Dual Decomposition Inference for Graphical Models over Strings*

Nanyun Peng and **Ryan Cotterell** and **Jason Eisner**
Department of Computer Science, Johns Hopkins University
{npeng1,ryan.cotterell,eisner}@jhu.edu

Abstract

We investigate dual decomposition for joint MAP inference of many strings. Given an arbitrary graphical model, we decompose it into small acyclic sub-models, whose MAP configurations can be found by finite-state composition and dynamic programming. We force the solutions of these subproblems to agree on overlapping variables, by tuning Lagrange multipliers for an adaptively expanding set of variable-length n-gram count features.

This is the first inference method for arbitrary graphical models over strings that does not require approximations such as random sampling, message simplification, or a bound on string length. *Provided that* the inference method terminates, it gives a certificate of global optimality (though MAP inference in our setting is undecidable in general). On our global phonological inference problems, it always terminates, and achieves more accurate results than max-product and sum-product loopy belief propagation.

1 Introduction

Graphical models allow expert modeling of complex relations and interactions between random variables. Since a graphical model with given parameters defines a probability distribution, it can be used to reconstruct values for unobserved variables. The **marginal inference** problem is to compute the posterior marginal distributions of these variables. The **MAP inference** (or **MPE**) problem is to compute the single highest-probability joint assignment to all the unobserved variables.

Inference in general graphical models is NP-hard even when the variables' values are finite discrete values such as categories, tags or domains. In this paper, we address the more challenging setting where the variables in the graphical models range over strings. Thus, the domain of the variables is an *infinite* space of discrete structures.

In NLP, such graphical models can deal with large, incompletely observed lexicons. They could be used to model diverse relationships among strings that represent spellings or pronunciations; morphemes, words, phrases (such as named entities and URLs), or utterances; standard or variant forms; clean or noisy forms; contemporary or historical forms; underlying or surface forms; source or target language forms. Such relationships arise in domains such as morphology, phonology, historical linguistics, translation between related languages, and social media text analysis.

In this paper, we assume a *given* graphical model, whose factors evaluate the relationships among observed and unobserved strings.[1] We present a dual decomposition algorithm for MAP inference, which returns a certifiably optimal solution when it converges. We demonstrate our method on a graphical model for phonology proposed by Cotterell et al. (2015). We show that the method generally converges and that it achieves better results than alternatives.

The rest of the paper is arranged as follows: We will review graphical models over strings in section 2, and briefly introduce our sample problem in section 3. Section 4 develops dual decomposition inference for graphical models over strings. Then our experimental setup and results are presented in sections 5 and 6, with some discussion.

2 Graphical Models Over Strings

2.1 Factor Graphs and MAP Inference

To perform inference on a graphical model (directed or undirected), one first converts the model to a factor graph representation (Kschischang et al., 2001). A **factor graph** is a finite bipartite

*This material is based upon work supported by the National Science Foundation under Grant No. 1423276.

[1] In some task settings, it is also necessary to discover the model topology along with the model parameters. In this paper we do not treat that structure learning problem. However, both structure learning and parameter learning need to call inference—such as the method presented here—in order to evaluate proposed topologies or improve their parameters.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 917–927,
Lisbon, Portugal, 17-21 September 2015. ©2015 Association for Computational Linguistics.

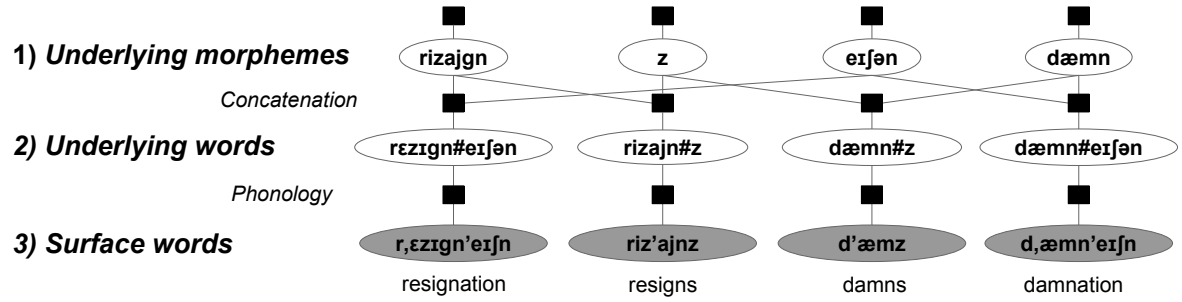


Figure 1: A fragment of the factor graph for the directed graphical model of Cotterell et al. (2015), displaying a possible assignment to the variables (ellipses). The model explains each observed surface word as the result of applying phonology to a concatenation of underlying morphemes. Shaded variables show the observed surface forms for four words: *resignation*, *resigns*, *damns*, and *damnation*. The underlying pronunciations of these words are assumed to be more similar than their surface pronunciations, because the words are known to share latent morphemes. The factor graph encodes what is shared. Each observed word at layer 3 has a latent underlying form at layer 2, which is a deterministic concatenation of latent morphemes at layer 1. The binary factors between layers 2 and 3 score each (underlying,surface) pair for its phonological plausibility. The unary factors at layer 1 score each morpheme for its lexical plausibility. See Cotterell et al. (2015) for discussion of alternatives.

graph over a set $\mathcal{X} = \{X_1, X_2, \ldots\}$ of variables and a set $\mathcal{F}$ of factors. An **assignment** to the variables is a vector of values $\mathbf{x} = (x_1, x_2, \ldots)$. Each factor $F \in \mathcal{F}$ is a real-valued function of $\mathbf{x}$, but it depends on a given x_i *only if* F is connected to X_i in the graph. Thus, a degree d-factor scores some length-d subtuple of $\mathbf{x}$. The score of the whole joint assignment simply sums over all factors:

$$\text{score}(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{F \in \mathcal{F}} F(\mathbf{x}). \quad (1)$$

We seek the $\mathbf{x}$ of maximum score that is consistent with our partial observation of $\mathbf{x}$. This is a generic constraint satisfaction problem with soft constraints. While our algorithm does not depend on a probabilistic interpretation of the factor graph,[2] it can be regarded as peforming maximum *a posteriori* (MAP) inference of the unobserved variables, under the probability distribution $p(\mathbf{x}) \stackrel{\text{def}}{=} (1/Z) \exp \text{score}(\mathbf{x})$.

