Linear Dynamic Programming and the Training of Sequence Estimators with Application to Musical Harmonic Analysis

Christopher Raphael and Eric Nichols

Abstract We consider the problem of finding an optimal path through a trellis graph when the arc costs are linear functions of an unknown parameter. In this context we develop an algorithm, Linear Dynamic Programming (LDP), that simultaneously computes the optimal path for all values of the parameter. We show how the LDP algorithm can be used for supervised learning of the arc costs for a dynamic-programming-based sequence estimator by minimizing empirical risk. We present an application to musical harmonic analysis in which we optimize the performance of our estimator by seeking the parameter value generating the sequence best agreeing with hand-labeled data.

1 Introduction

Dynamic programming (DP) is a well-established technique for finding the optimal path through a trellis graph in which the score of the path is represented as a sum of arc scores traversed along the path. The history of DP goes back at least to Bellman, [1], though perhaps much further. In this work we introduce an extension of the DP algorithm, we call linear dynamic programming (LDP). LDP also addresses a situation in which we seek the best scoring path through a trellis. However, in the LDP case the arc scores are known linear functions of an unknown parameter. In this context, LDP finds the optimal path *simultaneously* for *all* values of the parameter. LDP mirrors regular DP by recursively constructing the score of the best possible path to each intermediate trellis node. The score of this path, as a function of the parameter, is shown to be the maximum of a finite collection of linear functions. This form can be carried through the DP iteration exactly. While meaningful complexity

Christopher Raphael
School of Informatics, Indiana Univ., e-mail: craphael@indiana.edu

Eric Nichols
School of Informatics, Indiana Univ., e-mail: epnichols@gmail.com

analysis remains open, we demonstrate the feasibility of this approach in the domain of musical harmonic analysis.

While we find the LDP formulation interesting in its own right, we developed the approach with a specific aim: LDP serves as an alternative to maximum likelihood parameter estimation methods in sequence estimation problems using hidden Markov models (HMMs). One vexing aspect of the HMM training algorithms, for both labeled and unlabeled data, is the focus on data likelihood, rather than a criterion of more direct interest, such as recognition performance. LDP can be used to directly optimize recognition performance on a training set. This direct approach is embraced by a host of other machine learning algorithms, though we extend the approach to sequence estimators.

The algorithm we use to perform the LDP iteration is a close cousin to *value iteration* in partially observable Markov decision processes (POMDPs) [2], [3], [4], [5], [6], though our problem formulation seems to have little in common with POMDPs. We expect that the wealth of knowledge concerning POMDP solvers has much to contribute to our LDP approach, though, at present, we have not yet exploited this connection. This work demonstrates that the algorithmic approaches of POMDPs find application in a more general setting.

We demonstrate our training approach with an application to musical harmonic analysis. In this domain we associate a harmonic label, such as chord and key, to each measure or beat of the music, while seeking the optimal harmonic labeling of the music using DP. Our notion of optimality considers both agreement between our harmonic sequence and the observable data, as well as prior knowledge concerning harmonic sequences. We parameterize our DP trellis graph so that each arc score is a linear function of an unknown parameter that weights the contributions of several relevant features. We choose the parameter by finding the value that gives optimal performance on a labeled training set, as well as presenting the performance on separate test data.

2 Linear Dynamic Programming

2.1 Traditional Dynamic Programming

Suppose we have a trellis graph with finite set of nodes, S, as depicted in Figure 1. By "trellis", we mean that every node, $s \in S$, has an associated level, $l(s) \in \{1,\ldots,N\}$, while the arcs of the graph, $A \subset S \times S$, only connect nodes at adjacent levels:

$$A \subseteq \{(s,t) : l(t) = l(s)+1\}$$

More general definitions are possible. A path through the trellis is a sequence $(s_1,\ldots,s_n)$ such that $l(s_1) = 1$ and $(s_m, s_{m+1}) \in A$ for $m = 1,\ldots,n-1$. We will call a path $(s_1,\ldots,s_n)$ a complete path if $n = N$. We define the score of a path

as $c(s_1,\ldots,s_n) = \sum_{m=1}^{n-1} c(s_m, s_{m+1})$, where $c(s,t)$ is some fixed score for each arc $(s,t) \in A$.

