Tandem Inference: An Out-of-Core Streaming Algorithm for Very Large-Scale Relational Inference

Sriram Srinivasan[*]
UC Santa Cruz
ssriniv9@ucsc.edu

Eriq Augustine[*]
UC Santa Cruz
eaugusti@ucsc.edu

Lise Getoor
UC Santa Cruz
getoor@ucsc.edu

Abstract

Statistical relational learning (SRL) frameworks allow users to create large, complex graphical models using a compact, rule-based representation. However, these models can quickly become prohibitively large and not fit into machine memory. In this work we address this issue by introducing a novel technique called *tandem inference* (TI). The primary idea of TI is to combine grounding and inference such that both processes happen in tandem. TI uses an out-of-core streaming approach to overcome memory limitations. Even when memory is not an issue, we show that our proposed approach is able to do inference faster while using less memory than existing approaches. To show the effectiveness of TI, we use a popular SRL framework called Probabilistic Soft Logic (PSL). We implement TI for PSL by proposing a gradient-based inference engine and a streaming approach to grounding. We show that we are able to run an SRL model with over 1B cliques in under nine hours and using only 10 GB of RAM; previous approaches required more than 800 GB for this model and are infeasible on common hardware. To the best of our knowledge, this is the largest SRL model ever run.

1 Introduction

Statistical relational learning (SRL) (Richardson and Domingos 2006; Getoor and Taskar 2007; Raedt and Kersting 2011) is an effective method of combining weighted first-order logic with probabilistic inference to make high-quality, structured predictions. A characterizing trait of SRL is the ability to generate large graphical models from a small set of logical templates (*rules*). Several different SRL frameworks have been proposed, each exploring different types of graphical models, inference algorithms, or hyperparameter learning methods (Richardson and Domingos 2006; Bach et al. 2017; Venugopal, Sarkhel, and Gogate 2016). SRL methods have achieved state-of-the-art results in a varied set of domains such as image classification (Aditya, Yang, and Baral 2018), activity recognition (London et al. 2013), natural language processing (Ebrahimi, Dou, and Lowd 2016), recommender systems (Kouki et al. 2015), knowledge graphs (Pujara et al. 2013), diagnosis of physical systems (Chen, Chen, and Qian 2014), and information retrieval (Srinivasan et al. 2019b).

While SRL methods have seen a great deal of success, they face a major challenge when scaling to large datasets. The benefit of easily generating large graphical models from a few template rules can turn into a curse as the graphical models can quickly grow to intractable sizes that do not fit in memory. Consider a simple transitive rule: $Link(A, B) \wedge Link(B, C) \rightarrow Link(A, C)$, commonly used in conjunction with link prediction tasks. When this rule is instantiated (*grounded*) with a simple dataset of 1000 entries, a graphical model with a billion potentials is generated.

To address this issue, several approaches have been proposed. *Lifted inference* (de Salvo Braz, Amir, and Roth 2005; Singla and Domingos 2008; den Broeck et al. 2011; Kersting 2012; Sarkhel et al. 2014; Kimmig, Mihalkova, and Getoor 2015; Srinivasan et al. 2019a) is a commonly employed and effective approach that exploits symmetry in the data to generate a more compact model on which to perform inference. While effective in many settings, a key drawback of these approaches is that evidence or noisy data can break symmetries, making lifting less effective. To address this issue, *approximate lifting* approaches have also been proposed (Sen, Deshpande, and Getoor 2009; den Broeck, Choi, and Darwiche 2012; Venugopal and Gogate 2014; Das et al. 2019) which exploit approximate symmetries, allowing for greater and more robust compression. However, if the ground model lacks significant symmetry, approximate lifting may improve tractability only at the cost of correctness.

Several approaches orthogonal to lifted inference have also been proposed that attempt to perform efficient grounding by utilizing hybrid database approaches (Niu et al. 2011), exploiting the structure of rules (Augustine and Getoor 2018), perform efficient approximate counting for faster inference (Venugopal, Sarkhel, and Gogate 2015; Sarkhel et al. 2016; Das et al. 2016), or distributing models across multiple machines (Magliacane et al. 2015). However these methods, while quite useful, only provide partial solutions to grounding and efficient inference at scale. The hybrid database approach increases runtime substantially, ex-

[*]These authors contributed equally to this work.

ploiting rule structure requires large amounts of memory to store the ground model, approximating counting methods are applicable to discrete graphical models only, and distributing across several machines does not reduce the overall memory required to run a large program.

In this paper, we propose an alternate approach to scaling which performs inference in tandem with grounding. This enables us to scale SRL systems to large, previously intractable, models. Our approach, which we refer to as *tandem inference* (TI), uses a novel streaming grounding architecture and an out-of-core inference algorithm that utilizes a disk cache in order to consume a fixed amount of memory. This allows TI to scale unbounded by a machine's main memory. Furthermore, even with increased I/O overhead, our approach runs the entire process of grounding and inference in a fraction of the runtime required by traditional approaches. Since TI is orthogonal to lifting and some of the other strategies, it can be combined with them for further improvements.

The TI concept is general and can potentially be applied to several different SRL frameworks. In this paper, we show how it can be implemented in probabilistic soft logic (PSL) (Bach et al. 2017). PSL is a SRL framework that generates a special kind of undirected graphical model called a hinge-loss Markov random field (HL-MRF). A key distinguishing factor of a HL-MRF is that it makes a continuous relaxation on random variables (RVs) which transforms the inference problem into a convex optimization problem. This allows PSL to use optimizers such as alternating direction method of multipliers (ADMM) (Boyd et al. 2011) to perform efficient inference.

