Fully Parallel Inference in Markov Logic Networks

Kaustubh Beedkar, Luciano Del Corro, Rainer Gemulla

Max-Planck-Institut für Informatik
66123 Saarbrücken
{kbeedkar,corrogg,rgemulla}@mpi-inf.mpg.de

Abstract: Markov logic is a powerful tool for handling the uncertainty that arises in real-world structured data; it has been applied successfully to a number of data management problems. In practice, the resulting ground Markov logic networks can get very large, which poses challenges to scalable inference. In this paper, we present the first fully parallelized approach to inference in Markov logic networks. Inference decomposes into a grounding step and a probabilistic inference step, both of which can be cost-intensive. We propose a parallel grounding algorithm that partitions the Markov logic network based on its corresponding join graph; each partition is ground independently and in parallel. Our partitioning scheme is based on importance sampling, which we use for parallel probabilistic inference, and is also well-suited to other, more efficient parallel inference techniques. Preliminary experiments suggest that significant speedup can be gained by parallelizing both grounding and probabilistic inference.

1 Introduction

Real-world data is often inconsistent, noisy, or incomplete. Markov logic [RD06] is a recent, promising approach to handle such uncertainty in structured data. It has been employed successfully in a number of applications, including link prediction [RD06], entity resolution [SD06], information extraction [PD07], and ontology learning [PD10]. At its heart, Markov logic bridges the gap between first-order logic and probability theory: The former allows to encode and reason about deterministic information, the latter is well-suited to manage uncertainty.

A *Markov logic network* (MLN) is a set of first-order logic formulas called *rules*; each rule is associated with a numerical *weight*. For example, suppose that we are given a university database in which the `advisedBy` relation between students and professors is incomplete. MLNs allow us to use rules such as "if a student and a professor have a joint publication, then the student is advised by the professor" to approach this link prediction problem. This rule appears helpful but is inherently uncertain, i.e., it is true in many but not all cases. MLNs thus attach a weight (say, 2.5) to this rule; the weight is related to the confidence that instances of the rule are true. Formally, we obtain

$$2.5\colon \forall s.\forall p.\forall t.\, \texttt{student}(s) \wedge \texttt{professor}(p) \wedge \texttt{authorOf}(p,t) \wedge \texttt{authorOf}(s,t) \\ \implies \texttt{advisedby}(s,p).$$

Inference in Markov logic is performed by grounding the network using an *evidence database* of known facts and a set of constants. The output of this grounding step is a Markov network, on which probabilistic inference is performed subsequently. The evidence database and, even more so, the ground network can be very large. A (somewhat naive) grounding of the above rule with 1000 students, 10 professors, and 100 publications results in one million instances, connecting tens of thousands of variables (such as advisedBy(Anna, Bob)). In real-world applications, in which the involved datasets can be much larger, both grounding and probabilistic inference pose severe challenges to the scalability of Markov logic.

In this paper, we propose a fully parallel approach to scalable inference in Markov logic. Most prior work has focused on efficient grounding methods [SN09, MR10, NRDS11] or parallel probabilistic inference [GLGG11]. In contrast, our approach is holistic in that we parallelize both grounding and probabilistic inference. In more detail, we develop a simple yet effective technique to partition an MLN in such a way that each partition can be ground independently and in parallel (using a grounding method of choice). In addition to reducing grounding time, the resulting ground network is readily partitioned and well-suited for parallel probabilistic inference (again, using a method of choice). Our approach thus completely avoids expensive network partitioning and data redistribution steps after grounding.

The contributions of this paper are as follows: (1) We propose a framework for fully parallel inference in Markov logic networks. (2) We derive and analyze a parallel inference algorithm based on importance sampling. This algorithm may not be the best-performing choice in practice, but it provides valuable insight into how to obtain a good partitioning of a ground Markov logic network over a set of compute nodes. (3) We develop and analyze a practical algorithm for partitioning a ground Markov network based on minimum graph cuts. (4) Based on the insights obtained by our analysis, we propose a novel partitioning scheme for Markov logic networks. In contrast to prior work, we partition the network before we ground it. This approach avoids the need for partitioning and redistributing the ground network. (5) We present results of a preliminary experimental study on real-world data. Our results suggest that significant speedup can be gained by both parallelizing grounding and parallelizing probabilistic inference.

2 A Primer On Markov Logic Networks

Recall that a Markov logic network is a set of weighted rules (first-order logic formulas). Fig. 1 displays an excerpt of practical MLN used for predicting the advisedBy relation [RD06]; the example is very simple for expository reasons. The first rule states that advisees must be students and the second rule states that advisors must be professors or senior researchers. The higher weight of the second rule indicates that its instances are more likely to be satisfied than instances of the first rule.

To understand the semantics of a Markov logic network, we need to ground the network using a specific set of *constants*, i.e., students and professors. The

Rule	Weight	Formula
1	1.7	$\forall s.\forall p.\ \neg\texttt{student}(s) \implies \neg\texttt{advisedBy}(s,p)$
2	2.5	$\forall s.\forall p.\ \texttt{advisedBy}(s,p) \implies \texttt{hasPosition}(p,\texttt{Professor})$ $\vee\,\texttt{hasPosition}(p,\texttt{Senior Researcher})$

Figure 1: Excerpt of a Markov logic network for predicting the `advisedBy` relation

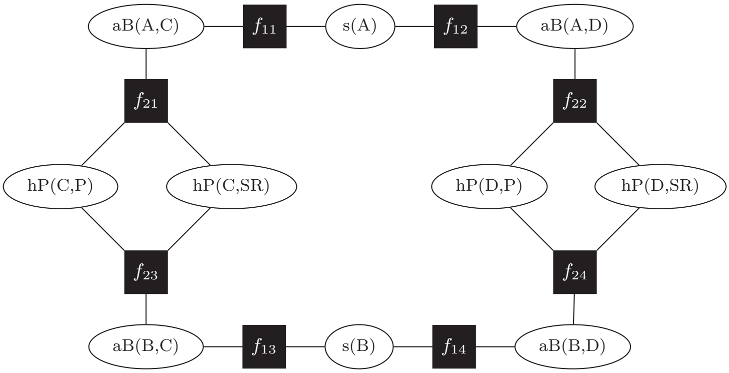


Figure 2: Factor graph representation of a ground MLN

output of the grounding process can be represented by a factor graph. A *factor graph* is a bipartite graph in which nodes correspond to either Boolean random variables (ovals) or factors (boxes). Each *Boolean variable* represents a grounding of an atom that occurs in an MLN rule. For example, suppose that Anna and Bob are students (domain of s) and that Charles and Debbie are professors (domain of p). We obtain the 10 Boolean variables shown in Fig. 2, e.g., $\texttt{s(A)}$ and $\texttt{aB(A,C)}$. Informally, *factors* represent groundings of MLN rules; see [RD06] for a formal definition. In our example, there are 8 factors; the two variables mentioned above are connected by factor f_{11}. Each factor can be seen as a function over the variables that it is connected to. Given a valuation of these variables, the factor outputs the (exponential of the) weight of its corresponding rule if satisfied under the valuation; otherwise the factor outputs 1. We obtain

$$f_{11}(\texttt{s(A)}, \texttt{aB(A,C)}) = \begin{cases} e^{1.7} & \text{if } \neg\texttt{s(A)} \implies \neg\texttt{aB(A,C)}, \\ 1 & \text{otherwise.} \end{cases}$$

Note that the exponentiation here implies that the value of a factor is always positive. Negative weights correspond to factor values less than one (rules that are "usually" wrong), zero weights correspond to factor value one (no influence), and positive weights correspond to factor values larger than one (usually true).