Dynamic programming (DP) seeks a complete path $s_1^*,\ldots,s_N^*$ having *maximal* score. We denote the optimal score to a node s_n at level n as

$$c^*(s_n) = \max_{s_1,\ldots,s_{n-1}} c(s_1,\ldots,s_{n-1},s_n)$$

where the maximum is over all paths ending in s_n. The well known Viterbi algorithm [7], with roots going at least as far back as [1], defines $c^*(s) = 0$ for states with $l(s) = 1$ and computes the function, c^*, recursively as

$$\begin{aligned} c^*(t) &= \max_{s \in \text{Pred}(t)} c^*(s) + c(s,t) \\ a(t) &= \arg \max_{s \in \text{Pred}(t)} c^*(s) + c(s,t) \end{aligned} \tag{1}$$

for $n = 2,\ldots,N$, where $\text{Pred}(t) = \{s \in S : (s,t) \in A\}$. We choose arbitrarily from the optimal predecessor when the arg max is not unique. An optimal path $s_1^*, s_2^*, \ldots, s_n^*$ to any state, s_n^*, at level n is then recovered by defining $s_m^* = a(s_{m+1}^*)$ for $m = n-1,\ldots,1$. A globally optimal complete path can be found by tracing back an optimally scoring state at level N,

$$s_N^* = \arg \max_{s:l(s)=N} c^*(s)$$

2.2 An Extension to Simultaneous Computation

Now suppose that the arc scores depend on some parameter $\theta \in \Re^D$ through $c_\theta(s,t) = \theta^t \beta(s,t) + \beta_0(s,t)$ where $\beta(s,t) \in \Re^D$ and $\beta_0(s,t) \in \Re$ are known. We

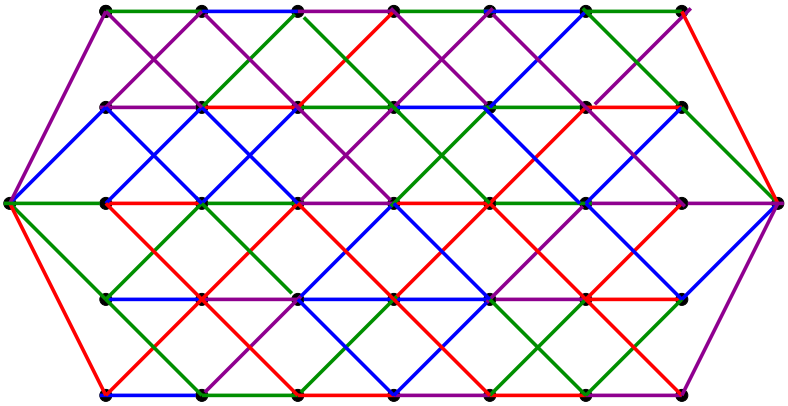

Fig. 1 A dynamic programming trellis structure.

now consider computing the optimal path through the DP recursion of Eqn. 1, *simultaneously* for *all* values of the parameter θ.

The key observation is the following. Note that the score of any particular path $s_1,\ldots,s_n$, viewed as a function of θ, given by

$$c_\theta(s_1,\ldots,s_n)=\sum_{m=1}^{n-1}\theta^t\beta(s_m,s_{m+1})+\beta_0(s_m,s_{m+1})$$

is clearly affine in θ. Thus the score of the *optimal* path to s_n, also viewed as a function of θ, is a maximum of affine functions

$$c^*_\theta(s_n)=\max_{s_1,\ldots,s_{n-1}} c_\theta(s_1,\ldots,s_n) \tag{2}$$

where the maximum is taken over all paths $s_1,\ldots,s_{n-1}$ ending in s_n. The number of paths in the maximization of Eqn. 2 grows exponentially in n, so such a representation will not be useful from an algorithmic perspective. However, many paths may be suboptimal for *all* values of θ; for such paths all descendant paths will also be suboptimal for all θ and can be eliminated from consideration.

