Appears in Proceedings of the IEEE International Conference on Data Mining (ICDM), 2012.

Scaling Inference for Markov Logic via Dual Decomposition

Feng Niu Ce Zhang Christopher Ré Jude Shavlik
Department of Computer Sciences
University of Wisconsin-Madison, USA
{*leonn, czhang, chrisre, shavlik*}*@cs.wisc.edu*

***Abstract*—Markov logic is a knowledge-representation language that allows one to specify large graphical models. However, the resulting large graphical models can make inference for Markov logic a computationally challenging problem. Recently, dual decomposition (DD) has become a popular approach for scalable inference on graphical models. We study how to apply DD to scale up inference in Markov logic.**

A standard approach for DD first partitions a graphical model into multiple tree-structured subproblems. We apply this approach to Markov logic and show that DD can outperform prior inference approaches. Nevertheless, we observe that the standard approach for DD is suboptimal as it does not exploit the rich structure often present in the Markov logic program. Thus, we describe a novel decomposition strategy that partitions a Markov logic program into parts based on its structure. A crucial advantage of our approach is that we can use specialized algorithms for portions of the input problem – some of which have been studied for decades, e.g., coreference resolution. Empirically, we show that our program-level decomposition approach outperforms both non-decomposition and graphical model-based decomposition approaches to Markov logic inference on several data-mining tasks.

I. Introduction

Markov logic [10] is a knowledge-representation language that uses weighted first-order logic to specify graphical models. It has been applied to a wide range of applications, including many in information extraction and data mining [9]. The resulting graphical models can be huge (hundreds of millions of nodes or more in our applications), and so a key technical challenge is the scalability and performance of Markov logic inference.

Semantically, a Markov logic program, or *Markov logic network (*MLN*)*, specifies a graphical model called a *Markov random field* (MRF). Thus, one approach to improving the scalability of MLNs is to apply inference techniques from the graphical-model literature. One such technique is *dual decomposition* [15], which has recently been applied to MRF inference [1], [2], [4], [14]. In addition to increased scalability, dual decomposition also offers the possibility of dual certificates[1] that can bound the distance of a solution from optimality, e.g., for *maximum a posteriori (MAP)* inference.

There are two steps in applying dual decomposition: (1) decompose the inference problem into multiple parts, and (2) iteratively combine solutions from individual parts. The intuition is that the individual parts will be more tractable than the whole problem – so much so that the improved performance of individual parts will compensate for the overhead of many iterations of repeated inference. Following the literature [2], [4], we first implement MRF-level decomposition and compare it with state-of-the-art MLN inference algorithms. On simpler MLN-generated MRFs, MRF-level decomposition can achieve competitive performance compared to monolithic inference (i.e., running a generic MLN inference algorithm without decomposition). On more complex MLN-generated MRFs, the performance and quality of both monolithic inference and MRF decomposition approaches may be suboptimal.

Our key observation is that MRF-level decomposition strategies in step (1) ignore valuable structural hints that often occur in MLN programs. For example, a large Markov logic program may have several "subroutines" that each perform a standard, well-studied task such as coreference resolution or labeling, e.g., with a *conditional random field* (CRF) [6]. In contrast to traditional approaches that either use a generic MLN inference algorithm or decompose into semantically meaningless parts like trees, our idea is to exploit this information so that we may use existing, specialized algorithms for such individual subtasks. For example, we may choose to solve a labeling task with a CRF and so use the Viterbi algorithm [6]. Importantly, even if we use different algorithms for each part, dual decomposition preserves the joint-inference property – i.e., different parts are not solved in isolation. The hope is to achieve higher efficiency and quality than monolithic approaches.

To illustrate this idea, we describe several such correspondences between MLN structure and specialized inference algorithms, including logistic regression and linear-chain conditional random fields. A user may declare (or an algorithm may detect) which of these specialized algorithms to use on an individual part of the MLN program. We call our prototype system Felix.

Experimentally, we validate that our (MRF-level and program-level) dual-decomposition approaches have superior performance than prior MLN inference approaches on several data-mining tasks. Our main results are that (1) on simple MLNs taken from the literature, MRF-level decomposition outperforms monolithic inference, and Felix has

[1]Namely, an lower (resp. upper) bound to a minimization (resp. maximization) problem.

competitive efficiency and quality compared to monolithic inference and MRF-level decomposition; and (2) on more complex MLNs (also taken from the literature), FELIX achieves substantially higher efficiency and quality than monolithic inference and MRF-level decomposition.

