Adding Linguistic Constraints to Document Image Decoding: Comparing the Iterated Complete Path and Stack Algorithms

Kris Popat[a], Daniel H. Greene[a], Justin K. Romberg[b], and Dan S. Bloomberg[a]

[a]Xerox Palo Alto Research Center, Palo Alto, California, USA
[b]ECE Dept., Rice University, Houston, Texas, USA

ABSTRACT

Beginning with an observed document image and a model of how the image has been degraded, Document Image Decoding recognizes printed text by attempting to find a most probable path through a hypothesized Markov source. The incorporation of linguistic constraints, which are expressed by a sequential predictive probabilistic language model, can improve recognition accuracy significantly in the case of moderately to severely corrupted documents. Two methods of incorporating linguistic constraints in the best-path search are described, analyzed and compared. The first, called the iterated complete path algorithm, involves iteratively rescoring complete paths using conditional language model probability distributions of increasing order, expanding state only as necessary with each iteration. A property of this approach is that it results in a solution that is exactly optimal with respect to the specified source, degradation, and language models; no approximation is necessary. The second approach considered is the *Stack* algorithm, which is often used in speech recognition and in the decoding of convolutional codes. Experimental results are presented in which text line images that have been corrupted in a known way are recognized using both the ICP and Stack algorithms. This controlled experimental setting preserves many of the essential features and challenges of real text line decoding, while highlighting the important algorithmic issues.

Keywords: Document Image Decoding, optical character recognition, lexical language modeling, hidden Markov models, dynamic programming, Viterbi algorithm, stack algorithm, list decoding, convolutional decoding

1. INTRODUCTION

Document Image Decoding (DID)[1,2] is a method of text recognition in document images that is based on a communications systems view of the document composition, printing, degradation, and scanning processes. Among the advantages of DID are high recognition accuracy in situations where extensive customization is allowable, the ability to recognize some higher-level structure along with the text, and the ability to extend and improve the system within a consistent probabilistic framework. Surprisingly, in most of the work on DID reported until now, the high recognition accuracy has been achieved despite a lack of any prior specification of which recognized strings are linguistically valid and which are not.

Recently, a technique for incorporating linguistic constraints into DID was proposed and partially explored using a simulated, one-dimensional Morse-code signaling scheme having known corruption parameters.[3] While useful in illustrating the functioning of the proposed algorithm, that treatment did not place the technique in sufficient perspective to draw conclusions about it. In this paper, we examine that technique more closely, and compare it with the *Stack* algorithm, which is a standard, widely used alternative. In addition, we replace the one-dimensional Morse-code setting with one involving synthetic two-dimensional text-line images. For methodological reasons, we continue to exercise tight control over the manner in which the images are produced and corrupted. Nevertheless, working on two-dimensional images of printed text improves both the realism and the relevance of the resulting comparison and analysis over the previous experimental framework.

Author email addresses: {popat|greene|bloomberg}@parc.xerox.com, jrom@rice.edu

[To appear in *Proceedings of IS&T/SPIE Electronic Imaging 2001: Document Recognition and Retrieval VIII* January 2001.]

1.1. Document Image Decoding

We briefly review the essential elements of traditional Document Image Decoding. For details, the reader is referred to Kopec and Chou.[2] In the DID framework, document images are regarded as having been produced by transitioning through a Markov source, which is a probabilistic finite-state machine. The source begins in a special *start* state and terminates in a special *stop* state. Each transition within the source causes the imaging of a character template (a bitmap) on the page at a current cursor location, then advances that location by a two-dimensional vector displacement in preparation for printing the next character. The set of character templates includes whitespace of various kinds. Formally, each transition in the source is assigned a four-tuple consisting of a character template, the two-dimensional displacement by which to advance the cursor, the prior probability of following that transition, and a string label. Note that the notion of prior probability here is quite limited; for instance, it does not take into account what previous transitions might have occurred on the same path through the Markov source. Every complete path through the source defines a document image and an associated transcription: the image is the union of the bitmaps imaged on each transition, and the transcription is the concatenation of the associated string labels. It should be noted that more than one complete path through the source may give rise to the same image and/or the same transcription.

After the document image has been formed in this way, it is assumed to be subjected to some form of random corruption, which is the cause of some uncertainty in the recognition process. Recognition proceeds by finding a complete path through the hypothesized Markov source that "best" explains the observed image. Specifically, a complete path is sought that is most probable considering the entire image as evidence, where the probability is computed on the basis of the prior probabilities of the transitions, the likelihoods of the associated imaged templates, and the random corruption process. Because multiple paths can correspond to the same transcription, choosing the most probable complete path is not the same as choosing the most probable transcription. The probability of a transcription is properly calculated by summing the probabilities of all of the complete paths that are consistent with that transcription. Nevertheless, experience has shown that choosing a message with the greatest complete-path probability is usually a good approximation to choosing the message with the highest posterior probability; this is known as the *Viterbi approximation* to the maximum *a posteriori* probability (MAP) decision rule. We use this approximation throughout. Thus, given an observed document image, the recognition task reduces to finding a MAP path through the Markov source. Traditionally, dynamic programming[4] has been used for this task.

1.2. Separable Markov Sources and Text Line Decoding

In the general DID setting, groups of states in the Markov source correspond to distinct regions of the document image, which expresses the view that different generative models are appropriate for regions of different types. For example, in a Markov source for the first page of a two-column paper, one group of states might correspond to the title line, another to the abstract block, and another to the left text column, etc. The most basic type of group is that for a text column. Text columns can be modeled by a two-level Markov source structure in which the top level characterizes and accounts for the vertical placement of lines of text, and the bottom level accounts for the formation of the text lines themselves.[5] A Markov source fitting this description is termed *separable*. Because linguistic constraints bear most strongly on recognition within text lines, we focus our attention there instead of on more general two-dimensional structures. The recognition problem thus becomes that of finding a MAP path through the lower-level text-line subsource for a given observed text line image.

