Implicitly Intersecting Weighted Automata using Dual Decomposition[*]

Michael J. Paul and **Jason Eisner**
Department of Computer Science / Johns Hopkins University
Baltimore, MD 21218, USA
{mpaul,jason}@cs.jhu.edu

Abstract

We propose an algorithm to find the best path through an intersection of arbitrarily many weighted automata, without actually performing the intersection. The algorithm is based on dual decomposition: the automata attempt to agree on a string by communicating about features of the string. We demonstrate the algorithm on the Steiner consensus string problem, both on synthetic data and on consensus decoding for speech recognition. This involves implicitly intersecting up to 100 automata.

1 Introduction

Many tasks in natural language processing involve functions that assign scores—such as log-probabilities—to candidate strings or sequences. Often such a function can be represented compactly as a weighted finite state automaton (WFSA). Finding the best-scoring string according to a WFSA is straightforward using standard best-path algorithms.

It is common to construct a scoring WFSA by combining two or more simpler WFSAs, taking advantage of the closure properties of WFSAs. For example, consider noisy channel approaches to speech recognition (Pereira and Riley, 1997) or machine translation (Knight and Al-Onaizan, 1998). Given an input f, the score of a possible English transcription or translation e is the sum of its language model score $\log p(e)$ and its channel model score $\log p(f \mid e)$. If each of these functions of e is represented as a WFSA, then their sum is represented as the intersection of those two WFSAs.

WFSA intersection corresponds to constraint conjunction, and hence is often a mathematically natural way to specify a solution to a problem involving multiple soft constraints on a desired string. Unfortunately, the intersection may be computationally inefficient in practice. The intersection of K WFSAs having $n_1, n_2, \ldots, n_K$ states may have $n_1 \cdot n_2 \cdots n_K$ states in the worst case.[1]

In this paper, we propose a more efficient method for finding the best path in an intersection without actually computing the full intersection. Our approach is based on *dual decomposition*, a combinatorial optimization technique that was recently introduced to the vision (Komodakis et al., 2007) and language processing communities (Rush et al., 2010; Koo et al., 2010). Our idea is to interrogate the several WFSAs separately, repeatedly visiting each WFSA to seek a high-scoring path in each WFSA that agrees with the current paths found in the other WSFAs. This iterative negotiation is reminiscent of message-passing algorithms (Sontag et al., 2008), while the queries to the WFSAs are reminiscent of loss-augmented inference (Taskar et al., 2005).

We remark that a general solution whose asymptotic worst-case runtime beat that of naive intersection would have important implications for complexity theory (Karakostas et al., 2003). Our approach is not such a solution. We have no worst-case bounds on how long dual decomposition will take to converge in our setting, and indeed it can fail to converge altogether.[2] However, when it does converge, we have a "certificate" that the solution is optimal.

Dual decomposition is usually regarded as a method for finding an optimal *vector* in $\mathbb{R}^d$, subject to several constraints. However, it is not obvious how best to represent strings as vectors—they

[*]The authors are grateful to Damianos Karakos for providing tools and data for the ASR experiments. This work was supported in part by an NSF Graduate Research Fellowship.

[1]Most regular expression operators combine WFSA sizes additively. It is primarily intersection and its close relative, composition, that do so multiplicatively, leading to inefficiency when two large WFSAs are combined, and to exponential blowup when many WFSAs are combined. Yet these operations are crucially important in practice.

[2]An example that oscillates can be constructed along lines similar to the one given by Rush et al. (2010).

have unbounded length, and furthermore the absolute position of a symbol is not usually significant in evaluating its contribution to the score.[3] One contribution of this work is that we propose a general, flexible scheme for converting strings to feature vectors on which the WFSAs must agree. In principle the number of features may be infinite, but the set of "active" features is expanded only as needed until the algorithm converges. Our experiments use a particular instantiation of our general scheme, based on n-gram features.

We apply our method to a particular task: finding the *Steiner consensus string* (Gusfield, 1997) that has low total edit distance to a number of given, unaligned strings. As an illustration, we are pleased to report that "`alia`" and "`aian`" are the consensus popular names for girls and boys born in the U.S. in 2010. We use this technique for consensus decoding from speech recognition lattices, and to reconstruct the common source of up to 100 strings corrupted by random noise. Explicit intersection would be *astronomically expensive* in these cases. We demonstrate that our approach tends to converge rather quickly, and that it finds good solutions quickly in any case.

2 Preliminaries

2.1 Weighted Finite State Automata

A weighted finite state automaton (WFSA) over the finite alphabet Σ is an FSA that has a cost or weight associated with each arc. We consider the case of real-valued weights in the tropical semiring. This is a fancy way of saying that the weight of a path is the sum of its arc weights, and that the weight of a string is the minimum weight of all its accepting paths (or ∞ if there are none).

When we intersect two WFSAs F and G, the effect is to add string weights: $(F \cap G)(x) = F(x) + G(x)$. Our problem is to find the x that *minimizes* this sum, but without constructing $F \cap G$ to run a shortest-path algorithm on it.