2.2 The String Case

Graphical models over strings have enjoyed some attention in the NLP community. Tree-shaped graphical models naturally model the evolutionary tree of word forms (Bouchard-Côté et al., 2007; Bouchard-Côté et al., 2008; Hall and Klein, 2010; Hall and Klein, 2011). Cyclic graphical models have been used to model morphological paradigms (Dreyer and Eisner, 2009; Dreyer and Eisner, 2011) and to reconstruct phonological underlying forms of words (Cotterell et al., 2015).

The variables in such a model are strings of unbounded length: each variable X_i is permitted to range over Σ^* where Σ is some fixed, finite alphabet. As in previous work, we assume that a degree-d factor is a d-way rational relation, i.e., a function of d strings that can be computed by a d-tape **weighted finite-state machine (WFSM)** (Mohri et al., 2002; Kempe et al., 2004). Such a machine is called an **acceptor (WFSA)** if $d = 1$ or a **transducer (WFST)** if $d = 2$.[3]

Past work has shown how to approximately *sample* from the distribution over $\mathbf{x}$ defined by such a model (Bouchard-Côté et al., 2007), or approximately compute the distribution's *marginals* using variants of **sum-product belief propagation** (BP) (Dreyer and Eisner, 2009) and expectation propagation (EP) (Cotterell and Eisner, 2015).

2.3 Finite-State Belief Propagation

BP iteratively updates **messages** between factors and variables. Each message is a vector whose elements score the possible values of a variable.

Murphy et al. (1999) discusses BP on cyclic ("loopy") graphs. For pedagogical reasons, suppose momentarily that all factors have degree ≤ 2 (this loses no power). Then BP manipulates only vectors and matrices—whose *dimensionality* depends on the number of possible values of the vari-

[2] E.g., it could be used for exactly computing the separation oracle when training a structural SVM (Tsochantaridis et al., 2005; Finley and Joachims, 2007). Another use is minimum Bayes risk decoding—computing the joint assignment having minimum expected loss—if the loss function does not decompose over the variables, but a factor graph can be constructed that evaluates the expected loss of any assignment.

[3] Finite-state software libraries often support only these cases. Accordingly, Cotterell and Eisner (2015, Appendix B.10) explain how to eliminate factors of degree $d > 2$.

ables. In the string case, they have *infinitely* many rows and columns, indexed by possible strings.

Dreyer and Eisner (2009) represented these infinite vectors and matrices by WFSAs and WFSTs, respectively. They observed that the simple linear-algebra operations used by BP can be implemented by finite-state constructions. The pointwise product of two vectors is the intersection of their WFSAs; the marginalization of a matrix is the projection of its WFST; a vector-matrix product is computed by composing the WFSA with the WFST and then projecting onto the output tape. For degree > 2, BP's rank-d tensors become d-tape WFSMs, and these constructions generalize.

Unfortunately, except in small acyclic models, the BP messages—which are WFSAs—usually become impractically large. Each intersection or composition involves a cross-product construction. For example, when finding the marginal distribution at a degree-d variable, intersecting d WFSA messages having m states each may yield a WFSA with up to m^d states. (Our models in section 6 include variables with d up to 156.) Combining *many* cross products, as BP iteratively passes messages along a path in the factor graph, leads to blowup that is exponential in the length of the path—which in turn is unbounded if the graph has cycles (Dreyer and Eisner, 2009), as ours do.

The usual solution is to prune or otherwise approximate the messages at each step. In particular, Cotterell and Eisner (2015) gave a principled way to approximate the messages using variable-length n-gram models, using an adaptive variant of Expectation Propagation (Minka, 2001).

2.4 Dual Decomposition Inference

In section 4, we will present a dual decomposition (DD) method that decomposes the original complex problem into many small subproblems that are free of cycles and high degree nodes. BP can solve each subproblem without approximation.[4]

The subproblems "communicate" through Lagrange multipliers that guide them towards agreement on a single global solution. This information is encoded in WFSAs that score possible values of a string variable. DD incrementally adjusts the WFSAs so as to encourage values that agree with the variable's average value across subproblems.

Unlike BP messages, the WFSAs in our DD method will be restricted to be variable-length n-gram models, similar to Cotterell and Eisner (2015). They may still grow over time; but DD often halts while the WFSAs are still small. It halts when its strings agree exactly, rather than when it has converged up to a numerical tolerance, like BP.

2.5 Switching Between Semirings

Our factors may be *nondeterministic* WFSMs. So when $F \in \mathcal{F}$ scores a given d-tuple of string values, it may accept that d-tuple along multiple different WFSM paths with different scores, corresponding to different alignments of the strings.

For purposes of MAP inference, we define F to return the *maximum* of these path scores. That is, we take the WFSMs to be defined with weights in the $(\max, +)$ semiring (Mohri et al., 2002). Equivalently, we are seeking the "best global solution" in the sense of choosing not only the strings x_i but also the alignments of the d-tuples.[5]

To do so, we must solve each DD subproblem in the same sense. We use **max-product BP**. This still applies the Dreyer-Eisner method of section 2.3. Since these WFSMs are defined in the $(\max, +)$ semiring, the method's finite-state operations will combine weights using $\max$ and $+$.

