

# Adaptive Stack Algorithm in Document Image Decoding

Kris Popat

Palo Alto Research Center

3333 Coyote Hill Road

Palo Alto, California, USA

{popat,greene}@parc.xerox.com

Daniel H. Greene

Tze-Lei Poo

Dept. of Electrical Engineering

Stanford University

Stanford, California, USA

leipoo@stanford.edu

## Abstract

*The Stack algorithm, which is a best-first search algorithm widely used in speech recognition, is modified for application to the problem of recognizing machine printed text in the Document Image Decoding (DID) framework. An iterative scheme is described wherein successively more stringent Stack searches are performed, each time using a model of the image that is updated on the basis of what was discovered on the previous iteration. In this way, the algorithm can adapt to realistic degradation patterns that are irregular and perhaps not well described by stationary models. The contribution of this work is twofold: (1) it represents a reliable method of estimating suitable parameter values for Stack decoding in DID, and (2) as a means of handling nonstationary degradation, it presents an alternative to another recently developed approach that is described elsewhere, the Iterated Complete Path algorithm, at potentially lower computational cost. Preliminary results are presented on text line images with simulated nonstationary noise.*

## 1 Introduction

Text is recognized in Document Image Decoding [3] by finding a character string that “best explains” the observed image. Specifically, a text document image is regarded as having been produced by a Markov imaging source that prints characters sequentially, choosing each character according to a probability distribution conditioned on previous characters (a *language model*), and changing the printing location for each character by a vector *set width* that depends on the previous character. The result is then regarded as having been subjected to a degradation process that is described by a random (or partially random) *channel*. This results in the observed document image. The channel specifies the probability, for each character and for each location in the image, that any specifiable configuration of pixel val-

ues will be observed. With the observed image regarded as fixed and the characters and positions as variable, the channel probability assignment turns into a likelihood function. The *posterior probability* of any sequence of characters imaged in a particular location on the page is the product of the *prior* probability assigned to it by the language model and the character likelihoods.

Within the DID framework, the task of recognizing text amounts to that of finding a sequence of characters that is most probable, in the sense that the paths through the Markov imaging source that are consistent with that sequence have the largest total posterior probability. It is common to make the simplifying *Viterbi* assumption [1], as we do here, that the probability of a sequence of characters is dominated by a single path through the Markov source. To simplify discussion and prototyping we make the additional simplifying assumption that the document image consists of a single text line, so that spatial position is one-dimensional (horizontal). This latter assumption is for convenience and does not limit the scope of the algorithm in principle.

To re-cast this problem into a domain for which search algorithms are well developed, it is useful to “unroll” possible paths through the Markov source against horizontal position. Any particular path can then be regarded as a sequence of transitions through a collection of nodes that represent state in the form of horizontal location and path history (linguistic context). The collection of all possible such paths constitutes a directed acyclic<sup>1</sup> graph, with a weight on each edge that corresponds to the sum of the log-likelihood and log-prior for the corresponding character, location, and linguistic context. Since the weights are logarithmic, the overall weight or *score* of any path is determined by simply summing the weights of the edges. Normally, the language model makes simplifying assumptions that induce equivalence classes among linguistic contexts, so that this graph collapses into something less than a full tree, but for the

---

<sup>1</sup>Avoiding cycles in the two-dimensional setting requires appropriate *scheduling* of nodes [3]; in one dimension it is sufficient that all characters have positive set width, which we assume.

moment it is instructive to imagine the graph of all possible paths without such simplification. In principle, a path with the highest posterior probability can be found by searching this graph exhaustively. In practice, this is not computationally feasible, even with the reduction in the size of the graph that results from the language model’s equivalence-partitioning of contexts. It is therefore generally necessary to content ourselves with searching exhaustively only a small fraction of the graph.