II. Related Work

Dual Decomposition: Dual decomposition is a classic and general technique in optimization that decomposes an objective function into multiple smaller subproblems; in turn, these subproblems communicate to optimize a global objective via Lagrange multipliers [15]. Recently, dual decomposition has been applied to inference in graphical models such as MRFs. In the master-slave scheme [3], [4], the MAP solution from each subproblem is communicated and the Lagrange multipliers are updated with the projected gradient method at each iteration. Our prototype implementation of FELIX uses the master-slave scheme. It is future work to adapt the closely related tree-reweighted (TRW) algorithms [2], [14] that decompose an MRF into a convex combination of spanning trees.

MLN Inference: There has been extensive research interest in techniques to improve the scalability and performance of MLN inference. For example, we have studied how to prune MRFs based on logical rules in MLNs [11], and use relational databases to improve the speed and scalability of constructing MRFs [7]. Our implementation builds on the open-source TUFFY system [7], which implements many of the above techniques.

III. Background

We first walk through an MLN program that extracts affiliation relationships between people and organizations from text. We then briefly review dual decomposition.

A. Markov Logic

An MLN consists of three parts: *schema*, *evidence*, and *rules*. To tell an MLN inference system what data will be provided or generated, the user provides a *schema*, i.e., definitions of a set of relations (or equivalently, predicates). The truth values for some relations are given; such relations are called the *evidence*. In the schema of Figure 1, the first three relations are evidence relations. In addition to evidence, there are also relations whose content we do not know; they are called *query relations*. A user may ask for predictions on some or all query relations.

In addition to schema and evidence, we also provide a set of MLN rules that encode our knowledge about the correlations and constraints over the relations. An MLN rule is a first-order logic formula associated with a real number (or infinity) called a *weight*. Infinite-weighted rules are called hard rules, which means that they must hold in any prediction that the MLN system makes. In contrast, rules with finite weights are soft rules: a positive weight indicates confidence in the rule's correctness.

Semantics: An MLN program defines a probability distribution over possible worlds. Formally, we first fix a schema σ (as in Figure 1) and a domain D. Given as input a set of formulae $\bar{F} = F_1, \ldots, F_N$ with weights $w_1, \ldots, w_N$, they define a probability distribution over *possible worlds* as follows. Given a formula F_k with free variables $\bar{x} = (x_1, \cdots, x_m)$, for each $\bar{d} \in D^m$ we create a new formula $g_{\bar{d}}$ called a *ground formula* where $g_{\bar{d}}$ denotes the result of substituting each variable x_i of F_k with d_i. We assign the weight w_k to $g_{\bar{d}}$. Denote by $G = (\bar{g}, w)$ the set of all such weighted ground formulae of $\bar{F}$. Essentially, G forms an MRF over a set of Boolean random variables (representing the truth value of each possible *ground tuple*). Let w be a function that maps each ground formula to its assigned weight. Fix an MLN $\bar{F}$, then for any possible world (instance) I, we say a ground formula g is *violated* if $w(g) > 0$ and g is false in I, or if $w(g) < 0$ and g is true in I. We denote the set of ground formulae violated in a world I as $V(I)$. The cost of the world I is

$$\mathrm{cost}_{\mathrm{MLN}}(I) = \sum_{g \in V(I)} |w(g)|. \tag{1}$$

Through $\mathrm{cost}_{\mathrm{MLN}}$, an MLN defines a probability distribution over all instances:

$$\Pr[I] = Z^{-1} \exp\left\{-\mathrm{cost}_{\mathrm{MLN}}(I)\right\},$$

where Z is a normalizing constant.