MAP inference in our setting is in general computationally undecidable.[6] However, if DD converges (as in our experiments), then *its solution is guaranteed to be the true MAP assignment.*

In section 6, we will compare DD with (loopy) max-product BP and (loopy) sum-product BP. These respectively approximate MAP inference and marginal inference over the entire factor graph. Marginal inference computes marginal string probabilities that sum (rather than maximize) over the choices of other strings *and the choices of paths*. Thus, for sum-product BP, we re-interpret the factor WFSMs as defined over the $(\text{logadd}, +)$ semiring. This means that the exponentiated score assigned by a WFSM is the sum of the exponentiated scores of the accepting paths.

[4] Such small BP problems commonly arise in NLP. In particular, using finite-state methods to decode a composition of several finite-state noisy channels (Pereira and Riley, 1997; Knight and Graehl, 1998) can be regarded as BP on a graphical model over strings that has a linear-chain topology.

[5] This problem is more specifically called **MPE inference**.

[6] The trouble is that we cannot bound the length of the latent strings. If we could, then we could encode them using a finite set of boolean variables, and solve as an ILP problem. But that would allow us to determine whether there exists a MAP assignment with score ≥ 0. That is impossible in general, because it would solve Post's Correspondence Problem as a simple special case (see Dreyer and Eisner (2009)).

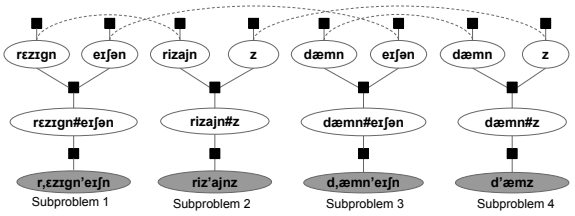


Figure 2: To apply dual decomposition, we choose to decompose 1 into one subproblem per surface word. Dashed lines connect two or more variables from different subproblems that correspond to the same variable in the original graph. The method of Lagrange multipliers is used to force these variables to have identical values. An additional unary factor attached to each subproblem variable (not shown) is used to incorporate its Lagrangian term.

3 A Sample Task: Generative Phonology

Before giving the formal details of our DD method, we give a motivating example: a recently proposed graphical model for morphophonology. Cotterell et al. (2015) defined a Bayesian network to describe the generative process of phonological words. Our Figure 1 shows a conversion of their model to a factor graph and explains what the variables and factors mean.

Inference on this graph performs *unsupervised discovery of latent strings*. Given observed surface representations of words (SRs), inference aims to recover the underlying representations (URs) of the words and their shared constituent morphemes. The latter can then be used to predict held-out SRs.

Notice that the 8 edges in the first layer of Figure 1 form a cycle; such cycles make BP inexact. Moreover, the figure shows only a schematic fragment of the graphical model. In the actual experiments, the graphical models have up to 829 variables, and the variables representing morpheme URs are connected to up to 156 factors (because many words share the same affix).

To handle the above challenges without approximation, we want to decompose the original problem into subproblems where each subproblem can be solved efficiently. In particular, we want the subproblems to be free of cycles and high-degree nodes. In our phonology example, each observed word along with its correspondent latent URs forms an ideal subproblem. This decomposition is shown in Figure 2.

While the subproblems can be solved efficiently in isolation, they may share variables, as shown by the dashed lines in Figure 2. DD repeatedly modifies and re-solves the subproblems until they agree on their shared variables.

4 Dual Decomposition

Dual decomposition is a general technique for solving constrained optimization problems. It has been widely used for MAP inference in graphical models (Komodakis et al., 2007; Komodakis and Paragios, 2009; Koo et al., 2010; Martins et al., 2011; Sontag et al., 2011; Rush and Collins, 2014). However, previous work has focused on variables X_i whose values are in $\mathbb{R}$ or a small finite set; we will consider the infinite set Σ^*.

4.1 Review of Dual Decomposition

To apply dual decomposition, we must partition the original problem into a union of K subproblems, each of which can be solved exactly and efficiently (and in parallel). For example, our experiments partition Figure 1 as shown in Figure 2.

Specifically, we partition the factors into K sets $\mathcal{F}^1, \ldots, \mathcal{F}^K$. Each factor $F \in \mathcal{F}$ appears in exactly one of these sets. This lets us rewrite the score (1) as $\sum_k \sum_{F \in \mathcal{F}^k} F(\mathbf{x})$. Instead of simply seeking its maximizer $\mathbf{x}$, we equivalently seek

$$\operatorname*{argmax}_{\mathbf{x}^1,\ldots,\mathbf{x}^K} \sum_{k=1}^{K} \Big(\sum_{F \in \mathcal{F}^k} F(\mathbf{x}^k) \Big) \text{ s.t. } \mathbf{x}^1 = \cdots = \mathbf{x}^K \tag{2}$$

If we dropped the equality constraint, (2) could be solved by *separately* maximizing $\sum_{F \in \mathcal{F}^k} F(\mathbf{x}^k)$ for each k. This "subproblem" is itself a MAP problem which considers only the factors $\mathcal{F}^k$ and the variables $\mathcal{X}^k$ adjacent to them in the original factor graph. The subproblem objective does not depend on the other variables.

We now attempt to enforce the equality constraint indirectly, by adding Lagrange multipliers that encourage agreement among the subproblems. Assume for the moment that the variables in the factor graph are real-valued (each x_i^k is in $\mathbb{R}$). Then consider the **Lagrangian relaxation** of (2),

$$\max_{\mathbf{x}^1,\ldots,\mathbf{x}^K} \sum_{k=1}^{K} \Big(\sum_{F \in \mathcal{F}^k} F(\mathbf{x}^k) + \sum_i \lambda_i^k \cdot x_i^k \Big) \tag{3}$$

This can still be solved by separate maximizations. For *any* choices of $\lambda_i^k \in \mathbb{R}$ having $(\forall i) \sum_k \lambda_i^k = 0$, it upper-bounds the objective of (2). Why? The solution to (2) achieves the same value in (3), yet (3) may do even better by considering solutions that do not satisfy the constraint. Our goal is to find λ_i^k values that tighten this upper bound as much as possible. *If* we can find λ_i^k values so that

the optimum of (3) satisfies the equality constraint, then we have a tight bound and a solution to (2).