If the truth value of a ground atom is known from the evidence database, we clamp the value of the corresponding Boolean variable appropriately. These *evidence variables* form the "data" for inferring statistics of the unknown variables. If a predicate is ground under *closed-world semantics*, we additionally clamp the values of ground atoms that do not occur in the evidence database to false; such a grounding semantic is useful for predicates that are completely known. Note that the factor graph can be simplified by eliminating evidence variables; this simplification is usually performed directly during grounding for efficiency reasons. Under the alternate *open-world semantics*, the value of the ground atoms that do not occur in the database are not clamped. In our ongoing example, we use the open-world semantics and an empty evidence database for simplicity.

After grounding, we perform probabilistic inference to make statements about the probability distribution of the unknown Boolean variables. We refer to predicates that we are ultimately interested in as *query predicates* (here `advisedBy`); the corresponding Boolean variables are referred to as *query variables*. The factor graph defines a probability distribution that assigns a probability $P(\boldsymbol{x})$ to each world $\boldsymbol{x}$, i.e., to each distinct assignment of values to the unknown variables $\boldsymbol{X}$:

$$P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z_P} \prod_{f \in F} f(\boldsymbol{x}_f). \tag{1}$$

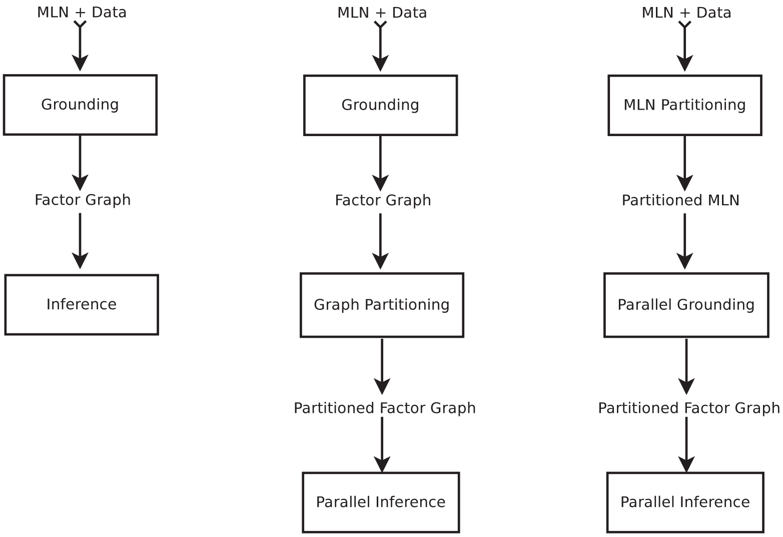
Here Z_P is a normalization constant that ensures summation to one, F denotes the set of all factors, and $\boldsymbol{x}_f$ denotes the values in $\boldsymbol{x}$ of the variables connected to factor f. Observe that the probability of a given world depends on both the number and the weights of the ground rules that are satisfied. If we increase the weight of a specific rule, then worlds that satisfy that rule become more likely while other worlds become less likely. In this paper, we focus on *marginal inference*, i.e., we want to infer the marginal distribution of each query variable (e.g., the probability that $\mathtt{aB}(\mathtt{A}, \mathtt{B})$ is true).

The meaningfulness of the result of probabilistic inference depends on the information captured in the MLN. In general, rules are created by domain experts or learned from training data. Weight assignment is very difficult for humans because different formulas correlate with each other; weights are thus almost always learned from training data. In this paper, we assume that we are given an already learned MLN. Nevertheless, our methods and techniques are also helpful for scaling up the learning process, which makes repeated use of an inference component.

A thorough and accessible treatment of Markov logic can be found in [RD06]; factor graphs and techniques for probabilistic inference are discussed in [KF09].

3 Related Work

A well-known implementation of Markov logic is Alchemy [KSRD05], which includes both learning and inference components. Inference is performed as described above: the MLN is grounded using the evidence database and a probabilistic inference algorithm is run on the resulting factor graph. This traditional approach is illustrated in Figure 3(a). A number of techniques have been proposed to speed up inference, including clustering of query literals [MR10], reducing the size of the ground network [SN09], incremental grounding [Rie08], in-database grounding [NRDS11], and task-specific probabilistic MAP inference [NZRS11]. In this



(a) Sequential (b) Partly parallel (c) Fully parallel

Figure 3: Comparison of approaches to parallel inference in Markov logic

paper, we propose a fully parallel framework for speeding up inference in Markov logic. Since our framework is oblivious to the actual grounding algorithm and (to a lesser extent) to the probabilistic inference algorithms being run at worker nodes, most of the techniques mentioned above still apply.

Parallel probabilistic inference has received a lot of attention in the literature [DVKMG09, ASW08, NASW07, NRDS11, GLGG11]. In the context of Markov logic, Tuffy [NRDS11] implements a partly parallel approach to inference. In contrast to the sequential approach taken by Alchemy, Tuffy runs a graph partitioning algorithm on the factor graph obtained by the grounding algorithms, and subsequently runs a simple parallel algorithm for probabilistic inference on the partitions; see Figure 3(b). Since Tuffy also uses efficient in-database grounding, it is highly efficient and scalable. A potential bottleneck of Tuffy is the graph partitioning step; finding a minimum-cost balanced partition is NP-hard even for quite simple MLNs [NRDS11]. In fact, Tuffy resorts to a simple heuristic algorithm because even state-of-the-art graph partitioners are too expensive in practice.

Our work is inspired by Tuffy but also parallelizes the grounding and graph partitioning steps. As shown in Fig. 3(c), we partition the Markov logic network *before grounding*. A key advantage of this approach is that the expensive graph partitioning step is performed on the small MLN (data independent) instead of on the large factor graph (data dependent). Each partition of the resulting partitioned MLN is ground independently and in parallel; the output of the grounding step is thus a readily partitioned factor graph. Our MLN partitioning is designed such that the partitioned factor graph is suitable for parallel probabilistic inference; our experiments suggest that this approach can outperform state-of-the-art graph partitioners in both speed and quality.

4 Parallel Probabilistic Inference

In general, exact probabilistic inference in ground Markov logic networks is intractable so that all existing implementations use some form of approximate inference; e.g., local search, belief propagation, or Markov Chain Monte Carlo (MCMC) sampling. In large applications, where factor graphs consist of millions of variables and factors, even approximate inference can be prohibitively expensive.

In this section, we show how to parallelize an arbitrary MCMC sampling technique via importance sampling. Our main objective is to gain understanding in how to partition a factor graph in a way that allows for efficient parallel inference. Our results are the basis for the MLN partitioning methods described in Sec. 5. Note that, in general, the MLN partitioning method needs to be designed for the specific probabilistic inference algorithm being used. As discussed later on, our framework is flexible enough to allow for a number of existing, potentially more efficient parallel inference methods [GLGG11].

4.1 Importance Sampling

In general, MCMC sampling techniques [KF09] explore the probability distribution of $\boldsymbol{X}$ by analyzing a subset $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(n)}$ of the possible worlds; these subsets are referred to as *samples*. MCMC methods differ in how these samples are generated; in general, these methods construct a Markov chain that consists of one state for each possible world and has a stationary distribution of (exactly or approximately) $P(\boldsymbol{X})$. Samples from the Markov chain are generally dependent, but the dependencies decrease the further apart the samples are taken. We thus assume that $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(n)}$ are independent samples from $\boldsymbol{X}$.