2.2 Dual Decomposition

The trick in dual decomposition is to decompose an intractable *global* problem into two or more tractable *subproblems* that can be solved independently. If we can somehow combine the solutions from the subproblems into a "valid" solution to the global problem, then we can avoid optimizing the joint problem directly. A valid solution is one in which the individual solutions of each subproblem all agree on the variables which are shared in the joint problem. For example, if we are combining a parser with a part-of-speech tagger, the tag assignments from both models must agree in the final solution (Rush et al., 2010); if we are intersecting a translation model with a language model, then it is the words that must agree (Rush and Collins, 2011).

More formally, suppose we want to find a global solution that is jointly optimized among K subproblems: $\operatorname{argmin}_x \sum_{k=1}^{K} f_k(x)$. Suppose that x ranges over vectors. Introducing an auxiliary variable x_k for each subproblem f_k allows us to equivalently formulate this as the following constrained optimization problem:

$$\operatorname*{argmin}_{\{x, x_1, \ldots, x_K\}} \sum_{k=1}^{K} f_k(x_k) \qquad \text{s.t. } (\forall k)\, x_k = x \tag{1}$$

For *any* set of vectors λ_k that sum to 0, $\sum_{k=1}^{K} \lambda_k = 0$, Komodakis et al. (2007) show that the following Lagrangian dual is a lower bound on (1):[4]

$$\min_{\{x_1, \ldots, x_K\}} \sum_{k=1}^{K} f_k(x_k) + \lambda_k \cdot x_k \tag{2}$$

where the Lagrange multiplier vectors λ_k can be used to penalize solutions that do not satisfy the agreement constraints $(\forall k)\, x_k = x$. Our goal is to maximize this lower bound and hope that the result does satisfy the constraints. The graphs in Fig. 2 illustrate how we increase the lower bound over time, using a subgradient algorithm to adjust the λ's. At each subgradient step, (2) can be computed by choosing each $x_k = \operatorname{argmin}_{x_k} f_k(x_k) + \lambda_k \cdot x_k$ separately. In effect, each subproblem makes an independent prediction x_k influenced by λ_k, and if these outputs do not yet satisfy the agreement constraints, then the λ_k are adjusted to encourage the subproblems to agree on the next iteration. See Sontag et al. (2011) for a detailed tutorial on dual decomposition.

[3]Such difficulties are typical when trying to apply structured prediction or optimization techniques to predict linguistic objects such as strings or trees, rather than vectors.

[4]The objective in (2) can always be made as small as in (1) by choosing the vectors $(x_1, \ldots x_K)$ that minimize (1) (because then $\sum_k \lambda_k \cdot x_k = \sum_k \lambda_k \cdot x = 0 \cdot x = 0$). Hence (2) $\leq$ (1).

3 WFSAs and Dual Decomposition

Given K WFSAs, $F_1, \ldots, F_K$, we are interested in finding the string x which has the best score in the intersection $F_1 \cap \ldots \cap F_K$. The lowest-cost string in the intersection of all K machines is defined as:

$$\operatorname*{argmin}_x \sum_k F_k(x) \tag{3}$$

As explained above, the trick in dual decomposition is to recast (3) as independent problems of the form $\operatorname{argmin}_{x_k} F_k(x_k)$, subject to constraints that all x_k are the same. However, it is not so clear how to define agreement constraints on strings. Perhaps a natural formulation is that F_k should be urged to favor strings x_k that would be read by $F_{k'}$ along a similar path to that of $x_{k'}$. But F_k cannot keep track of the state of $F_{k'}$ for all k' without solving the full intersection—precisely what we are trying to avoid.

Instead of requiring the strings x_k to be equal as in (1), we will require their *features* to be equal:

$$(\forall k)\, \gamma(x_k) = \gamma(x) \tag{4}$$

Of course, we must define the features. We will use an infinite feature vector $\gamma(x)$ that completely characterizes x, so that agreement of the feature vectors implies agreement of the strings. At each subgradient step, however, we will only allow finitely many elements of λ_k to become nonzero, so only a finite portion of $\gamma(x_k)$ needs to be computed.[5]

We will define these "active" features of a string x by constructing some unweighted deterministic FSA, G (described in §4). The active features of x are determined by the collection of arcs on the accepting path of x in G. Thus, to satisfy the agreement constraint, x_i and x_j must be accepted using the same arcs of G (or more generally, arcs that have the same features).

We relax the constraints by introducing a collection $\lambda = \lambda_1, \ldots, \lambda_K$ of Lagrange multipliers, and defining $G_{\lambda_k}(x)$ such that the features of G are weighted by the vector λ_k (*all* of whose nonzero elements must correspond to features in G). As in (2), we assume $\lambda \in \Lambda$, where $\Lambda = \{\lambda : \sum_k \lambda_k = 0\}$. This gives the objective:

$$h(\lambda) = \min_{\{x_1,\ldots,x_K\}} \sum_k \left(F_k(x_k) + G_{\lambda_k}(x_k)\right) \tag{5}$$

This minimization fully decomposes into K subproblems that can be solved independently. The kth subproblem is to find $\operatorname{argmin}_{x_k} F_k(x_k) + G_{\lambda_k}(x_k)$, which is straightforward to solve with finite-state methods. It is the string on the lowest-cost path through $H_k = F_k \cap G_{\lambda_k}$, as found with standard path algorithms (Mohri, 2002).