In the absence of any linguistic constraints, a suitable subsource model for a text line consists of a start state, a single interior state, and a stop state. The interior state has one self-transition for each character template in the font. Generation of a text line begins in the start state, with the cursor at a predetermined horizontal position in the text-line image. The first transition is into the interior state, and subsequent transitions loop back into that state, each time imaging a character and advancing the cursor horizontally. After the text line has thus been produced, a final transition is made into the stop state, whereupon the process terminates. Typically, the cursor positions specified for the start and stop states are the left- and right-most printable pixel locations in the image, respectively. A complete path through this subsource can be conveniently represented by a trellis diagram, wherein nodes represent horizontal pixel locations along the baseline, and directed edges represent the possible transitions that connect the nodes. A complete path is obtained by following a sequence of edges from the start node to the stop node. Each edge is labeled with a score that can be interpreted as a posterior probability: the product of the prior probability of the transition and the likelihood of the corresponding imaged template in the spatial location that the edge spans. It is convenient to represent these scores as logarithms. In our discussion of various search algorithms, we will generalize the trellis graph structure considerably in Section 3. As mentioned, in the absence of linguistic constraints, finding the highest-score path can be accomplished by a straightforward application of dynamic programming. Dynamic programming on a trellis weighted with log likelihoods is often referred to as the *Viterbi algorithm*,[6] and we will so refer to it here.

2. SOFT LINGUISTIC CONSTRAINTS

As described above, DID makes no use of prior knowledge about which recognized transcriptions are more linguistically valid than others. It is desirable for the purpose of reducing error rate to provide DID with a means of preferring linguistically valid transcriptions over less valid ones.

DID chooses the transcription that corresponds to a path that has the highest posterior probability, where that probability is computed as the sum, over all transitions along the path, of the sum of the log likelihood of that transition and an unconditional log prior probability of making that transition. A natural way of expressing a prior preference for transcriptions that are deemed valid is to make the prior probabilities attached to each transition depend on which transitions were followed previously along the path. Assigning these probabilities is the function of a *language model*, which is discussed next. However, making the prior probability depend on anything besides the identity of the originating node is a violation of the Markov assumption, and greatly complicates the search problem. Discussion of best-path search in the presence of a language model is taken up in detail in Section 3.

2.1. Probabilistic Language Model

Linguistic validity can be measured by means of a probabilistic language model, which in its full generality is a probability distribution over all finite sequences over the alphabet of string labels attached to transitions in the Markov source. Usually, the string labels on the transitions are single characters. Hence, for simplicity, we will refer to the string labels attached to the transitions as *characters,* and to sequences of the string labels as *strings*. For use in DID we restrict the language model to be factorable as a sequence of probability distributions over individual characters, each conditioned on a subset of preceding characters. These conditional probabilities then replace the unconditional prior probabilities on the transitions in the original DID formulation. Let the alphabet be $\mathcal{A}$, and let $v_1, \ldots, v_n$ denote a string with $v_i \in \mathcal{A}, i = 1, \ldots, n$. Let τ be a termination symbol, and let $\mathcal{A}' = \mathcal{A} \cup \{\tau\}$. We view strings as having been formed by the following process. Characters are generated sequentially according to a sequence of conditional probability distributions

$$p_i(v_i|v_1, \ldots, v_{i-1}) = p_i(v_i|\phi_i(v_1, \ldots, v_{i-1})) \tag{1}$$

where $v_i \in \mathcal{A}'$, $v_1, \ldots, v_{i-1} \in \mathcal{A}$, $i = 1, 2, \ldots$, and the function $\phi_i(v_1, \ldots, v_{i-1})$ maps contexts into equivalence classes. The string terminates when the symbol τ is generated; in terms of the Markov source this corresponds to a transition into the stop state.

For simplicity, we remove the dependence of p in (1) on i, and restrict $\phi_i(v_1, \ldots, v_{i-1})$ to be of the form

$$\phi_i(v_1, \ldots, v_{i-1}) = \begin{cases} (v_1, \ldots, v_{i-1}) & \text{if } i \leq N \\ (v_{i-N+1}, \ldots, v_{i-1}) & \text{otherwise} \end{cases} \tag{2}$$

for a fixed small integer N. With these restrictions, (1) is referred to as a *character N-gram language model*, hereinafter referred to simply as an N-gram model:

$$p_i(v_i|v_1, \dots, v_{i-1}) = p(v_i|v_{i-N_i+1}, \dots, v_{i-1}) \tag{3}$$

where $N_i = \min\{i, N\}$. Although the search techniques to be considered in Section 3 will remain practical when a class of language models more general than N-grams is used, N-grams are simple and effective in capturing important statistical regularity in natural language strings. We therefore adopt their use here. To illustrate the ability of the N-gram to model English, we exhibit a pseudorandom string that was obtained by sequentially sampling from a 5-gram model trained on the Brown corpus[7]:

```
And fine alone other Itality and agains she tumor his result, from
of Brannot faculous shall precentative inter at lear time to much a
relanguages proposal?  Baer fall over Open-megaw of most used agreen
during represearchbishes of chlor] But the first position a little
rear intenant year-olds to And also if we should beef childing face
graduates
```

The implementation and training of the N-gram model used in this study are as were reported in the previous "Morse code" paper.[3]