Given samples $\boldsymbol{X}^{(1)}, \boldsymbol{X}^{(2)}, \ldots, \boldsymbol{X}^{(n)}$, we can estimate the expected value of any function $h(\boldsymbol{X})$ over the possible worlds as follows:

$$\mu = \mathrm{E}\,[\, h(\mathbf{X})\,] \approx \frac{1}{n} \sum_{i=1}^{n} h(\boldsymbol{X}^{(i)}) = \hat{\mu}.$$

In our setting of marginal estimation, we use functions of form $h(\boldsymbol{X}) = I_X$ for some query variable $X \in \boldsymbol{X}$. Here, indicator I_X takes value 1 if X is true; otherwise it takes value 0. For example, if $h(\boldsymbol{X}) = I_{\mathtt{aB(A,C)}}$, then μ is equal to the probability that Anna is advised by Charlie. Our estimate $\hat{\mu}$ of μ is simply the fraction of samples in which Charlie advised Anna. Under our assumptions, $\hat{\mu}$ has variance

$$\mathrm{Var}\,[\,\hat{\mu}\,] = \mathrm{Var}\Big[\, \frac{1}{n} \sum_{i=1}^{n} h(\mathbf{X}^{(\mathbf{i})}) \,\Big] = \frac{\mathrm{Var}\,[\, h(\mathbf{X})]\,]}{n}, \tag{2}$$

which decreases linearly in the number n of samples.

Two aspects play a key role in selecting a sampling method: the time to obtain a sample $\boldsymbol{X}^{(i)}$ and the number of required samples to reach the desired accuracy.

Importance sampling (IS) is a method to increase efficiency by replacing the *target distribution* $P(\boldsymbol{X})$ by a new *proposal distribution* $Q(\boldsymbol{X})$. The increase in efficiency is obtained by either reducing the variance of the estimator (numerator of (2)) or by increasing the number of samples that can be obtained in a given amount of time (denominator). Since samples are taken from the "wrong" distribution $Q(\boldsymbol{X})$, we assign an *importance weight* $w(\boldsymbol{X}^{(i)}) = P(\boldsymbol{X}^{(i)})/Q(\boldsymbol{X}^{(i)})$ to each sample from the proposal distribution. We have

$$\mathrm{E}_P\,[\,h(\mathbf{X})\,] = \int h(\mathbf{x})P(\mathbf{x})\;\mathrm{d}\mathbf{x} = \int h(\mathbf{x})w(\mathbf{x})Q(\mathbf{x})\;\mathrm{d}\mathbf{x} = \mathrm{E}_Q\,[\,h(\mathbf{X})w(\mathbf{X})\,].$$

Here we use subscripts P and Q to indicate the distribution w.r.t. which the expectation is taken. Note that $\mathrm{E}_Q\,[\,w(\boldsymbol{X})\,] = 1$ and that, for example, $w(\boldsymbol{X}^{(i)}) > 1$ if possible world $\boldsymbol{X}^{(i)}$ is more likely under P than under Q. Thus the importance of sample $\boldsymbol{X}^{(i)}$ (when sampled from Q) is higher than average because we see the sample with lower probability than required.

The previous discussion assumes that the target distribution $P(\mathbf{X})$ is known. In the case of factor graphs, however, computing $P(\boldsymbol{x})$ for some possible world $\boldsymbol{x}$ is intractable in general because we do not know the normalization constant Z_P of Eq. (1). Instead, we make use of an *unnormalized distribution* $\tilde{P}(\boldsymbol{X}) = Z_P P(\boldsymbol{X}) = \prod_{f\in F} f(\boldsymbol{X}_f)$ (similarly $\tilde{Q}$). We obtain the *normalized IS estimator*

$$\hat{\mu}_{\mathrm{IS}} = \frac{\sum_{i=1}^{n} \tilde{w}(\mathbf{X}^{(i)})h(\mathbf{X}^{(i)})}{\sum_{i=1}^{n} \tilde{w}(\mathbf{X}^{(i)})} \approx \mathbb{E}_Q[h(\mathbf{X})], \tag{3}$$

where $\tilde{w}(\boldsymbol{X}^{(i)}) = \tilde{P}(\boldsymbol{X}^{(i)})/\tilde{Q}(\boldsymbol{X}^{(i)})$ are *unnormalized importance weights.* Estimator $\hat{\mu}_{\mathrm{IS}}$ is consistent,[1] asymptotically unbiased, and has approximate variance [KF09]

$$\mathrm{Var}_Q(\hat{\mu}_{\mathrm{IS}}) \approx \frac{(1 + \mathrm{Var}_Q[w(\mathbf{X})])\,\mathrm{Var}_P[h(\mathbf{X})]}{n}. \tag{4}$$

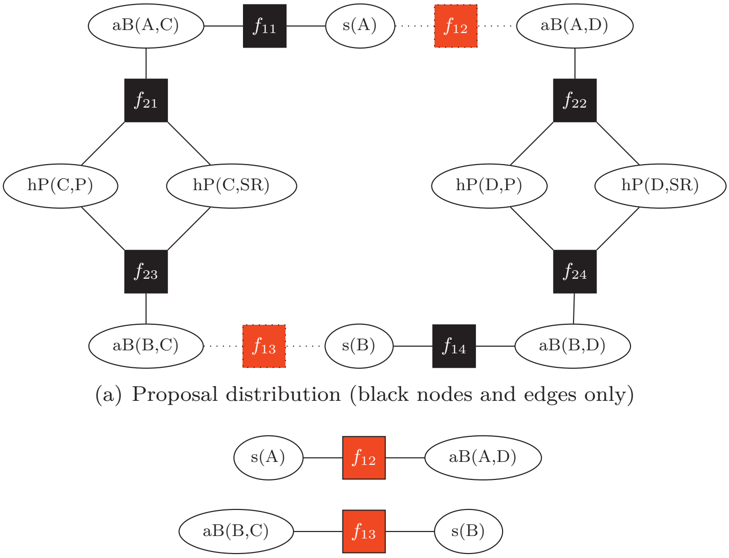
Thus the performance of estimator $\hat{\mu}_{\mathrm{IS}}$ is governed by (i) properties of the proposal and target distributions and (ii) the number of samples that can be taken in a given amount of time. Note that if $P = Q$, we obtain the standard Monte Carlo estimator and variance given in Eq. (2).

See [KF09] for a more thorough treatment of importance sampling.

4.2 Parallel Probabilistic Inference with Importance Sampling

Given a *target factor graph* $G_P = (V, F_P)$ obtained from grounding the MLN, we parallelize probabilistic inference by constructing a *proposal factor graph* $G_Q = (V, F_Q)$ from G_P in such a way that sampling from G_Q can be easily parallelized. Here G_P represents the target distribution P while G_Q represents the proposal distribution Q. Since G_P and G_Q can represent different distributions, we use

[1] Here we assume that $Q(\mathbf{x}) \neq 0$ whenever $P(\mathbf{x})h(\mathbf{x}) \neq 0$, which holds for our choice of Q.



(a) Proposal distribution (black nodes and edges only)

(b) Importance weight (cut)

Figure 4: Partitioning of the factor graph for the university network

importance sampling to ensure that our estimates of marginal probabilities are (asymptotically) correct.

Denote by k the number of available processors. We obtain G_Q by partitioning the set of variables V into k partitions $V_1, \dots, V_k$ such that $\bigcup_i V_i = V$ and $V_i \cap V_j = \emptyset$ for $i \neq j$, $1 \leq i, j \leq k$; we discuss how to actually obtain these partitions in subsequent sections. During the sampling process, processor i will be responsible for sampling the variables in partition V_i. For example, consider the ground university network reproduced in Fig. 4(a). Setting $k = 2$, a potential partitioning is given by setting

$$V_1 = \{\, \texttt{s(A)}, \texttt{aB(A,C)}, \texttt{aB(B,C)}, \texttt{hP(C,P)}, \texttt{hP(C,SR)} \,\}$$

and

$$V_2 = \{\, \texttt{s(B)}, \texttt{aB(A,D)}, \texttt{aB(B,D)}, \texttt{hP(D,P)}, \texttt{hP(D,SR)} \,\}.$$

We can now construct a factor graph $G_i = (V_i, F_i)$ for each partition by considering only a "local" subset of the factors from G_P, i.e., by setting

$$F_i = \{\, f \in F_P : \text{all neighbors of } f \text{ are contained in } V_i \,\}.$$

In our example, we have $F_1 = \{\, f_{11}, f_{21}, f_{23} \,\}$ and $F_2 = \{\, f_{22}, f_{24}, f_{14} \,\}$. The proposal factor graph G_Q is given by the union of the factor graphs for each partition, i.e., $F_Q = \bigcup_i F_i$. In our ongoing example, the proposal factor graph consists of the black nodes and edges in Fig. 4(a).