The dual problem we wish to solve is $\max_{\lambda \in \Lambda} h(\lambda)$, where $h(\lambda)$ itself is a min over $\{x_1, \ldots, x_K\}$. We optimize λ via projected subgradient ascent (Komodakis et al., 2007). The update equation for λ_k at iteration t is then:

$$\lambda_k^{(t+1)} = \lambda_k^{(t)} + \eta_t \left(\gamma(x_k^{(t)}) - \frac{\sum_{k'} \gamma(x_{k'}^{(t)})}{K} \right) \tag{6}$$

where $\eta_t > 0$ is the step size at iteration t. This update is intuitive. It moves away from the current solution and toward the average solution (where they differ), by increasing the cost of the former's features and reducing the cost of the latter's features.

This update may be very dense, however, since $\gamma(x)$ is an infinite vector. So we usually only update the elements of λ_k that correspond to the small finite set of *active* features (the other elements are still "frozen" at 0), denoted Θ. This is still a valid subgradient step. This strategy is incorrect only if the updates for all active features are 0—in other words, only if we have achieved equality of the currently active features and yet still the $\{x_k\}$ do not agree. In that case, we must choose some inactive features that are still unequal and allow the subgradient step to update their λ coefficients to nonzero, making them active. At the next step of optimization, we must expand G to consider this enlarged set of active features.

4 The Agreement Machine

The agreement machine (or constraint machine) G can be thought of as a way of encoding features of

[5] The simplest scheme would define a binary feature for each string in Σ^*. Then the nonzero elements of λ_k would specify punishments and rewards for outputting various strings that had been encountered at earlier iterations: "Try subproblem k again, and try harder not to output `michael` this time, as it still didn't agree with other subproblems: try `jason` instead." This scheme would converge glacially if at all. We instead focus on featurizations that let subproblems negotiate about *substrings*: "Try again, avoiding `mi` if possible and favoring `ja` instead."

strings on which we enforce agreement. There are a number of different topologies for G that might be considered, with varying degrees of efficiency and utility. Constructing G essentially amounts to feature engineering; as such, it is unlikely that there is a universally optimal topology of G. Nevertheless, there are clearly *bad* ways to build G, as not all topologies are guaranteed to lead to an optimal solution. In this section, we lay out some abstract guidelines for appropriate G construction, before we describe specific topologies in the later subsections.

Most importantly, we should design G so that it accepts all strings in $F_1 \cap \ldots \cap F_K$. This is to ensure that it accepts the string that is the optimal solution to the joint problem. If G did not accept that string, then neither would $H_k = F_k \cap G$, and our algorithm would not be able to find it.

Even if H_k can accept the optimal string, it is possible that this string would never be the best path in this machine, regardless of λ. For example, suppose G is a single-state machine with self-loops accepting each symbol in the alphabet (i.e. a unigram machine). Suppose H_k outputs the string aaa in the current iteration, but we would like the machines to converge to aaaaa. We would lower the weight of $\lambda_{\mathtt{a}}$ to encourage H_k to output more of the symbol a. However, if H_k has a cyclic topology, then it could happen that a negative value of $\lambda_{\mathtt{a}}$ could create a negative-weight cycle, in which the lowest-cost path through H_k is infinitely long. It might be that adjusting $\lambda_{\mathtt{a}}$ can change the best string to either aaa or aaaaaaaaa... (depending on whether a cycle after the initial aaa has positive or negative weight), but never the optimal aaaaa. On the other hand, if G instead encoded 5-grams, this would not be a problem because a path through a 5-gram machine could accept aaaaa without traversing a cycle.

Finally, agreeing on (active) features does not necessarily mean that all x_k are the same string. For example, if we again use a unigram G (that is, $\Theta = \Sigma$, the set of unigrams), then $\gamma_\Theta(\mathtt{abc}) = \gamma_\Theta(\mathtt{cba})$, where γ_Θ returns a feature vector where all but the active features are zeroed out. In this instance, we satisfy the constraints imposed by G, even though we have not satisfied the constraint we truly care about: that the strings agree.

To summarize, we will aim to choose Θ such that G has the following characteristics:

1. The language $\mathcal{L}(F_k \cap G) = \mathcal{L}(F_k)$; i.e. G does not restrict the set of strings accepted by F_k.
2. When $\gamma_\Theta(x_i) = \gamma_\Theta(x_j)$, typically $x_i = x_j$.
3. $\exists \lambda \in \Lambda$ s.t. $\operatorname{argmin}_x F_k(x) + G_{\lambda_k}(x) = \operatorname{argmin}_x \sum_{k'} F_{k'}(x)$, i.e., the optimal string can be the best path in $F_k \cap G$.[6] This may not be the case if G is cyclic.

The first of these is required during every iteration of the algorithm in order to maintain optimality guarantees. However, even if we do not satisfy the latter two points, we may get lucky and the strings themselves will agree upon convergence, and no further work is required. Furthermore, the unigram machine G used in the above examples, despite breaking these requirements, has the advantage of being very efficient to intersect with F. This motivates our "active feature" strategy of using a simple G initially, and incrementally altering it as needed, for example if we satisfy the constraints but the strings do not yet match. We discuss this in §4.2.