2.2. Auxiliary Upper Bound Functions

In addition to the conditional probabilities provided by (3), one of the search techniques to be considered in Section 3 requires that a set of upper-bound functions of a particular form be defined on the probabilities in addition to the probabilities themselves. We describe them now, although their purpose will not be fully clear until Section 3.3. For a fixed N-gram, we define a sequence of auxiliary functions $q_0, \dots, q_{N_i-1}$ as

$$q_k(v_i|v_{i-k}, \dots, v_{i-1}) = \max_{v_{i-N_i+1}, \dots, v_{i-k-1}} p(v_i|v_{i-N_i+1}, \dots, v_{i-1}) \tag{4}$$

which for each k provides an upper bound on the probability that can be assigned by the N-gram to v_i when immediately preceded by $(v_{i-k}, \dots, v_{i-1})$. For example, q_0 specifies the maximum probability that can be assigned by the model to v_i in *any* context, while at the other extreme, q_{N-1} is simply another name for p. Note that for any fixed string section $(v_{i-N_i+1}, \dots, v_i)$, q_k is nonincreasing in k. Note further that for sufficiently large N, q_0 will be close to 1 for every character. This is because every character, no matter how infrequent in absolute terms, becomes nearly certain in *some* context. As k increases, the bound tightens.

3. BEST-PATH SEARCH IN THE PRESENCE OF A LANGUAGE MODEL

The introduction of even a modestly complex language model (e.g. $N = 4$) immediately creates computational challenges for the decoding algorithm. The Viterbi approach cannot be extended in the obvious way (i.e., by taking advantage of the N-gram's limited memory to conduct the search in an expanded trellis in which the Markov assumption is once again obeyed), because the state space for the search problem now includes both image information and language model information, and so the size of the state space has increased significantly. For example, when $N = 4$, the state space grows from W to $W|\mathcal{A}|^3$, where W is the width in pixels of the text-line image, and $|\mathcal{A}|$ is the alphabet size. We must therefore consider algorithms that do not explore the entire state space. Here we benefit from algorithmic techniques from related domains: convolutional coding,[8] speech processing,[9] and artificial intelligence,[4] that can be adapted to DID.

While language modeling is helpful in reducing errors, in DID the image normally has more influence than the language model in determining the best decoding. Accordingly, one approach is to factor the problem, solving it first with template matching without a language model, generating the K-best solutions, and then rescoring these solutions with the language model to obtain the best solution.[10] While the computation of the K-best solutions is feasible for small K, experience shows that K must be very large (on the order of 500 for an average text line) to include the best solution.[11] It appears that several short, independent sections of ambiguity in a line of text, each with only a few nearly equivalent decodings, can combine to create a large number of nearly equivalent decodings for the entire line. To be both efficient and effective, the language model must be more tightly integrated with the best-path search algorithm than is achieved by K-best rescoring.

Prior to stating the two specific algorithms to be compared, we will set the stage by identifying two broad categories of search algorithms: *bound-based* techniques that find the best solution by guaranteeing that the states omitted by the search cannot be part of the best solution, and *estimation-based* techniques that focus the search on the most relevant portions of the state space.

3.1. Search Techniques Based on Bounding

Discussion is simplified if we abstract the problem to a graph-theoretic setting. Consider a directed graph $G = (V, E)$, where the nodes V correspond to states, which in our case consist of positions in the image and context relevant to the language model, and the edges E correspond to transitions. Incorporation of position as part of the state, and the requirement that each template have nonzero width, ensure that the graph has no cycles. In a manner analogous to the scoring of edges in the trellis described in Section 1.2, the edges E are such that each is associated with a character template and is weighted with a score that is the sum of two components: the log likelihood of matching the character at the image position associated with the originating node, and the log of the conditional probability (3) returned by the language model when supplied with the context specified by the originating node. The use of conditional rather than unconditional language model probabilities distinguishes this graph from the graph developed in Section 1.2. The best decoding is taken to be the largest-weight path through this graph. If the graph were small enough there would be many different algorithms that could be used to find the best path — dynamic programming has already been mentioned. For large graphs, a popular bounding technique, the A* algorithm,[4] uses a function $h(v)$ which is easily computed, and which gives an upper bound on the best possible path from a node v to the stop state. The search then proceeds by iteratively exploring the edges E leaving the most promising state, as measured by the function $g(v) + h(v)$, where $g(v)$ is the actual partial path score from source to v, and $h(v)$ is an upper bound on the weight of any path from v to the stop state. Even though it does not explore the whole graph, the A* algorithm finds the best solution, because when the search reaches the stop state, all unexplored nodes have lower $g(v) + h(v)$ than the stop state, and so they cannot be part of the best solution.

The success of the A* algorithm depends entirely on the bounding function $h(v)$. Easily computed and accurate bounds $h(v)$ lead to very efficient algorithms. A* approaches have been used in speech processing,[9] where $h(v)$ can be created by a first (backward) pass of the algorithm with either no language model or a language model that is small enough to allow for a Viterbi computation of the best possible interpretation of the remainder of the signal. A second (forward) pass then uses an A* algorithm with a bounding function that is based on the first pass.

For linear decoding applications, the A* algorithm suffers from a scaling problem — the accuracy of the bound decreases with length of the path remaining to the stop state. As a consequence, an A* search will explore a disproportionately large amount of state space near the beginning of the line that ultimately will not prove relevant to the best solution. As was shown recently,[3] an alternative approach, the Iterated Complete Path (ICP) algorithm first used by Kam and Kopec[5] in a different context, can be extended to the language modeling problem. Rather than using a two pass approach in which the bounding function is developed entirely in the first pass, the ICP approach iteratively refines the bound near the most promising portions of the graph. This approach will be discussed in detail in Section 3.3.

3.2. Search Techniques Based on Estimation

Returning to the general problem of finding the best path in a large graph, there is a class of algorithms that make use of heuristic or estimated computations to guide the search. These algorithms find the shortest path on a subset of the graph; however, the subset explored is not determined with the rigor of the bounding algorithms described above. Instead, exploration is guided by predictive estimates of the scores of unexplored path segments.