The partitioning approach described above ensures that in G_Q, there are no connections in between the set of variables in V_i and the set of variables in V_j,

$i \neq j$. One can show that this implies that the variables in V_i are probabilistically independent from the variables in V_j. This allows us to parallelize sampling from G_Q: Each processor i runs an arbitrary MCMC sampling scheme on factor graph G_i. To obtain a sample from the entire proposal distribution G_Q, we take the union of the samples produced by each processor (conceptually, see below).

We can view G_Q as a factor graph obtained from G_P by "dropping" some of the factors. Since these factors have no influence on the samples obtained from G_Q, we need to account for their effect via the importance weights. Denote by $G_{\text{cut}} = (V_{\text{cut}}, F_{\text{cut}})$ a *cut factor graph* that consists of the factors dropped from G_P and the variables directly connected to these factors; i.e.,

$$F_{\text{cut}} = F_P \setminus F_Q$$

and

$$V_{\text{cut}} = \{\, v \in V : \text{v is connected to at least one factor in } F_{\text{cut}} \,\}.$$

In our example, we have $F_{\text{cut}} = \{\, f_{12}, f_{13} \,\}$ and $V_{\text{cut}} = \{\, \texttt{s(A)}, \texttt{s(B)}, \texttt{aB(B,C)}, \texttt{aB(A,D)} \,\}$. The corresponding factor graph is shown in Fig. 4(b). To obtain the unnormalized importance weights, observe that

$$\begin{aligned}\tilde{w}(\boldsymbol{X}) &= \frac{\tilde{P}(\boldsymbol{X})}{\tilde{Q}(\boldsymbol{X})} = \frac{\prod_{f \in F_P} f(\boldsymbol{X}_f)}{\prod_{f \in F_Q} f(\boldsymbol{X}_f)} = \frac{\left(\prod_{f \in F_Q} f(\boldsymbol{X}_f)\right)\left(\prod_{f \in F_{\text{cut}}} f(\boldsymbol{X}_f)\right)}{\prod_{f \in F_Q} f(\boldsymbol{X}_f)} \\ &= \prod_{f \in F_{\text{cut}}} f(\boldsymbol{X}_f).\end{aligned}$$

Thus G_{cut} can be used to compute the importance weights using the values of only the variables in V_{cut}.

To summarize, we parallelize probabilistic inference as follows: We first partition the factor graph and distribute the partitions across the set of compute nodes. Each node computes a sample of its local factor graph using an arbitrary MCMC sampling method. After the samples have been obtained, we communicate the values of the cut variables (i.e., V_{cut}) to a coordinator node, which subsequently computes and broadcasts the unnormalized importance weight. Finally, each node updates the marginals of its local variables using Eq. (3).

4.3 The Optimal Partitioning

Recall Eq. (4), which defines the variance of the normalized IS estimator. For a fixed number n of samples, the variance depends on (1) the target distribution and (2) the variance of the importance weights $w(\boldsymbol{X})$. Since (1) is not affected by the partitioning, we subsequently focus on how to minimize (2). In our setting, we have

$$w(\boldsymbol{X}) = \frac{Z_Q}{Z_P} \tilde{w}(\boldsymbol{X}),$$

where Z_P and Z_Q are the normalization constants of G_P and G_Q, respectively. We obtain

$$\mathrm{Var}_Q[w(\boldsymbol{X})] = \frac{Z_Q^2}{Z_P^2}\,\mathrm{Var}_Q[\tilde{w}(\boldsymbol{X})] = \frac{\mathrm{Var}_Q[\tilde{w}(\boldsymbol{X})]}{E_Q^2[\tilde{w}(\boldsymbol{X})]} = \mathrm{CV}_Q^2(\tilde{w}(\boldsymbol{X})),$$

where $\mathrm{CV}_Q(\tilde{w}(\boldsymbol{X})) = \sqrt{\mathrm{Var}_Q[\tilde{w}(\boldsymbol{X})]}/E_Q[\tilde{w}(\boldsymbol{X})]$ denotes the *coefficient of variation* (CV) of $\tilde{w}(\boldsymbol{X})$. In the derivation, we used the fact that $E[w(\boldsymbol{X})] = 1$ and thus $E[\tilde{w}(\boldsymbol{X})] = Z_P/Z_Q$. To minimize the variance of the IS estimator $\hat{\mu}_{\mathrm{IS}}$, we thus need to minimize the CV of $\tilde{w}(\boldsymbol{X})$ under proposal Q. To do this, we need to infer the ratio Z_Q/Z_P and the joint distribution of the variables in V_{cut} (both of which depend on our choice of proposal), i.e., we need to run inference. Since our ultimate goal is to actually parallelize inference, such an approach is impractical.

The situation is further complicated by the fact that the partitioning that minimizes the CV is not necessarily the best performing one in practice. Recall that in order to compute the unnormalized importance weight, we need to obtain a sample from each of the partitions. If the partitions are balanced (e.g., equal number of variables and/or factors), parallelization is effective because every processor has roughly the same amount of work. If partitions are imbalanced, parallalization may not be effective since one of the processors may require significantly more time to obtain its sample than the other processors. Thus with k processors, we obtain a speedup somewhere in between 1 (no balancing) and k (perfect balancing).

We conclude that the optimal partitioning trades off statistical efficiency and parallelization benefits. In principle, we can determine this optimal partitioning given an appropriate model of sampling cost. However, such an approach is impractical: the benefits of parallelization are outweighed by the cost of determining the partitioning. For this reason, we settle for a “good” partitioning instead of the optimal one.

4.4 A Good Partitioning

To obtain a practicable procedure that finds a good but not necessarily the best partitioning, we (1) do not minimize but bound the CV of the unnormalized importance weights and (2) always create balanced partitions (i.e., we do not trade off variance and computational cost). With these relaxations, we are able to apply existing hypergraph partitioning algorithms to our problem. Moreover, these relaxations form the basis for the parallel grounding method described in Sec. 5.

