Randomized Decoding for Selection-and-Ordering Problems

Pawan Deshpande, Regina Barzilay and David R. Karger
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{pawand,regina,karger}@csail.mit.edu

Abstract

The task of selecting and ordering information appears in multiple contexts in text generation and summarization. For instance, methods for title generation construct a headline by selecting and ordering words from the input text. In this paper, we investigate decoding methods that simultaneously optimize selection and ordering preferences. We formalize decoding as a task of finding an acyclic path in a directed weighted graph. Since the problem is NP-hard, finding an exact solution is challenging. We describe a novel decoding method based on a randomized color-coding algorithm. We prove bounds on the number of color-coding iterations necessary to guarantee any desired likelihood of finding the correct solution. Our experiments show that the randomized decoder is an appealing alternative to a range of decoding algorithms for selection-and-ordering problems, including beam search and Integer Linear Programming.

1 Introduction

The task of selecting and ordering information appears in multiple contexts in text generation and summarization. For instance, a typical multidocument summarization system creates a summary by selecting a subset of input sentences and ordering them into a coherent text. Selection and ordering at the word level is commonly employed in lexical realization. For instance, in the task of title generation, the headline is constructed by selecting and ordering words from the input text.

Decoding is an essential component of the selection-and-ordering process. Given selection and ordering preferences, the task is to find a sequence of elements that maximally satisfies these preferences. One possible approach for finding such a solution is to decompose it into two tasks: first, select a set of words based on individual selection preferences, and then order the selected units into a well-formed sequence. Although the modularity of this approach is appealing, the decisions made in the selection step cannot be retracted. Therefore, we cannot guarantee that selected units can be ordered in a meaningful way, and we may end up with a suboptimal output.

In this paper, we investigate decoding methods that simultaneously optimize selection and ordering preferences. We formalize decoding as finding a path in a directed weighted graph.[1] The vertices in the graph represent units with associated selection scores, and the edges represent pairwise ordering preferences. The desired solution is the highest-weighted acyclic path of a prespecified length. The requirement for acyclicity is essential because in a typical selection-and-ordering problem, a well-formed output does not include any repeated units. For instance, a summary of multiple documents should not contain any repeated sentences.

[1] We assume that the scoring function is local; that is, it is computed by combining pairwise scores. In fact, the majority of models that are used to guide ordering (i.e., bigrams) are local scoring functions.

Proceedings of NAACL HLT 2007, pages 444–451,
Rochester, NY, April 2007.

Since the problem is NP-hard, finding an exact solution is challenging. We introduce a novel randomized decoding algorithm[2] based on the idea of color-coding (Alon et al., 1995). Although the algorithm is not guaranteed to find the optimal solution on any single run, by increasing the number of runs the algorithm can guarantee an arbitrarily high probability of success. The paper provides a theoretical analysis that establishes the connection between the required number of runs and the likelihood of finding the correct solution.

Next, we show how to find an exact solution using an integer linear programming (ILP) formulation. Although ILP is NP-hard, this method is guaranteed to compute the optimal solution. This allows us to experimentally investigate the trade-off between the accuracy and the efficiency of decoding algorithms considered in the paper.

We evaluate the accuracy of the decoding algorithms on the task of title generation. The decoding algorithms introduced in the paper are compared against beam search, a heuristic search algorithm commonly used for selection-and-ordering and other natural language processing tasks. Our experiments show that the randomized decoder is an appealing alternative to both beam search and ILP when applied to selection-and-ordering problems.

2 Problem Formulation

In this section, we formally define the decoding task for selection-and-ordering problems. First, we introduce our graph representation and show an example of its construction for multidocument summarization. (An additional example of graph construction for title generation is given in Section 6.) Then, we discuss the complexity of this task and its connection to classical NP-hard problems.

2.1 Graph Representation

We represent the set of selection units as the set of vertices V in a weighted directed graph G. The set of edges E represents pairwise ordering scores between all pairs of vertices in V. We also add a special source vertex s and sink vertex t. For each vertex v in V, we add an edge from s to v and an edge from v to t. We then define the set of all vertices as $V^* = V \cup \{s, t\}$, and the set of all edges as $E^* = E \cup \{(s, v) \; \forall \; v \in V\} \cup \{(v, t) \; \forall \; v \in V\}$.