To improve the method, recall that subproblem k considers only variables $\mathcal{X}^k$. It is indifferent to the value of X_i if $X_i \notin \mathcal{X}^k$, so we just leave x_i^k undefined in the subproblem's solution. We treat that as automatically satisfying the equality constraint; thus we do not need any Lagrange multiplier λ_i^k to force equality. Our final solution $\mathbf{x}$ ignores undefined values, and sets x_i to the value agreed on by the subproblems that *did* consider X_i.[7]

4.2 Substring Count Features

But what do we do if the variables are strings? The Lagrangian term $\lambda_i^k \cdot x_i^k$ in (3) is now ill-typed. We replace it with $\boldsymbol{\lambda}_i^k \cdot \boldsymbol{\gamma}(x_i^k)$, where $\boldsymbol{\gamma}(\cdot)$ extracts a real-valued **feature vector** from a string, and $\boldsymbol{\lambda}_i^k$ is a vector of Lagrange multipliers.

This corresponds to changing the constraint in (2). Instead of requiring $x_i^1 = \cdots = x_i^K$ for each i, we are now requiring $\boldsymbol{\gamma}(x_i^1) = \cdots = \boldsymbol{\gamma}(x_i^K)$, i.e., these strings must agree *in their features*.

We want each possible string to have a unique feature vector, so that matching features forces the actual strings to match. We follow Paul and Eisner (2012) and use a **substring count feature** for each $w \in \Sigma^*$. In other words, $\boldsymbol{\gamma}(x)$ is an infinitely long vector, which maps each w to the number of times that w appears in x as a substring.[8]

Computing $\boldsymbol{\lambda}_i^k \cdot \boldsymbol{\gamma}(x_i^k)$ in (3) remains possible because in practice, $\boldsymbol{\lambda}_i^k$ will have only finitely many nonzeros. This is so because our feature vector $\boldsymbol{\gamma}(x)$ has only finitely many nonzeros for any string x, and the subgradient algorithm in section 4.3 below always updates $\boldsymbol{\lambda}_i^k$ by adding multiples of such $\boldsymbol{\gamma}(x)$ vectors.

We will use a further trick below to prevent rapid growth of this finite set of nonzeros. Each variable X_i maintains an **active set** of features, $\mathcal{W}_i$. Only these features may have nonzero Lagrange multipliers. While the active set can grow over time, it will be finite at any given step.

Given the Lagrange multipliers, subproblem k of (3) is simply MAP inference on the factor graph consisting of the variables $\mathcal{X}^k$ and factors $\mathcal{F}^k$ *as well as* an extra unary factor G_i^k at each $X_i \in \mathcal{X}^k$:

$$G_i^k(\mathbf{x}^k) \stackrel{\text{def}}{=} \boldsymbol{\lambda}_i^k \cdot \boldsymbol{\gamma}(x_i^k) \tag{4}$$

These unary factors penalize strings according to the Lagrange multipliers. They can be encoded as WFSAs (Allauzen et al., 2003; Cotterell and Eisner, 2015, Appendices B.1–B.5), allowing us to solve the subproblem by max-product BP as usual. The topology of the WFSA for G_i^k depends only on $\mathcal{W}_i$, while its weights come from $\boldsymbol{\lambda}_i^k$.

4.3 Projected Subgradient Method

We aim to adjust the collection $\boldsymbol{\lambda}$ of Lagrange multipliers to *minimize* the upper bound (3). Following Komodakis et al. (2007), we solve this convex **dual problem** using a projected subgradient method. We initialize $\boldsymbol{\lambda} = \mathbf{0}$ and compute (3) by solving the K subproblems. Then we take a step to adjust $\boldsymbol{\lambda}$, and repeat in hopes of eventually satisfying the equality condition.

The projected subgradient step is

$$\boldsymbol{\lambda}_i^k := \boldsymbol{\lambda}_i^k + \eta \cdot \Big(\mu_i - \boldsymbol{\gamma}(x_i^k)\Big) \tag{5}$$

where $\eta > 0$ is the current step size, and μ_i is the mean of $\boldsymbol{\gamma}(x_i^{k'})$ over all subproblems k' that consider X_i. This update modifies (3) to encourage solutions x^k such that $\gamma(x_i^k)$ comes closer to μ_i.

For each i, we update all $\boldsymbol{\lambda}_i^k$ at once to preserve the property that $(\forall i) \sum_k \lambda_i^k = 0$. However, we are only allowed to update components of the $\boldsymbol{\lambda}_i^k$ that correspond to features in the active set $\mathcal{W}_i$. To ensure that we continue to make progress even after we agree on these features, we first expand $\mathcal{W}_i$ by adding the *minimal* strings (if any) on which the x_i^k do not yet all agree. For example, we will add the `abc` feature only when the x_i^k already agree on their counts of its substrings `ab` and `bc`.[9]

Algorithm 1 summarizes the whole method. Table 1 illustrates how one active set $\mathcal{W}_i$ (section 4.3) evolves, in our experiments, as it tries to enforce agreement on a particular string x_i.