Our key contributions are as follows: 1) we propose a general framework, TI, which uses streaming grounding and out-of-core streaming inference to perform memory efficient, large-scale inference in SRL frameworks; 2) we derive a stochastic gradient descent-based inference method (SGD) and show that the SGD-based method can outperform the traditionally used ADMM-based method; 3) we develop an efficient streaming grounding architecture and SGD-based out-of-core inference system that runs faster than previous state-of-the-art systems; 4) through experiments on two large models, FRIENDSHIP-500M and FRIENDSHIP-1B, which require over 400GB and 800GB of memory respectively, we show that, using just 10GB of memory, we can perform inference on FRIENDSHIP-500M in under four hours and FRIENDSHIP-1B in under nine hours; and 5) we perform an empirical evaluation on eight realworld datasets to validate the speed and accuracy of TI. In addition to enabling inference on models too large to fit into memory, on our largest realworld dataset which does fit in memory, TI is 8 times faster than traditional approaches.

2 Background: PSL and HL-MRFs

A HL-MRF is a probabilistic graphical model generated using a probabilistic programming language called PSL. A PSL model ($\mathcal{M}$) is defined through a set of weighted first-order logic rules (template rules) which are instantiated with data ($\mathcal{D}$) (a.k.a. grounded) to generate several instances of the template rules, called ground rules, which form a HL-MRF. Each ground rule corresponds to a clique potential in the HL-MRF. A HL-MRF can be formally defined as:

Definition 1 (Hinge-loss Markov random field)**.** *Let* $\mathbf{y} = \{y_1, y_2, ..., y_n\}$ *be* n *RVs,* $\mathbf{x} = \{x_1, x_2, ..., x_m\}$ *be* m *observed variables or evidence, and* $\phi = \{\phi_1, \phi_2, ..., \phi_K\}$ *be* K *potential functions such that* $\phi_i(\mathbf{x}, \mathbf{y}) = \max(\ell_i(\mathbf{x}, \mathbf{y}), 0)^{d_i}$ *or* $\ell_i(\mathbf{x}, \mathbf{y})^{d_i}$*. Where* ℓ_i *is a linear function and* $d_i \in \{1, 2\}$ *provides a choice of two different loss functions,* $d_i = 1$ *(i.e., linear) and* $d_i = 2$ *(i.e, quadratic). For weights* $\mathbf{w} \in \{w_1, w_2, ..., w_K\}$ *a hinge-loss energy function can be defined as:*

$$\mathbf{E}(\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{K} w_i \phi_i(\mathbf{x}, \mathbf{y}) \; ; \textit{s.t.}, \mathbf{y} \in [0, 1] \; ; \; \mathbf{x} \in [0, 1] \quad (1)$$

and the HL-MRF is defined as:

$$P(\mathbf{y}|\mathbf{x}, \mathbf{w}) = \frac{1}{Z(\mathbf{y})} exp(-E(\mathbf{y}|\mathbf{x})) \quad (2)$$

where $Z(\mathbf{y}) = \int_{\mathbf{y}} exp(-E(\mathbf{y}|\mathbf{x}))$*.*

The template rules in PSL define the interaction between different observed and unobserved RVs. Each predicate of a rule in PSL generates continuous RVs $\mathbf{x}, \mathbf{y} \in [0, 1]$. Therefore, a rule can be interpreted as continuous relaxations of Boolean logical connectives. For example, $a \vee b$ corresponds to the hinge potential $\min(1, a + b)$, and $a \wedge b$ corresponds to $\max(0, a + b - 1)$ (see (Bach et al. 2017) for full details). The combination of all ground rules define the hinge-loss energy function. Once the energy function is generated the task of inference is to find the maximum aposteriori estimate (MAP) of the RVs $\mathbf{y}$ given evidence $\mathbf{x}$. This is performed by maximizing the density function or minimizing the energy function in (1). MAP inference is expressed as:

$$\underset{\mathbf{y}}{\operatorname{argmax}}\, P(\mathbf{y}|\mathbf{x}) = \underset{\mathbf{y}}{\operatorname{argmin}}\, \mathbf{E}(\mathbf{y}|\mathbf{x}) \quad (3)$$

The above expression is a convex optimization problem and is efficiently solved by a convex optimizer such as ADMM.

We illustrate how HL-MRFs are instantiated using PSL with the following example:

Example 1. *Consider a social network with users* $\mathbf{U} \in \{U_1, \ldots, U_u\}$ *and friendship links* $\mathbf{F} \in [0, 1]^{u \times u}$*, where Friend*$(U_1, U_2)$ *denotes the degree of friendship between* U_1 *and* U_2*, near* 0 *if they are not friends and near* 1 *if they are. Let* $\mathbf{F_o} \subset \mathbf{F}$ *be the observed friendship links and* $\mathbf{F_u} = \mathbf{F} \setminus \mathbf{F_o}$ *be the unobserved links. The task is to predict a value for all unobserved friendship links,* $\mathbf{F_u}$*, given the observed friendship links,* $\mathbf{F_o}$*. Let LocalPredictor*(U_1, U_2) *denote a prediction of Friend*(U_1, U_2) *made by a local classifier using the non-relational attributes of* U_1 *and* U_2*. Using these, we can define a simple PSL model to collectively infer labels for all unobserved edges* $\mathbf{F_u}$ *as:*

$$w_1 : \textit{LocalPredictor}(U_1, U_2) \rightarrow \textit{Friend}(U_1, U_2)$$
$$w_2 : \textit{Friend}(U_1, U_2) \wedge \textit{Friend}(U_2, U_3) \rightarrow \textit{Friend}(U_1, U_3)$$

where w_1 *and* w_2 *are non-negative weights for the rules. A HL-MRF is generated by grounding the above model with*

users $\mathbf{U}$ *and friendship links* $\mathbf{F}$. *Each instantiation of the predicate Friend with a member of* $\mathbf{F_u}$ *is represented by unobserved RV* y_i. *Each ground rule, e.g.*
w_1 : *LocalPredictor*($\mathtt{Ann}, \mathtt{Bob}$) $\rightarrow$ *Friend*($\mathtt{Ann}, \mathtt{Bob}$), *generates a hinge-loss potential* ϕ_i.

