x_a x_b$. This results in an *Integer Polynomial Program*. To be more precise, a k[th] order trivial aggregation may be performed to formulate the MAP inference problem as an integer k[th] order mathematical program. The program can be further linearized to an equivalent ILP using the classic linearization method outlined by Watters [1967], as previously done in a different MAP inference method by Sarkhel et al. [2014]. However, the resulting ILP would carry much more variables and constraints and last longer to be solved than the one from the traditional ILP formulation.

To overcome this issue, we propose a novel relaxed translation of MLNs to ILPs by associating auxiliary variables with disjunctive clauses. This is done through higher-order aggregation rules.

Definition 4 (Higher-Order Aggregation Rules) *Let $G \subseteq \mathcal{G}$ be a set of n ground clauses with weight w_G with the aggregation order of $k > 1$, i.e., can be aggregated of order k with respect* to *a ground clause c. Each ground clause L_i such that $L_i \vee c \in G$ is a disjunction of at most k unnegated or negated ground atoms. For each L_i with length 1, apply the First-Order Aggregation Rules. For the remainder:*

I. *Corresponding to each L_i, define a new binary ILP variable x_{s_i}. Choose one of the following Bound Constraints (BCs) after the sign of w_G and add it to the ILP:*

Upper-BC: $x_{s_i} \leq \sum_{\ell \in L^+(L_i)} x_\ell + \sum_{\ell \in L^-(L_i)} (1 - x_\ell)$
if $w_G > 0$ or $w_G = \infty$, and
Lower-BC: $\sum_{\ell \in L^+(L_i)} x_\ell + \sum_{\ell \in L^-(L_i)} (1 - x_\ell) \leq |L_i| . x_{s_i}$
if $w_G < 0$.

II. *In case of a finite weight ($w_G \neq \infty$), replace the corresponding terms in the ILP objective with a new integer variable $z_G \in \{0, .., n\}$ with coefficient w_G:*

$$\text{maximize } w_G z_G + \text{rest of the objective.}$$

If $w_G > 0$, replace the corresponding ILP constraints with:

$$\sum_{i=1}^{n} x_{s_i} + \sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) \geq z_G,$$

and if $w_G < 0$, with:

$$\sum_{i=1}^{n} x_{s_i} \leq z_G,$$
$$n x_\ell \leq z_G \quad \text{for every } \ell \in L^+(c),$$
$$n(1 - x_\ell) \leq z_G \quad \text{for every } \ell \in L^-(c).$$

III. *In case of an infinite weight ($w_G = \infty$), only replace the corresponding ILP constraints with:*

$$\sum_{i=1}^{n} x_{s_i} + \sum_{\ell \in L^+(c)} n x_\ell + \sum_{\ell \in L^-(c)} n(1 - x_\ell) \geq n.$$

The Upper-BC and Lower-BC pair if used altogether, represent the disjunction operator(s) of L_i in the ILP. Intuitively, to correspond a logical OR operator for two Boolean variables: $c \equiv a \vee b$ in the integer space, one may add two inequalities: $x_c \leq x_a + x_b \leq 2x_c$. Similar inequalities would represent the disjunction of *k* Boolean variables. Now if x_c is involved in either maximization or minimization, one can drop the corresponding inessential constraint. The reader is referred to the original paper [Amirian and Shiry Ghidary, 2019] for the soundness proof.

4 Leveraging RDBMS and Heuristics

In the original paper, we showed analytically a major difference between the first-order and higher-order aggregations. First-order aggregation, if applicable, reduces the ILP size on the order of the number of aggregated clauses, which results in much faster program solving. This is also verified through experiments on the artificial data. Desired metrics may be obtained to compare the efficiency of possible clustering schemes. However, this is not trivial because efficiency is an abstract measure that relates to the cost of the aggregation procedure in terms of size and runtime, along with the cost of solving the ILP. Moreover, involving a metric in the algorithm would acquire time-consuming computations. Instead, we propose a heuristic approach to choose an appropriate clustering scheme for aggregation as described in the following algorithm. In order to benefit from query optimization technologies, this algorithm leverages RDBMS to compute the compactness of possible aggregations. The idea of RDBMS leverage for MAP inference in Markov Logic was inspired by Riedel [2008] and evolved by Niu et al. [2011] and Noessner et al. [2013]. They used RDBMS in the grounding phase and also in finding the violated constraints, but the computation of counting features was not implemented with RDBMS because it requires storing or regeneration of the ground table which is often inefficient for first-order

aggregation. In the case of higher-order aggregations however, counting the number of rows is required several times which is done by the `COUNT DISTINCT` query. It is a standard SQL query used for counting the number of distinguishable rows on the specified columns in a table. Extensionally, MySQL permits using it multiple times in a single query.