4.4 Past Work: Implicit Intersection

Our DD algorithm is an extension of one that Paul and Eisner (2012) developed for the simpler **implicit intersection** problem. Given many WFSAs $F_1, \ldots, F_K$, they were able to find the string x with maximum total score $\sum_{k=1}^K F_k(x)$. (They applied this to solve instances of the NP-hard **Steiner**

[7] Without this optimization, the Lagrangian term $\lambda_i^k \cdot x_i^k$ would have driven x_i^k to match that value anyway.

[8] More precisely, the number of times that w appears in BOS x EOS, where BOS, EOS are distinguished boundary symbols. We allow w to start with BOS and/or end with EOS, which yields prefix and suffix indicator features.

[9] In principle, we should check that they also (still) agree on `a`, `b`, and `c`, but we skip this check. Our active set heuristic is almost identical to that of Paul and Eisner (2012).

Algorithm 1 DD for graphical models over strings

initialize the active set $\mathcal{W}_i$ for each variable $X_i \in \mathcal{X}$
initialize $\boldsymbol{\lambda}_i^k = \mathbf{0}$ for each X_i and each subproblem k
for t = 1 to T **do** ▷ max number of iterations
  for k = 1 to K **do** ▷ solve all primal subproblems
    if any of the $\boldsymbol{\lambda}_i^k$ have changed **then**
      run max-product BP on the acyclic graph defined by variables $\mathcal{X}^k$ and factors $\mathcal{F}^k$ and G_i^k
      extract MAP strings: $\forall i$ with $X_i \in \mathcal{X}^k$, x_i^k is the label of the max-scoring accepting path in the WFSA that represents the belief at X_i
  for each $X_i \in \mathcal{X}$ **do** ▷ improve dual bound
    if the defined strings x_i^k are not all equal **then**
      Expand active feature set $\mathcal{W}_i$ ▷ section 4.3
      Update each $\boldsymbol{\lambda}_i^k$ ▷ equation (5)
      Update each G_i^k from $\Theta_i, \boldsymbol{\lambda}_i^k$ ▷ see (4)
  if none of the X_i required updates **then**
    return any defined x_i^k (all are equal) for each i
return $\{x_i^1, \ldots, x_i^K\}$ for each i ▷ failed to converge

string problem, i.e., finding the string x of minimum total edit distance to a collection of $K \approx 100$ given strings.) The naive solution to this problem would be to find the highest-weighted path in the intersection $F_1 \cap \cdots \cap F_K$. Unfortunately, the intersection of WFSAs takes the Cartesian product of their state sets. Thus materializing this intersection would have taken time exponential in K.

To put this another way, inference is NP-hard even on a "trivial" factor graph: a *single* variable X_1 attached to K factors. Recall from section 2.3 that BP would solve this via the expensive intersection above. Paul and Eisner (2012) instead applied DD with one subproblem per factor. We generalize their method to handle arbitrary factor graphs, with multiple latent variables and cycles.

4.5 Block Coordinate Update

We also explored a possible speedup for our algorithm. We used a **block coordinate update** variant of the algorithm when performing inference on the phonology problem and observed an empirical speedup. Block coordinate updates are widely used in Lagrangian relaxation and have also been explored specifically for dual decomposition.

In general, block algorithms minimize the objective by holding some variables fixed while updating others. Sontag et al. (2011) proposed a sophisticated block method called MPLP that considers all values of variable X_i instead of the ones obtained from the best assignments for the subproblems. However, it is not clear how to apply their technique to string-valued variables. Instead, the algorithm we propose here is much simpler—it divides the *primal* variables into groups and updates each group's associated *dual* variables in turn, using a single subgradient step (5). Note that this way of partitioning the dual variables has the nice property that we can still use the projected subgradient update we gave in (5) and preserve the property that $(\forall i) \sum_k \lambda_i^k = 0$.

In the graphical model for generative phonology, there are two types of underlying morphemes in the first layer: word stems and word affixes. Our block coordinate update algorithm thus alternates between subgradient updates to the dual variables for the stems and the dual variables for the affixes. Note that when performing block coordinate update on the dual variables, the primal variables are *not* held constant, but rather are chosen by optimizing the corresponding subproblem.

Iter#	x_i^1	x_i^2	x_i^3	x_i^4	$\Delta\mathcal{W}_i$
1	ϵ	ϵ	ϵ	ϵ	∅
3	g	g	g	g	∅
4	gris	griz	griz	griz	{s, z, is, iz, s$ z$ }
5	gris	grizo	griz	griz	{o, zo, o$ }
14	griz	grizo	griz	griz	∅
17	griz	griz	griz	griz	∅
18	griz	griz	grize	griz	{ e, ze, e$ }
19	gris	griz	griz	griz	∅
31	griz	griz	griz	griz	∅

Table 1: One variable's active set as DD runs. This variable is the unobserved stem morpheme shared by the Catalan words gris, grizos, grize, grizes. The second column shows the current set of solutions from the 4 subproblems having copies of this variable. The third column shows the new substrings that are then added to the active set, to try to enforce agreement via their Lagrange multipliers. The table does not show iterations in which these columns have not changed. However, those iterations still update the Lagrange multipliers to more strongly encourage agreement (if needed). Although agreement is achieved at iterations 1, 3, and 17, it is then disrupted—the subproblems' solutions change because of Lagrange-multiplier pressures on their *other* variables (suffixes that do *not* agree yet). At iteration 31, the variable returns to agreement on griz, and never changes again.

5 Experimental Setup

5.1 Datasets

We compare DD to belief propagation, using the graphical model for generative phonology discussed in section 3. Inference in this model aims to reconstruct underlying morphemes. Since our focus is inference, we will evaluate these reconstructions directly (whereas Cotterell et al. (2015) evaluated their ability to predict novel surface forms using the reconstructions).