Inference: We focus on MAP inference that finds a most likely world, i.e., a world with the lowest cost. MAP inference is essentially a mathematical optimization problem that is intractable, and so existing MLN systems implement generic algorithms for inference.[2]

B. Dual Decomposition

We illustrate the basic idea of *dual decomposition* with an example. Consider the problem of minimizing a real-valued function $f(x_1, x_2, x_3)$. Suppose that f can be written as

$$f(x_1, x_2, x_3) = f_1(x_1, x_2) + f_2(x_2, x_3).$$

Further suppose that we have black boxes to solve f_1 and f_2 (plus linear terms). To apply these black boxes to minimize f we need to cope with the fact that f_1 and f_2 share the variable x_2. Following dual decomposition, we can rewrite $\min_{x_1,x_2,x_3} f(x_1, x_2, x_3)$ into the form

$$\min_{x_1,x_{21},x_{22},x_3} f_1(x_1, x_{21}) + f_2(x_{22}, x_3) \text{ s.t. } x_{21} = x_{22},$$

where we essentially make two copies of x_2 and enforce that they are identical. The significance of such rewriting is that we can apply Lagrangian relaxation to the equality constraint to decompose the formula into two independent

[2] FELIX also supports *marginal inference* using dual decomposition.

pSimHard(per1, per2) homepage(per, page) faculty(org, per) *affil(per, org) *pCoref(per1, per2)	faculty('MIT', 'Chomsky') homepage('Joe', 'Doc201') ...	weight rule $+\infty$ pCoref(p, p) (F_1) $+\infty$ pCoref$(p1, p2)$ => pCoref$(p2, p1)$ (F_2) $+\infty$ pCoref(x, y), pCoref(y, z) => pCoref(x, z) (F_3) 6 pSimHard$(p1, p2)$ => pCoref$(p1, p2)$ (F_4) 8 faculty(o, p) => affil(p, o) (F_5)
Schema	**Evidence**	**Rules**

Figure 1. An example MLN program that performs two tasks: 1. discover affiliation relationships between people and organizations (affil), 2. resolve coreference among people mentions (pCoref). The remaining three relations are evidence relations.

pieces. To do this, we introduce a scalar variable $\lambda \in \mathbb{R}$ (called a *Lagrange multiplier*) and define $g(\lambda) =$

$$\min_{x_1,x_{21},x_{22},x_3} f_1(x_1, x_{21}) + f_2(x_{22}, x_3) + \lambda(x_{21} - x_{22}).$$

For any λ, we have $g(\lambda) \leq \min_{x_1,x_2,x_3} f(x_1, x_2, x_3)$.[3] Thus, the tightest bound is to maximize $g(\lambda)$; the problem $\max_\lambda g(\lambda)$ is a *dual problem* for the problem on f. If the optimal solution of this dual problem is feasible (here, $x_{21} = x_{22}$), then the dual optimal solution is also an optimum of the original program [15, p. 168].

The key benefit of this relaxation is that, instead of a single problem on f, we can now compute $g(\lambda)$ by solving two independent problems (each problem is grouped by parentheses) that may be easier to solve:

$$g(\lambda) = \min_{x_1,x_{21}} (f_1(x_1, x_{21}) + \lambda x_{21}) + \min_{x_{22},x_3} (f_2(x_{22}, x_3) - \lambda x_{22}).$$

To compute $\max_\lambda g(\lambda)$, we can use standard techniques such as *projected subgradient* [15, p. 174]. Notice that dual decomposition can be used for MLN inference if x_i are truth values of ground tuples and one defines f to be $\text{cost}_{\text{MLN}}(I)$.

Decomposition Choices: The dual decomposition technique leaves open the question of *how* to decompose a function f. We need to answer this question if we want to apply dual decomposition to MLNs.

IV. Dual Decomposition for MLNs

The two approaches that we implement for dual decomposition work at different levels of abstraction: at the MRF level or at the MLN-program level. Still, the two methods are similar: and both pass messages in a master-slave scheme, and produce a MAP solution after inference in a similar way. As a result, we are able to implement both approaches on top of the Tuffy [7] system. We describe each step of the process in turn: decomposition (Section IV-A), master-slave message passing (Section IV-B), and production of the solution (Section IV-C).

[3]The search space of LHS is a superset of RHS, therefore we always have LHS $\leq$ RHS. One can always take x21 = x22 = x3 in the minimization, and the value of the two object functions are equal.

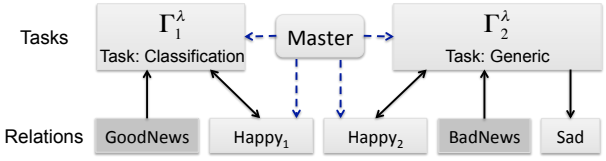

Figure 2. A program-level decomposition for Example 1. Shaded boxes are evidence relations. Solid arrows are data flow; dash arrows are control.