To simplify the representation, we remove all vertex weights in our graph structure and instead shift the weight for each vertex onto its incoming edges. For each pair of distinct vertices $(v, u) \in V$, we set the weight of edge $e_{v,u}$ to be the sum of the logarithms of the selection score for u and the pairwise ordering score of (v, u).

We also enhance our graph representation by grouping sets of vertices into *equivalence classes*. We introduce these classes to control for redundancy as required in many selection-and-ordering problems.[3] For instance, in title generation, an equivalence class may consist of morphological variants of the same stem (i.e., *examine* and *examination*). Because a typical title is unlikely to contain more than one word with the same stem, we can only select a single representative from each class.

Our task is now to find the highest weighted acyclic path starting at s and ending at t with k vertices in between, such that no two vertices belong to the same equivalence class.

2.2 Example: Decoding for Multidocument Summarization

In multidocument summarization, the vertices in the decoding graph represent sentences from input documents. The vertices may be organized into equivalence classes that correspond to clusters of sentences conveying similar information. The edges in the graph represent the combination of the selection and the ordering scores. The selection scores encode the likelihood of a sentence to be extracted, while pairwise ordering scores capture coherence-based precedence likelihood. The goal of the decoder is to find the sequence of k non-redundant sentences that optimize both the selection and the ordering scores. Finding an acyclic path with the highest weight will achieve this goal.

[2] The code is available at `http://people.csail.mit.edu/pawand/rand/`

[3] An alternative approach for redundancy control would be to represent all the members of an equivalence class as a single vertex in the graph. However, such an approach does not allow us to select the best representative from the class. For instance, one element in the equivalence class may have a highly weighted incoming edge, while another may have a highly weighted outgoing edge.

2.3 Relation to Classical Problems

Our path-finding problem may seem to be similar to the tractable shortest paths problem. However, the requirement that the path be long makes it more similar to the the Traveling Salesman Problem (TSP). More precisely, our problem is an instance of the *prize collecting traveling salesman problem*, in which the salesman is required to visit k vertices at best cost (Balas, 1989; Awerbuch et al., 1995).

Since our problem is NP-hard, we might be pessimistic about finding an exact solution. But our problem has an important feature: the length k of the path we want to find is small relative to the number of vertices n. This feature distinguishes our task from other decoding problems, such as decoding in machine translation (Germann et al., 2001), that are modeled using a standard TSP formulation. In general, the connection between n and k opens up a new range of solutions. For example, if we wanted to find the best length-2 path, we could simply try all subsets of 2 vertices in the graph, in all 2 possible orders. This is a set of only $O(n^2)$ possibilities, so we can check all to identify the best in polynomial time.

This approach is very limited, however: in general, its runtime of $O(n^k)$ for paths of length k makes it prohibitive for all but the smallest values of k. We cannot really hope to avoid the exponential dependence on k, because doing so would give us a fast solution to an NP-hard problem, but there is hope of making the dependence "less exponential." This is captured by the definition of *fixed parameter tractability* (Downey and Fellows, 1995). A problem is fixed parameter tractable if we can make the exponential dependence on the parameter k *independent* of the polynomial dependence on the problem size n. This is the case for our problem: as we will describe below, an algorithm of Alon et al. can be used to achieve a running time of roughly $O(2^k n^2)$. In other words, the path length k only exponentiates a small constant, instead of the problem size n, while the dependence on n is in fact quadratic.