3 Tandem Inference

In order to define our proposed tandem inference (TI) algorithm we introduce two components: the *grounding generator* (GG) and the *inference engine* (IE). The GG supports *streaming grounding*, which is the process of generating ground rules in small batches without materializing all grounding results into memory. The IE supports *streaming inference*, which is the process of performing inference using a single potential at a time. Fig. 1a shows the system architecture of TI. The GG takes as input the data $\mathcal{D}$ and the model $\mathcal{M}$, which GG uses to generate the ground model. With respect to storage, the GG can leverage the hard disk, the database, and RAM, while the IE can only utilize RAM.

The process flow of TI is shown in the network sequence diagrams given in Fig. 1b (for the first round of inference) and 1c (for subsequent rounds of inference). The IE begins by requesting a potential function (also called a *ground term*) from the GG. During the first round of inference, the GG utilizes the database to generate ground rules. Once a ground rule is created, it is converted into a ground term, written to a disk cache, and then passed onto the IE. On subsequent rounds of inference, the GG uses the disk cache alone to provide ground terms to the IE. As each term is returned from the GG, the IE will optimize the term and update any relevant RVs held in RAM. After all terms have been seen, IE will then begin a new round of inference until convergence criteria are met. Since the GG uses a fixed-size in-memory cache and the IE discards terms after use, there is a maximum amount of memory in use by TI at any given time.

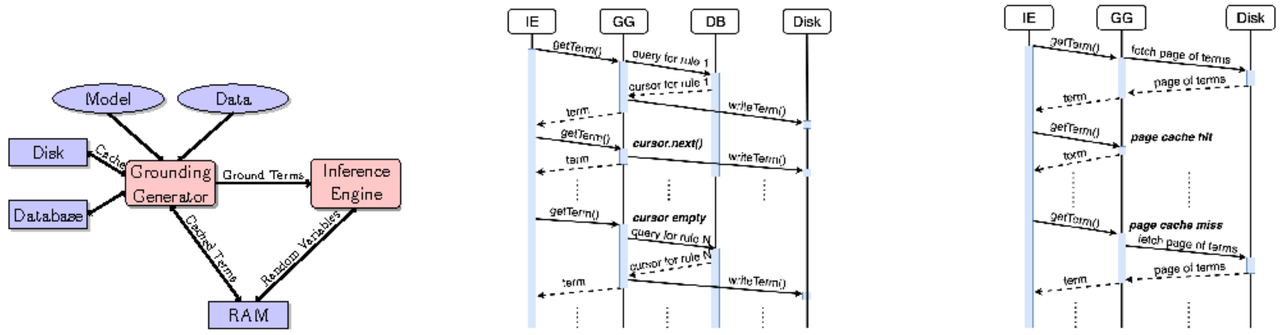
3.1 Streaming Grounding

Streaming grounding is the TI component that is responsible for providing ground terms one at a time to the IE. To support streaming grounding, the underlying SRL framework must be able to construct a single ground rule without instantiating large portions of the model, a process we will refer to as *partial grounding*. Constructing the full ground network is the SRL phase that is most prone to running out of memory, so it is imperative that this process can be broken up into small chunks. PSL is one of several SRL frameworks that supports *bottom-up grounding*(Augustine and Getoor 2018), which frames the problem of grounding a rule as constructing and executing a SQL query. Relational database management systems (RDBMSs) have a built-in way to fetch only a portion of the query results through cursors(Garcia-Molina, Ullman, and Widom 2008). A cursor works as an iterator for a query's results. If the RDBMS and content of the SQL query allows for it, cursors can return results as they are generated by the query instead of waiting for the full query to complete. Cases that force a database to materialize all results before returning any records include sorting or deduplicating the results, both of which are avoided in PSL grounding queries.

During the initial iteration of streaming grounding, the database must be queried to fetch the ground terms. The process described here is also shown as a network sequence diagram in Fig. 1b. The rules in the model, $\mathcal{M}$, will be iterated over until all have been grounded. When asked for a term, the GG will first check if it has an open database cursor. If there is no cursor or the current cursor has been exhausted, then the database will be queried for the next rule and a new cursor will be constructed. If all rules have been grounded, then the GG will inform the IE that there are no more ground terms for this iteration. With an open cursor, the next result tuple will be fetched. The tuple will then be instantiated into a ground rule and checked for triviality. A ground rule is trivial if it will not affect the result of inference; for example, a ground rule with a logical expression that is already satisfied by observed variables is considered trivial. If the newly instantiated ground rule is invalid, the process is repeated with the next result from the cursor until a valid ground rule is instantiated. After a ground rule is validated, it is converted into a ground term. This term is then put into a cache buffer and eventually passed on to the IE. Once the cache buffer is full, or there are no more ground rules, it is written to disk.