A. Decomposition

In decomposition, we partition the input structure and set up auxiliary structures to support message passing. For MRF-level decomposition, we partition an MRF into multiple trees that are linked via auxiliary singleton factors; for program-level decomposition, we partition an MLN into subprograms that are linked via auxiliary singleton rules.

MRF-level Decomposition: The input to the decomposition algorithm is an MRF which is the result of grounding an input MLN. The MRF is represented as a *factor graph* (i.e., a bipartite graph between ground-tuple nodes and ground-formula factors).[4] Following Komodakis et al. [3], [4] and Wainwright et al. [14], we decompose the MRF into a collection of trees (smaller factor graphs with disjoint factors) that cover this factor graph, i.e., each node in the factor graph is present in one or more trees.[5] Nodes that are in multiple trees may take conflicting values; to allow messages to be passed in the next step, we create a special singleton factor for each copy of a shared node.

Program-level Decomposition: In contrast, Felix performs decomposition at the program-level: the input is the (first-order) rule set of an MLN program, and Felix partitions it into multiple tasks each of which can be solved with different algorithms. In our program-level approach, we share at the granularity of relations, i.e., an entire relation is shared or not. This choice allows Felix to use a relational database management system (RDBMS) for all data movement, which can be formulated as SQL queries; in turn, this allows us to ignore low-level issues like memory management. We illustrate the idea with an example.

Example 1 Consider a simple MLN which we call Γ:

1	GoodNews(p) => Happy(p)	ϕ_1
1	BadNews(p) => Sad(p)	ϕ_2
5	Happy(p) <=> $\neg$Sad(p)	ϕ_3

[4]This representation allows for non-pairwise MRFs.

[5]We use a greedy algorithm to find these trees. We describe more details in the full version [8].

where GoodNews and BadNews are evidence and the other two relations are queries. Consider the decomposition $\Gamma_1 = \{\phi_1\}$ and $\Gamma_2 = \{\phi_2, \phi_3\}$. Γ_1 and Γ_2 share the relation Happy, so we create two copies of this relation: $\texttt{Happy}_1$ and $\texttt{Happy}_2$, one for each subprogram. We introduce Lagrange multipliers λ_p, one for each possible ground tuple $\texttt{Happy}(p)$. We thereby obtain a new program Γ^λ:

$$\begin{array}{lll} 1 & \texttt{GoodNews}(p) => \texttt{Happy}_1(p) & \phi_1' \\ \lambda_p & \texttt{Happy}_1(p) & \varphi_1 \\ 1 & \texttt{BadNews}(p) => \texttt{Sad}(p) & \phi_2 \\ 5 & \texttt{Happy}_2(p) <=> \neg\texttt{Sad}(p) & \phi_3' \\ -\lambda_p & \texttt{Happy}_2(p) & \varphi_2 \end{array}$$

where each φ_i represents a set of singleton rules, one for each value of p (i.e., for each specific person in a given testbed). This program contains two subprograms, $\Gamma_1^\lambda = \{\phi_1', \varphi_1\}$ and $\Gamma_2^\lambda = \{\phi_2, \phi_3', \varphi_2\}$, that can be solved independently with any inference algorithm.

As illustrated in Figure 2, the output of our decomposition method is a bipartite graph between a set of subprograms and a set of relations. In a program-level approach, FELIX attaches an inference algorithm to each subprogram; we call this pairing of algorithm and subprogram a *task*. We discuss how to select decompositions and assign algorithms in Section V.

B. Message Passing

We apply the master-slave message passing scheme [3], [4] for both MRF-level and program-level decomposition. The master-slave approach alternates between two steps: (1) perform inference on each part independently to obtain each part's predictions on shared variables, and (2) a process called the *Master* examines the (possibly conflicting) predictions and sends messages in the form of Lagrange multipliers to each task. The reader can consult [3], [4] for details.[6]

C. Producing the Final Solution

When inference stops, for some variables all of their copies might not have the same value. The last step is to choose a final solution based on solutions from the decomposed parts.

MRF-level Decomposition: To obtain a solution to the original MLN from individual solutions on each tree, we use the heuristic proposed by Komogorov et al. [2] (and used in Komodakis et al. [4]) that sequentially fixes shared-variable values based on max-product messages in individual trees.