One extreme example of such a technique is a "greedy" algorithm, which grows a single path by always following a best outgoing edge at each node until the stop node is reached. The greedy approach is prone to missing good solutions because of its inability to backtrack when the path no longer looks promising. An effective estimation-based search strategy must have a way of selectively postponing work on partially explored paths to return to earlier nodes in the graph to explore alternate decodings. The opposite of the greedy approach is to systematically explore *all* of the short paths before moving on (i.e., breadth-first search), which will end up exploring the entire graph before determining a solution. The more interesting and useful estimation-based algorithms lie between these two extremes. One approach is the Stack algorithm, in which the estimates allow partial paths of differing lengths to be effectively compared. We believe that the Stack algorithm is an attractive choice in the DID setting for a number of reasons, and we will discuss it in detail in Section 3.4.

Two other estimation-based strategies are worth mentioning: the *List* and *Beam* search algorithms. In these, only paths of similar lengths are directly compared. In the context of signal processing, these algorithms are referred to as *time-synchronous.* List search uses a simple K-best heuristic to determine which paths are explored. All nodes v at the same image position are processed together, and only the K-best nodes (according to $g(v)$, the weight of the path from the source to v) are explored. List search suffers from the need to carry forward K paths at every position. Where the noise is low, the extra paths are unnecessary, and where the noise is high, it is likely that the best path will be lost. Beam search attempts to alleviate this problem by varying the number of paths explored, rather than using a fixed number K. In Beam search, all partial paths scoring within a threshold of the best current partial path are explored.

We have described two basic classes of algorithm that avoid searching the entire graph. One uses bounds on the scores of unexplored path segments, while the other uses estimates. We next discuss, in detail, the bound-based ICP algorithm applied to DID in the presence of a language model. Section 3.4 will present a corresponding discussion of the estimate-based Stack algorithm.

3.3. Iterated Complete Path Algorithm

As mentioned, the ICP algorithm involves iteratively rescoring complete paths using language models of increasing order. We describe it here in terms of the graph defined above. More details can be found in previous work.[3] To begin our description of the algorithm, we need some additional structure in the graph. We introduce a class of *less refined* nodes V^* with states that include position in the image, but which have an incomplete linguistic context – that is, a linguistic context that is less specific than can be distinguished by the language model. A node v' is considered to be a refinement of a node v if both nodes have the same spatial position and if the incomplete linguistic context of v can be completed to the context at v' by prepending a character. For example, a node with linguistic context `ABCD` is a refinement of one with context `BCD`, which itself is a refinement of one with context `CD`, and so forth until there is a node at the same spatial position having an empty or *null* linguistic context. Edges that originate in V^* are weighted with context-conditional upper bounds rather than with true language model scores. That is, the language-model component of the score assigned to such an edge is the logarithm of the bound q_k defined in Equation (4), where k is determined by the length of the context of the originating node. The ICP algorithm then proceeds as follows:

1 The graph G is initialized to contain only those nodes that contain the null context (the least refined nodes).

2 A Viterbi algorithm is used to compute a candidate best path p in G.

3 Along p the nodes in V^* are replaced with refinement nodes that are consistent with the transcription of p. Depending on the level of refinement, these new nodes may be members of either V or V^*.

4 Steps 2 and 3 are repeated until the best path p contains only nodes from V, that is, the path is completely refined.

So, for example, if G contains a node from V^* that has linguistic context `CD`, and the node appears on the current best path p having transcription `ABCDEF`, then a new, more refined node with context `BCD` is added to the graph for the next iteration. It is possible to implement G with implicit edges, whose weights can be derived from precomputed language model and template match scores, so that the expansion of the graph in step 3 simply involves adding more refined nodes along the path p.

At every iteration, the best path computation is on a graph that contains both exact language model scores (on edges leaving nodes in V) and upper bounds on language model scores (on edges leaving nodes in V^*). Paths involving nodes in V^* will only get worse when refined, so as soon as we encounter a current best path p that contains only nodes in V, we can terminate the algorithm knowing that the remaining nodes in V^* cannot be refined to yield a better path.

In principle, there is no limit on the amount of the graph that the ICP algorithm can explore. In practice however, the ICP algorithm is able to take advantage of the tendency of the image match scores to dominate over the language model scores in the overall path score. Typically, the linguistic constraints exert more of a "tiebreaking" influence on the decoding decision. This allows it to rule out the vast majority of poor paths quickly, even when those paths are "propped up" by the relatively loose language-model bounds associated with the least-refined nodes. Convergence is particularly fast in the low-noise regime, as will be seen in the experimental results presented in Section 4. When the noise increases beyond a certain range, the ICP algorithm becomes less feasible, but in such cases it can be stopped at any time to yield a good current guess. Note that in the high-noise regime, the connection between the best path and the true transmitted message becomes tenuous, so that it makes little sense to insist on finding the true best path in such cases, particularly if doing so comes at a high computational cost.

3.4. Stack Algorithm

The essential idea of the *Stack* algorithm[9,8] is to provide a means of directly comparing paths of differing lengths, so that at each step the most promising path can be extended. At the heart of the Stack algorithm is a priority queue that is used to determine the most promising path according to the formula $g(v) + h(v)$, where $g(v)$ is the actual score of the partial path from the source to v, and $h(v)$ is a prediction of the score of the best remaining path from v to the sink. We can interpret $g(v) + h(v)$ as being a measure of the overall potential of the partial path ending in node v to be continued to a high-scoring full path. The form of this expression is identical to that used in the A* algorithm. However, the role of $h(v)$ differs significantly from its role in A*, making the algorithm behave very differently. In A*, $h(v)$ is a strict upper bound on the score of the best remaining path, whereas in the Stack algorithm, $h(v)$ is an estimate of that score. As a result, the guarantee provided by A* that a true best path will be found when the algorithm terminates is lost. In compensation, there is typically a significant reduction in complexity in both the design and running of the algorithm. An estimate is usually simpler to define and compute than a bound, and its use in place of a bound prevents the exploration of many extraneous partial paths.