After the initial iteration of TI, ground terms are fetched from the disk cache in the order they were written during the initial iteration of streaming grounding. The process described here is shown as a network sequence diagram in Fig. 1c. The disk cache is written as several pages, one page to a file. The number of terms written to a page can be configured and the effect of this configuration is explored in Section 4.2. When asked for a term, the GG will first ensure that a cache page is loaded. If the current page has been exhausted and there are no more pages, then the IE will be informed that there are no more ground terms for this iteration. Each page is written in binary to minimize I/O. The first 16 bytes of a page contains the number of terms written to the page and the size in bytes of all the terms in the page. Because each term may contain a different number of RVs, the exact size of each term is not known until it is generated. Each term is then read into a preallocated term structure. A free list of available terms large enough to fill a page is maintained to minimize memory allocations. After all terms have been read into the memory cache, the next term from the cache will be returned to the IE until the page is exhausted.

The GG also provides the ability to return ground terms in a semi-random order. The effectiveness of randomizing term order is dependent upon the optimization algorithm employed by the IE. During the initial iteration of streaming grounding, the order of the terms is dependent upon the order that results are returned from the database. However, during subsequent iterations, there are more opportunities to induce randomness. First, the order that pages are accessed can be randomized. Second, the terms in each page can be shuffled in-place after they have been read from disk. Although not fully random since every term is guaranteed to be no more than a page length away from other terms in the same page, these steps provide a good amount of randomness without increasing the number of I/O operations or memory required.



(a) Block diagram showing the TI system architecture.

(b) Network sequence diagram for iteration 1 of TI.

(c) Network sequence diagram for iterations 2 through T of TI.

Figure 1: The architecture of TI.

3.2 Streaming Inference

The second core component of TI is streaming inference. The open-source PSL implementation uses ADMM to minimize the convex objective produced by a HL-MRF. While ADMM is an efficient algorithm, it places a high memory requirement on the system. However, our primary goal of using TI is to alleviate any memory constraints for performing inference. ADMM in PSL works by creating Lagrange variables (LVs) and a local copy of the RVs (LRVs) for every potential. Every potential is optimized w.r.t. the LRVs and an average over all LRVs is taken to obtain a global assignment for the RVs. All ground rules, LRVs, and LVs are kept in memory to make this process efficient. However, the need to keep all this information in memory makes ADMM less than ideal for streaming inference. To replace ADMM, we propose a gradient-based optimization to solve Equation 3.

Stochastic Gradient Descent for PSL
Consider the energy function from Equation 1. To facilitate easy gradient computation, this can be re-written as:

$$E(\mathbf{y}|\mathbf{c}) = \sum_{i=1}^{K} w_i \phi_i(\mathbf{y}, c_i) \tag{4}$$

$$\phi_i(\mathbf{y}, c_i) = \begin{cases} (\mathbf{y}^T \mathbf{q}_i - c_i)^{d_i} & \text{if linear loss} \\ \max((\mathbf{y}^T \mathbf{q}_i - c_i), 0)^{d_i} & \text{otherwise} \end{cases}$$

$$c_i = \mathbf{x}^T \dot{\mathbf{q}}_i$$

where $\mathbf{q}_i \in \{0, 1, -1\}^n$ and $\dot{\mathbf{q}}_i \in \{0, 1, -1\}^m$ are respective n-dimensional and m-dimensional ternary vectors which indicates if a variable participates positively, negatively, or not at all in potential function ϕ_i. The scalar c_i incorporates the information of all the observed variables participating in ϕ_i and we use $\mathbf{c}$ to represent the vector of scalars c_i. As a reminder, in full gradient descent (GD), we iteratively optimize by taking weighted steps in the direction of the energy function's (Equation 4) gradient until convergence or for T steps. The weight of the steps is determined by the learning rate, η. Additionally for PSL, Equation 1 has a restriction that $\mathbf{y} \in [0, 1]^n$, therefore after each step we need to project $\mathbf{y}$ back in to the box $[0, 1]$ through truncation. The gradient step update at every step t can be represented as:

$$\mathbf{y}_t = \mathbf{y}_{t-1} - \eta \nabla_{\mathbf{y}} E(\mathbf{y}|\mathbf{c}) \tag{5}$$

$$\mathbf{y}_t = \min(\max(\mathbf{y}_t, 0), 1) \tag{6}$$

$$\text{where } \nabla_{\mathbf{y}} E(\mathbf{y}|\mathbf{c}) = \sum_{i=1}^{K} w_i \nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i)$$

The gradient computation for the energy function involves computing projected gradients for the potential functions. The projected gradients can be written as:

$$\nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i) = \begin{cases} 0 & \text{if hinge \& } \mathbf{y}^T \mathbf{x}_i \le c_i \\ w_i \mathbf{q}_i & \text{if } d_i = 1 \\ 2 w_i \mathbf{q}_i (\mathbf{y}^T \mathbf{q}_i - c_i) & \text{otherwise} \end{cases} \tag{7}$$

Using the above equations, we can compute the gradient update and run the process to convergence. Since the objective is convex, the right choice of η will guarantee convergence. However, an issue with GD is that at every step it needs to compute the full gradient, requiring all terms. This is expensive and does not support our streaming approach. To make it more compatible with our streaming approach, we use stochastic gradient descent (SGD). In SGD, the gradient is computed w.r.t. a single potential ϕ_i and an update to the variables can be made without examining all terms.

$$\mathbf{y}_t = \mathbf{y}_{t-1} - \eta w_i \nabla_{\mathbf{y}} \phi_i(\mathbf{y}, c_i) \tag{8}$$

The variables are then projected as in Equation 6. This update is aligned with our streaming approach, and allows the IE to request a single potential, perform an update on the participating RVs, and continue on to the next potential.

