A Convexity-based Generalization of Viterbi for Non-Deterministic Weighted Automata

Marc Dymetman
Xerox Research Centre Europe
Grenoble, France
marc.dymetman@xrce.xerox.com

Abstract

We propose a novel approach for the max-string problem in acyclic nondeterministic weighted FSA's, which is based on a convexity-related notion of domination among intermediary results, and which can be seen as a generalization of the usual dynamic programming technique for finding the max-path (a.k.a. Viterbi approximation) in such automata.

1 Introduction

Let A be an acyclic weighted finite-state automaton (WFSA) on a vocabulary V with weights in the (extended) set of non-negative reals $\mathbb{R}_+^\infty = [0, \infty]$, which we assume to be combined multiplicatively.

We can consider two problems. The first one, *max-path*, is to find the *path* π of maximum weight in the automaton, that is, the path that maximizes the product of the weights associated with its transitions; the second one, *max-string*, is to find the *string* x in V^* that maximizes the sum of the weights of all the paths that yield x. While the max-string problem is often the most important in principle, it is much more difficult to solve than the max-path problem; in fact (Casacuberta and de la Higuera, 2000) show that the problem is NP-hard: they describe a class of acyclic weighted automata that encode the Satisfiability problem (SAT) in such a way that identifying the max-string in such automata in polynomial time would imply a polynomial solution to SAT. In practice, one tends to use the max-path solution as a proxy to the max-string solution; this approximation employs the *Viterbi* algorithm (Viterbi, 1967), and is widely used in speech recognition, machine translation and other NLP tasks. The contribution of this paper is to propose a novel approach for the max-string problem over the "sum-times" semiring $K_s \equiv (\mathbb{R}_+^\infty, +, \cdot, 0, 1)$, involving a generalization of the Viterbi procedure.

A naive approach to the max-string problem would consist in enumerating all the paths, summing the weights of paths corresponding to the same string, and outputting the maximum string.

Another, more appealing, approach consists in noting that in the case of a *deterministic* weighted automaton A', the max-string and max-path problems coincide, and therefore in trying to determinize A, and then apply the standard Viterbi algorithm. However, while existing techniques for determinizing a weighted automaton (Mohri, 1997; Mohri, 2009) work reasonably well in some practical cases over the "max-times" semiring $(\mathbb{R}_+^\infty, \max, \cdot, 0, 1)$,[1] they often — rather counter-intuitively — lead to combinatorial explosion when working in the sum-times semiring, even in cases where the automaton is acyclic and where the classical (unweighted) determinization of A does not explode (Buchsbaum et al., 1998).[2] While the applications of determinization cited in (Mohri, 2009) to such domains as speech recognition tend to focus on the max-times semiring, we are aware of one application where determinization is based on the sum-times semiring, but in a slightly different formal situation (May and Knight, 2006). In this paper, the authors generalize the determinization technique of (Mohri, 1997) from string to tree automata, and then address the question of determinizing a weighted tree

[1] This semiring is isomorphic to the more common "tropical" semiring, through a logarithmic mapping.

[2] A simple example of a cyclic automaton over the sum-times semiring which is not determinizable at all is given in (Aminof et al., 2011).

automaton generating a set of trees, where each tree represents a possible translation of a fixed source sentence, in the context of a syntax-based SMT system (they also present an application to data-oriented parsing). In this way they are often able, at least for short sentences, to find the translation tree of maximum total weight in reasonable time. A similar technique, directly based on (Mohri, 1997), but using the sum-times semiring over a string automaton, could presumably be tempted for the weighted lattices produced by a phrase-based translation system such as Moses (Koehn et al., 2007), but we are not aware of any such attempt.

The novelty of our approach to the max-string problem is that it bypasses the need for a preliminary determinization of the automaton before applying Viterbi, but instead proposes to apply a generalization of Viterbi directly to the original non-deterministic automaton. Let us now describe this approach.

Elements of V are called symbols, elements of V^* strings. It will be convenient to assume that the automaton A has a special form: (i) it has exactly one initial state q_0 and one final state q_f, (ii) there is a special "end-of-string" symbol \$ in V, (iii) \$ only appears on transitions to q_f, and no other symbol can appear on such a transition, (iv) transitions labeled with \$ have weight 1 or 0.[3]

In a nutshell and informally, the main idea is then the following. Consider a string $x = a_1 a_2 \ldots a_k$. If A were deterministic, then, starting from q_0, this string would end in a single state q and would assign to this state a certain weight w equal to the product of the weights associated with the transitions of x; then, if some other string x' also ended in q, but with a higher weight w', then we would know that x could not be a prefix of the max-string for A; this observation contains the essence of the Viterbi procedure: for each state q, it is sufficient to only "remember" the prefix string producing the highest weight at q. Now, when A is non-deterministic, the string x can end in *several* states $q_1, ..., q_m$ simultaneously, with weights $w_1, ..., w_m$; in this case, if it happens that some other string x' ends in the same states, but with weights $w'_1, ..., w'_m$ s.t. $w'_1 > w_1, ..., w'_m > w_m$, then it can be shown that x cannot be a prefix of the max-string for A, and we can then discard x; as we will see, we can also discard x under weaker "domination" conditions, namely when the weight vector $w = (w_1, ..., w_m)$ associated with x belongs to a certain kind of convex hull of "dominating" vectors associated with a set S of prefix strings. Using this observation, we only need to explicitely store the set S of dominating prefix strings; whatever the suffix used to go to the final state, at least one of these dominating prefixes will lead to a better result with this suffix than using a dominated prefix x with the same suffix.