4.1 N-Gram Construction of G

In principle, it is valid to use any G that satisfies the guidelines above, but in practice, some topologies will lead to faster convergence than others.

Perhaps the most obvious form is a simple vector encoding of strings, e.g. "a at position 1", "b at position 2", and so on. As a WFSA, this would simply have one state represent each position, with arcs for each symbol going from position i to $i+1$. This is essentially a unigram machine where the loops have been "unrolled" to also keep track of position.

However, early experiments showed that with this topology for G, our algorithm converged very slowly, if at all. What goes wrong? The problem stems from the fact that the strings are unaligned and of varying length, and it is difficult to get the strings to agree quickly at specific positions. For example, if two subproblems have b at positions 6 and 8 in the current iteration, they might agree at position 7—but our features don't encourage this. The Lagrangian update would discourage accepting b at 6 and encourage b at 8 (and vice versa), without giving credit

[6]It is not always possible to construct a G to satisfy this property, as the Lagrangian dual may not be a tight bound to the original problem.

for meeting in the middle. Further, these features do not encourage the subproblems to preserve the relative order of neighboring symbols, and strings which are almost the same but slightly misaligned will be penalized essentially everywhere. This is an ineffective way for the subproblems to communicate.

In this paper, we focus on the feature set we found to work the best in our experiments: the strings should agree on their n-gram features, such as "number of occurrences of the bigram ab." Even if we don't yet know precisely where ab should appear in the string, we can still move toward convergence if we try to force the subproblems to agree on whether and how often ab appears at all.

To encode n-gram features in a WFSA, each state represents the $(n-1)$-gram history, and all arcs leaving the state represent the final symbol in the n-gram, weighted by the score of that n-gram. The machine will also contain start and end states, with appropriate transitions to/from the n-gram states. For example, if the trigram abc has weight λ_{abc}, then the trigram machine will encode this as an arc with the symbol c leaving the state representing ab, and this arc will have weight λ_{abc}. If our feature set also contains 1- and 2-grams, then the arc in this example would incorporate the weights of all of the corresponding features: $\lambda_{\mathsf{abc}} + \lambda_{\mathsf{bc}} + \lambda_{\mathsf{c}}$.

A drawback is that these features give no information about *where* in the string the n-grams should occur. In a long string, we might want to encourage or discourage an n-gram in a certain "region" of the string. Our features can only encourage or discourage it *everywhere* in the string, which may lead to slow convergence. Nevertheless, in our particular experimental settings, we find that this works better than other topologies we have considered.

Sparse N-Gram Encoding A full n-gram language model requires $\approx |\Sigma|^n$ arcs to encode as a WFSA. This could be quite expensive. Fortunately, large n-gram models can be compacted by using failure arcs (ϕ-arcs) to encode *backoff* (Allauzen et al., 2003). These arcs act as ϵ-transitions that can be taken only when no other transition is available. They allow us to encode the sparse subset of n-grams that have nonzero Lagrangians. We encode G such that all features whose λ value is 0 will back off to the next largest n-gram having nonzero weight. This form of G still accepts Σ^* and has the same weights as a dense representation, but could require substantially fewer states.

4.2 Incrementally Expanding G

As mentioned above, we may need to alter G as we go along. Intuitively, we may want to start with features that are cheap to encode, to move the parameters λ to a good part of the solution space, then incrementally bring in more expensive features as needed. Shorter n-grams require a smaller G and will require a shorter runtime per iteration, but if they are too short to be informative, then they may require many more iterations to reach convergence. In an extreme case, we may reach a point where the subproblems all agree on n-grams currently in Θ, but the actual strings still do not match. Waiting until we hit such a point may be unnecessarily slow. We experimented with periodically increasing n (e.g. adding trigrams to the feature set if we haven't converged with bigrams after a fixed number of iterations), but this is expensive, and it is not clear how to define a schedule for increasing the order of n. We instead present a simple and effective heuristic for bringing in more features.

The idea is that if the subproblem solutions currently disagree on counts of the bigrams ab and bc, then an abc feature may be unnecessary, since the subproblems could still make progress with only these bigram constraints. However, once the subproblems agree on these two bigrams, but disagree on trigram abc, we bring this into the feature set Θ. More generally, we add an $(n+1)$-gram to the feature set if the current strings disagree on its counts despite agreeing on its n-gram prefix and n-gram suffix (which need not necessarily be Θ). This *selectively* brings in larger n-grams to target portions of the strings that may require longer context, while keeping the agreement machine small.

Algorithm 1 gives pseudocode for our complete algorithm when using n-gram features with this incremental strategy. To summarize, we solve for each x_k using the current λ_k, and if all the strings agree, we return them as the optimal solution. Otherwise, we update λ_k and repeat. At each iteration, we check for n-gram agreement, and bring in select $(n+1)$-grams to the feature set as appropriate.