An important factor to make SGD work in practice is the correct choice of η, and many approaches have been proposed to compute adaptive learning rates (Ruder 2016). One of the more popular and successful methods for adaptive learning rates is SGD-ADAM (Kingma and Ba 2014). In this work we investigate three approaches: 1) using SGD-ADAM, 2) using a time-decaying learning rate (i.e., $\eta = \frac{\eta}{t}$), and 3) using a constant learning rate η. In our experiments, we observe that a time-decaying learning rate is more effective

Algorithm 1: SGD for PSL

```
   Data: list of ground terms $\phi = \{\phi_1, \phi_2, ..., \phi_K\}$
   Result: RVs $\mathbf{y}$
 1 $\eta$ = learning rate;
 2 $\mathbf{y} \sim Unif(0,1)^n$;
 3 $t = 1$;
 4 while not converged and $t <= T$ do
 5     for $i \in \{1 \dots K\}$ do
 6         Update $\mathbf{y}$ using $\phi_i$ with Equation 8;
 7         $\mathbf{y} = \min(\max(\mathbf{y}, 0), 1)$;
 8     end
 9     $t = t + 1$;
10     Update $\eta$;
11 end
```

than more complicated mechanisms such as SGD-ADAM. Finally, the overall process of performing non-streaming inference using SGD in PSL is shown in Algorithm 1.

Now that we have a streaming grounding infrastructure and the SGD-based PSL IE, we can perform TI. The algorithm for TI is the same as Algorithm 1, except that the potentials are read from the GG instead of from memory.

4 Empirical Evaluation

In this section, we evaluate the performance of our proposed method on variety of realworld and two large synthetic datasets. We answer the following questions: Q1) Can we perform inference on large ground models that were previously intractable? Q2) Is streaming faster than and as accurate as traditional inference? Q3) How much memory does TI use? Q4) Can a gradient-based optimizer converge faster than ADMM? Q5) What is the best strategy for selecting the learning rate? We answer Q1 and Q2 in Section 4.1, Q3 in Section 4.2, and Q4 and Q5 in Section 4.3. For all our experiments, we set the max number of iterations, T, to 500, a convergence tolerance of 10^{-6}, and a machine with 400GB of memory.

We perform our experiments on eight realworld datasets and two synthetic datasets from a data generator previously used to test scaling in PSL (Augustine and Getoor 2018)[1]. The details of the datasets are as follows:

CITESEER: a collective classification dataset with 2,708 scientific documents, seven document categories, and 5,429 directed citation links.

CORA: a collective classification dataset with 3,312 documents, six categories, and 4,591 directed citation links.

EPINIONS: a trust prediction dataset with 2,000 users and 8,675 directed links which represent positive and negative trust between users.

NELL: a knowledge graph construction dataset originally derived from the NELL project with 27,843 entity labels and 12,438 relations.

CITESEER-ER: an entity resolution dataset with a citation network of 1136 authors references and 864 paper references.

[1]Models, data, and code: https://github.com/linqs/aaai-ti

LASTFM: an artist recommendation dataset with 1,892 users, 17,632 artists, 92,834 user-artist ratings, and 12,717 friendship links.

JESTER: a joke recommendation dataset with 2,000 users, 100 jokes, and 200,000 user-joke ratings, sampled from the larger JESTER-FULL dataset.

JESTER-FULL: the full Jester dataset. Contains 73,421 users, 100 jokes, and 7.3M user-joke ratings. To the best of our knowledge, this is the first time the full Jester dataset has been used in with an SRL framework.

FRIENDSHIP-500M: a synthetic link prediction dataset with 2,000 users and 4M unobserved edges.

FRIENDSHIP-1B: similar to FRIENDSHIP-500M containing 2,750 users and 7.5M unobserved edges.

Table 1 provides details on number of rules in each model, the number of ground rules generated, and the amount of memory required to hold each model in memory.

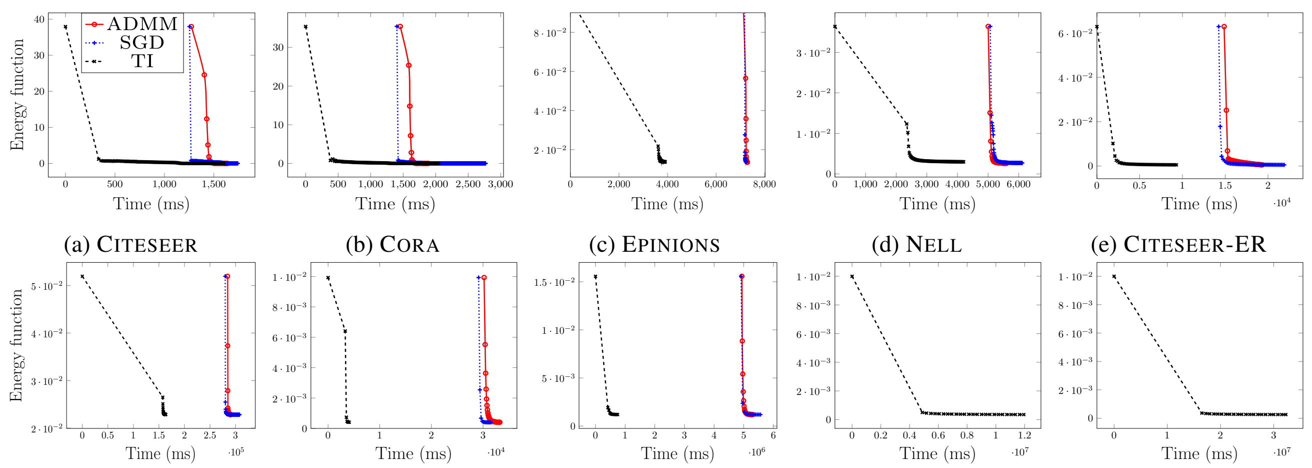
Dataset	Rules	Ground Rules	Random Variables	Memory (GB)	Source
CITESEER	10	36K	10K	0.10	Bach et al. (2017)
CORA	10	41K	10K	0.11	Bach et al. (2017)
EPINIONS	20	14K	1K	0.12	Bach et al. (2017)
NELL	26	91K	24K	0.13	Pujara et al. (2013)
CITESEER-ER	9	541K	485K	0.24	Bhattacharya and Getoor (2007)
LASTFM	22	1.4M	18K	0.45	Kouki et al. (2015)
JESTER	7	1M	50K	0.49	Bach et al. (2017)
JESTER-FULL	8	110M	3.6M	110	Goldberg et al. (2001)
FRIENDSHIP-500M	4	500M	4M	400+	Augustine and Getoor (2018)
FRIENDSHIP-1B	4	1B	7.6M	800+	Augustine and Getoor (2018)