To this end, define the *surviving* paths to be

$$B(s_n)=\bigcup_{\theta\in\Re^D}\{(s_1,\ldots,s_n):c_\theta(s_1,\ldots,s_n)=c^*_\theta(s_n)\}$$

These are the paths ending in s_n that are optimal for *some* value of θ. Then we have

$$c^*_\theta(s_n)=\max_{(s_1,\ldots,s_n)\in B(s_n)} c_\theta(s_1,\ldots,s_n) \tag{3}$$

since the discarded paths contribute nothing to the maximum of Eqn. 2. The paths in $B(s_n)$ are those that *could* be prefixes of an optimal complete path for some θ.

The function of θ, $c^*_\theta(s_n)$, is rather interesting geometrically. First of all, as a maximum of affine functions, $c^*_\theta(s_n)$ must be convex. For each path $(s_1,\ldots,s_n)$, define the associated region of optimality to be

$$R(s_1,\ldots,s_n)=\{\theta:c^*_\theta(s_n)=c_\theta(s_1,\ldots,s_n)\}$$

Thus $R(s_1,\ldots,s_n)$ is non-empty if and only if $(s_1,\ldots,s_n)\in B(s_n)$. Each nonempty region, $R(s_1,\ldots,s_n)$, can be shown to be a simplex, and on such regions $c^*_\theta(s_n)$ is, by definition, affine in θ. Figure 2 depicts a possible configuration of the regions, $R(s_1,\ldots,s_n)$, for a two-dimensional parameter space, $\theta=(\theta_1,\theta_2)$. On each region R_k of the figure, $c^*_\theta(s_n)$ is affine in θ. The affine functions associated with two neighboring regions are equal along the segment that separates the regions.

The essential computation of our linear dynamic programming (LDP) algorithm is to compute the sets $\{B(s)\}_{l(s)=n+1}$ recursively from the sets $\{B(s)\}_{l(s)=n}$, as follows. Since, for any fixed θ, an optimal path at level $n+1$ must be an extension of some optimal path at level n, we know that

$$B(t) \subseteq \tilde{B}(t) \stackrel{\text{def}}{=} \bigcup_{(s,t)\in A} B(s) \circ t$$

where by $B(s) \circ t$ we mean extending the paths in $B(s)$ by t. We will obtain $B(t)$ from $\tilde{B}(t)$ by removing any superfluous paths, $(s_1,\ldots,s,t)$ whose score, $c_\theta(s_1,\ldots,s,t)$ is suboptimal for all θ. That is, $B(t)$ is the *smallest* subset of $\tilde{B}(t)$ having

$$\max_{\pi\in B(t)} c_\theta(\pi) = \max_{\pi\in\tilde{B}(t)} c_\theta(\pi) \tag{4}$$

This "filter" computation, which allows us to compute $B(t)$ from $\tilde{B}(t)$, is the subject of a good deal of research in the POMDP community [2], [3], [4], [5], [6], as it forms the computational workhorse for value iteration techniques. There are many techniques for performing filtering, though the search for computationally attractive approaches is a source of ongoing research in POMDPs. We will not not discuss filtering techniques here in any detail. However, A popular approach due to White [6] iteratively constructs $B(t)$ by comparing each new affine function of $\tilde{B}(t)$ with a set of current "survivors" already shown to be somewhere optimal. By solving