Our factor graphs have a similar topology to the pedagogical fragment shown in Figure 1. How-

ever, they are actually derived from datasets constructed by Cotterell et al. (2015), which are available with full descriptions at `http://hubal.cs.jhu.edu/tacl2015/`. Briefly:

EXERCISE Small datasets of Catalan, English, Maori, and Tangale, drawn from phonology textbooks. Each dataset contains 55 to 106 surface words, formed from a collection of 16 to 55 morphemes.

CELEX Larger datasets of German, English, and Dutch, drawn from the CELEX database (Baayen et al., 1995). Each dataset contains 1000 surface words, formed from 341 to 381 underlying morphemes.

5.2 Evaluation Scheme

We compared three types of inference:

DD Use DD to perform *exact MAP inference*.

SP Perform *approximate marginal inference* by sum-product loopy BP with pruning (Cotterell et al., 2015).

MP Perform *approximate MAP inference* by max-product loopy BP with pruning. DD and SP improve this baseline in different ways.

DD predicts a string value for each variable. For SP and MP, we deem the prediction at a variable to be the string that is scored most highly by the belief at that variable.

We report the fraction of predicted morpheme URs that exactly match the gold-standard URs proposed by a human (Cotterell et al., 2015). We also compare these predicted URs to one another, to see how well the methods agree.

5.3 Parameterization

The model of Cotterell et al. (2015) has two factor types whose parameters must be chosen.[10] The first is a unary factor M_ϕ. Each underlying-morpheme variable (layer 1 of Figure 1) is connected to a copy of M_ϕ, which gives the prior distribution over its values. The second is a binary factor S_θ. For each surface word (layer 3), a copy of S_θ gives its conditional distribution given the corresponding underlying word (layer 2). M_ϕ and S_θ respectively model the *lexicon* and the *phonology* of the specific language; both are encoded as WFSMs.

M_ϕ is a 0-gram generative model: at each step it emits a character chosen uniformly from the alphabet Σ with probability ϕ, or halts with probability $1-\phi$. It favors shorter strings in general, but ϕ determines how weak this preference is.

S_θ is a sequential edit model that produces a word's SR by stochastically copying, inserting, substituting, and deleting the phonemes of its UR. We explore two ways of parameterizing it.

Model 1 is a simple model in which θ is a scalar, specifying the probability of copying the next character of the underlying word as it is transduced to the surface word. The remaining probability mass $1-\theta$ is apportioned equally among insertion, substitution and deletion operations.[11] This models phonology as "noisy concatenation"—the minimum necessary to account for the fact that surface words cannot quite be obtained as simple concatenations of their shared underlying morphemes.

Model 2 is a replication of the much more complicated parametric model of Cotterell et al. (2015), which can handle linguistic phonology. Here the factor S_θ is a *contextual* edit FST (Cotterell et al., 2014). The probabilities of competing edits in a given context are determined by a log-linear model with weight vector θ and features that are meant to pick up on phonological phenomena.

5.4 Training

When evaluating an inference method from section 5.2, we use the same inference method both for prediction and within training.

We train Model 1 by grid search. Specifically, we choose $\phi \in [0.65, 1)$ and $\theta \in [0.25, 1)$ such that the predicted forms maximize the joint score (1) (always using the $(\max, +)$ semiring).

For Model 2, we compared two methods for training the ϕ and θ parameters (θ is a vector):

Model 2S Supervised training, which *observes* the "true" (hand-constructed) values of the URs. This idealized setting uses the best possible parameters (trained on the test data).

Model 2E Expectation maximization (EM), whose E step *imputes* the unobserved URs.

EM's E step calls for *exact marginal inference*, which is intractable for our model. So we substitute the same inference method that we are test-

[10] The model also has a three-way factor, connecting layers 1 and 2 of Figure 1. This represents deterministic concatenation (appropriate for these languages) and has no parameters.

[11] That is, probability mass of $(1-\theta)/3$ is divided equally among the $|\Sigma|$ possible insertions; another $(1-\theta)/3$ is divided equally among the $|\Sigma|-1$ possible substitutions; and the final $(1-\theta)/3$ is allocated to deletion.

ing. This gives us three approximations to EM, based on DD, SP and MP. Note that DD specifically gives the Viterbi approximation to EM—which sometimes gets better results than true EM (Spitkovsky et al., 2010). For MP (but not SP), we extract only the 1-best predictions for the E step, since we study MP as an approximation to DD.

As initialization, our first E step uses the trained version of Model 1 for the same inference method.

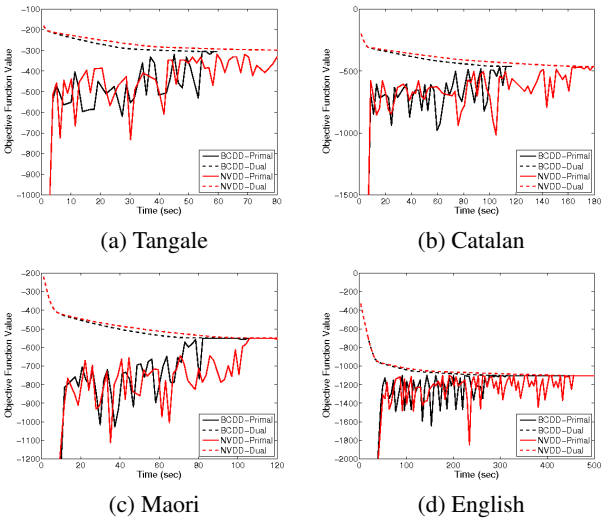
5.5 Inference Details

We run SP and MP for 20 iterations (usually the predictions converge within 10 iterations). We run DD to convergence (usually < 600 iterations). DD iterations are much faster since each variable considers d strings, not d *distributions* over strings. Hence DD does not intersect distributions, and many parts of the graph settle down early because discrete values can converge in finite time.[12]

We follow Paul and Eisner (2012, section 5.1) fairly closely. In particular: Our stepsize in (5) is $\eta = \alpha/(t + 500)$, where t is the iteration number; $\alpha = 1$ for Model 2S and $\alpha = 10$ otherwise. We proactively include all 1-gram and 2-gram substring features in the active sets $\mathcal{W}_i$ at initialization, rather than adding them only as needed. At iterations 200, 400, and 600, we proactively add all 3-, 4-, and 5-gram features (respectively) on which the counts still disagree; this accelerates convergence on the few variables that have not already converged. We handle negative-weight cycles as Paul and Eisner do. If we had ever failed to converge within 2000 iterations, we would have used their heuristic to extract a prediction anyway.