An alternative but equivalent formulation is to measure the potential of each node v as the difference between $g(v)$ and an expected score $t(v)$. Here, we simplify the computation of $t(v)$ by making it depend only on spatial position, so that $t(v)$ is the expected score for a generic "promising" node at the same spatial position. The function $t(v)$ can be thought of as *tilting* the best-first search computation on $g(v)$ so that it favors paths further in the graph. For this reason, $t(v)$ is referred to as a *tilt function*.

Clearly the choice of $t(v)$, or equivalently $h(v)$, is critical to the running of the algorithm. In our DID problem, the log-posterior-probability weighting of the graph edges, under a stationarity assumption, suggests that $t(v)$ should grow linearly with the position of v in the image. The slope of $t(v)$ is the crucial parameter, and must be chosen carefully. Intuitively, the slope sets an expectation for the score of a path based on its length. Partial paths with scores that are better than their expected value according to this measure are rewarded by being explored further, while partial paths that are worse than expected are more likely to be abandoned. Setting too low a slope sets too low an expectation, so that an arbitrary path can be rewarded for simply for being long, effectively "locking out" other paths that may be better. At the other extreme, setting too high a slope penalizes length, resulting in an explosion of short paths that must be considered. Since low-slope tilt functions generally allow for faster computations, with less accurate and more "greedy" results, the tilt function provides a means of trading-off accuracy with computation time. We are actively exploring the problem of adapting the tilt function in an automatic way. As will be seen in the experimental results in Section 4, with a good tilt function, the stack algorithm performs well in both the low noise and high noise regimes.

It is natural to ask whether it is possible to obtain the computational benefits of estimation-based searching without the strong dependence on a tilt function. The original motivation for the tilt function was to compare paths of different lengths. We could organize our search so that only paths of similar lengths are directly compared. This is the basis for the *List* and *Beam* search strategies. After some experimentation, we have chosen to focus on the Stack algorithm rather than the List or Beam algorithms. Stack appears to have lower computation costs in the low noise regime (where there is no difference in accuracy), and comparable accuracy in the high noise regime. In situations where the determination of an appropriate tilt function is difficult and moderate additional computational cost can be tolerated, the ICP approach described in the preceeding section becomes attractive because it is based only on relatively weak assumptions and does not require the setting of critical parameters.

4. EXPERIMENTAL RESULTS

We illustrate the Stack and ICP algorithms by applying DID to simulated low-resolution grayscale text-line images. Ideal text line images are corrupted by spatial lowpass filtering, subsampling, and additive noise. Specifically, the lowpass filter is a separable 11-tap FIR filter, the subsampling is 3×3, and the noise process consists of adding zero-mean Gaussian pseudorandom numbers with a standard deviation of σ, followed by quantization and clipping back into the range $[0, \ldots, 255]$. Example degraded text lines are shown for several values of σ in Figure 1. Except for noting that σ is a free parameter, the details of the degradation process and the corresponding template-match function are not material to our discussion of the algorithms.

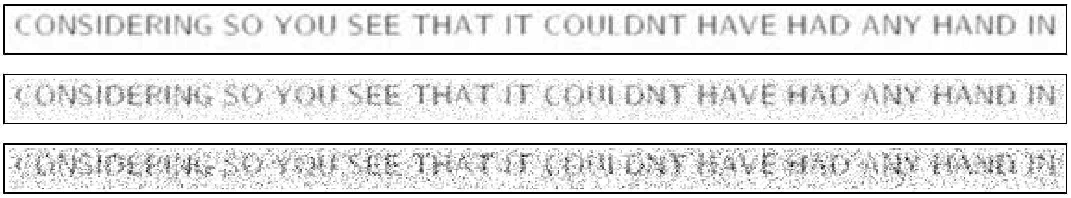

Figure 1. Example text line images for $\sigma = 5$ (top), $\sigma = 25$ (middle), $\sigma = 50$ (bottom).

Training and test data are taken from an electronic version of Lewis Carroll's *Through the Looking Glass*,[12] with all characters mapped to upper case and all punctuation symbols except periods, commas, and question marks removed. Blank lines are omitted, and the remaining 3,118 lines of text are divided into two equal-size portions: a training segment, consisting of the even-numbered lines, and a test portion, consisting of the odd-numbered lines. In this simulation, only the language model needs to be trained, as all other models in the system are known accurately in advance. Training the language model on even-numbered lines and testing the system on odd-numbered lines makes it likely that the language model is both relevant and accurate, while still preventing severe overfitting. The alphabet is taken to be the set of symbols that actually appears in the union of the training and test data.

The preprocessing step of mapping symbols to uppercase and removing most forms of punctuation is a holdover from earlier work on the ICP algorithm in which Morse code was used for signaling instead of images of printed text.[3] Here, the preprocessing is retained because of the computational simplicity that results by severely restricting the size of the alphabet.

A final simplification in the present simulation is the omission of an unlabeled single-pixel-space transition in the Markov source. Its absence simplifies implementation of the algorithms, but sacrifices a degree of realism in that the important synchronization issues that normally arise in decoding are largely avoided. Moreover, omitting this transition requires that the correct horizontal positions of the start- and stop-nodes be specified to the decoder in advance for each text line.