Program-level Decomposition: If a shared relation is not subject to any hard rules, FELIX takes majority votes from the predictions of related tasks. (If all copies of this relation have converged, the votes would be unanimous.) To ensure that hard rules in the input MLN program are not violated in the final output, we insist that for any query relation r, all hard rules involving r (if any) be assigned to a single task, and that the final value of r be taken from this task.[7] This guarantees that the final output is a possible world for Γ (provided that the hard rules are satisfiable).

V. SPECIALIZED TASKS

With program-level decomposition, we can use different algorithms for different subprograms. FELIX uses specialized algorithms to handle tasks, which can be more efficient than generic MLN inference. As an existence proof, we describe several tasks that are common in text processing.[8]

Classification: Classification is a fundamental statistical problem and ubiquitous in applications; it arises in Markov logic as a query predicate $R(x, y)$ with hard rules like

$$R(x, y_1) \wedge y_1 \neq y_2 => \neg R(x, y_2),$$

which mandates that each object (represented by a possible value of x) can only be assigned at most one label (represented by a possible value of y).

If the only query relation in a subprogram Γ_i is R and R is mentioned at most once in each rule in Γ_i (except the rule above), then Γ_i is essentially a *logistic regression* (LR) classification model. The inference problem for LR is trivial given model parameters (here rule weights) and feature values (here ground formulae).

Suppose there are N objects and K labels. Then the memory requirement for LR is $O(NK)$. On the other hand, it would require $N\binom{K}{2}$ factors to represent the above rule in an MRF. For tasks such as entity linking (e.g., mapping textual mentions to Wikipedia entities), the value of K could be in the millions.

Other Tasks: FELIX also supports other tasks, including (1) Segmentation and (2) Coreference. Segmentation encodes linear-chain CRF models [6] that contain both unigram and bigram features. Coreference takes a set of N strings and decides which strings represent the same real-world entity. FELIX implements the Viterbi algorithm [6] for segmentation and Singh et al.'s algorithm [12] for coreference.[9]

VI. EXPERIMENTS

Our main hypotheses are that (1) MRF-level decomposition can outperform monolithic inference, and (2) FELIX can outperform both MRF-level decomposition and monolithic inference. We also validate that specialized algorithms indeed have higher efficiency and quality than generic MLN inference algorithms or MRF decomposition.

[6]For MRF-level decomposition, we can perform *exact* inference by running the max-product algorithm [5] on each component.

[7]This policy might result in cascaded subtasks. More sophisticated policies are an interesting future direction.

[8]We describe how FELIX automatically detects these tasks given an MLN program in the full version [8].

[9]We will describe these tasks in more details in the full version [8].

	# Relations in MLN	# MLN Rules	# Evidence Tuples	# MRF Factors	DB Size
IE	17	34	150K	137K	11MB
IERJ	18	357	150K	564K	44MB
NER	2	4	10K	1.7M	134MB
KBP	7	6	4.3M	20M	1.6GB
KBP+	7	6	240M	64B	5.1TB

Table I
DATASET SIZES.

Datasets and MLNs: We use two publicly available MLN testbeds from ALCHEMY's website.[10] In addition, we create an MLN program for named-entity recognition based on skip-chain CRFs [13], and an MLN program that we developed for TAC-KBP, a knowledge-base population challenge.[11] Table I shows some statistics of these datasets.

We describe the MLN testbeds from ALCHEMY's website: (1) **IE**, where one performs segmentation on Cora citations using unigram and bigram features (see the "Isolated" program [9]); (2) **IERJ**, where one performs joint segmentation and entity resolution on Cora citations (see the "Jnt-Seg-ER" program [9]). The last two are (3) **NER**, where one performs named-entity recognition on a dataset with 10K tokens using the skip-chain CRF model [13] (encoded in three MLN rules); and (4) **KBP**, which is an implementation of the TAC-KBP (knowledge-base population) challenge using an MLN that performs entity linking (mapping textual mentions to Wikipedia entities), slot filling (mapping co-occurring mentions to a set of possible relationships), entity-level knowledge-base population, and fact verification from an existing partial knowledge base.