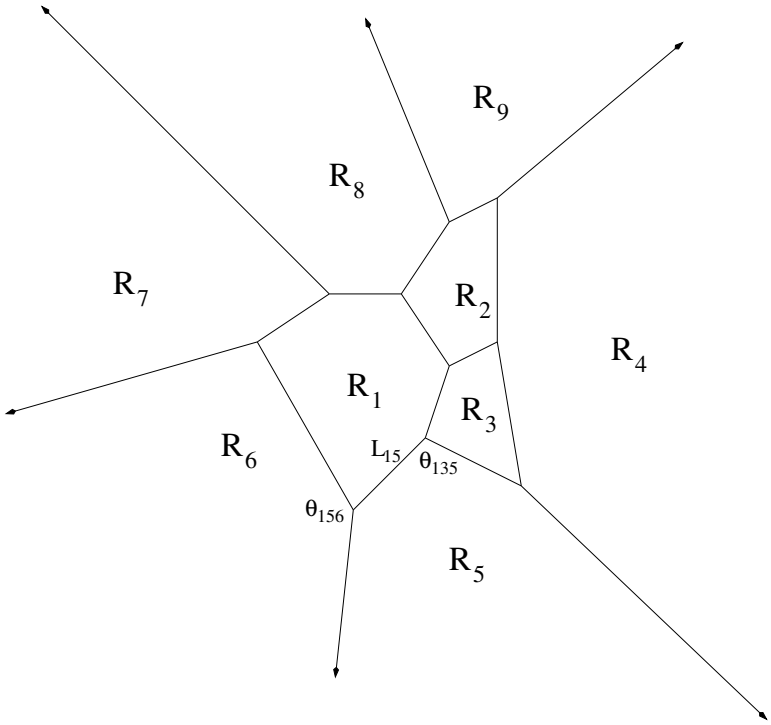

Fig. 2 $c^*_\theta(s_n)$ viewed as a function of θ. The $R(s_1,\ldots,s_n)$ regions for various paths are the simplices labeled as R_k in the figure.

a linear program, the new function can be shown either to not be in $B(t)$, or the algorithm identifies a new member of $\tilde{B}(t)$ that must be in $B(t)$.

The LDP algorithm constructs a search tree of possible paths. In the tree, a path at depth n corresponds to a path through the first n levels of the trellis. Such a tree is depicted in Fig. 3 for a trellis having only two states per level, labeled 0 and 1, thus two children for each nonterminal node. Each surviving path $s_1,\ldots,s_n$ in the search tree has an associated set, $R(s_1,\ldots,s_n)$, of parameter values, θ, for which the path is optimal (with respect to other paths ending in s_n). Thus, unlike in the regular DP computation, we may have many surviving paths ending in state s_n — each optimal for a different range of parameter values. The task of the filter operation is to determine which sets, $R(s_1,\ldots,s_n)$, are empty since these paths need not be considered further in the search tree. Along a particular path $s_1,s_2,\ldots$, we have

$$R(s_1) \supseteq R(s_1,s_2) \supseteq R(s_1,s_2,s_3)\ldots$$

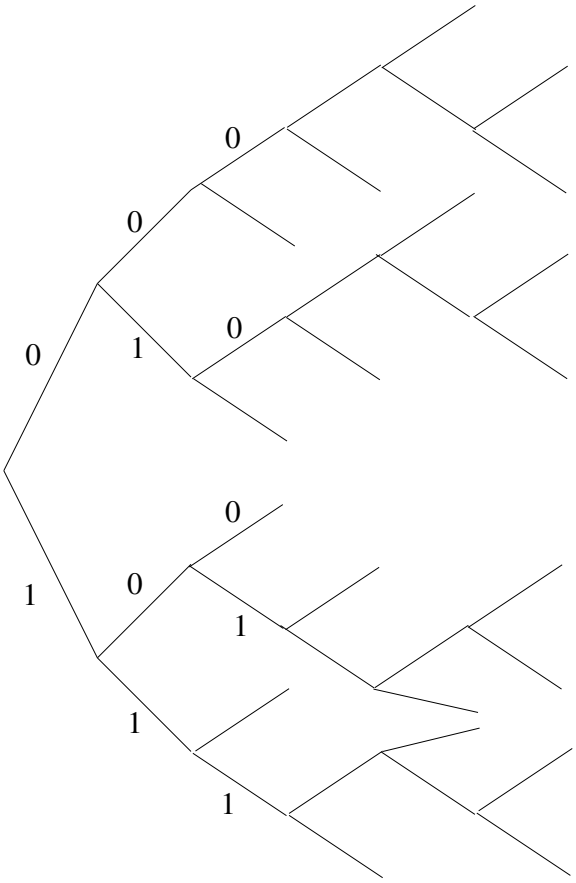

Fig. 3 The search tree of potentially optimal paths generated by the LDP algorithm for a trellis graph having only two states, 0 and 1, for each level. A terminal node in the graph indicates a path $s_1,\ldots,s_n$ for which $R(s_1,\ldots,s_n)$ is empty. Such a path requires no exploration of its children in the graph.