Because of all of these simplifications — the severely restricted alphabet, the known relevance and accuracy of the language model, the absence of a single-pixel-space transition, and perfect prior knowledge of the image formation and degradation models — DID recognition accuracy is much higher in this simulation than would be expected were it applied to actual scanned images of comparable subjective quality. This phenomenon is illustrated in Figure 2, where a completely illegible corrupted text line image (a) is correctly decoded by two of the algorithms. While it is largely a matter of methodological preference whether to explore algorithmic issues in a simulated setting, the simulation should still be realistic, and in this respect expanding the alphabet and inclusion of a single-pixel-space transition would be an important improvement. This will be taken up in future work. Nevertheless, the present simulation is felt to retain enough realism to allow the essential characteristics of the search algorithms to be analyzed qualitatively.

(a) 

(b) CDNSIOEBING SD YOU SEE TP?IT.IJ, LIJULONT KAVE HAD ANX HAND IN

(c) CDNSIOEBING SD YOU SEE THAT IT COULDNT HAVE HAD ANX HAND IN

(d) CONSIDERING SO YOU SEE THAT IT COULDNT KAVE HAD ANY PAND ID

(e) CONSIDERING SO YOU SEE THAT IT COULDNT HAVE HAD ANY HAND IN

(f) CONSIDERING SO YOU SEE THAT IT COULDNT HAVE HAD ANY HAND IN

Figure 2. Result of decoding a severely degraded text line image: **(a)** illegible corrupted text line ($\sigma = 100$); **(b)** Viterbi without a language model; **(c)** Viterbi with a unigram language model; **(d)** Stack-1.0 (see text for an explanation) with a 5-gram; **(e)** Stack-2.0 with a 5-gram; and **(f)** ICP with a 5-gram. The apostrophe in "couldn't" is removed in a preprocessing step (see text).

We now discuss our simulation of the Stack algorithm. For each level σ of noise considered, the expected slope $H(\sigma)$ of the path score as a function of pixel position is estimated as the sum of a per-pixel-column language model component and a per-pixel-column likelihood component. The former is based on an empirical entropy rate obtained by scoring the training data with the language model and accounting for the variable widths of the characters. The latter is calculated on the basis of the noise process and the marginal distribution of uncorrupted pixel values. Figure 3 indicates the sensitivity of the space-complexity of the algorithm to the choice of tilt function slope, as well as a large difference in space-complexity for two different lines at a fixed slope value, suggesting a generally large variability of space-complexity from line to line for a given slope value.

In the experiments that compare the Stack algorithm to ICP and Viterbi, the tilt function is given in terms of the expected slope $H(\sigma)$. Specifically, the slope is set at $\alpha H(\sigma)$, where α ranges over the values 1.0, 1.5, and 2.0. The

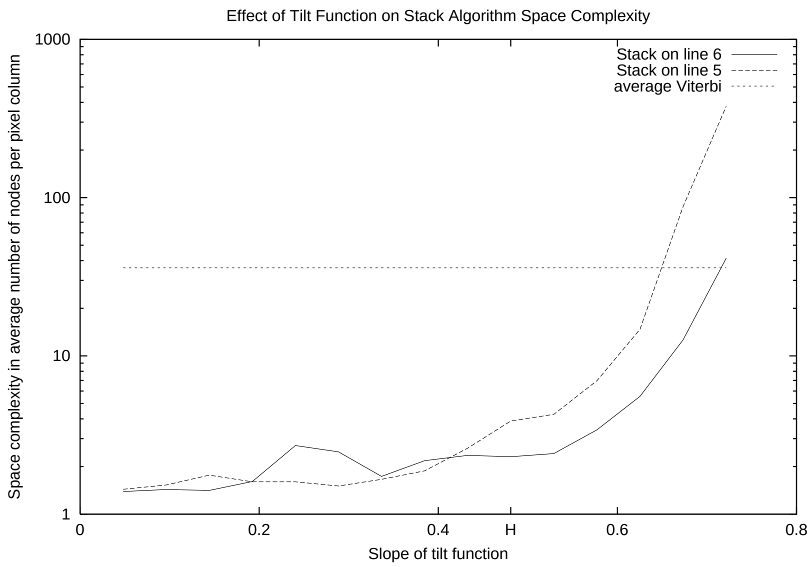

Figure 3. Space-complexity of the Stack algorithm measured in graph nodes per horizontal pixel position, as a function of tilt slope, for two different text lines both corrupted with the same noise level $\sigma = 75$. The "ideal" slope predicted on the basis of empirical entropy rate is indicated by H. Also shown, for reference, is the space-complexity of the standard Viterbi algorithm on the basic DID trellis (i.e., without a language model).

Stack algorithm using a particular value of α is labeled *Stack-*α in the graphs. To prevent a computational explosion for the occasional text lines for which the specified slope turns out to be too high, a strict upper limit of 10^6 nodes is imposed on the size of the graph for any one text line. When that limit is exceeded, then the slope is reduced in steps of $H(\sigma)/10$ until the limit is obeyed.

The basic ICP algorithm has no parameters besides the language model itself, and its implementation directly follows the description given in Section 3.3. We have equipped the algorithm with a mechanism for early stopping so that the best path found prior to termination can be recorded and returned, but we have not had occasion to use early stopping in the experiments reported on here. It is mentioned because this capability can be important in practice.

Figures 4-7 characterize the performance of the various algorithms as a function of noise level on the first 50 lines of the test data, with the exception that for ICP at $\sigma = 100$, only the first 32 lines are used. Figure 4 presents error rates, calculated as the minimum average per-character number of substitutions, insertions, and deletions, each weighted equally, that must be performed to transform the recognized text into the original. The error-rate performance of ICP stands out; it is consistently and significantly better than the others.