Table 1: Details of models used and their memory consumption for non-streaming inference. Memory usage for FRIENDSHIP-500M and FRIENDSHIP-1B are estimates. The memory consumed by TI depends on the page size chosen. In our comparison experiments, we use a page size of 10M which uses about 10GB of memory.

4.1 Scale, Speed, and Convergence

We begin by examining the inference time and convergence of TI, SGD, and ADMM on all ten datasets (Q1 & Q2). Weights for the rules in each model are learned and re-scaled to be in the range $[0, 1]$. Both SGD and TI use a time-decayed learning rate and the initial learning rate η needs to be tuned. For the LASTFM, FRIENDSHIP-500M, and FRIENDSHIP-1B datasets we use $\eta = 0.1$, for CITESEER-ER we use $\eta = 10$, and for all the other datasets we use $\eta = 1.0$. The rational for choosing these learning rates is explained in Section 4.3. TI also has a page size which can be tuned based on the amount of memory available. Since our machine has 400GB RAM, we choose a page size of 10M for all datasets, which uses about 10GB of memory. We discuss further details about trade-offs in page size, memory, and computation in Section 4.2.

Scaling to Large Datasets: Fig. 2i and 2j show the inference convergence w.r.t. time (in milliseconds) for FRIENDSHIP-500M and FRIENDSHIP-1B. TI was able to run the FRIENDSHIP-500M dataset in under four hours using only 10GB of memory. Both SGD and ADMM exhausted the 400GB of memory available on the machine and failed to run. Similarly, we observe that the FRIENDSHIP-1B dataset, which we estimate to require more than 800GB to hold in



(a) CITESEER (b) CORA (c) EPINIONS (d) NELL (e) CITESEER-ER

(f) LASTFM (g) JESTER (h) JESTER-FULL (i) FRIENDSHIP-500M (j) FRIENDSHIP-1B

Figure 2: Comparison of the runtimes of TI, ADMM, and SGD on 10 datasets.

memory, was able to run on the same machine in under nine hours using only 10GB of memory. These results answer Q1 affirmatively, TI can successfully perform inference on large ground models that were previously intractable.

Speed and Convergence: In order to address Q2, Fig. 2 shows the inference convergence for all datasets using all three approaches. The time shown includes both the grounding and inference phases (which happen together in TI). In all datasets except CITESEER and CORA, we observe that TI converges before the other methods even fully finish grounding! In CITESEER and CORA, possibly due to some rules with high weights, tuning the learning rate is difficult, and, after the first steep drop, SGD and TI take many more iterations to converge compared to ADMM. As the ground model size increases, we observe more significant timing differences. In EPINIONS, CITESEER-ER, and LASTFM, we see that TI finishes the entire process of grounding and inference in just half the time taken by ADMM and SGD. For JESTER, TI is over 5 times faster than both ADMM and SGD. For our largest realworld dataset, JESTER-FULL, TI is about 8 times faster than both ADMM and SGD. This shows that TI is faster than traditional approaches especially on larger datasets and converges to the same function value.

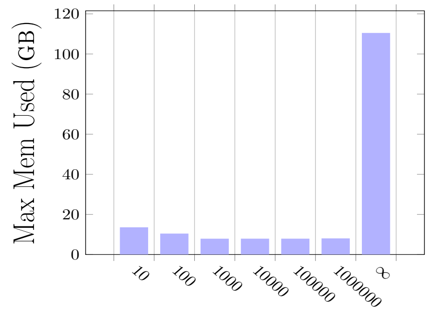
4.2 Memory Efficiency

To answer Q3, how much memory TI uses, and test the effect of page size, we run TI on the JESTER-FULL dataset with page sizes between 10 and 1M potentials. We run SGD to establish baseline behavior. Since SGD holds all components in memory, we consider it to have an infinite page size. We measure the maximum memory usage during the entire run, the mean number of I/O operations performed in a single iteration of optimization, and the mean time to complete a single iteration of optimization. Because PSL is written in Java, the memory usage we report is the size of the JVM's heap. I/O operations are measured by the number of calls made to Java's low-level I/O methods `FileInputStream.read` and `FileOutputStream.write`. All reported values are averaged over 10 runs.