Finally, there is another instance where we might

Algorithm 1 The dual decomposition algorithm with n-gram features.

```
Initialize $\Theta$ to some initial set of $n$-gram features.
for $t = 1$ to $T$ do
  for $k = 1$ to $K$ do
    Solve $x_k = \operatorname{argmin}_x (F_k \cap G_{\lambda_k})(x)$ with a shortest-path algorithm
  end for
  if $(\forall i, j) x_i = x_j$ then
    return $\{x_1, \ldots, x_K\}$
  else
    $\Theta = \Theta \cup \{z \in \Sigma^* :$ all $x_k$ agree on the features corresponding to the length-$(|z| - 1)$ prefix and suffix of $z$, but not on $z$ itself$\}$
    for $k = 1$ to $K$ do
      Update $\lambda_k$ according to equation (6)
      Create $G_{\lambda_k}$ to encode the features $\Theta$
    end for
  end if
end for
```

need to expand G, which we omit from the pseudocode for conciseness. If both F_k and G are cyclic, then there is a chance that there will be a negative-weight cycle in $F_k \cap G_{\lambda_k}$. (If at least one of these machines is acyclic, then this is not a problem, because their intersection yields a finite set.) In the case of a negative-weight cycle, the best path is infinitely long, and so the algorithm will either return an error or fail to terminate. If this happens, then we need to backtrack, and either decrease the subgradient step size to avoid moving into this territory, or alter G to expand the cycles. This can be done by unrolling loops to keep track of more information—when encoding n-gram features with G, this amounts to expanding G to encode higher order n-grams. When using a sparse G with ϕ-arcs, it may also be necessary to increase the minimum n-gram history that is used for back-off. For example, instead of allowing bigrams to back off to unigrams, we might force G to encode the full set of bigrams (not just bigrams with nonzero λ) in order to avoid cycles in the lower order states. Our strategy for avoiding negative-weight cycles is detailed in §5.1.

5 Experiments with Consensus Decoding

To best highlight the utility of our approach, we consider applications that must (implicitly) intersect a large number of WFSAs. We will demonstrate that, in many cases, our algorithm converges to an exact solution on problems involving 10, 25, and even 100 machines, all of which would be hopeless to solve by taking the full intersection.

We focus on the problem of solving for the Steiner consensus string: given a set of K strings, find the string in Σ^* that has minimal total edit distance to all strings in the set. This is an NP-hard problem that can be solved as an intersection of K machines, as we will describe in §5.2. The consensus string also gives an implicit multiple sequence alignment, as we discuss in §6.

We begin with the application of minimum Bayes risk decoding of speech lattices, which we show can reduce to the consensus string problem. We then explore the consensus problem in depth by applying it to a variety of different inputs.

5.1 Experimental Details

We initialize Θ to include both unigrams and bigrams, as we find that unigrams alone are not productive features in these experiments. As we expand Θ, we allow it to include n-grams up to length five.

We run our algorithm for a maximum of 1000 iterations, using a subgradient step size of $\alpha/(t + 500)$ at iteration t, which satisfies the general properties to guarantee asymptotic convergence (Spall, 2003). We initialize α to 1 and 10 in the two subsections, respectively. We halve α whenever we hit a negative-weight cycle and need to backtrack. If we still get negative-weight cycles after $\alpha \leq 10^{-4}$ then we reset α and increase the minimum order of n which is encoded in G. (If n is already at our maximum of five, then we simply end without converging.) In the case of non-convergence after 1000 iterations, we select the best string (according to the objective) from the set of strings that were solutions to any subproblem at any point during optimization.

Our implementation uses OpenFST 1.2.8 (Allauzen et al., 2007).

5.2 Minimum Bayes Risk Decoding for ASR

We first consider the task of automatic speech recognition (ASR). Suppose x^* is the true transcription (a string) of an spoken utterance, and $\pi(w)$ is an ASR system's probability distribution over possible transcriptions w. The *Bayes risk* of an output transcription x is defined as the expectation

$\sum_w \pi(w)\,\ell(x,w)$ for some loss function ℓ (Bickel and Doksum, 2006). Minimum Bayes risk decoding (Goel and Byrne, 2003) involves choosing the x that minimizes the Bayes risk, rather than simply choosing the x that maximizes $\pi(x)$ as in MAP decoding.

As a reasonable approximation, we will take the expectation over just the strings $w_1, \ldots, w_K$ that are most probable under π. A common loss function is the Levenshtein distance because this is generally used to measure the word error rate of ASR output. Thus, we seek a consensus transcription

$$\operatorname*{argmin}_x \sum_{k=1}^{K} \pi_k\, d(x, w_k) \tag{7}$$

that minimizes a weighted sum of edit distances to all of the top-K strings, where high edit distance to more probable strings is more strongly penalized. Here $d(x,w)$ is the unweighted Levenshtein distance between two strings, and $\pi_k = \pi(w_k)$. If each $\pi_k = 1/K$, then argmin_x is known as the *Steiner consensus string*, which is NP-hard to find (Sim and Park, 2003). Equation (7) is a weighted generalization of the Steiner problem.