Recall that the unnormalized importance weight is given by $\tilde{w}(\boldsymbol{X}) = \prod_{f \in F_{\mathrm{cut}}} f(\boldsymbol{X}_f)$. From our definition of factor functions, we know that f can only take two possible values: e^{w_f} (corresponding formula satisfied) or 1 (otherwise). Denote by $f_{\min}$ ($f_{\max}$) the smaller (larger) of the two values. Given a cut F_{cut}, we obtain

$$\tilde{w}_{\min} = \prod_{f \in F_{\mathrm{cut}}} f_{\min} \leq \tilde{w}(\boldsymbol{X}) \leq \prod_{f \in F_{\mathrm{cut}}} f_{\max} = \tilde{w}_{\max}.$$

To bound the CV of $\tilde{w}(\boldsymbol{X})$, we compute the largest CV realizable by any distribution on the interval $[\tilde{w}_{\min}, \tilde{w}_{\max}]$. We thus replace the actual distribution of

$\tilde{w}(\boldsymbol{X})$ by its worst-case distribution; this trick allows us to avoid running inference while still obtaining guarantees on the quality of estimation. One can show that the largest CV is realized by a distribution on $\{\tilde{w}_{\min}, \tilde{w}_{\max}\}$. Such a distribution is parameterized by a parameter $p_{\max}$ for the probability of observing $\tilde{w}_{\max}$; set $p_{\min} = 1 - p_{\max}$ and $\bar{w} = p_{\min}\tilde{w}_{\min} + p_{\max}\tilde{w}_{\max}$. We obtain

$$\mathrm{CV}_Q[\tilde{w}] \leq \max_{0 \leq p_{\max} \leq 1} \frac{\sqrt{p_{\min}(\tilde{w}_{\min} - \bar{w})^2 + p_{\max}(\tilde{w}_{\max} - \bar{w})^2}}{\bar{w}}. \tag{5}$$

One way to compute the above bound is to (conceptually) "normalize" the factor graph before running inference. The *normalized factor graph* can be obtained by replacing every factor function $f \in F$ by $f'(\boldsymbol{X}) = f(\boldsymbol{X})/f_{\min}$; the distribution represented by the factor graph is not affected by such a scaling. After normalization, we have $\tilde{w}'_{\min} = 1$ so that the bound of (5) depends only on $\tilde{w}'_{\max}$. In fact, the *normalized bound* is monotonically increasing in $\tilde{w}'_{\max}$ so that we pick the partitioning that minimizes $\tilde{w}'_{\max} = \prod_{f' \in F'_{\mathrm{cut}}} f'_{\max}$. If the so-obtained bound on the CV is close to 0, importance sampling is guaranteed to be statistically efficient. In contrast, if the bound is large, the performance of IS depends on the distribution of the cut variables; IS may or may not be effective.

We can use existing weighted hypergraph partitioning algorithms to find the partitioning that minimizes $\tilde{w}'_{\max}$. A hypergraph partitioning algorithm [KL70] finds a partitioning of the variables such that the sum of the weights of the hyperedges that cross partitions is minimized. Note that minimizing $\tilde{w}'_{\max}$ is equivalent to minimizing $\log(\tilde{w}'_{\max}) = \sum_{f' \in F'_{\mathrm{cut}}} \log(f'_{\max})$, which is in summation form. To construct the *weighted hypergraph* for $G = (V, F)$, replace each factor $f \in F$ by a hyperedge that connects the variables in $\boldsymbol{X}_f$; the hyperedge is assigned weight $\log(f'_{\max})$. The minimum cost hypergraph partitioning is exactly the partitioning with the smallest possible bound on the CV. Note that the use of hypergraph partitioning for probabilistic inference is not novel (e.g., see [NRDS11]). To the best of our knowledge, however, the connection between hypergraph partitioning and the effectiveness of parallel probabilistic inference has not been established before.

In order to facilitate parallel processing, we modify the above approach and search for a *balanced* partitioning instead of an arbitrary partitioning. The perhaps simplest possible approach—which we also used in our experiments—is to require that each partition contains roughly the same number of variables. The hope is that the cost of obtaining a sample is then roughly the same for each partition. A key advantage of this particular balancing condition is that it is directly supported by most hypergraph partitioners. In fact, our experiments suggest that the so-obtained partitions work well in practice.

5 Parallel Grounding

Since both grounding and graph partitioning can be expensive, we develop an MLN partitioning technique that significantly reduces partitioning cost and also parallelizes the grounding step. The key idea of our approach is to partition

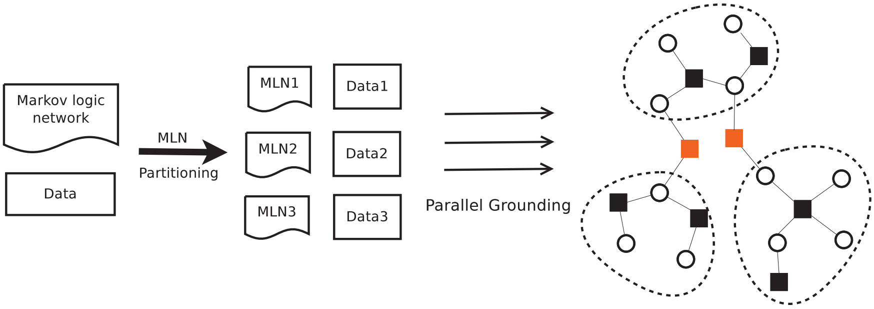


Figure 5: MLN partitioning and parallel grounding ($k = 3$).

the MLN directly—i.e., before grounding—and to ground the resulting partitioned MLNs in parallel. The MLN partitioning step draws from ideas from the parallel databases; it exploits the connection between grounding an MLN and computing a set of database joins [NRDS11]. Fig. 5 illustrates our approach.

5.1 Grounding via Database Queries

To establish the connection between grounding and database joins, recall the university MLN of Fig. 1 as well as its grounding shown in Fig. 2. The first rule of the MLN states that advisees must be students and involves predicates $\texttt{student}(s)$ and $\texttt{advisedBy}(s, p)$. Suppose that we create a *hypothetical database relation* for each predicate that occurs in the MLN; a predicate's relation contains a tuple for each ground instance of the predicate as well as a globally unique *identifier*. For students Anna and Bob, and professors Charles and Debbie, we obtain

$$\begin{aligned}\texttt{Student}(\underline{sid}, s_1) &= \{\, (1, \texttt{Anna}),\ (2, \texttt{Bob}) \,\}\,,\\ \texttt{AdvisedBy}(\underline{aBid}, s_2, p_1) &= \{\, (3, \texttt{Anna}, \texttt{Charles}),\ (4, \texttt{Anna}, \texttt{Debbie}),\ (5, \texttt{Bob}, \texttt{Charles}),\\ &\qquad (6, \texttt{Bob}, \texttt{Debbie}) \,\}.\end{aligned}$$

Observe that these relations precisely capture the set of variables in the ground MLN corresponding to the student and advisedBy predicates, and that each variable is assigned a unique identifier. To additionally capture the factors that connect these variables, we perform a database join for each clause in the MLN and project to the variable identifiers. The join condition consists of the set of equality predicates induced by the MLN rule. For example, the first rule of Fig. 1 is given by $\texttt{s}(s) \vee \neg\texttt{aB}(s)$ (here in CNF). The corresponding database query and result is

$$\begin{aligned}\texttt{R1} = \pi_{sid,aBid}(\texttt{Student} \bowtie_{s_1=s_2} \texttt{AdvisedBy}) &= \{\, (1,3),\ (1,4),\ (2,5),\ (2,6) \,\}\\ &= \{\, f_{11},\ f_{12},\ f_{13},\ f_{14} \,\}\,.\end{aligned}$$

Here R1 contains a single tuple for each factor obtained from grounding the first rule; a factor's tuple consists of the variable identifiers of its arguments.

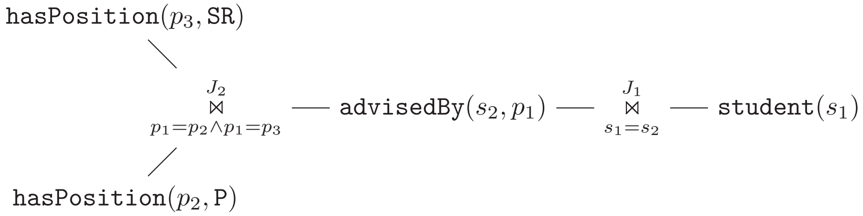


Figure 6: Join graph for the MLN of Fig. 1.