Among those datasets, IE, and NER are simpler programs as they only have unigram and sparse bigram rules and classification constraints with very few labels; IERJ, and KBP are more complex programs as they involve transitivity rules and classification constraints with many labels. To test the scalability of FELIX, we also run the KBP program on a 1.8M-doc TAC-KBP corpus ("**KBP+**").

Experimental Setup: We run TUFFY [7] and ALCHEMY as state-of-the-art monolithic MLN inference systems. We implement MRF-level decomposition and program-level decomposition (i.e., FELIX) approaches on top of the open-source TUFFY system.[12] As TUFFY has similar or superior performance to ALCHEMY on each dataset, here we use TUFFY as a representative for state-of-the-art MLN inference and report ALCHEMY's performance in the technical-report version of this paper. We use the following labels for these three approaches: (1) **TUFFY**, in which TUFFY performs RDBMS-based grounding and uses WalkSAT as its inference algorithm; (2) **TREE**, in which we replace TUFFY's WalkSAT algorithm with tree-based MRF-level dual decomposition as described in Section IV; and (3) **FELIX**, in which we implement program-level dual decomposition as described in Sections IV and V. Table II lists FELIX's decomposition scheme for each dataset.

[10]http://alchemy.cs.washington.edu/
[11]http://nlp.cs.qc.cuny.edu/kbp/2010/
[12]http://research.cs.wisc.edu/hazy/tuffy

	LR	CRF	Coref	WalkSAT
IE	1	0	0	1
IERJ	1	0	1	1
NER	0	1	0	1
KBP	2	0	0	1

Table II
NUMBER OF TASKS OF EACH TYPE IN THE DECOMPOSITIONS USED BY FELIX.

All three approaches are implemented in Java and use PostgreSQL 9.0.4 as the underlying database system. Unless specified otherwise, all experiments are run on a RHEL 6.1 server with four 2.00GHz Intel Xeon CPUs (40 total physical cores plus hyperthreading) and 256 GB of RAM.

A. Overall Efficiency and Quality

We validate that TREE can outperform TUFFY and that FELIX in turn outperforms both TREE and TUFFY on complex programs. To support these claims, we compare the efficiency and quality of all three approaches on the datasets listed above. We run each system on each dataset for 5000 seconds (except for the largest dataset KBP, for which we run 5 hours), and plot the MLN cost against runtime.

From Figure 3 we see that, on the two simpler programs (i.e., IE and NER), all three approaches were able to converge to about the same result quality. Although TREE and TUFFY have similar performance on IE, on NER the TREE approach obtains a low-cost solution within two minutes whereas TUFFY takes more than one hour to reach comparable quality. FELIX has slower convergence behavior than TUFFY and TREE on IE, but it does converge to a similar solution. We note that alternate step-size rules may improve FELIX's performance on these datasets.

On the more complex programs (i.e., IERJ and KBP), FELIX achieves dramatically better performance compared to TUFFY and TREE: while TUFFY and TREE fail to find a feasible solution (i.e., a solution with finite cost) after 5000 seconds on each of these datasets, FELIX converges to a feasible solution within minutes. There are complex structures in these MLNs; e.g., transitivity for entity resolution and uniqueness constraints for entity linking in KBP. TUFFY and TREE were not able to find feasible solutions that satisfy such complex constraints, and the corresponding curves were obtained by replacing hard rules with a "softened" version with weight 100. Still, we see that the results from both TUFFY and TREE are substantially worse than FELIX. We conclude that overall the FELIX approach is able to achieve significantly higher efficiency and quality than TUFFY and TREE approaches.

Scalability: To test scalability, we also run FELIX on the large KBP+ dataset with a parallel RDBMS (from Greenplum Inc.). This MLN converges within a few iterations;

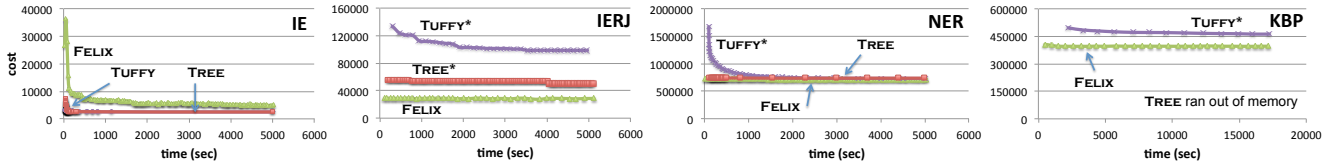

Figure 3. High-level performance results of various approaches to MLN inference. For each dataset and each system, we plot a time-cost curve. Labels ending with an asterisk indicate that some points in the corresponding curves correspond to infeasible solutions (i.e., solutions with infinite cost), and the curves were obtained by "softening" hard rules to have weight 100.