3 Related Work

Decoding for selection-and-ordering problems is commonly implemented using beam search (Banko et al., 2000; Corston-Oliver et al., 2002; Jin and Hauptmann, 2001). Being heuristic in nature, this algorithm is not guaranteed to find an optimal solution. However, its simplicity and time efficiency make it a decoding algorithm of choice for a wide range of NLP applications. In applications where beam decoding does not yield sufficient accuracy, researchers employ an alternative heuristic search, A* (Jelinek, 1969; Germann et al., 2001). While in some cases A* is quite effective, in other cases its running time and memory requirements may equal that of an exhaustive search. Time- and memory-bounded modifications of A* (i.e., IDA-A*) do not suffer from this limitation, but they are not guaranteed to find the exact solution. Nor do they provide bounds on the likelihood of finding the exact solution. Newly introduced methods based on local search can effectively examine large areas of a search space (Eisner and Tromble, 2006), but they still suffer from the same limitations.

As an alternative to heuristic search algorithms, researchers also employ exact methods from combinatorial optimization, in particular integer linear programming (Germann et al., 2001; Roth and Yih, 2004). While existing ILP solvers find the exact solution eventually, the running time may be too slow for practical applications.

Our randomized decoder represents an important departure from previous approaches to decoding selection-and-ordering problems. The theoretically established bounds on the performance of this algorithm enable us to explicitly control the trade-off between the quality and the efficiency of the decoding process. This property of our decoder sets it apart from existing heuristic algorithms that cannot guarantee an arbitrarily high probability of success.

4 Randomized Decoding with Color-Coding

One might hope to solve decoding with a dynamic program (like that for shortest paths) that grows an optimal path one vertex at a time. The problem is that this dynamic program may grow to include a vertex already on the path, creating a cycle. One way to prevent this is to remember the vertices used on each partial path, but this creates a dynamic program with too many states to compute efficiently.

Instead, we apply a *color coding* technique of

Alon et al (1995). The basic step of the algorithm consists of randomly coloring the graph vertices with a set of colors of size r, and using dynamic programming to find the optimum length-k path *without repeated colors*. (Later, we describe how to determine the number of colors r.) Forbidding repeated colors excludes cycles as required, but remembering only colors on the path requires less state than remembering precisely which vertices are on the path. Since we color randomly, any single iteration is not guaranteed to find the optimal path; in a given coloring, two vertices along the optimal path may be assigned the same color, in which case the optimal path will never be selected. Therefore, the whole process is repeated multiple times, increasing the likelihood of finding an optimal path.

Our algorithm is a variant of the original color-coding algorithm (Alon et al., 1995), which was developed to detect the existence of paths of length k in an unweighted graph. We modify the original algorithm to find the highest weighted path and also to handle equivalence classes of vertices. In addition, we provide a method for determining the optimal number of colors to use for finding the highest weighted path of length k.

We first describe the dynamic programming algorithm. Next, we provide a probabilistic bound on the likelihood of finding the optimal solution, and present a method for determining the optimal number of colors for a given value of k.

Dynamic Programming Recall that we began with a weighted directed graph G to which we added artificial *start* and *end* vertices s and t. We now posit a *coloring* of that graph that assigns a color c_v to each vertex v aside from s and t. Our dynamic program returns the maximum score path of length $k+2$ (including the artificial vertices s and t) from s to t with no repeated colors.

Our dynamic program grows *colorful paths*—paths with at most one vertex of each color. For a given colorful path, we define the *spectrum* of a path to be the set of colors (each used exactly once) of nodes on the *interior* of the path—we exclude the starting vertex (which will always be s) and the ending vertex. To implement the dynamic program, we maintain a table $q[v, S]$ indexed by a path-ending vertex v and a spectrum S. For vertex v and spectrum S, entry $q[v, S]$ contains the value of the maximum-score colorful path that starts at s, terminates at v, and has spectrum S in its interior.

We initialize the table with length-one paths: $q[v, \emptyset]$ represents the path from s to v, whose spectrum is the empty set since there are no interior vertices. Its value is set to the score of edge (s, v). We then iterate the dynamic program k times in order to build paths of length $k + 1$ starting at s. We observe that the optimum colorful path of length ℓ and spectrum S from s to v must consist of an optimum path from s to u (which will already have been found by the dynamic program) concatenated to the edge (u, v). When we concatenate (u, v), vertex u becomes an interior vertex of the path, and so its color must not be in the preexisting path's spectrum, but joins the spectrum of the path we build. It follows that

$$q[v, S] = \max_{(u,v)\in G, c_u \in S, c_v \notin S} q[u, S - \{c_u\}] + w(u, v)$$

After k iterations, for each vertex v we will have a list of optimal paths from s to v of length $k + 1$ with all possible spectra. The optimum length-$k + 2$ colorful path from s to t must follow the optimum length-$k + 1$ path of *some* spectrum to *some* penultimate vertex v and then proceed to vertex t; we find it by iterating over all such possible spectra and all vertices v to determine $argmax_{v,S} q[v, S] + w(v, t)$.

