A An Example Run of the Algorithm in Figure 3

Figure 1 gives an example run of the algorithm. After 31 iterations the algorithm detects that the dual is no longer decreasing rapidly enough, and runs for $K = 10$ additional iterations, tracking which constraints are violated. Constraints $y(6) = 1$ and $y(10) = 1$ are each violated 10 times, while other constraints are not violated. A recursive call to the algorithm is made with $C = \{6, 10\}$, and the algorithm converges in a single iteration, to a solution that is guaranteed to be optimal.

B Speeding up the DP: A* Search

In the algorithm depicted in Figure 3, each time we call $Optimize(C \cup C', u)$, we expand the number of states in the dynamic program by adding hard constraints. On the graph level, adding hard constraints can be viewed as expanding an original state in $Y'$ to $2^{|C|}$ states in $Y'_C$, since now we keep a bit-string $b_C$ of length $|C|$ in the states to record which words in $C$ have or haven’t been translated. We now show how this observation leads to an A* algorithm that can significantly improve efficiency when decoding with $C \neq \emptyset$.

For any state $s = (w_1, w_2, n, l, m, r, b_C)$ and Lagrange multiplier values $u \in \mathbb{R}^N$, define $\beta_C(s, u)$ to be the maximum score for any path from the state $s$ to the end state, under Lagrange multipliers $u$, in the graph created using constraint set $C$. Define $\pi(s) = (w_1, w_2, n, l, m, r)$, that is, the corresponding state in the graph with no constraints ($C = \emptyset$). Then for any values of $s$ and $u$, we have

$$\beta_C(s, u) \leq \beta_0(\pi(s), u)$$

That is, the maximum score for any path to the end state in the graph with no constraints, forms an upper bound on the value for $\beta_C(s, u)$.

This observation leads directly to an A* algorithm, which is exact in finding the optimum solution, since we can use $\beta_0(\pi(s), u)$ as the admissible estimates for the score from state $s$ to the goal (the end state). The $\beta_0(s', u)$ values for all $s'$ can be calculated by running the Viterbi algorithm using a backwards path. With only $1/2^{|C|}$ states, calculating $\beta_0(s', u)$ is much cheaper than calculating $\beta_C(s, u)$ directly. Guided by $\beta_0(s', u)$, $\beta_C(s, u)$ can be calculated efficiently by using A* search.

Using the A* algorithm leads to significant improvements in efficiency when constraints are added. Section 6 presents comparison of the running time with and without A* algorithm.
which constraints are violated most often. After constraints, we have

\[ y \]

have not been violated during the

\[ K \]

iterations. Thus, hard constraints for word 6 and 10 are added. After adding the constraints, we have \( y^i(i) = 1 \) for \( i = 1 \ldots N \), and the translation is returned, with a guarantee that it is optimal.

Figure 1: An example run of the algorithm in Figure 3. At iteration 32, we start the \( K = 10 \) iterations to count which constraints are violated most often. After \( K \) iterations, the count for 6 and 10 is 10, and all other constraints have not been violated during the \( K \) iterations. Thus, hard constraints for word 6 and 10 are added. After adding the constraints, we have \( y^i(i) = 1 \) for \( i = 1 \ldots N \), and the translation is returned, with a guarantee that it is optimal.