Model 1 suffers from a symmetry-breaking problem. Many edits have identical probability, and when we run inference, many assignments will tie for highest scoring configuration. This can prevent DD from converging and makes performance hard to measure. To break these ties, we add "jitter" separately to each copy of M_ϕ in Figure 1. Specifically, if F_i is the unary factor attached to X_i, we expand our 0-gram model $F_i(x) = \log((p/|\Sigma|)^{|x|} \cdot (1-p))$ to become $F_i(x) = \log(\prod_{c\in\Sigma} p_{c,i}^{|x|_c} \cdot (1-p))$, where $|x|_c$ denotes the count of character c in string x, and $p_{c,i} \propto (p/|\Sigma|) \cdot \exp \varepsilon_{c,i}$ where $\varepsilon_{c,i} \sim N(0, 0.01)$ and we preserve $\sum_{c\in\Sigma} p_{c,i} = p$.

[12] A variable need not update $\boldsymbol{\lambda}$ if its strings agree; a subproblem is not re-solved if none of its variables updated $\boldsymbol{\lambda}$.



(a) Tangale (b) Catalan

(c) Maori (d) English

Figure 3: The primal-dual curve of NVDD v.s. BCDD on 4 EXERCISE languages. BCDD always converges faster.

6 Experimental Results

6.1 Convergence and Speed of DD

As linguists know, reconstructing an underlying stem or suffix can be difficult. We may face insufficient evidence or linguistic irregularity—or regularity that goes unrecognized because the phonological model is impoverished (Model 1) or poorly trained (early EM iterations on Model 2). DD may then require extensive negotiation to resolve disagreements among subproblems. Furthermore, DD must renegotiate as conditions change elsewhere in the factor graph (Table 1).

DD converged in all of our experiments. Note that DD (section 4.3) has converged when all the equality constraints in (2) are satisfied. In this case, we have found the true MAP configuration.

In section 4.5, we discussed a block coordinate update variation (BCDD) of our DD algorithm. Figure 3 shows the convergence behavior of BCDD against the naive projected subgradient algorithm (NVDD) on the four EXERCISE languages under Model 1. The dual objective (3) always upper-bounds the primal score (i.e., the score (1) of an assignment derived heuristically from the current subproblem solutions). The dual decreases as the algorithm progresses. When the two objectives meet, we have found an optimal solution to the primal problem. We can see in Figure 3 that our DD algorithm converges quickly on the four EXERCISE languages and BCDD converges consistently faster than NVDD. We use BCDD in the remaining experiments.

When DD runs fast, it is competitive with the

	DD	SP	MP	Gold
DD		92.74%	90.55%	**96.92%**
SP			95.22%	94.63%
MP				90.63%

(a) The 4 EXERCISE languages under Model 1

	DD	SP	MP	Gold
DD		88.05%	85.19%	**89.66%**
SP			92.64%	85.71%
MP				83.46%

(b) The 3 CELEX languages under Model 1

	DD	SP	MP	Gold
DD		96.53%	100%	**98.67%**
SP			96.53%	96.05%
MP				**98.67%**

(c) The 3 CELEX languages under Model 2S (EXERCISE dataset gives 100% everywhere)

	DD	SP	MP	Gold
DD		92.43%	89.39%	**98.18%**
SP			96.73%	95.42%
MP				90.74%

(d) The 4 EXERCISE languages under Model 2E

Table 2: Pairwise agreement (on morpheme URs) of DD, SP, MP and the gold standard, for each group of inference problems. Boldface is highest accuracy (agreement with gold).

other methods. It is typically faster on the EXERCISE data, and a few times slower on the CELEX data. But we stop the other methods after 20 iterations, whereas *DD runs until it gets an exact answer*. We find that this runtime is unpredictable and sometimes quite long. In the grid search for training Model 1, we observed that changes in the parameters (ϕ, θ) could cause the runtime of DD inference to vary by 2 orders of magnitude. Similarly, on the CELEX data, the runtime on Model 1 (over 10 different $N = 600$ subsets of English) varied from about 1 hour to nearly 2 days.[13]

6.2 Comparison of Inference

For each language, we constructed several different unsupervised prediction problems. In each problem, we observe some size-N subset of the words in our dataset, and we attempt to predict the URs of the morphemes in those words. For each CELEX language, we took $N = 600$, and used three of the size-N training sets from (Cotterell et al., 2015). For each EXERCISE language, we took N to be one less than the dataset size, and used all $N + 1$ subsets of size N, again similar to (Cotterell et al., 2015). We report the unweighted macro-average of all these accuracy numbers.

[13]Note that our implementation is not optimized; e.g., it uses Python (not Cython).

We compare DD, SP, and MP inference on each language under different settings. Table 2 shows aggregate results, as an unweighted average over multiple languages and training sets. We present various additional results at `http://cs.jhu.edu/~npeng/emnlp2015/`, including a per-language breakdown of the results, runtime numbers, and significance tests.

The results for Model 1 are shown in Tables 2a and 2b. As we can see, in both datasets, dual decomposition performed the best at recovering the URs, while MP performed the worst. Both DD and MP are doing MAP inference, so the differences reflect the search error in MP. Interestingly, DD agrees more with SP than with MP, even though SP uses marginal inference.