Amplification The algorithm of Alon et al., and the variant we describe, are somewhat peculiar in that the probability of finding the optimal solution in one coloring iteration is quite small. But this can easily be dealt with using a standard technique called *amplification* (Motwani and Raghavan, 1995). Suppose that the algorithm succeeds with small probability p, but that we would like it to succeed with probability $1 - \delta$ where δ is very small. We run the algorithm $t = (1/p) \ln 1/\delta$ times. The probability that the algorithm fails every single run is then $(1 - p)^t \leq e^{-pt} = \delta$. But if the algorithm succeeds on even one run, then we will find the optimum answer (by taking the best of the answers we see).

No matter how many times we run the algorithm, we cannot absolutely guarantee an optimal answer. However, the chance of failure can easily be driven to negligible levels—achieving, say, a one-in-a-billion chance of failure requires only $20/p$ itera-

tions by the previous analysis.

Determining the number of colors Suppose that we use r random colors and want to achieve a given failure probability δ. The probability that the optimal path has no repeated colors is:

$$1 \cdot \frac{r-1}{r} \cdot \frac{r-2}{r} \cdots \frac{r-(k-1)}{r}.$$

By the amplification analysis, the number of trials needed to drive the failure probability to the desired level will be inversely proportional to this quantity. At the same time, the dynamic programming table at each vertex will have size 2^r (indexing on a bit vector of colors used per path), and the runtime of each trial will be proportional to this. Thus, the running time for the necessary number of trials T_r will be proportional to

$$1 \cdot \frac{r}{r-1} \cdot \frac{r}{r-2} \cdots \frac{r}{r-(k-1)} \cdot 2^r$$

What $r \geq k$ should we choose to minimize this quantity? To answer, let us consider the ratio:

$$\begin{aligned} \frac{T_{r+1}}{T_r} &= \left(\frac{r+1}{r}\right)^k \cdot \frac{r-(k-1)}{r+1} \cdot 2 \\ &= 2(1+1/r)^k(1-k/(r+1)) \end{aligned}$$

If this ratio is less than 1, then using $r+1$ colors will be faster than using r; otherwise it will be slower. When r is very close to k, the above equation is tiny, indicating that one should increase r. When $r \gg k$, the above equation is huge, indicating one should decrease r. Somewhere in between, the ratio passes through 1, indicating the optimum point where neither increasing nor decreasing r will help. If we write $\alpha = k/r$, and consider large k, then $\frac{T_{r+1}}{T_r}$ converges to $2e^{\alpha}(1-\alpha)$. Solving numerically to find where this is equal to 1, we find $\alpha \approx .76804$, which yields a running time proportional to approximately $(4.5)^k$.

In practice, rather than using an (approximate) formula for the optimum r, one should simply plug all values of r in the range $[k, 2k]$ into the running-time formula in order to determine the best; doing so takes negligible time.

5 Decoding with Integer Linear Programming

In this section, we show how to formulate the selection-and-ordering problem in the ILP framework. We represent each edge (i, j) from vertex i to vertex j with an indicator variable $I_{i,j}$ that is set to 1 if the edge is selected for the optimal path and 0 otherwise. In addition, the associated weight of the edge is represented by a constant $w_{i,j}$.

The objective is then to maximize the following sum:

$$\max_I \sum_{i \in V} \sum_{j \in V} w_{i,j} I_{i,j} \tag{1}$$

This sum combines the weights of edges selected to be on the optimal path.