This follows since, by the basic reasoning of DP, if $(s_1,\ldots,s_n)$ is an optimal path for some fixed θ, then so is $(s_1,\ldots,s_{n-1})$. That is, if $\theta \in R(s_1,\ldots,s_n)$ then $\theta \in R(s_1,\ldots,s_{n-1})$

The LDP algorithm constructs this search tree level by level, generating children for each tree node at level n whose corresponding path, $(s_1,\ldots,s_n)$, has nonempty $R(s_1,\ldots,s_n)$. If computationally necessary, perhaps additional nodes (paths) are pruned. The paths that reach the final trellis node are the paths that are optimal for some value of θ.

3 Linear Dynamic Programming for Training in Sequence Estimation

The hidden Markov model (HMM) has proved to be a powerful and flexible approach for analyzing data sequences, with successes in many application domains including speech recognition [8], gesture recognition [9], handwriting recognition [10], various applications in Bioinformatics, e.g. [11], musical score following [12], and many others. In the HMM, recognition is often accomplished using dynamic programming (DP) to find the mostly likely sequence of hidden states given the observed data. This corresponds to finding the best scoring path through the state space trellis. One of the most attractive aspects of the HMM is automatic training procedures for estimating model parameters. However, a possible weakness of these training methods is their focus on a criterion not directly related to recognition performance. That is, HMM training algorithms such as the Baum-Welch algorithm, for unlabeled data, and straightforward empirical probability, for labeled data, optimize the *data likelihood* rather than a quantity, such as *recognition error rate*, that explicitly measures the quality of labellings produced by the recognition algorithm. In this section we show how LDP can serve as a reasonable alternative to maximum likelihood techniques for parameter estimation in sequence estimation problems.

Suppose we observe a sequence of N data observations and wish to estimate a corresponding sequence of labels that explain the data. A time-honored approach builds a $|S| \times N$ trellis graph where S is our collection of possible labels or states. Labeling of the data can be accomplished by assigning arc scores in a reasonable manner and computing the trellis path giving the best score through DP. Of course, the resulting sequence strongly depends on the choice arc scores. At a minimum the arc scores should encourage a reasonable "vertical" correspondence between the labels and associated data observations. For instance, some data observations are likely to occur under some states, so arcs leading to these states should receive high scores. In addition, the arc scores can be chosen to prefer *a priori* desirable label sequences over less plausible ones, perhaps even enforcing certain "horizontal" constraints on the label sequence.

Now we introduce a way of *learning* the arc scores automatically. Suppose that our arc scores are each known linear functions of an unknown parameter, θ. We will choose θ as the value whose associated DP state sequence most closely matches a

ground truth sequence. This value of θ can be found with LDP. Thus we address a problem of *supervised* learning. We emphasize that we are *not* trying to optimize the DP score over θ — this problem is unbounded if θ is unrestricted. Rather, we seek θ giving the best agreement between its associated DP state sequence and the ground truth. Thus, our approach *directly* optimizes a criterion we care about, such as the number of recognition errors we commit. Our optimality criterion, however, can be anything we choose — thus we may optimize a loss function incorporating a more nuanced assessment of the "badness" of different errors than does the 0-1 loss function. We will present such a loss function in the next section. In contrast, traditional HMM training techniques optimize the data likelihood, which is not of direct relevance to recognition performance.

Our approach does not address the issue of *generalization error* in any meaningful sense, since we simply optimize the unregularized performance on a training set. Thus we hope that the training set is large enough that we do not over-fit during this process.

The approach of optimizing performance on a training set is certainly an old and well-established one in the machine learning community. What is novel here is our LDP method for optimizing training performance over a family of *sequence* estimators. We know of no other work that seeks to directly optimize the performance of a sequence estimator on a training set.