Depending on the method chosen, a partial search can result in either a provably optimum solution (using variants of *branch and bound* [7, 5]) or in a solution that only approximates an optimal one, but which may be obtained at possibly lower computational cost. The Stack algorithm [1, 2] is in the latter category. It begins by exploring (i.e., evaluating the partial score along) every edge originating in the start-node or *root*. At any time in the search, the portion of the graph that has been exhaustively explored is delineated by a *frontier* of nodes. The frontier is pushed successively deeper in the graph by exploring edges that originate from that node which is deemed currently most *promising*, taking into account both the known accumulated score from the root to the node, and an estimate of the score of the best continuation to an end-state. Hence the term “best-first” in the description of this search technique. When the frontier comes to include an end-state node, the path to that node is taken as the solution and the algorithm terminates. The estimate of the score of the best completion from any node plays a crucial role. If it is too pessimistic, then nodes are rewarded for being deeper in the graph, and the frontier grows in a *greedy* manner. In the *extremely greedy* regime, deeper nodes will always exceed expectation by virtue of their mere depth, resulting in their yet deeper successors being explored in a runaway condition that we term *lockout* (other, possibly better but less deeply explored paths are effectively “locked out” from consideration). At the other extreme, when the estimate is too optimistic, any explored edge will tend not to live up to expectations, effectively resulting in breadth-first, fully exhaustive search. This is termed the *explosive* regime, as the overall computational cost quickly becomes prohibitive as the estimated scores of the best path completions become increasingly optimistic.

In our experience, the main determining factor behind both the search efficacy and the computationally efficiency of the Stack algorithm when applied to DID is the choice of score estimation policy for path completions. The matter becomes even more critical when the image degradation is non-uniform; a non-adaptive estimation policy will almost certainly result in explosive behavior upon entry of the frontier into a region of comparatively high degradation. In Section 3 an iterative scheme is described wherein the score estimation policy is determined on the basis of information learned from the previous, and by design less stringent,

search run. While the approach described in that section constitutes the main subject of this paper, there are implementation issues which are crucial to successful application of the Stack algorithm to DID irrespective of path-score estimation or adaptation; these are summarized in Section 2. Section 4 presents results and concludes.

## 2 Stack Algorithm Implementation

A general description of the Stack algorithm is available elsewhere [1, 2]; here, we focus on those aspects of our implementation that are specific to DID.

The nodes in the portion of the graph explored by the Stack algorithm are represented explicitly; the edges need not be stored, as the nodes embody the state information necessary to recover the edges unambiguously. The “best-first” policy is realized by inserting nodes to be explored into a priority queue according to their “promise” in the sense given in Section 1, and the highest-priority node in this queue is the one selected for removal and exploration at each step.

Our implementation essentially treats the graph as if it were a tree, with one important qualification. A hashing function is defined to reflect the equivalence class induced by the language model; different (*position, linguistic context*) pairs are hashed to the same value if the positions are the same and if the linguistic contexts are treated as equivalent by the language model. When a newly explored node hashes to a value that was previously encountered, then one of three actions is taken. If the new node has less promise than the one previously encountered, then it is discarded. If it has greater promise and the previously encountered node is still in the queue, then the latter is replaced by the new node. Finally, if a more-promising hash-equivalent node has been encountered but has already been popped off the queue for subsequent exploration, then the new node is placed in a separate *redo* queue which, when non-empty, is always preferred over the main priority queue as the source of the next node to be explored. The effect is to ensure that successor nodes known to be worthy of exploration are not required to fight their way back to the top of the queue at potentially large computational expense. This approach to “duplicate exploration suppression” approximates the behavior of seemingly more straightforward solution: re-scoring the descendants of the previously encountered hash-equivalent node to reflect the improvement represented by the newly found node. Carrying this out would require adding an additional member (a forward pointer) to the node data structure, which would represent a significant cost in space complexity; therefore, the redo-queue approach is taken instead. This approach was developed and refined in earlier work in collaboration with Justin K. Romberg [6].

While advantageous in a general setting, the suppres-

sion of exploration of duplicate partial paths achieved by the foregoing strategy becomes absolutely indispensable when Stack is applied specifically to DID. The reason is the presence in the DID framework of a linguistically neutral pseudo-character that corresponds to a single-pixel space. The purpose of this pseudo-character is to provide for fine alignment of the characters in the hypothesis text against the image, effectively correcting possible errors in the specification of set-widths of the other characters. The presence of this pseudo-character also has the effect of greatly multiplying the number of distinct path segments that map to the same string, and if there were not a way of quickly pruning provably worse equivalent paths, the computational complexity would likely be prohibitive.