To ensure that the selected edges form a valid acyclic path starting at s and ending at t, we introduce the following constraints:

Source-Sink Constraints Exactly one edge originating at source s is selected:

$$\sum_{j \in V} I_{s,j} = 1 \tag{2}$$

Exactly one edge ending at sink t is selected:

$$\sum_{i \in V} I_{i,t} = 1 \tag{3}$$

Length Constraint Exactly $k+1$ edges are selected:

$$\sum_{i \in V} \sum_{j \in V} I_{i,j} = k+1 \tag{4}$$

The $k+1$ selected edges connect $k+2$ vertices including s and t.

Balance Constraints Every vertex $v \in V$ has in-degree equal to its out-degree:

$$\sum_{i \in V} I_{i,v} = \sum_{i \in V} I_{v,j} \quad \forall\, v \in V^* \tag{5}$$

Note that with this constraint, a vertex can have at most one outgoing and one incoming edge.

Redundancy Constraints To control for redundancy, we require that at most one representative from each equivalence class is selected. Let Z be a set of vertices that belong to the same equivalence class. For every equivalence class Z, we force the total out-degree of all vertices in Z to be at most 1.

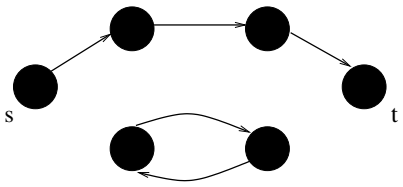

Figure 1: A subgraph that contains a cycle, while satisfying constraints 2 through 5.

$$\sum_{i \in Z} \sum_{j \in V} I_{i,j} \leq 1 \quad \forall\, Z \subseteq V \tag{6}$$

Acyclicity Constraints The constraints introduced above do not fully prohibit the presence of cycles in the selected subgraph. Figure 1 shows an example of a selected subgraph that contains a cycle while satisfying all the above constraints.

We force acyclicity with an additional set of variables. The variables $f_{i,j}$ are intended to number the edges on the path from 1 to $k+1$, with the first edge getting number $f_{i,j} = k+1$, and the last getting number $f_{i,j} = 1$. All other edges will get $f_{i,j} = 0$. To enforce this, we start by ensuring that only the edges selected for the path ($I_{i,j} = 1$) get nonzero f-values:

$$0 \leq f_{i,j} \leq (k+1)\, I_{i,j} \quad \forall\, i, j \in V \tag{7}$$

When $I_{i,j} = 0$, this constraint forces $f_{i,j} = 0$. When $I_{i,j} = 1$, this allows $0 \leq f_{i,j} \leq k+1$. Now we introduce additional variables and constraints. We constrain demand variables d_v by:

$$d_v = \sum_{i \in V} I_{i,v} \quad \forall\, v \in V^* - \{s\} \tag{8}$$

The right hand side sums the number of selected edges entering v, and will therefore be either 0 or 1. Next we add variables a_v and b_v constrained by the equations:

$$a_v = \sum_{i \in V} f_{i,v} \tag{9}$$

$$b_v = \sum_{i \in V} f_{v,i} \tag{10}$$

Note that a_v sums over f values on all edges entering v. However, by the previous constraints those f-values can only be nonzero on the (at most one) selected edge entering v. So, a_v is simply the f-value on the selected edge entering v, if one exists, and 0 otherwise. Similarly, b_v is the f-value on the (at most one) selected edge leaving v.

Finally, we add the constraints

$$a_v - b_v = d_v \quad v \neq s \tag{11}$$

$$b_s = k+1 \tag{12}$$

$$a_t = 1 \tag{13}$$

These last constraints let us argue, by induction, that a path of length exactly $k+1$ must run from s to t, as follows. The previous constraints forced exactly one edge leaving s, to some vertex v, to be selected. The constraint $b_s = k+1$ means that the f-value on this edge must be $k+1$. The balance constraint on v means some edge must be selected leaving v. The constraint $a_v - b_v = d_v$ means this edge must have f-value k. The argument continues the same way, building up a path. The balance constraints mean that the path must terminate at t, and the constraint that $a_t = 1$ forces that termination to happen after exactly $k+1$ edges.[4]

For those familiar with max-flow, our program can be understood as follows. The variables I force a flow, of value 1, from s to t. The variables f represent a flow with supply $k+1$ at s and demand d_v at v, being forced to obey "capacity constraints" that let the flow travel only along edges with $I = 1$.