Algorithm CHOOSEFORAGGREGATION

Input $\mathcal{F}$: a first-order disjunctive clause
Input $\mathcal{G}$: a SQL table, containing all essential ground clauses of $\mathcal{F}$
Output appropriate clustering scheme as the target for aggregation

1: L ← $Literals\_of\ (\mathcal{F})$
2: $\mathcal{V}$ ← $Variables\_of$ (L)
3: **if** |L| = 1
4: *candidate_sets* ← {∅}
5: **else**
6: *candidate_sets* ← {}
7: **foreach** non-empty $V \subseteq \mathcal{V}$
8: *identical_literals* ← $\{\ell_i \in L \mid Variables_{of}(\ell_i) \subseteq V\}$
9: **if** *identical_literals* ∉ *candidate_sets* **and** *identical_literals* ≠ L
10: add *identical_literals* to *candidate_sets*
11: **if** *candidate_sets* contains any sets of literals with the size of |L| - 1
12: **return** best of them greedily to be further first-order aggregated
13: **else**
14: *query* ← `'SELECT '`
15: **foreach** *identical_part* ∈ *candidate_sets*
16: *query* ← *query* + `'COUNT (DISTINCT '` + $Serialize$ (*identical_part*) + `') as '` + $Caption$ (*identical_part*) [+ `','`] // *except for the last loop*
17: *query* ← *query* + `'FROM '` + $\mathcal{G}$
18: execute *query* into *#clusters*
19: *best_candidate_sets* ← {argmin$_{\#clusters}$ *candidate_sets*}
20: **return** argmax$_{\text{cardinality}}$ *best_candidate_sets*

The algorithm is provided with a disjunctive clause and a table containing all of its essential groundings in the rows and its constituting variables in the columns. Inessential groundings (e.g. by the substitution of $\{k \mapsto Bob,\ p \mapsto Rose\}$ in Example 1) should have been previously ignored in order to produce an effective output. The algorithm first acquires the set of all literals that constitute the clause and also all variables that participate in grounding (lines 1, 2). In lines 3-6, a set is initialized for storing all feasible partitioning schemes of the literals. Each *candidate set* is a representative for the set of identical literals (i.e., clause c in Definition 3), so the complementary literals are considered distinct. The cardinality of a candidate set represents the clause length minus its proposing order of aggregation. ∅ is by default a feasible candidate set. However, it is added only for first-order aggregation as it would not be efficient for higher orders, as considered in Definition 3. Lines 7-10 collect candidate sets for aggregation. One may trivially collect the sets based on ground literals; however, we build the collection method upon non-empty sets of variables, in order to avoid producing infeasible candidate sets. It starts with partitioning the variables. Each partitioning scheme of variables specifies a feasible partitioning scheme of literals which in turn corresponds to a clustering scheme of the essential ground clauses. Line 11 discovers if any candidates are present for first-order aggregation, to be in case the arguments of a call to the procedure (line 12). If there exist more than one candidate, one would be chosen greedily as done in CPA. Lines 14-18 contain the code for generating and executing the SQL query that counts the number of clusters aggregated according to each partitioning scheme. Each single clause that is not aggregated is considered in a singleton cluster. We involve the query output *#clusters* as a lookup table which returns the number of clusters given a candidate set. Finally, the candidate partitioning scheme that offers the minimum number of clusters is selected. If there exist multiple solutions, the one with the lowest aggregation order is returned (lines 19-20). This is motivated from the analysis on the artificial data; keeping other parameters fixed, lower-order aggregations often produce slightly more efficient ILPs.

5 Empirical Evaluation

We implemented the proposed methods within an evolutionary extension of ROCKIT, namely XEGGORA, besides fixing minor bugs of the previous system. Finally, we proved enhancements with comprehensive experiments upon two benchmark sets. The first set of artificial benchmarks explores the effect of higher-order aggregation as compared to the first-order aggregation with respect to the length of clauses. The results confirm the intuition developed through the theoretic side. On practical benchmarks for the entire system, we showed that FCA retains the enhancements of previous works on instances where higher-order aggregation is not applicable, but provides a large benefit in instances where it is.

References

[Amirian and Shiry Ghidary, 2019] Mohammad Mahdi Amirian and Saeed Shiry Ghidary. Xeggora: Exploiting Immune-to-Evidence Symmetries with Full Aggregation in Statistical Relational Models, *Journal of Artificial Intelligence Research*, 66: 33-56, 2019.

[Huynh and Mooney, 2009] Tuyen N. Huynh and Raymond J. Mooney. Max-margin weight learning for Markov logic networks. In *Proceedings of ECML PKDD 2009, Part I, volume 5781 of Lecture Notes in Computer Science*, pages 564-579, Springer, 2009.

[Kimmig et al., 2015] Angelika Kimmig, Lilyana Mihalkova, and Lisa Getoor. Lifted graphical models: a survey. *Machine Learning*, 99: 1-45, 2015.

[Niu et al., 2011] Feng Niu, Christopher Ré, AnHai Doan, and Jude Shavlik. Tuffy: Scaling up statistical inference in Markov logic networks using an RDBMS. In *Proceedings of the VLDB Endowment*, 4(6): 373-384, 2011.

[Noessner et al., 2013] Jan Noessner, Mathias Niepert, and Heiner Stuckenschmidt. RockIt: Exploiting Parallelism and Symmetry for MAP Inference in Statistical Relational Models. In *Proceedings of AAAI*, pages 739-745, 2013.

[Richardson and Domingos, 2006] Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62: 107-136, 2006.

[Riedel, 2008] Sebastian Riedel. Improving the accuracy and efficiency of MAP inference for Markov Logic. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, pages 468-475, 2008.

[Sarkhel et al., 2014] Somdeb Sarkhel, Deepak Venugopal, Parag Singla, and Vibhav Gogate. An Integer Polynomial Programming Based Framework for Lifted MAP Inference. *Advances in Neural Information Processing Systems*, pages 3302-3310, 2014.

[Watters, 1967] Lawrence J. Watters. Reduction of Integer Polynomial Programming Problems to Zero-One Linear Programming Problems. *Operations Research*, 15(6): 1171-1174, 1967.

[Wolsey, 1998] Laurence A. Wolsey. *Integer Programming*. Wiley-Interscience, New York, 1998.