Given an input string w_k, it is straightforward to define our WFSA F_k such that $F_k(x)$ computes $\pi_k\, d(x, w_k)$. A direct construction of F_k is as follows. First, create a "straight line" WFSA whose single path accepts (only) w_k; each each state corresponds to a position in w_k. These arcs all have cost 0. Now add various arcs with cost π_k that permit edit operations. For each arc labeled with a symbol $a \in \Sigma$, add competing "substitution" arcs labeled with the other symbols in Σ, and a competing "deletion" arc labeled with ϵ; these have the same source and target as the original arc. Also, at each state, add a self-loop labeled with each symbol in Σ; these are "insertion" arcs. Each arc that deviates from w_k has a cost of π_k, and thus the lowest-cost path through F_k accepting x has weight $\pi_k\, d(x, w_k)$.

The consensus objective in Equation (7) can be solved by finding the lowest-cost path in $F_1 \cap \ldots \cap F_K$, and we can solve this best-path problem using the dual decomposition algorithm described above.

5.2.1 Experiments

We ran our algorithm on Broadcast News data, using 226 lattices produced by the IBM Attila decoder

```
  0  <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> WE WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I DON'T WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> WELL I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> THEY WANT TO BE TAKING A DEEP BREATH NOW </s>
300  <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> WE WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I DON'T WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> WELL I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> WELL WANT TO BE TAKING A DEEP BREATH NOW </s>
375  <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I DON'T WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
472  <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
     <s> I WANT TO BE TAKING A DEEP BREATH NOW </s>
```

Figure 1: Example run of the consensus problem on $K = 25$ strings on a Broadcast News utterance, showing $x_1, \ldots, x_5$ at the 0th, 300th, 375th, and 472nd iterations.

(Chen et al., 2006; Soltau et al., 2010) on a subset of the NIST dev04f data, using models trained by Zweig et al. (2011). For each lattice, we found the consensus of the top $K = 25$ strings.

85% of the problems converged within 1000 iterations, with an average of 147.4 iterations. We found that the true consensus was often the most likely string under π, but not always—this was true 70% of the time. In the Bayes risk objective we are optimizing in equation (7)—the expected loss—our approach averaged a score of 1.59, while always taking the top string gives only a slightly worse average of 1.66. 8% of the problems encountered negative-weight cycles, which were all resolved either by decreasing the step size or encoding larger n-grams.

5.3 Investigating Consensus Performance with Synthetic Data

The above experiments demonstrate that we can exactly find the best path in the intersection of 25 machines—an intersection that could not feasibly be constructed in practice. However, these experiments do not exhaustively explore how dual decomposition behaves on the Steiner string problem in general.

Above, we experimented with only a fixed number of input strings, which were generally similar to one another. There are a variety of other inputs to the consensus problem which might lead to different behavior and convergence results, however. If we were to instead run this experiment on DNA sequences (for example, if we posit that the strings are all mutations of the same ancestor), the alphabet {A,T,C,G}

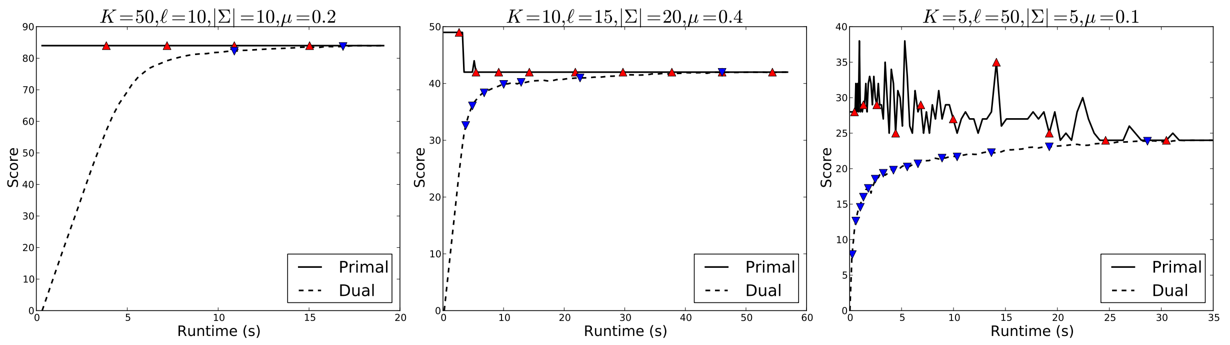

Figure 2: The algorithm's behavior on three specific consensus problems. The curves show the current values of the primal bound (based on the best string at the current iteration) and dual bound $h(\lambda)$. The horizontal axis shows runtime. Red upper triangles are placed every 10 iterations, while blue lower triangles are placed for every 10% increase in the size of the feature set Θ.