6 Experimental Set-Up

Task We applied our decoding algorithm to the task of title generation. This task has been extensively studied over the last six years (Banko et al., 2000; Jin and Hauptmann, 2001). Title generation is a classic selection-and-ordering problem: during title realization, an algorithm has to take into account both the likelihood of words appearing in the title and their ordering preferences. In the previous approaches, beam search has been used for decoding. Therefore, it is natural to explore more sophisticated decoding techniques like the ones described in this paper.

Our method for estimation of selection-and-ordering preferences is based on the technique described in (Banko et al., 2000). We compute the

[4] The network flow constraints allow us to remove the previously placed length constraint.

likelihood of a word in the document appearing in the title using a maximum entropy classifier. Every stem is represented by commonly used positional and distributional features, such as location of the first sentence that contains the stem and its TF*IDF. We estimate the ordering preferences using a bigram language model with Good-Turing smoothing.

In previous systems, the title length is either provided to a decoder as a parameter, or heuristics are used to determine it. Since exploration of these heuristics is not the focus of our paper, we provide the decoder with the actual title length (as measured by the number of content words).

Graph Construction We construct a decoding graph in the following fashion. Every unique content word comprises a vertex in the graph. All the morphological variants of a stem belong to the same equivalence class. An edge (v, u) in the graph encodes the selection preference of u and the likelihood of the transition from v to u.

Note that the graph does not contain any auxiliary words in its vertices. We handle the insertion of auxiliary words by inserting additional edges. For every auxiliary word x, we add one edge representing the transition from v to u via x, and the selection preference of u. The auxiliary word set consists of 24 prepositions and articles extracted from the corpus.

Corpus Our corpus consists of 547 sections of a commonly used undergraduate algorithms textbook. The average section contains 609.2 words. A title, on average, contains 3.7 words, among which 3.0 are content words; the shortest and longest titles have 1 and 13 words respectively. Our training set consists of the first 382 sections, the remaining 165 sections are used for testing. The bigram language model is estimated from the body text of all sections in the corpus, consisting of 461,351 tokens.

To assess the importance of the acyclicity constraint, we compute the number of titles that have repeated content words. The empirical findings support our assumption: 97.9% of the titles do not contain repeated words.

Decoding Algorithms We consider three decoding algorithms: our color-coding algorithm, ILP, and beam search.[5] The beam search algorithm can only consider vertices which are not already in the path.[6]

To solve the ILP formulations, we employ a Mixed Integer Programming solver *lp_solve* which implements the Branch-and-Bound algorithm. We implemented the rest of the decoders in Python with the Psyco speed-up module. We put substantial effort to optimize the performance of all of the algorithms. The color-coding algorithm is implemented using parallelized computation of coloring iterations.

7 Results

Table 1 shows the performance of various decoding algorithms considered in the paper. We first evaluate each algorithm by the running times it requires to find all the optimal solutions on the test set. Since ILP is guaranteed to find the optimal solution, we can use its output as a point of comparison. Table 1 lists both the average and the median running times. For some of the decoding algorithms, the difference between the two measurements is striking — 6,536 seconds versus 57.3 seconds for ILP. This gap can be explained by outliers which greatly increase the average running time. For instance, in the worst case, ILP takes an astounding 136 hours to find the optimal solution. Therefore, we base our comparison on the median running time.

The color-coding algorithm requires a median time of 9.7 seconds to find an optimal solution compared to the 57.3 seconds taken by ILP. Furthermore, as Figure 2 shows, the algorithm converges quickly: just eleven iterations are required to find an optimal solution in 90% of the titles, and within 35 iterations all of the solutions are found. An alternative method for finding optimal solutions is to employ a beam search with a large beam size. We found that for our test set, the smallest beam size that satisfies this condition is 1345, making it twenty-three times slower than the randomized decoder with respect to the median running time.