Although the aggregate results on the EXERCISE dataset show a large improvement of DD over both of the BP algorithms, the gain all comes from the English language. SP actually does better than DD on Catalan and Maori, and MP also gets better results than DD on Maori, tying with SP.

For Model 2S, all inference methods achieved 100% accuracy on the EXERCISE dataset, so we do not show a table. The results on the CELEX dataset are shown in Table 2c. Here both DD and MP performed equally well, and outperformed BP—a result like (Spitkovsky et al., 2010). This trend is consistent over all three languages: DD and MP always achieve similar results and both outperform SP. Of course, one advantage of DD in the setting is that it actually finds the true MAP prediction of the model; the errors are known to be due to the model, not the search procedure.

For Model 2E, we show results on the EXERCISE dataset in Table 2d. Here the results resemble the pattern of Model 1.

7 Conclusion and Future Work

We presented a general dual decomposition algorithm for MAP inference on graphical models over strings, and applied it to an unsupervised learning task in phonology. The experiments show that our DD algorithm converges and gets better results than both max-product and sum-product BP.

Techniques should be explored to speed up the DD method. Adapting the MPLP algorithm (Sontag et al., 2011) to the string-valued case would be a nontrivial extension. We could also explore other serial update schemes, which generally speed up message-passing algorithms over parallel update.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of ACL*, pages 40–47.

R. Harald Baayen, Richard Piepenbrock, and Leon Gulikers. 1995. The CELEX lexical database on CD-ROM.

Alexandre Bouchard-Côté, Percy Liang, Thomas L Griffiths, and Dan Klein. 2007. A probabilistic approach to diachronic phonology. In *Proceedings of EMNLP-CoNLL*, pages 887–896.

Alexandre Bouchard-Côté, Percy Liang, Thomas Griffiths, and Dan Klein. 2008. A probabilistic approach to language change. In *Proceedings of NIPS*.

Ryan Cotterell and Jason Eisner. 2015. Penalized expectation propagation for graphical models over strings. In *Proceedings of NAACL-HLT*, pages 932–942, Denver, June. Supplementary material (11 pages) also available.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2014. Stochastic contextual edit distance and probabilistic FSTs. In *Proceedings of ACL*, Baltimore, June. 6 pages.

Ryan Cotterell, Nanyun Peng, and Jason Eisner. 2015. Modeling word forms using latent underlying morphs and phonology. *Transactions of the Association for Computational Linguistics*, 3:433–447.

Markus Dreyer and Jason Eisner. 2009. Graphical models over multiple strings. In *Proceedings of EMNLP*, pages 101–110, Singapore, August.

Markus Dreyer and Jason Eisner. 2011. Discovering morphological paradigms from plain text using a Dirichlet process mixture model. In *Proceedings of EMNLP*, pages 616–627, Edinburgh, July.

Thomas Finley and Thorsten Joachims. 2007. Parameter learning for loopy markov random fields with structural support vector machines. In *ICML Workshop on Constrained Optimization and Structured Output Spaces*.

David Hall and Dan Klein. 2010. Finding cognate groups using phylogenies. In *Proceedings of ACL*.

David Hall and Dan Klein. 2011. Large-scale cognate recovery. In *Proceedings of EMNLP*.

André Kempe, Jean-Marc Champarnaud, and Jason Eisner. 2004. A note on join and auto-intersection of n-ary rational relations. In Loek Cleophas and Bruce Watson, editors, *Proceedings of the Eindhoven FASTAR Days (Computer Science Technical Report 04-40)*, pages 64–78. Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Netherlands, December.

Kevin Knight and Jonathan Graehl. 1998. Machine transliteration. *Computational Linguistics*, 24(4).

Nikos Komodakis and Nikos Paragios. 2009. Beyond pairwise energies: Efficient optimization for higher-order MRFs. In *Proceedings of CVPR*, pages 2985–2992. IEEE.

Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. 2007. MRF optimization via dual decomposition: Message-passing revisited. In *Proceedings of ICCV*, pages 1–8. IEEE.

Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. 2010. Dual decomposition for parsing with non-projective head automata. In *Proceedings of EMNLP*, pages 1288–1298.

F. R. Kschischang, B. J. Frey, and H. A. Loeliger. 2001. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, February.

André Martins, Mário Figueiredo, Pedro Aguiar, Eric P. Xing, and Noah A. Smith. 2011. An augmented lagrangian approach to constrained map inference. In *Proceedings of ICML*, pages 169–176.

Thomas P. Minka. 2001. Expectation propagation for approximate Bayesian inference. In *Proceedings of UAI*, pages 362–369.

Mehryar Mohri, Fernando Pereira, and Michael Riley. 2002. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. 1999. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of UAI*, pages 467–475.

Michael J. Paul and Jason Eisner. 2012. Implicitly intersecting weighted automata using dual decomposition. In *Proceedings of NAACL*, pages 232–242.

Fernando C. N. Pereira and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. In Emmanuel Roche and Yves Schabes, editors, *Finite-State Language Processing*. MIT Press, Cambridge, MA.

Alexander M. Rush and Michael Collins. 2014. A tutorial on dual decomposition and Lagrangian relaxation for inference in natural language processing. Technical report available from `arXiv.org` as `arXiv:1405.5208`.

David Sontag, Amir Globerson, and Tommi Jaakkola. 2011. Introduction to dual decomposition for inference. *Optimization for Machine Learning*, 1:219–254.

Valentin I. Spitkovsky, Hiyan Alshawi, Daniel Jurafsky, and Christopher D. Manning. 2010. Viterbi training improves unsupervised dependency parsing. In *Proceedings of CoNLL*, page 917, Uppsala, Sweden, July.

I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. 2005. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, September.