The corresponding space-complexity of each algorithm as a function of noise level is illustrated in Figure 5. The extremely low complexity of Stack-1.0 makes it attractive in very low-noise situations, while higher noise levels are likely to require an adaptive tilt-slope selection procedure to achieve good performance with low complexity. Note that the ICP algorithm grows the trellis very little beyond what is already required for Viterbi; the excess is noticeable only at the highest noise levels. The time-complexity of ICP versus Viterbi is harder to gauge. ICP requires evaluating at least N complete paths versus only one for Viterbi. This number grows with noise level as shown in Figure 6, gradually for typical noise levels and more quickly as the level of corruption becomes extreme. However, the additional path evaluations required by ICP reuse the likelihood component of the edge scores, which can be memoized. On the other hand, the language model components of the scores need to be computed for each pass, and the cost of doing so depends on the details of the language model. A careful analysis of ICP time-complexity would involve the details of how the language model is evaluated, particularly the bound functions; for

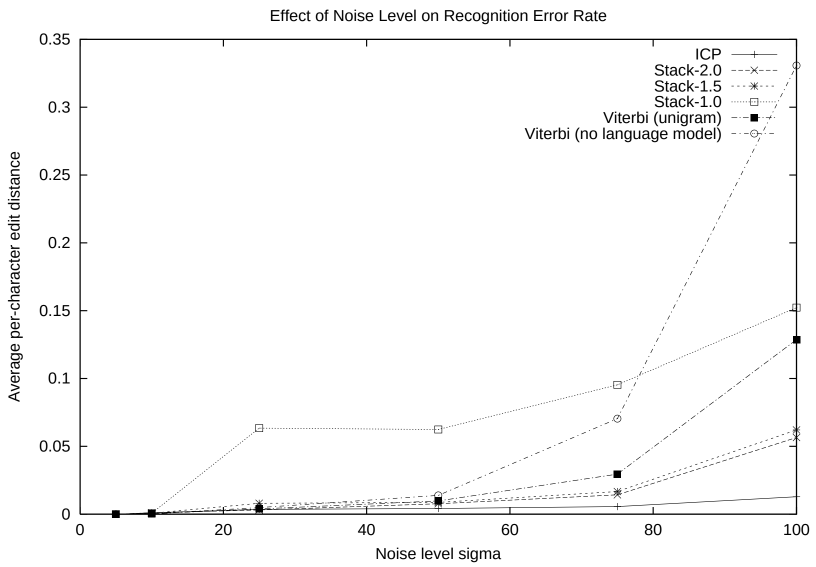

Figure 4. Error rates for the various algorithms, measured in edit distance normalized by the length of the text, using unit costs for insertions, deletions, and substitutions.

now we simply observe that the time-complexity of each additional scoring pass in ICP is variable.

For both the Viterbi and Stack algorithms, time-complexity is directly proportional to space-complexity. Relative to the Viterbi algorithm, the Stack algorithm saves a factor of $|\mathcal{A}|$ per node in time-complexity if we assume the cost of scoring an edge is the same in both cases, because in Stack the outgoing edges are scored only when a node is taken off the priority queue and expanded to $|\mathcal{A}|$ new nodes. In practice, the cost of scoring an edge is slightly higher in Stack than in Viterbi because of the presence of a language model, and a detailed analysis of time-complexity (not attempted here) would have to take this difference into account.

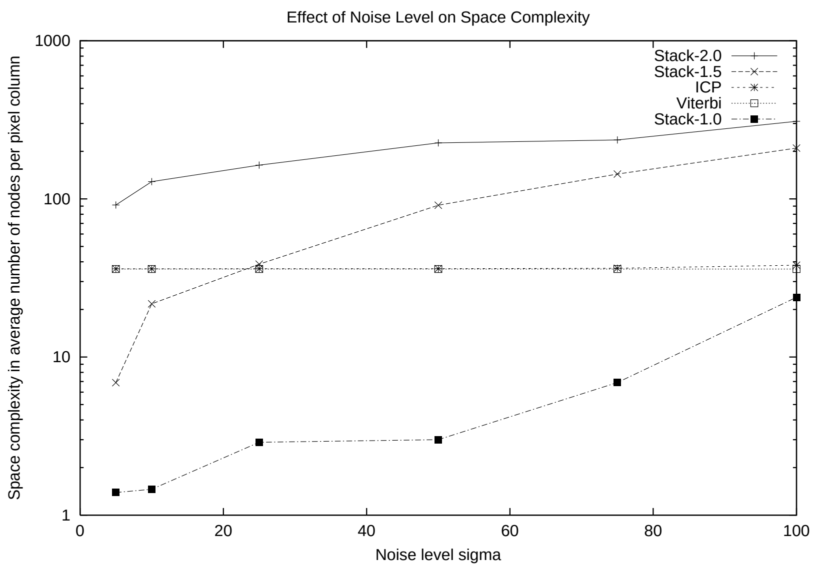

Figure 5. space-complexity as a function of noise level for the various algorithms.

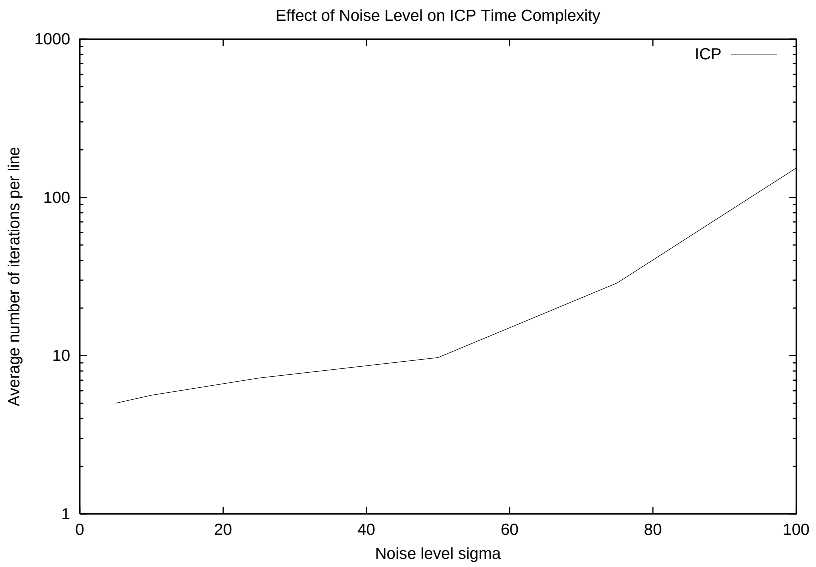

Figure 6. Number of ICP iterations required as a function of noise level.