To summarize, we (conceptually) ground an MLN by instantiating a set of relations (corresponding to the ground variables) and computing a database join for each clause of the MLN (corresponding to the factors). This grounding process can be described with a *join graph* $G = (\mathscr{R}, \mathscr{J})$, where $\mathscr{R} = \{ R_1, \ldots, R_n \}$ denotes a set of relations and $\mathscr{J} = \{ J_1, \ldots, J_m \}$ denotes a set of joins (factors). Each $J \in \mathscr{J}$ is a hyperedge annotated with a join condition; the hyperedge connects to the relations occurring in the join condition. Fig. 6 shows the join graph for our running example; here we use predicates instead of relations for brevity. Note that in the foregoing discussion, we have ignored the evidence database and assumed open-world semantics. When an evidence database is given and grounding is performed under closed-world semantics, we directly use the relations from the evidence database (which have an additional Boolean `Value` attribute) instead of the hypothetical relations. To simplify exposition, however, we continue to focus on open-world grounding without an evidence database unless stated otherwise.

5.2 Partitioning a Markov Logic Network

Parallel database systems make use of vertical and horizontal partitioning techniques to parallelize query processing. In the following, we discuss how these techniques can be applied to the (hypothetical) relations of an MLN to allow for parallel grounding. We focus on horizontal partitioning throughout; similar techniques can be used to handle vertical partitioning.

A *horizontal partitioning* of a relation R w.r.t. a set A of its attributes is a set $R_1, \ldots, R_k$ of relations such that $\bigcup_i R_i = R$ and $\pi_A(R_i) \cap \pi_A(R_j) = \emptyset$ whenever $i \neq j$, where $1 \leq i, j \leq k$. For example, when $k = 2$, a horizontal partitioning of the AdvisedBy relation w.r.t. attribute p is given by $\texttt{AdvisedBy}_1 = \{ (3, \texttt{A}, \texttt{C}), (5, \texttt{B}, \texttt{C}) \}$ and $\texttt{AdvisedBy}_2 = \{ (4, \texttt{A}, \texttt{D}), (6, \texttt{B}, \texttt{D}) \}$. Such a partitioning can be performed at the MLN level, i.e., before grounding. Continuing the example, we obtain two new predicates $\texttt{advisedBy}_1(s, p_{11})$ and $\texttt{advisedBy}_2(s, p_{12})$ with domains $\mathrm{dom}(p_{11}) = \{ C \}$ and $\mathrm{dom}(p_{12}) = \{ D \}$. Horizontal partitioning allows us to spread the ground variables of a single relation across multiple compute nodes (partition i is stored on node i). In fact, if the partitioning is performed at the MLN level, we can ground each partition in parallel directly at its respective compute node.

Our aim is to partition the relations in a way amenable to parallel probabilis-

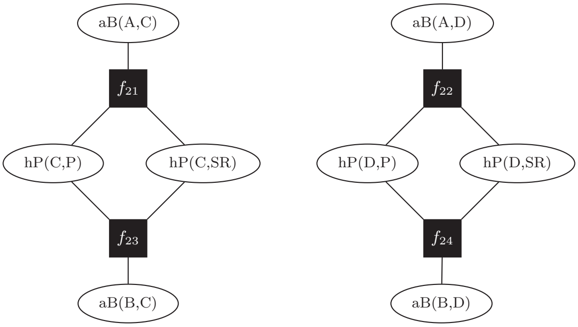


Figure 7: Partial factor graph obtained from grounding rule 2 of Fig. 1

tic inference, i.e., to minimize the number of factors spanning multiple partitions in addition to balancing the number of variables per partition. For example, consider the factor graph obtained when grounding only the second rule of our example MLN of Fig. 1. As can be seen in Fig. 7, this factor graph consists of two connected components, i.e., two subgraphs that do not share any factors. Using horizontal partitioning, we can identify and exploit the existence of multiple connected components directly at the rule level. The key idea is to make use of the co-partitioning techniques used in parallel databases: Two relations R_1 and R_2 are *co-partitioned* with respect to a set of attributes A_1 and A_2 (of equal domains), respectively, if R_l is horizontally partitioned w.r.t. A_l into $R_{l1}, \ldots, R_{lk}$, $l \in \{1, 2\}$, and $\pi_{A_1}(R_{1i}) \cap \pi_{A_2}(R_{2j}) = \emptyset$ for $1 \leq i, j \leq k$ and $i \neq j$. For example, suppose that we partition relation

$$\texttt{HasPosition}(hPid, p_2, v) = \{ (7, \texttt{C}, \texttt{P}), (8, \texttt{C}, \texttt{SR}), (9, \texttt{D}, \texttt{P}), (10, \texttt{D}, \texttt{SR}) \}$$

on attribute p to obtain relations $\texttt{HasPosition}_1 = \{ (7, \texttt{C}, \texttt{P}), (8, \texttt{C}, \texttt{SR}) \}$ and $\texttt{HasPosition}_2 = \{ (9, \texttt{D}, \texttt{P}), (10, \texttt{D}, \texttt{SR}) \}$. Then relations AdvisedBy (see above) and HasPosition are co-partitioned w.r.t. attributes p_1 and p_2, respectively.

If R_1 and R_2 are co-partitioned on A_1 and A_2, we can perform join $J = R_1 \bowtie_{A_1=A_2} R_2$ *locally* at each compute node: Each node i computes $R_{1i} \bowtie_{A_1=A_2} R_{2i}$. This fact is heavily exploited for efficient join processing in parallel database systems. In our setting, horizontal partitions consist of sets of ground variables and join results of sets of factors and their arguments. Suppose that join J above corresponds to some MLN rule. Since R_1 and R_2 are co-partitioned w.r.t. to the join attributes, we can not only compute join J locally at each compute node, but also guarantee that the local join results contain only local variables. In other words, all factors produced by a co-partitioned join will be local.

Our MLN partitioning technique aims to co-partition the relations in an MLN such that the number (or sum of weights) of the factors that span partitions is minimized. Since there are generally several formulas in which each relation occurs, and since we cannot partition a relation on multiple sets of attributes, we may not be able to find a co-partitioning that localizes all joins. We thus aim to find a good partitioning, i.e., one in which "important" joins are localized. Our partitioning technique is based on the following insight: The number of ground variables or factors of a predicate or rule, respectively, is given by the size of its corresponding relation or join. In our example, there are $|\texttt{Student}| = 2$ ground variables for `student` predicate, $|\texttt{AdvisedBy}| = 4$ ground variables for the `advisedBy` predicate, and $|\texttt{R1}| = 4$ factors for the first rule. Cardinality estimation techniques from the database literature allow us to estimate these quantities accurately; we can thus estimate the size of a factor graph for a given MLN and, more importantly, for a given horizontal partitioning on an MLN.

Revisiting the example of Fig. 6, the total number of factors in the factor graph is given by $|J_1| + |J_2|$. If we co-partition `aB` and `hP` on attribute p, then J_2 becomes local. In doing so, however, we cannot co-partition `aB` and `s` so that join J_1 is not local. Vice versa, if we choose a co-partitioning that localizes J_1, join J_2 will be non-local. The optimum choice of partitioning depends on the join sizes $|J_1|$ and $|J_2|$, as well as on the weights associated with the corresponding MLN rules. Intuitively, we want to localize joins with high cardinality and high weight; joins with low cardinality and low weight do not incur a high cost when not localized.