K	ℓ	$\|\Sigma\|$	μ	**Conv.**	**Iters.**	**Red.**
5	100	5	0.1	68%	257 (±110)	24%
5	100	5	0.2	0%	–	8%
5	50	5	0.1	80%	123 (± 65)	20%
5	50	5	0.2	10%	436 (±195)	18%
10	50	5	0.1	69%	228 (±164)	18%
10	50	5	0.2	0%	–	8%
10	50	5	0.4	0%	–	3%
10	30	10	0.1	100%	50 (± 69)	13%
10	30	10	0.2	93%	146 (±142)	20%
10	30	10	0.4	0%	–	16%
10	15	20	0.1	100%	26 (± 6)	1%
10	15	20	0.2	98%	43 (± 18)	10%
10	15	20	0.4	63%	289 (±217)	18%
10	15	20	0.8	0%	–	11%
25	15	20	0.1	98%	30 (± 5)	0%
25	15	20	0.2	92%	69 (±112)	6%
25	15	20	0.4	55%	257 (±149)	16%
25	15	20	0.8	0%	–	12%
50	10	10	0.2	68%	84 (±141)	0%
50	10	10	0.4	21%	173 (± 94)	9%
100	10	10	0.2	44%	147 (±220)	0%
100	10	10	0.4	13%	201 (±138)	6%

Table 1: A summary of results for various consensus problems, as described in §5.3.

is so small that n-grams are likely to be repeated in many parts of the strings, and the lack of position information in our features could make it hard to reach agreement. Another interesting case is when the input strings have little or nothing in common—can we still converge to an optimal consensus in a reasonable number of iterations?

We can investigate many different cases by creating synthetic data, where we tune the number of input strings K, the length of the strings, the size of the vocabulary $|\Sigma|$, as well as how similar the strings are. We do this by randomly generating a base string $x^* \in \Sigma^\ell$ of length ℓ. We then generate K random strings $w_1, \ldots, w_K$, each by passing x^* through a noisy edit channel, where each position has independent probability μ of making an edit. For each position in x^*, we uniformly sample once among the three types of edits (substitution, insertion, deletion), and in the case of the first two, we uniformly sample from the vocabulary (excluding the current symbol for substitution). The larger μ, the more mutated the strings will be. For small μ or large K, the optimal consensus of $w_1, \ldots, w_K$ will usually be x^*.

Table 1 shows results under various settings. Each line presents the percentage of 100 examples that converge within the iteration limit, the average number of iterations to convergence (± standard deviation) for those that converged, and the reduction in the objective value that is obtained over a simple baseline of choosing the best string in the input set, to show how much progress the algorithm makes between the 0th and final iteration.

As expected, a higher mutation probability slows convergence in all cases, as does having longer input strings. These results also confirm our hypothesis that a small alphabet would lead to slow convergence when using small n-gram features. For these types of strings, which might show up in biological data, one would likely need more informative constraints than position-agnostic n-grams.

Figure 2 shows example runs on problems generated at three different parameter settings. We plot the objective value as a function of runtime, showing both the primal objective (3) that we hope to minimize, which we measure as the quality of the best solution among the $\{x_k\}$ that are output at the current iteration, and the dual objective (5) that our algorithm is maximizing. The dual problem (which is concave in λ) lower bounds the primal. If the two functions ever touch, we know the solution to the dual problem is in the set of feasible solutions to the original primal problem we are attempting to solve, and indeed must be optimal. The figure shows that the dual function always has an initial value of 0, since we initialize each $\lambda_k = 0$, and then F_k will simply return the input w_k as its best solution (since w_k has zero distance to itself). As the algorithm begins to enforce the agreement constraints, the value of the relaxed dual problem gradually worsens, until it fully satisfies the constraints.

These plots indicate the number of iterations that have passed and the number of active features. We see that the time per iteration increases as the number of features increases, as expected, because more (and longer) n-grams are being encoded by G.

The three patterns shown are typical of almost all the trials we examined. When the solution is in the original input set (a likely occurrence for large K or small $\mu \cdot \ell$), the primal value will be optimal from the start, and our algorithm only has to *prove* its optimality. For more challenging problems, the primal solution may jump around in quality at each iteration before settling into a stable part of the space.

To investigate how different n-gram sizes affect convergence rates, we experiment with using the entire set of n-grams (for a fixed n) for the duration of the optimization procedure. Figure 3 shows convergence rates (based on both iterations and runtime) of different values of n for one set of parameters. While bigrams are very fast (average runtime of 14s among those that converged), this converged within 1000 iterations only 78% of the time, and the remaining 22% end up bringing down the average speed (with an overall average runtime over a minute). All larger n-grams converged every time; trigrams had an average runtime of 32s. Our algorithm, which begins with bigrams but brings in more features (up to 5-grams) as needed, had an average runtime of 19s (with 98% convergence).

6 Discussion and Future Work

An important (and motivating) property of Lagrangian relaxation methods is the certificate of optimality. Even in instances where approximate algorithms perform well, it could be useful to have a true optimality guarantee. For example, our algorithm can be used to produce reference solutions, which are important to have for research purposes.

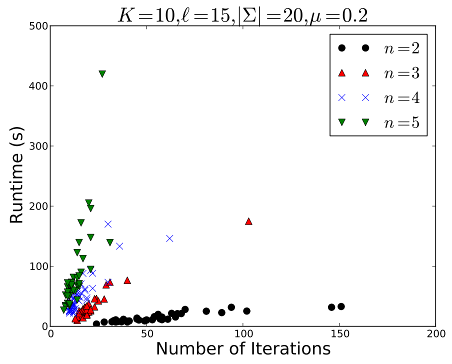

Figure 3: Convergence rates for a fixed set of n-grams.