It is useful to note that by modifying the hash function to ignore linguistic context and consider only spatial position, while at the same time employing a deliberately optimistic score estimation function, the Stack search can be made to simulate the behavior of standard dynamic programming (often called the *Viterbi algorithm* in a communications systems setting). This approach was in fact used to produce some of the comparative results obtained in this study.

The implementation of the remaining major components of the system was carried out as reported elsewhere. Specifically, the language model was implemented as described in [5] (sans the upper-bound generating mechanism), while the likelihood model used was the grayscale formulation reported in [4].

### 3 Iterative Adaptation

It was remarked previously that the policy used in estimating the score of path completions plays a crucial role in determining the performance of Stack, and that the policy must be sensitive to variations in the inherent difficulty of recognition across regions of the document image. To emphasize the spatial variability and the strong dependence on the image-dependent character likelihoods, we refer to the setting of this policy as *adaptation*. Furthermore, noting that the expected score of path completion will depend primarily on the *length* of the path remaining, we restrict consideration to policies that represent the estimated score of path completion as an array indexed by position.

Two observations then lead to an iterative adaptation strategy. The first observation is that when the path-completion score estimates are pessimistic so that the Stack algorithm behaves in a greedy manner, the computational cost of obtaining a solution (albeit suboptimal) is a small fraction of the cost associated with finding a near-optimal solution by any known means. Therefore, if eventually an optimal or near-optimal solution is to be found, as we assume, then running the Stack in greedy mode as a preliminary step can be considered to contribute no significant ad-

ditional computational cost. The second important observation is that the cumulative likelihood score at a particular location along a suboptimal path found in this inexpensive way contains useful information about the corresponding score for an *optimal* path. Specifically, one can expect *at least* as much cumulative likelihood from the optimal path, and if we have reason to believe that the suboptimal path is any good, then we should expect *around* as much as well.<sup>2</sup> These two considerations lead to the following general adaptation strategy:

1. Initialize the expected score array to ensure greedy behavior (the precise initialization is not critical).
2. Run Stack to obtain a candidate path through the graph.
3. Use the observed cumulative likelihood scores obtained along this path to set the expected score array for the next iteration.
4. If the total path score has not changed or if a pre-set limit on allowable computational complexity has been reached, stop; otherwise, return to Step 2.

Several issues had to be addressed to make this scheme workable. First and simplest, the candidate path does not specify a cumulative likelihood at every pixel location, but rather only at those corresponding to nodes along the path. This may be satisfactorily remedied by interpolation; we have found that linear interpolation works well, as one might expect by assuming at least local stationarity of the noise and by considering the additivity of the log-likelihoods of the columns in the character templates.

A more serious problem is that the procedure as stated does not provide a means for estimating the language model component of the expected cumulative score. We considered several approaches: (1) use a static estimate based on an assumed entropy rate of English; (2) use a static estimate based on empirical cross-entropy of the trained language model applied to hold-out data; (3) use the language model scores of the candidate path in segments where they are high enough to be thought reliable (under the assumption that errorful text will have higher empirical cross-entropy than correct text) and interpolate elsewhere; and (4) use the language model scores of the candidate path, but choose an interpolation scheme that will ensure that the resulting expected scores will not lock out linguistically better paths on later iterations. Of these, we have so far had the greatest success with the third approach, although we consider the fourth to be the most promising in terms of robustness.

A final major consideration is the possibility of premature termination in a local optimum, which are in fact encountered when the algorithm is applied as described above.

<sup>2</sup>We have been careful here to speak only of the likelihood component of the score only — the statements do not hold for the language-model component.