Does the decoding accuracy impact the quality of the generated titles? We can always trade speed for accuracy in heuristic search algorithms. As an extreme, consider a beam search with a beam of size 1: while it is very fast with a median running time

[5]The combination of the acyclicity and path length constraints require an exponential number of states for A* since each state has to preserve the history information. This prevents us from applying A* to this problem.

[6]Similarly, we avoid redundancy by disallowing two vertices from the same equivalence class to belong to the same path.

	Average (s)	Median (s)	ROUGE-L	Optimal Solutions (%)
Beam 1	0.6	0.4	0.0234	0.0
Beam 80	28.4	19.3	0.2373	64.8
Beam 1345	368.6	224.4	0.2556	100.0
ILP	6,536.2	57.3	0.2556	100.0
Color-coding	**73.8**	**9.7**	**0.2556**	**100.0**

Table 1: Running times in seconds, ROUGE scores, and percentage of optimal solutions found for each of the decoding algorithms.

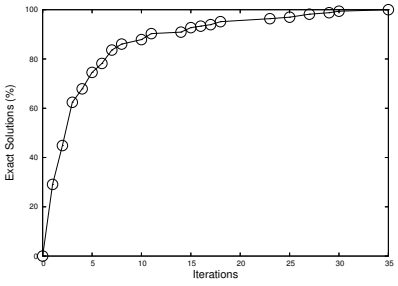

Figure 2: The proportion of exact solutions found for each iteration of the color coding algorithm.

of less than one second, it is unable to find any of the optimal solutions. The titles generated by this method have substantially lower scores than those produced by the optimal decoder, yielding a 0.2322 point difference in ROUGE scores. Even a larger beam size such as 80 (as used by Banko et al. (2000)) does not match the title quality of the optimal decoder.

8 Conclusions

In this paper, we formalized the decoding task for selection-and-ordering as a problem of finding the highest-weighted acyclic path in a directed graph. The presented decoding algorithm employs randomized color-coding, and can closely approximate the ILP performance, without blowing up the running time. The algorithm has been tested on title generation, but the decoder is not specific to this task and can be applied to other generation and summarization applications.

9 Acknowledgements

The authors acknowledge the support of the National Science Foundation (CAREER grant IIS-0448168 and grant IIS-0415865). We also would like to acknowledge the MIT NLP group and the anonymous reviewers for valuable comments.

References

N. Alon, R. Yuster, U. Zwick. 1995. Color-coding. *Journal of the ACM (JACM)*, 42(4):844–856.

B. Awerbuch, Y. Azar, A. Blum, S. Vempala. 1995. Improved approximation guarantees for minimum-weight k-trees and prize-collecting salesmen. In *Proceedings of the STOC*, 277–283.

E. Balas. 1989. The prize collecting traveling salesman problem. *Networks*, 19:621–636.

M. Banko, V. O. Mittal, M. J. Witbrock. 2000. Headline generation based on statistical translation. In *Proceedings of the ACL*, 318–325.

S. Corston-Oliver, M. Gamon, E. Ringger, R. Moore. 2002. An overview of amalgam: A machine-learned generation module. In *Proceedings of INLG*, 33–40.

R. G. Downey, M. R. Fellows. 1995. Fixed-parameter tractability and completeness II: On completeness for $W[1]$. *Theoretical Computer Science*, 141(1–2):109–131.

J. Eisner, R. W. Tromble. 2006. Local search with very large-scale neighborhoods for optimal permutations in machine translation. In *Proceedings of the HLT-NAACL Workshop on Computationally Hard Problems and Joint Inference in Speech and Language Processing*.

U. Germann, M. Jahr, K. Knight, D. Marcu, K. Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *Proceedings of the EACL/ACL*, 228–235.

F. Jelinek. 1969. A fast sequential decoding algorithm using a stack. *IBM Research Journal of Research and Development*.

R. Jin, A. G. Hauptmann. 2001. Automatic title generation for spoken broadcast news. In *Proceedings of the HLT*, 1–3.

R. Motwani, P. Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press, New York, NY.

D. Roth, W. Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the CONLL*, 1–8.