4 Application to Harmonic Analysis

The problem of *functional harmonic analysis* seeks to partition a piece of music into labeled sections, the labels giving the local *harmonic state*. The label usually consists of a key, e.g. C Major or G minor, and a chord symbol such as I, ii, iii, IV, V, vi, vii for the triads built on the scale degrees indicated by the roman numerals. For instance, the label (A major, IV) corresponds to the triad built on the fourth scale degree, (D,F♯,A), in the key of A major. Since harmony represents a significant part of what listeners respond to in music, its analysis is fundamental to a host of musical applications including expressive rendering, improvisational accompaniment systems, and may also constitute a one-dimensional reduction of music suitable for some search and retrieval applications. Several efforts have addressed this problem of automatic harmonic analysis, including [13], [14], [15]. While the problem holds promise for a wide range of musical applications, the evaluation of such work remains difficult, primarily due to the scarcity of ground truth data as well as a suitable evaluation metric (not all errors are equally bad).

Our recognition approach uses DP to find the best scoring path through the lattice composed by an $S \times N$ array of states, where N is the number of measures or beats in the piece and S is the number of possible harmonic labels we consider. Our score function consists of two components: a data score and a path score. The data score encourages close agreement between each measure label and the pitches of that measure. The path score rewards paths that are more musically plausible, inde-

pendent of the data. Our recognized sequence is then computed as the lattice path that minimizes the sum of these scores. We focus here on the problem of *learning* the data and path scores in a way that optimizes the path quality using a hand-labeled training set.

More explicitly, our score function, $C_\theta(s_1 \ldots, s_N)$, is composed as

$$C_\theta(s_1^N) = D_\theta(s_1, \ldots, s_N) + P_\theta(s_1, \ldots, s_N) \tag{5}$$

where the path, $(s_1, \ldots, s_N)$ is a sequence of labels, one for each measure. The data score, $D_\theta(s_1, \ldots, s_N) = \sum_{n=1}^{N} d_\theta(s_n, x_n)$, is represented as

$$d_\theta(s_n, x_n) = \sum_{i=1}^{3} \sum_{j=1}^{4} \theta_{ij}^d \delta_{ij}(s_n, x_n) \tag{6}$$

where x_n is the collection of pitches in the nth measure and the counts, $\delta_{ij}(s_n, x_n)$, are as follows. Each harmonic label, s_n, divides the possible chromatic pitches into four categories: those that are

1. the root of the chord
2. in the chord but not the root
3. in the scale but not the chord
4. outside the scale

Similarly, the pitches in a measure are divided into those that begin

1. on the downbeat of the measure
2. on a beat but not the downbeat
3. elsewhere in the measure

In Eqn. 6, $\delta_{ij}(s_n, x_n)$ counts the notes in the nth measure that are in position category i and of chromatic type j. To avoid degeneracies in which families of parameter assignments correspond to essentially identical choices, we further assume $\sum_j \theta_{ij}^d = 0$, thus reducing the effective length of the parameter θ^d.

The path score $P_\theta(s_1^N) = \sum_{n=1}^{N-1} p_\theta(s_n, s_{n+1})$ is the sum of modulation and progression components:

$$p_\theta(s_n, s_{n+1}) = \theta^m M(s_n, s_{n+1}) + \sum_{i=1}^{3} \theta_i^h H_i(s_n, s_{n+1})$$

where $M(s_n, s_{n+1})$ is an indicator function for key change. If $M(s_n, s_{n+1}) = 1$, the $\{H_i\}$ terms act as indicators for various classes of harmonic motion such as progressive and regressive. As above, we assume $\sum_i \theta_i^h = 0$. In total, considering the linear constraints, our parameter θ has dimension 12.

The LDP algorithm terminates with a collection of paths that are each optimal on a particular region of θ values. We perform training by selecting a value for θ whose associated path is the best of the surviving paths, according to some measure of goodness. This measure of goodness is *entirely distinct* from the score function

described above, and can be chosen arbitrarily. In the case of harmonic analysis, we believe some errors are worse than others and should be penalized accordingly. For instance, the difference between the ii chord and the IV chord may be subtle and subjective in some cases. We address this issue by assigning a penalty for recognizing the true chord (from hand-labeled ground truth) with a possibly different chord as the number of pitch classes in the symmetric difference between the two chords. Thus there is no cost for getting a chord correct and cost of 2 for confusing chords ii and IV, since each has one pitch class not contained in the other. Similarly, our cost for the key attribute is formed by counting the number of pitch classes in the symmetric difference of the two scales. These two attributes are weighted equally and summed over the entire analysis sequence to form our goodness function for comparing paths.