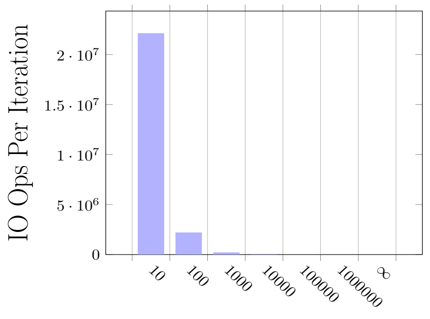
Fig. 3a shows the peak memory usage over several runs using different page sizes for TI. Naively, we expect the amount of memory used to decrease with the page size: the fewer ground terms held in memory, the less overall memory is used. However, instead we see that very small pages sizes (10 and 100) lead to more memory being used. To understand why, we must remember that Java is a garbage collected language and that memory marked for garbage collection will still count as being in use. The small page sizes cause many I/O operations to happen in quick succession. Every I/O operation requires Java to allocate memory buffers used in that operation. Therefore, the discarded buffers are eventually cleaned up but not before being counted in the memory total. A native language like C that can directly make system calls and that does not have to go through a virtual machine can avoid these extra allocations and inflated memory cost. Forcing the JVM to garbage collect more frequently[2] shows a more consistent memory usage over all page sizes. This is because all the page sizes still fit into memory and Java will continue to accumulate memory until a point where garbage collection is triggered. This point depends on the maximum size of the heap and is common among all the runs. From this experiment we can conclude that although using a smaller page size will use less persistent memory, the JVM's garbage collector will keep most reasonable page sizes at the same memory usage levels.

Fig. 3b shows the number of I/O operations per optimization iteration. SGD does not perform any I/O operations. As the page size increases, the number of I/O operations decreases. The impact of these additional I/O operations for a small page sizes can be seen in Fig. 3c (per-iteration runtime). The different page sizes result in approximately the

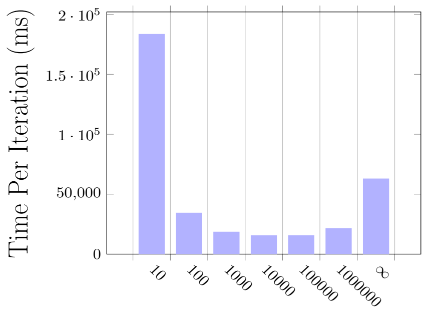
[2]A smaller value for the JVM parameter `XX:NewRatio` is used to force more frequent garbage collection.



(a) Maximum memory usage for TI over multiple page sizes.



(b) Number of I/O operations per optimization iteration of TI over multiple page sizes.

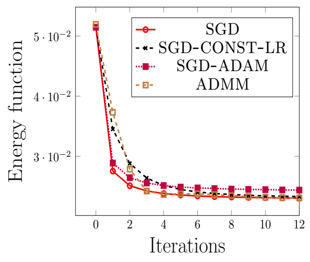


(c) Runtime per optimization iteration of TI over multiple page sizes.

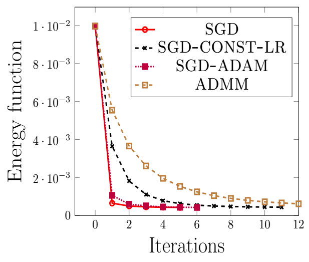
Figure 3: Memory usage, I/O usage, and speed of TI on the JESTER-FULL dataset w.r.t. page size. Page sizes listed as ∞ are run with SGD, which does not use pages.

same number of bytes read from disk and since the number of potentials is the same, the time spent in optimization will also be the approximately the same. However, the overhead of the additional I/O operations causes substantially slower iterations for page sizes of 10 and 100. For larger page sizes (1000+), the difference in I/O overhead becomes negligible and all have similar per iteration runtime.

4.3 Optimizer Efficiency and Learning Rate



(a) LASTFM dataset.



(b) JESTER dataset.

Figure 4: The effect of different optimizers on convergence.

To answer Q4, can SGD converge as fast as ADMM, and Q5, how to choose learning rate, we run experiments on the LASTFM and JESTER datasets. The results here extend to other datasets. Here, we compare four different approaches: SGD with a decaying learning rate (SGD), SGD with a constant learning rate (SGD-CONST-LR), SGD with an adaptive learning rate (SGD-ADAM), and ADMM.

SGD vs. ADMM: Fig. 4 shows the convergence of different approaches w.r.t. number of iterations for the LASTFM and JESTER datasets. Here, we observe that SGD and ADMM converge to the same function value in different number of iterations. Typically, SGD takes fewer iterations to converge than ADMM. However, this is heavily dependent on the learning rate chosen for SGD. If one cannot find the right learning rate, then it is possible for SGD to take significantly more iterations than ADMM.

Choice of Learning Rate: From Fig. 4 we observe that SGD-CONST-LR is the slowest to converge, and there seems to be little difference between SGD which uses time-decayed learning rate and SGD-ADAM which uses an adaptive learning rate. SGD and SGD-CONST-LR have an initial learning rate to be chosen. We observed that this can be chosen in range $\eta \in [\frac{1}{10\max(\mathbf{w})}, \frac{10}{\max(\mathbf{w})}]$ for SGD, and for SGD-CONST-LR, $\eta \in [\frac{1}{100\max(\mathbf{w})}, \frac{100}{\max(\mathbf{w})}]$. Thus, we choose an η of 1.0 and 0.1 for JESTER and LASTFM respectively for SGD, and η of 0.01 for SGD-CONST-LR. SGD-ADAM has four hyperparameter α, $\beta 1$, $\beta 2$, and ϵ to tune. This makes it harder to get ideal performance with SGD-ADAM. Further, SGD-ADAM uses additional parameters equal to three times the number of RVs to perform adaptive tuning. In our experiments, we choose $\alpha = 0.01$, and use $\beta 1 = 0.9$, $\beta 2 = 0.999$, and $\epsilon = 10^{-8}$ as suggested by (Kingma and Ba 2014). From our evaluation we conclude that the simpler strategy, SGD with decaying learning rate performs just as well as SGD-ADAM, the more complicated adaptive strategy.