As indicated above, our partitioning method (described in detail in the next section) relies on accurate join size estimates. There exists a vast amount of literature on join size estimation [SS94], which can be readily exploited. When we use open-world grounding, however, join size estimation is particularly simple. Recall that under open-world grounding, all relations are complete, i.e., they consist of every possible tuple. The size of a complete relation R with attributes $A_1, \ldots, A_n$ is given by $|\mathrm{dom}(A_1)||\mathrm{dom}(A_2)| \cdots |\mathrm{dom}(A_n)|$. When we compute the join $J = R_1 \bowtie_{R_1.A=R_2.A} R_2$ of two complete relations R_1 and R_2, the resulting join size is given by $|J| = |R_1||R_2|/|\mathrm{dom}(A)|$. When R_1 and R_2 are both partitioned into k equally-sized partitions, and at least one of the two relations is not partitioned on A, then the number of local factors is given by $|J|/k$ and consequently the number of non-local factors by $|J|(1 - 1/k)$. Of course, join size estimation is much more involved when grounding is performed under closed-world semantics. The simplest way to handle closed-world grounding is to use open-world join size estimation as described above, but to use closed-world semantics when actually grounding each partition. We use this simple approach in our experiments; more elaborate join size estimation techniques are likely to further improve our results.

5.3 Finding a Good Partitioning

To find a good MLN partitioning, we encode the partitioning problem as a 0/1 integer linear program (ILP), which we solve using a standard ILP solver.

In what follows, we use index i for the join edges, index j for attributes, and index l for relations of the join graph. Denote by $A(R_l)$ the set of attributes in relation R_l and by $R(J_i)$ the multiset of relations occuring in join J_i. We assume that $A(R_{l_1}) \cap A(R_{l_2}) = \emptyset$ whenever $l_1 \neq l_2$. We encode each join edge J_i via a set C_i as follows: For each pair of R_{l_1} and R_{l_2} in $R(J_i)$, C_i contains the set E_{l_1,l_2} of pairs of attributes that are equalized by the join condition. Each pair contains one attribute from R_{l_1} and one from R_{l_2}. In our running example, we have $R(J_2) =$

$\{\,\texttt{hP}(p_2,\texttt{SR}),\texttt{hP}(p_3,\texttt{P}),\texttt{aB}(s_2,p_1)\,\}$ and $C_2 = \{\,\{\,(p_2,p_3)\,\},\{\,(p_1,p_3)\,\},\{\,(p_2,p_3)\,\}\,\}$. Note that we treat $\texttt{hP}(p_2,\texttt{SR})$ and $\texttt{hP}(p_2,\texttt{P})$ as different relations since they are disjoint. Denote by x_{A_j} a 0/1 variable that takes a value 1 if the relation R_l that contains attribute A_j (i.e., $A_j \in A(R_l)$) is going to be partitioned on A_j. Furthermore, denote by y_{J_i} a 0/1 variable that takes a value 1 if J_i is localized by the partitioning $\{\,x_{A_j}\,\}$. In our example, we have

$$y_{J_2} = [(x_{p_2} \wedge x_{p_3}) \wedge (x_{p_1} \wedge x_{p_3}) \wedge (x_{p_2} \wedge x_{p_3})].$$

Generally, denote by $\mathscr{P}_i$ a set of sets of attribute pairs, where each set of attribute pairs is obtained by selecting a single element of each $E_{l_1,l_2} \in C_i$ (in our example, $\mathscr{P}_2 = C_2$). Only sets of pairs that mutually share at least one attribute are included in $\mathscr{P}_i$. Then the optimal partitioning can be obtained by solving the ILP

$$\begin{aligned}
\max \quad & \sum_{J_i} |J_i| \cdot w_{J_i} \cdot y_{J_i} && && (1)\\
\text{s.t.} \quad & \sum_{A_j \in A(R_l)} x_{A_j} \le 1 && \text{for all } R_l, && (2)\\
& y_{J_i} = \bigvee_{P \in \mathscr{P}_i} \Big[\bigwedge_{(A_{j_1},A_{j_2}) \in P} (x_{A_{j_1}} \wedge x_{A_{j_2}}) \Big] && \text{for all } J_i, && (3)\\
& x_{A_j} \in \{\,0,1\,\} && \text{for all } A_j. &&
\end{aligned}$$

The objective function (1) maximizes the weighted sum of the sizes of local joins; weight w_{J_i} is taken from the MLN rule corresponding to join J_i. One can show that this objective function is equivalent to minimizing the number of non-local factors. Depending on the particular probabilistic inference algorithm being used, other objective functions may be more suitable (e.g., sum of joins sizes or number of variables with at least one non-local factor). Condition (2) ensures that every relation is partitioned on only one attribute (the ILP can be extended to support partitioning on multiple attributes). Finally, condition (3) computes the y_{J_i} variables using Boolean formulas. The formulas can be converted to a pure ILP by introducing appropriate helper variables.

Once the ILP has been solved, we horizontally partition the relations based on the solution. In more detail, we split the domain of each partitioning attribute into k equally sized parts; e.g., using range-based or hash partitioning. Relations that are not partitioned are randomly distributed across the compute cluster; this ensures that the number of query variables is the same across nodes so that partitions are balanced.[2] Note that the cost of our MLN partitioning strategy is virtually independent of both k and the domain size, which ensures good scalability.

6 Experimental Evaluation

We conducted a number of preliminary experiments to gauge the viability of our fully parallel approach to MLN inference. In particular, we (1) compared

[2] An alternative approach is to extend the ILP with support for vertical partitioning.

existing graph partitioning algorithms with our MLN partitioning method in terms of both computational cost and result quality, (2) investigated whether bounding the CV of the unnormalized importance weights indeed leads to good partitionings, and (3) whether parallel probabilistic inference reduces overall cost. Our results provide initial justification for our approach.

6.1 Experimental Setup

We implemented a prototype system that consists of components for MLN partitioning, grounding, and parallel probabilistic inference. All algorithms were implemented in C++ using pthreads for parallel processing; during probabilistic inference, the partitioned factor graph resided in main memory and each partition was processed by a single core. We used the GNU Linear Programming Kit (GLPK) to solve the ILP. All experiments were run on a machine with an 2.4GHz Intel Xeon CPU E5530 (8 cores) and 48GB of main memory.

Most of our experiments were conducted on the UW-CSE dataset.[3] This real-word dataset contains information about the Department of Computer Science and Engineering of the University of Washington, including relationships between professors, students, courses, and publications. The dataset is associated with an MLN that aims to predict the `advisedBy` and `tempAdvisedBy` relationship; the MLN consists of 22 predicates and 94 clauses. From the MLN, we removed 6 clauses because they contain an existential quantifier; these clauses are not yet supported by our prototype. We learned the weights of the rules in the modified MLN using Alchemy as described in [RD06]. After grounding (AI dataset), the factor graph consists of roughly 9000 query variables and 1M factors. Although the UW-CSE dataset is relatively small, the benefits of parallel processing did materialize in our experiments.

6.2 MLN Partitioning

In our first experiment, we evaluated the computational cost and quality of partitioning of both traditional ground-first methods and our proposed partition-first approach. For the former, we used two different hypergraph partitioners: (1) PaToH,[4] a state-of-the-art hypergraph partitioner and (2) the graph partitioning heuristic used by Tuffy [NRDS11].[5] The experiments were conducted on the UW-CSE dataset with $k = 2$ and $k = 4$ partitions.