Finally, Figure 7 characterizes the performance of the algorithms in terms of what they are actually trying to optimize: the average path score (shown relative to the score of ground truth, and normalized by path length). Note that ICP scores slightly better than ground truth at the highest noise level. This is because at such a high noise level, ground-truth ceases to be a highest-score path. Had a sufficiently steep tilt function been used with Stack, it too would have found a highest-score path.

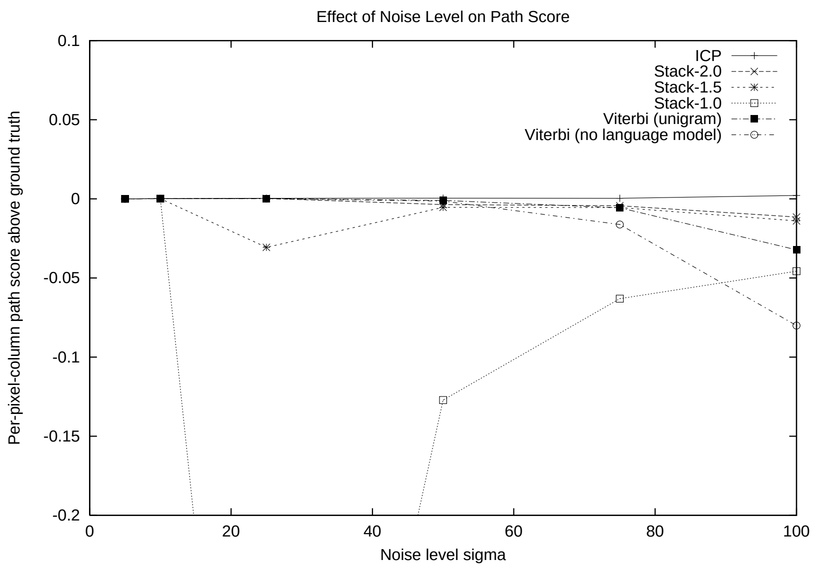

Figure 7. Average edge scores along paths found by the algorithms discussed in the text, relative to the average edge scores for ground truth.

5. CONCLUSION

We have discussed the problem of integrating soft linguistic constraints directly into the best-path search procedure in Document Image Decoding, and have identified and compared two algorithms for this purpose. The first is the ICP algorithm, a bound-based iterative technique that is guaranteed to yield a truly best path but which can be computationally expensive in situations where the document image has been severely corrupted. The second is the Stack algorithm, an estimation-based technique which is computationally attractive but which does not guarantee that a path with the highest score will be found. An additional difference is that the Stack algorithm requires that a tilt function be specified, and the functioning of Stack depends critically on it. The basic ICP algorithm has no parameters aside from the language model itself, but it does require that the language model be amenable to computing bounding functions of a particular form. We have also observed that the ICP algorithm can be employed as an approximate algorithm via early stopping, though we have not explored that aspect in detail. The experimental results presented were based on a simplified simulation of DID which used a restricted alphabet and in which no provision was made for fine variable spacing between character templates. Removing these simplifications to improve realism in further simulations is the subject of ongoing work.

ACKNOWLEDGMENTS

This work has benefitted from discussions with Henry Baird, Thorsten Brants, Tom Breuel, Francine Chen, Gary Kopec, Tom Minka, and Les Niles.

REFERENCES

1. G. E. Kopec, "Multilevel character templates for document image decoding," in *Proceedings of the IS&T/SPIE 1997 Intl. Symposium on Electronic Imaging: Science & Technology*, (San Jose), February 1997.
2. G. E. Kopec and P. A. Chou, "Document image decoding using Markov source models," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**, pp. 602–617, June 1994.
3. K. Popat, D. Bloomberg, and D. Greene, "Adding linguistic constraints to document image decoding," in *Proceedings of the 4th international workshop on document analysis systems*, International Association of Pattern Recognition, December 2000.
4. M. Stefik, *Introduction to Knowledge Systems*, Morgan Kaufmann, San Francisco, 1995.
5. A. C. Kam and G. E. Kopec, "Document image decoding by heuristic search," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18**, pp. 945–950, September 1996.
6. G. D. Forney, Jr., "The Viterbi algorithm," *Proceedings of the IEEE* **61**, pp. 268–278, March 1973.
7. W. N. Francis and H. Kucera, *Brown corpus maunal: manual of information to accompany a standard corpus of present-day edited American English, for use with digital computers*, Brown University, Providence, Rhode Island, 1964.
8. R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*, IEEE Press, 1999.
9. F. Jelinek, *Statistical Methods for Speech Recognition*, MIT Press, Cambridge, Massachusetts, 1997.
10. R. Schwartz and S. Austin, "A comparison of several approximate algorithms for finding multiple (n-best) sentence hypotheses," in *ICASSP 91: 1991 International Conference on Acoustics, Speech and Signal Processing*, vol. 1, pp. 701–704, IEEE, (Toronto, Ont., Canada), May 1991.
11. M. R. Said, "Unpublished work." Xerox Palo Alto Research Center, CA, August 1997.
12. L. Carroll, *Through the Looking Glass*, Project Gutenberg, Milennium Fulcrum ed., 1991.