Task	System	Initial	Final	Cost	F1
CRF	FELIX	35 s	35 s	4.6e5	0.90
	TUFFY	148 s	186 s	6.4e5	0.14
	TREE	150 s	911 s	4.7e5	0.16
	ALCHEMY	740 s	760 s	1.4e6	0.10

Table III
PERFORMANCE AND QUALITY COMPARISON ON INDIVIDUAL TASKS. "INITIAL" (RESP. "FINAL") IS THE TIME WHEN A SYSTEM PRODUCED THE FIRST (RESP. CONVERGED) RESULT. FOR TUFFY, THE HARD RULES WERE SOFTENED BECAUSE OTHERWISE NO SOLUTION WAS FOUND.

an iteration takes about five hours in FELIX. TUFFY and ALCHEMY failed to run on this dataset.

B. Specialized Tasks

We next validate that the ability to integrate specialized tasks into MLN inference is key to FELIX's higher efficiency and quality. To do this, we demonstrate that FELIX's specialized algorithms outperform generic MLN inference algorithms in both quality and efficiency when solving specialized tasks. To evaluate this claim, we run FELIX, TUFFY, TREE, and ALCHEMY on three MLN programs encoding a CRF task.[13] We measure application quality (F1 scores) on a subset of the CoNLL 2000 chunking dataset.[14] As shown in Table III, while it always takes less than a minute for FELIX on CRF, the other approaches take much longer. Moreover, FELIX has the best inference quality (i.e., cost) and application quality (i.e., F1).

VII. CONCLUSION

We study how to apply dual decomposition to Markov logic inference. We find that MRF-level decomposition empirically outperforms traditional, monolithic approaches to MLN inference on some programs. However, MRF-level decomposition ignores valuable structural hints in Markov logic programs. Thus, we propose an alternative decomposition strategy that partitions an MLN program into high-level tasks (e.g., classification) that can be solved with specialized algorithms. On several datasets, we empirically show that our program-level decomposition approach outperforms both monolithic inference and MRF-level decomposition approaches to MLN inference.

[13]We leave similar experiments on LR and CC to the full version [8].
[14]http://www.cnts.ua.ac.be/conll2000/chunking/

VIII. ACKNOWLEDGEMENTS

We gratefully acknowledge the support of the DARPA Machine Reading Program under prime contract no. FA8750-09-C-0181. CR is also generously supported by NSF CAREER award under IIS-1054009, ONR award N000141210041, and gifts or research awards from Google, Greenplum, and Oracle. The opinions and conclusions in this paper are not necessarily those of our sponsors.

REFERENCES

[1] V. Jojic, S. Gould, and D. Koller. Accelerated dual decomposition for map inference. In *ICML*, 2010.
[2] V. Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *PAMI*, 2006.
[3] N. Komodakis and N. Paragios. Beyond pairwise energies: Efficient optimization for higher-order MRFs. In *CVPR*, 2009.
[4] N. Komodakis, N. Paragios, and G. Tziritas. MRF optimization via dual decomposition: Message-passing revisited. In *ICCV*, 2007.
[5] F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.
[6] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.
[7] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: Scaling up statistical inference in Markov logic networks using an RDBMS. In *VLDB 2011*.
[8] F. Niu, C. Zhang, C. Ré, and J. Shavlik. Scaling inference for Markov logic via dual decomposition. Technical report, University of Wisconsin-Madison, 2012.
[9] H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI*, 2007.
[10] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 2006.
[11] 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.
[12] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *ACL-HLT*, 2011.
[13] C. Sutton and A. McCallum. Collective segmentation and labeling of distant entities in information extraction. Technical Report 04-49, University of Massachusetts, 2004.
[14] M. Wainwright, T. Jaakkola, and A. Willsky. MAP estimation via agreement on trees: Message-passing and linear programming. *IEEE Transactions on Information Theory*, 2005.
[15] L. Wolsey. *Integer Programming*. Wiley, 1998.