Quality. We used a number of metrics to evaluate the quality of the partitioned factor graph: (i) the number $|F_{\text{cut}}|$ of factors in the cut (measure of the computational cost for weight computation), (ii) the sum $\log(\tilde{w}_{\max})$ of the loga-

[3] `http://alchemy.cs.washington.edu/data/uw-cse`

[4] `http://bmi.osu.edu/~umit/software.html#patoh`

[5] The Tuffy heuristic sorts all factors by descending weight. It then scans the factors sequentially and tries to assign the variables of the current factor to the current partition (if not yet assigned). If the partition size would exceed some threshold by doing so, Tuffy finalizes the current partition and starts a new, empty partition.

k	Approach	Factors in cut	Weight of cut	Balancing	Runtime
$k = 2$	PaToH	4678	1109.04	0.000	948.288s
	Tuffy	4686	1108.66	0.000	1.092s
	MLN part.	4690	1108.47	0.000	0.003s
$k = 4$	PaToH	63001	64500.40	0.012	952.254s
	Tuffy	7040	1662.46	0.000	1.288s
	MLN part.	7023	1662.84	0.000	0.003s

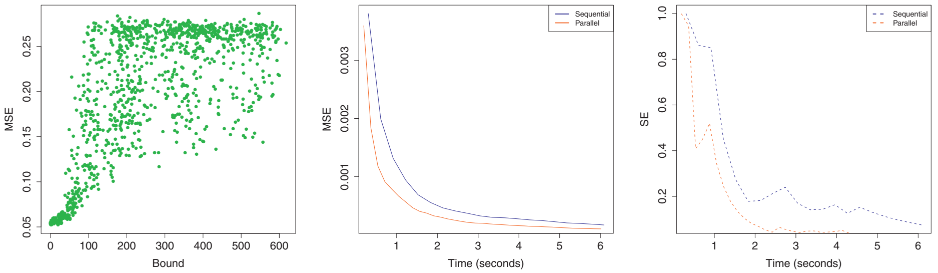
Table 1: Comparison of various graph partitioning approaches on the UW-CSE dataset

rithmic weights of these factors (measure of effectiveness of importance sampling), and (iii) the CV of the partition sizes in terms of number of variables (measure of balancing, 0 indicates perfect balancing). Our results are shown in Table 1. For $k = 2$ partition, PaToH, Tuffy, and our MLN partitioning approach give similar results. For $k = 4$, however, PaToH does not find a good partitioning, whereas Tuffy and our proposed MLN approach perform much better. Again, Tuffy and MLN partitioning perform comparably. Perhaps surprisingly, the simple Tuffy heuristic worked extremely well on the UW-CSE dataset. We conclude that MLN partitioning can indeed find good graph partitionings and may even outperform state-of-the-art partitioners.

Computational cost. We focus on the computational cost of computing the factor graph partitioning in terms of wall-clock time, i.e., we exclude the time for actually grounding the network. Table 1 shows our results. We found that PaToH is two orders of magnitude slower than Tuffy, which in turn is orders of magnitude slower than MLN partitioning. The excessive runtime of PaToH is due to the fact that PaToH is a generic hypergraph partitioner; it is unaware of and thus cannot exploit the structure of the ground factor graph. Tuffy is significantly faster because it uses an inexpensive heuristic partitioning algorithm. It is a viable option on datasets for which this heuristic works well. In both approaches, we need to distribute the partitions across the compute cluster after grounding to perform parallel probabilistic inference. This data distribution step is avoided by our MLN partitioning approach because partitions are grounded directly at their respective compute nodes. Moreover, since MLN partitioning is performed on the MLN level, it is independent of the size of the evidence database and thus much faster than the ground-first methods.

6.3 Parallel Probabilistic Inference

Quality. We conducted a simple experiment to investigate whether our bounding strategy for the CV of the unnormalized importance weights is effective. We generated a set of 1100 random graphs with 1000 variables and 1000 factors each. Each factor was connected to between 1 and 3 variables picked at random; the weights were sampled from a Normal$(0.67, 25)$ distribution. We ran a



(a) Effectiveness of bound (synthetic data)

(b) Average MSE (UW-CSE)

(c) Maximum SE (UW-CSE)

Figure 8: Sequential and parallel probabilistic inference (4 threads)

large number of sampling steps (10000) of a Gibbs sampler [KF09]; the resulting marginal probabilities were taken as ground truth. We then randomly partitioned the variables into two equally-sized sets and computed (1) the corresponding bound on the CV and (2) the mean square error (MSE) of the estimated marginal probabilities obtained by running a fixed number of 10000 importance sampling steps. Fig. 8(a) plots the bound vs. the MSE; each data point corresponds to one random partitioning. As can be seen, the MSE is small when the CV bound is small, which indicates that our bounding strategy is effective. Moreover, the MSE varies significantly when the CV bound is large. The reason for this behavior is that the bound is based on a worst-case distribution; when the actual distribution is far from worst-case, we may obtain better results than suggested by the bound.

Computational cost. We evaluated parallel probabilistic inference using importance sampling on $k = 4$ partitions; the partitions have been created using MLN partitioning as described in the previous section. As before, we obtained the ground truth by running a large number of Gibbs sampling steps (250k steps on each variable) on the unpartitioned factor graph. Fig. 8 plots the average MSE as well as the highest square error (SE) on an individual variable for both a sequential sampler and a parallel sampler with 4 threads. In both cases, the parallel inference method converged much faster than the sequential method so that importance sampling was effective.[6]

7 Conclusion and Future Work

We proposed a fully parallel approach to inference in Markov logic networks. In contrast to prior work, we parallelized not just the final probabilistic inference step, but also the intermediate grounding and graph partitioning steps. In more detail, we described and analyzed a simple parallel probabilistic inference algorithm based on importance sampling. Our analysis clarifies the connection between importance

[6] Note that this particular dataset is characterized by fast mixing times, which makes even the sequential sampler unusually fast.

sampling on factor graphs and graph partitioning. Since graph partitioning can be expensive, we leveraged ideas from parallel database systems to partition the Markov logic network itself, i.e., before grounding the network. Our MLN partitioning technique reduces partitioning time by multiple orders of magnitude, while producing partitionings competitive to state-of-the-art graph partitioners. Our experiments give initial evidence that both MLN partitioning and parallel probabilistic inference can speed up inference in Markov logic networks significantly. Future work includes better handling of closed-world semantics, better sampling methods, and a fully distributed implementation of our approach.

References

[ASW08] A. Asuncion, P. Smyth, and M. Welling. Asynchronous Distributed Learning of Topic Models. In *NIPS*, 2008.

[DVKMG09] F. Doshi-Velez, D. Knowles, S. Mohamed, and Z. Ghahramani. Large Scale Nonparametric Bayesian Inference: Data Parallelisation in the Indian Buffet Process. In *NIPS*, 2009.

[GLGG11] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin. Parallel Gibbs Sampling: From Colored Fields to Thin Junction Trees. *Journal of Machine Learning Research*, 2011.

[KF09] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[KL70] B. W. Kernighan and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell system technical journal*, 1970.

[KSRD05] S. Kok, P. Singla, M. Richardson, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, Department of Computer Science and Engineering, University of Washington, 2005.

[MR10] L. Mihalkova and M. Richardson. Speeding up inference in statistical relational learning by clustering similar query literals. In *ILP*, 2010.

[NASW07] D. Newman, A. Asuncion, P. Smyth, and M. Welling. Distributed Inference for Latent Dirichlet Allocation. In *NIPS*, 2007.

[NRDS11] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: scaling up statistical inference in Markov logic networks using an RDBMS. *PVLDB*, 2011.

[NZRS11] F. Niu, C. Zhang, C. Ré, and J. Shavlik. Felix: Scaling Inference for Markov Logic with an Operator-based Approach. *CoRR*, 2011.

[PD07] H. Poon and P. Domingos. Joint Inference in Information Extraction. In *AAAI*, 2007.

[PD10] H. Poon and P. Domingos. Unsupervised ontology induction from text. In *ACL*, 2010.

[RD06] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 2006.

[Rie08] S. Riedel. Improving the Accuracy and Efficiency of MAP Inference for Markov Logic. In *UAI*, 2008.

[SD06] P. Singla and P. Domingos. Entity Resolution with Markov Logic. In *ICDM*, 2006.

[SN09] J. Shavlik and S. Natarajan. Speeding up inference in Markov logic networks by preprocessing to reduce the size of the resulting grounded network. In *IJCAI*, 2009.

[SS94] Arun N. Swami and K. Bernhard Schiefer. On the Estimation of Join Result Sizes. In *EDBT*, 1994.