Under a sum-of-pairs Levenshtein objective, the exact multi-sequence alignment can be directly obtained from the Steiner consensus string and vice versa (Gusfield, 1997). This implies that our exact algorithm could be also used to find exact multi-sequence alignments, an important problem in natural language processing (Barzilay and Lee, 2003) and computational biology (Durbin et al., 2006) that is almost always solved with approximate methods.

We have noted that some constraints are more useful than others. Position-specific information is hard to agree on and leads to slow convergence, while pure n-gram constraints do not work as well for long strings where the position may be important. One avenue we are investigating is the use of a non-deterministic G, which would allow us to encode latent variables (Dreyer et al., 2008), such as loosely defined "regions" within a string, and to allow for the encoding of alignments between the input strings. We would also like to extend these methods to other combinatorial optimization problems involving strings, such as inference in graphical models over strings (Dreyer and Eisner, 2009).

To conclude, we have presented a general framework for applying dual decomposition to implicit WFSA intersection. This could be applied to a number of NLP problems such as language model and lattice intersection. To demonstrate its utility on a large number of automata, we applied it to consensus decoding, determining the true optimum in a reasonable amount of time on a large majority of cases.

References

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing statistical language models. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 40–47.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Proceedings of the 12th International Conference on Implementation and Application of Automata*, CIAA'07, pages 11–23.

Regina Barzilay and Lillian Lee. 2003. Learning to paraphrase: An unsupervised approach using multiple-sequence alignment. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 16–23.

Peter J. Bickel and Kjell A. Doksum. 2006. *Mathematical Statistics: Basic Ideas and Selected Topics*, volume 1. Pearson Prentice Hall.

Stanley F. Chen, Brian Kingsbury, Lidia Mangu, Daniel Povey, George Saon, Hagen Soltau, and Geoffrey Zweig. 2006. Advances in speech transcription at IBM under the DARPA EARS program. *IEEE Transactions on Audio, Speech & Language Processing*, 14(5):1596–1608.

Markus Dreyer and Jason Eisner. 2009. Graphical models over multiple strings. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, EMNLP '09, pages 101–110. Association for Computational Linguistics.

Markus Dreyer, Jason R. Smith, and Jason Eisner. 2008. Latent-variable modeling of string transductions with finite-state methods. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1080–1089, Honolulu, October.

R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. 2006. *Biological Sequence Analysis*. Cambridge University Press.

Vaibhava Goel and William J. Byrne. 2003. Minimum Bayes risk methods in automatic speech recognition. In Wu Chou and Biing-Hwang Juan, editors, *Pattern Recognition in Speech and Language Processing*. CRC Press.

Dan Gusfield. 1997. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge University Press.

George Karakostas, Richard J Lipton, and Anastasios Viglas. 2003. On the complexity of intersecting finite state automata and NL versus NP. *Theoretical Computer Science*, pages 257–274.

Kevin Knight and Yaser Al-Onaizan. 1998. Translation with finite-state devices. In *AMTA'98*, pages 421–437.

N. Komodakis, N. Paragios, and G. Tziritas. 2007. MRF optimization via dual decomposition: Message-Passing revisited. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8.

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 the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1288–1298.

Mehryar Mohri. 2002. Semiring frameworks and algorithms for shortest-distance problems. *J. Autom. Lang. Comb.*, 7:321–350, January.

Fernando C. N. Pereira and Michael Riley. 1997. Speech recognition by composition of weighted finite automata. *CoRR*.

Alexander M. Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through Lagrangian relaxation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 72–82.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, EMNLP '10, pages 1–11.

Jeong Seop Sim and Kunsoo Park. 2003. The consensus string problem for a metric is NP-complete. *J. of Discrete Algorithms*, 1:111–117, February.

H. Soltau, G. Saon, and B. Kingsbury. 2010. The IBM Attila speech recognition toolkit. In *Proc. IEEE Workshop on Spoken Language Technology*, pages 97–102.

David Sontag, Talya Meltzer, Amir Globerson, Yair Weiss, and Tommi Jaakkola. 2008. Tightening LP relaxations for MAP using message-passing. In *24th Conference in Uncertainty in Artificial Intelligence*, pages 503–510. AUAI Press.

David Sontag, Amir Globerson, and Tommi Jaakkola. 2011. Introduction to dual decomposition for inference. In Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright, editors, *Optimization for Machine Learning*. MIT Press.

James C. Spall. 2003. *Introduction to Stochastic Search and Optimization*. John Wiley & Sons, Inc., New York, NY, USA, 1 edition.

Ben Taskar, Vassil Chatalbashev, Daphne Koller, and Carlos Guestrin. 2005. Learning structured prediction models: A large margin approach. In *Proceedings of* *the 22nd international conference on Machine learning*, ICML '05, pages 896–903.

Geoffrey Zweig, Patrick Nguyen, Dirk Van Compernolle, Kris Demuynck, Les E. Atlas, Pascal Clark, Gregory Sell, Meihong Wang, Fei Sha, Hynek Hermansky, Damianos Karakos, Aren Jansen, Samuel Thomas, Sivaram G. S. V. S., Sam Bowman, and Justine T. Kao. 2011. Speech recognition with segmental conditional random fields: A summary of the JHU CLSP 2010 Summer Workshop. In *ICASSP*.