Rather than trying to optimize *simultaneously* over our 12-dimensional parameter space, instead we have successively optimized over the various one-dimensional components of θ, using a simple one-dimensional implementation of the LDP algorithm. While it is clearly preferable to perform simultaneous optimization, we focus here on the essential idea of using LDP training a sequence recognizer. However, the simultaneous optimization problem remains a source of ongoing research for us, with potential to draw on the relevant POMDP literature. The one-dimensional filtering problem is quite simple since, given a collection of one-dimensional linear functions, we only need discover which are maximal over some interval. It is straightforward to solve this one-dimensional problem in a computationally efficient manner, though we omit the details here.

Our first experiments involved performing the LDP algorithm with no pruning. In this case we observed a steady increase in the number of surviving paths at each stage of the trellis, though far less than the exponentially growing number of paths present if no filtering is performed. Figure 4 shows how the number of surviving paths grows as a function of trellis level over several one-dimensional iterations of our algorithm. This demonstrates that the overwhelming majority of paths are

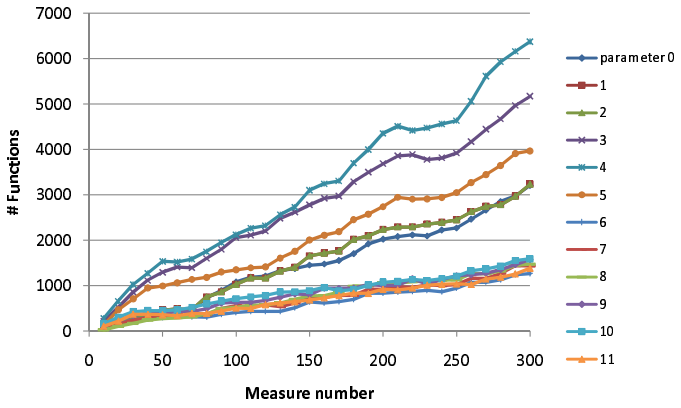

Fig. 4 Number of linear functions for each iteration of the algorithm.

pruned, but also suggests difficulties with the scalability of the basic algorithm. We expect that some pruning, with the possibility of losing potentially optimal paths, will be necessary in more ambitious problems.

In the remaining experiments we used techniques analogous to "beam search" to decrease the number of paths that are propagated through the algorithm. In this case, as we progress through the trellis, we narrow the considered range of each particular component, θ_k, according to our path quality metric, to focus on what *seems* to be the right region in parameter space. In effect, we "tunnel in" on the best region in the one-dimensional parameter space under consideration. Figure 5 shows our symmetric difference "goodness" measure for the surviving paths after a typical iteration of the algorithm for this beam search version. Unlike the $c^*_\theta(s_n)$ function, which is convex in θ, our "goodness" measure can have arbitrary dependence on θ.

Using this procedure, we trained our algorithm on the *Grande Valse Brilliante* of Chopin using hand-labeled ground truth. In this experiment we limited the set of possible harmonic labels to the 27 (key,chord) pairs encountered in the piece. Note that, due to the nature of our data and path scores in Eqn. 5, parameter values trained from a restricted set of harmonic labels still can still be applied to recognition problems using a different set of labels. Using this LDP implementation we were able to improve our path quality measure from 2203 to 173, starting with a random initial configuration for θ. The resulting configuration after our optimization terminates corresponded to a total summed symmetric difference (SSD) of 73 between the recognized chord pitch classes and the ground truth chord pitch classes, as well as an SSD of 100 between the two scale sequences. This corresponds to 0.24 chord errors and 0.33 scale errors per measure.