5 Conclusion and Future Work

In this paper we introduce tandem inference, TI, a new out-of-core method for performing inference on large ground models that don't fit into main memory. To make TI possible, we introduce a streaming method for both grounding and inference. Through experiments on ten datasets, we have shown that TI can not only reduce runtime by up to eight times, but it can do so using a fixed amount of memory. The fixed memory nature of TI enables the SRL community to scale to problems that were previously unreachable.

While this paper introduces the fundamentals of TI, there remain several areas for research. Incorporating lifted inference is a promising extension to TI. Because TI is orthogonal to lifting, these two can be combined to speed up inference further. Next, despite impressive performance on large datasets, the overall process of TI is largely sequential; parallelizing TI can be another way to speeding up inference further. Another interesting avenue for research is to create a hybrid IE using ADMM and SGD. SGD often minimizes quickly during the first few iterations, however may take many more iterations to fully converge (especially if the learning rate is poorly selected). Conversely, ADMM converges more slowly than SGD, but more steadily. A hybrid IE could start with SGD and then switch to ADMM after the first few iterations. Finally, TI can be extended to any other SRL framework that can support streaming grounding and streaming inference.

6 Acknowledgements

This work was partially supported by the National Science Foundation grants CCF-1740850 and IIS-1703331, AFRL and the Defense Advanced Research Projects Agency. We also thank the reviewers for their constructive feedback that helped shape the paper.

References

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

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

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

Bhattacharya, I., and Getoor, L. 2007. Collective entity resolution in relational data. *TKDD* 1:1–36.

Boyd, S. P.; Parikh, N.; Chu, E.; Peleato, B.; and Eckstein, J. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *FTML*.

Chen, P.; Chen, F.; and Qian, Z. 2014. Road traffic congestion monitoring in social media with hinge-loss Markov random fields. In *ICDM*.

Das, M.; Wu, Y.; Khot, T.; Kersting, K.; and Natarajan, S. 2016. Scaling lifted probabilistic inference and learning via graph databases. In *SDM*.

Das, M.; Dhami, D. S.; Kunapuli, G.; Kersting, K.; and Natarajan, S. 2019. Fast relational probabilistic inference and learning: Approximate counting via hypergraphs. In *AAAI*.

de Salvo Braz, R.; Amir, E.; and Roth, D. 2005. Lifted first-order probabilistic inference. In *IJCAI*.

den Broeck, G. V.; Taghipour, N.; Meert, W.; Davis, J.; and Raedt, L. D. 2011. Lifted probabilistic inference by first-order knowledge compilation. In *IJCAI*.

den Broeck, G. V.; Choi, A.; and Darwiche, A. 2012. Lifted relax, compensate and then recover: From approximate to exact lifted probabilistic inference. In *UAI*.

Ebrahimi, J.; Dou, D.; and Lowd, D. 2016. Weakly supervised tweet stance classification by relational bootstrapping. In *EMNLP*.

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

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

Goldberg, K.; Roeder, T.; Gupta, D.; and Perkins, C. 2001. Eigentaste: A constant time collaborative filtering algorithm. *IR* 4:133–151.

Kersting, K. 2012. Lifted probabilistic inference. In *ECAI*.

Kimmig, A.; Mihalkova, L.; and Getoor, L. 2015. Lifted graphical models: a survey. *MLJ* 99:1–45.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. In *ICLR*.

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

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

Magliacane, S.; Stutz, P.; Groth, P.; and Bernstein, A. 2015. foxPSL: A fast, optimized and extended psl implementation. *IJAR* 67:111–121.

Niu, F.; Ré, C.; Doan, A.; and Shavlik, J. 2011. Tuffy: Scaling up statistical inference in Markov logic networks using an rdbms. In *VLDB*.

Pujara, J.; Miao, H.; Getoor, L.; and Cohen, W. 2013. Knowledge graph identification. In *ISWC*.

Raedt, L. D., and Kersting, K. 2011. Statistical relational learning. In *Encyclopedia of Machine Learning*. Springer.

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

Ruder, S. 2016. An overview of gradient descent optimization algorithms. *CoRR* abs/1609.04747.

Sarkhel, S.; Venugopal, D.; Singla, P.; and Gogate, V. 2014. Lifted MAP inference for Markov logic networks. In *AIStats*.

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

Sen, P.; Deshpande, A.; and Getoor, L. 2009. Bisimulation-based approximate lifted inference. In *UAI*.

Singla, P., and Domingos, P. M. 2008. Lifted first-order belief propagation. In *AAAI*.

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

Srinivasan, S.; Rao, N. S.; Subbaian, K.; and Getoor, L. 2019b. Identifying facet mismatches in search via micrographs. In *CIKM*.

Venugopal, D., and Gogate, V. 2014. Evidence-based clustering for scalable inference in Markov logic. In *ECML*.

Venugopal, D.; Sarkhel, S.; and Gogate, V. 2015. Just count the satisfied groundings: scalable local-search and sampling based inference in MLNs. In *AAAI*.

Venugopal, D.; Sarkhel, S.; and Gogate, V. 2016. Magician: Scalable inference and learning in Markov logic using approximate symmetries. Technical report, Department of Computer Science, The University of Memphis.