Although we are not yet in a position to be definitive in our discussion, we have found an apparently reliable heuristic to avoid the problem: scale the entire interpolated expected score array in Step 3 of the adaptation procedure above by a factor slightly greater than one (in our experiments, 1.01). While remarkably reliable in practice, in terms of principle we find stronger arguments against such a heuristic than in support of it. Specifically, it is not invariant to the manner in which the likelihoods are normalized, which elsewhere is assumed to be non-critical [3, 4]. Still, the effectiveness of this simple heuristic in experiments indicates the local optima problem is not a serious one from a practical standpoint; search for a principled alternative is a subject of ongoing work.

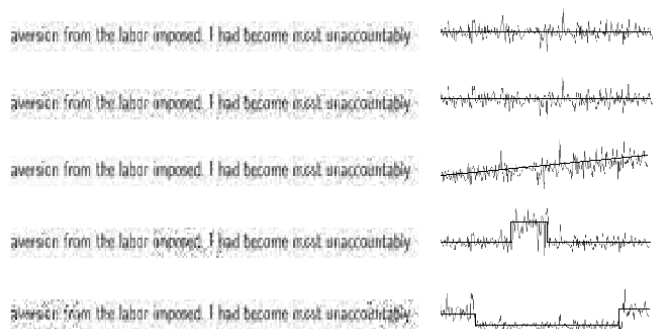
## 4 Results and Conclusions

The method of Section 3 was applied to twenty-one lines of text taken from *The Gold Bug* by Edgar Allan Poe, imaged in T<sub>E</sub>X's Computer Modern Sans Serif font at low resolution and in grayscale to simulate digital-camera acquisition, and corrupted by additive zero-mean Gaussian noise with horizontally varying standard deviation. Several standard-deviation profiles were used: a *ramp*, corresponding to a gradual noise-level change across the image as might result from an illumination gradient; a high-amplitude pulse against a low-amplitude background, simulating the effect of a *coffee* stain on the page; a low-noise segment against a high-noise background, which we have termed *fringe*; and finally two flat (stationary) profiles, one with *high* noise level and the other *medium*. These are illustrated in Figure 1 for one of the test lines.

The algorithm was allowed to iterate until convergence on each of the twenty-one test lines. The average resulting recognition accuracies for each of the noise profiles, weighing character insertions, deletions, and substitutions equally, were: .996 (medium), .987 (high), .862 (ramp), .954 (coffee), and 0.915 (fringe). Further testing using computationally more expensive search procedures indicated that these error rates are in fact the best attainable with the chosen channel and language models.

The significant finding is the low overall computational cost of the adaptive iterative scheme. Both running time and memory requirements are directly proportional to the number of nodes created, so it is natural to use as a complexity measure the ratio of this number to the size of a standard Viterbi algorithm lattice for the same text line. Averaged over the twenty-one lines and over all noise profiles, this relative complexity was found to be 0.205 — about one-fifth the cost of Viterbi. This is all the more significant when one notes that Stack integrates a language model while standard Viterbi does not.

Based on our findings we conclude that an iterative adap-



**Figure 1. A line from the test set imaged using noise profiles (from top to bottom): medium, high, ramp, coffee, fringe. On the right is the corresponding array of vertical projections with the actual standard-deviation profile superimposed, to more clearly indicate the noise pattern.**

tation strategy using multiple Stack decoding runs of increasing stringency can yield good decoding results at relatively low computational cost, even in the presence of non-stationary degradation. Further research is required to understand its merits relative to the Iterated Complete Path [5] approach, and to develop more principled strategies for initialization and stopping.

## References

- [1] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1998.
- [2] R. Johannesson and K. S. Zigangirov. *Fundamentals of Convolutional Coding*. IEEE Press, 1999.
- [3] G. E. Kopec and P. A. Chou. Document image decoding using Markov source models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):602–617, 1994.
- [4] K. Popat. Decoding of text lines in grayscale document images. In *Proceedings of the 2001 International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2001)*, Salt Lake City, Utah, May 2001. IEEE. IMDSP-L1.2.
- [5] 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.
- [6] J. K. Romberg, D. H. Greene, and K. Popat. Unpublished work. Xerox Palo Alto Research Center, 2000.
- [7] M. Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann, San Francisco, 1995.