We then applied this learned parameter to the Chopin *Petit Chien* (the "Minute Waltz") using a collection of 14 labels. This resulted in an SSD of 186 chord pitch

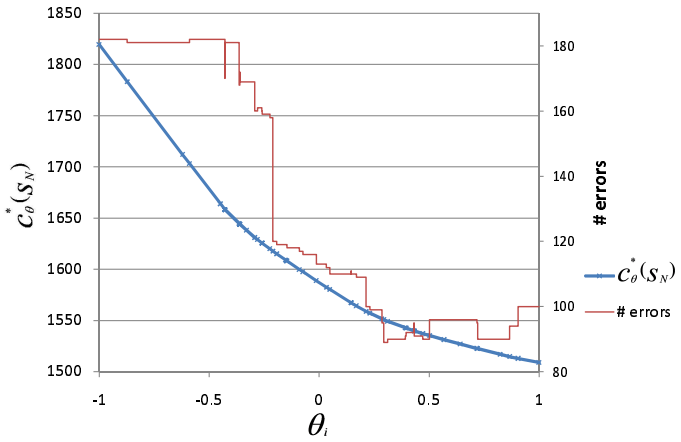

Fig. 5 After performing LDP on a particular component of θ, the resulting optimal cost and our symmetric difference penalty function, both as a functions of θ_k. Training is accomplished by optimizing the latter criterion over the possible paths indexed by θ_k.

class errors and 40 scale pitch class errors, corresponding to 1.33 chord errors and 0.29 scale errors per measure. Finally, we tried retraining the algorithm with a larger collection of 80 labels and used the learned θ with the test data and a larger collection of 132 chord labels. This experiment resulted in a better chord error rate of 1.06 but much worse scale error rate of 1.2, suggesting that the selected best path often "borrowed" chords from other keys. The lack of any significant quantity of ground truth data, or agreement on the collection of possible labels, makes comparisons with other approaches difficult.

The analysis is available for download as a midi file at http://www.music.informatics.indiana.edu/papers/informs08/ that demonstrates the resulting analysis by superimposing the the recognized triads over the piano music while printing out the chord labels as they are played.

References

1. Bellman R., "Dynamic Programming", *Proceedings of the International Computer Music Conference, 1984*, Princeton University Press, Princeton, New Jersey, 1957.
2. Kaelbling L, Littman M., and Cassandra A., "Planning and Acting in Partially Observable Stochastic Domains", *Artificial Intelligence* 101, 1-2, pp. 99-134, 1998.
3. Murphy K., "A survey of POMDP solution techniques", *Technical Report, U. C. Berkeley*, 2000.
4. Cassandra A., Littman M., Zhang N., "Incremental Pruning: A Simple, Fast, Exact Method for Partially Observable Markov Decision Processes", *Proc. Thirteenth Annual Conf. on Uncertainty in Artif. Intel. (UAI–97)* 1997.
5. Sondik E. J., "The optimal control of partially observable Markov processes", *Ph.D. Thesis* Stanford Univ., 1971.
6. White C. C., "Partially observed Markov decision processes: A survey", *Annals of Operations Research* 32, 1991.
7. Viterbi A., "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm", *IEEE Transactions on Information Theory*, 13-2, pp. 260-269, 1967.
   w
8. Rabiner L., "A tutorial on hidden markov models and selected applications in speech recognition", *Proceedings of the IEEE*, 77-2, pp. 257-285, 1989.
9. Starner T. and Pentland A., "Real-Time American Sign Language Recognition From Video Using Hidden Markov Models", *SCV95, 5B Systems and Applications*, 1995.
10. Hu J., Brown M. K., and Turin W., "HMM based on-line handwriting recognition", 18, pp. 1039-1045, 1996.
11. Karplus K., Barrett C., and Hughey R., "Hidden Markov models for detecting remote protein homologies", *Bioinformatics*, 14-10, 846-856, 1998.
12. Raphael C., "Automatic Segmentation of Acoustic Musical Signals Using Hidden Markov Models", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21-4, 360-370, 1999.
13. Pardo B. and Birmingham W. P., "Algorithms for Chordal Analysis", *Computer Music Journal*, 26-2, 27-49, 2002.
14. Raphael C., Stoddard J., "Harmonic analysis with probabilistic graphical models", *Proceedings of the International Symposium on Music Informatics Retrieval (ISMIR)* 2003.
15. Temperley D., "The cognition of basic musical structures", *MIT Press* Cambridge, MA, 2001.
16. Vercoe B. "The Synthetic Performer in the Context of Live Performance", *Proceedings of the International Computer Music Conference, 1984* ICMA, Paris, France, 1984.