[3]These conditions are not restrictive, as it is easy to transform any A into this form, by adding to each final state of the original automaton an outgoing edge of weight 1 with label \$ and target q_f. It can be verified that the weight of a string of symbols $a_1 a_2 \ldots a_k$ relative to the original automaton is equal to that of the string $a_1 a_2 \ldots a_k\$$ relative to the transformed automaton.

2 Preliminaries

Automata and transition matrices Let us define $U = V \setminus \{\$\}$. The weighted automaton A can be viewed as associating, with each symbol $a \in U$ a transition matrix, that we will also call a, of dimension $D \times D$ over the non-negative reals $\mathbb{R}_+^\infty$, where D is the number of non-final states in A; the coordinate a_{ij} of that matrix is equal to the weight of the transition of label a between q_i and q_j, this weight being null if there is no such transition. The initial state q_0 of the automaton can be identified with the D-dimensional line vector $(1, 0, ..., 0)$, and the distribution of weights over the (non-final) states of A after having consumed the string $a_1 a_2 \ldots a_k$ is then given by the D-dimensional line vector $(1, 0, ..., 0) \cdot a_1 \cdot a_2 \ldots \cdot a_k$, where the $a_1, \ldots, a_k$'s are identified with their matrices.

Due to our assumptions on the symbol \$, we can identify \$ with a D-dimensional *column* vector $(w_0, w_1, ..., w_{D-1})^\top$, where w_i is equal to 1 or to 0. The weight relative to the automaton of a string of the form $a_1 a_2 \ldots a_p\$$ is then obtained by computing the scalar value $(1, 0, ..., 0) \cdot a_1 \cdot a_2 \cdots a_p \cdot \$$, which can also be viewed as a scalar product of a line vector with the column vector \$.

Convex, ortho and ortho-convex hulls We now need to introduce the notions of convex, ortho, and ortho-convex hulls. Let d be a positive integer, and let S be a set (finite or not) of d-dimensional vectors over the non-negative reals. We say that:

- The vector u is in the *convex-hull* (or *c-hull*) of S iff we can write u as a finite sum $u = \sum_j \alpha_j s_j$, with $s_j \in S,\ j \in [1, m],\ \sum_j \alpha_j =$

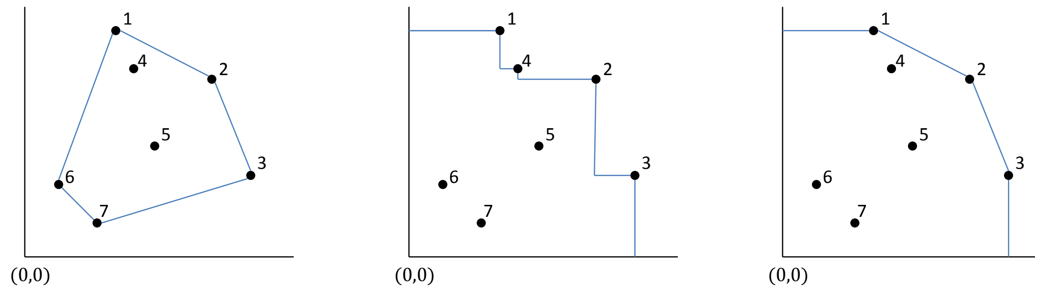

Figure 1: The sets $\{1,2,3,6,7\}$ [left pane], $\{1,2,3,4\}$ [middle pane], and $\{1,2,3\}$ [right pane] are subsets of dominators of the set $W = \{1,2,3,4,5,6,7\}$, respectively relative to the notions of convex-hull, ortho-hull, and ortho-convex hull.

$1, \alpha_j \geq 0$. This is the standard notion.

- The vector u is in the *ortho-hull* (or *o-hull*) of S iff there exists a $v \in S$ s.t. $u \leq v$, where the inequality is interpreted to hold for each coordinate. The ortho-hull is in general not convex.
- The vector u is in the *ortho-convex-hull* (or *oc-hull*) of S iff u is in the ortho-hull of the convex-hull of S. It is easy to check that the ortho-convex-hull is convex.[4]

When a set W of d-vectors is contained in the hull of some subset $S \subset W$, then we will say that S is a set of "dominators" of W, relative to the specific notion of hull used. Figure 1 illustrates these notions for dimension $d = 2$.

Lemma *If x is in the convex-hull (resp. ortho-hull, ortho-convex-hull) of S, and if z is a non-negative d-vector, then there exists $s \in S$ such that $x.z \leq s.z$, where . denotes scalar product.*[5]

3 Algorithm

Algorithm 1 is our main algorithm. The integer k corresponds to a stage of the algorithm. W is a set of pairs of the form (*prefix*, *vector*), where *prefix* is a string not ending in \$, and *vector* is the D-dimensional vector associated with this prefix; for each stage k of the algorithm, $W = W_k$ contains only prefixes of length k, and $S = S_k$ is a subset of dominators of W_k; $F = F_k$ is either the empty set or is a singleton set containing a pair of the form (*string*, *number*), where *string* is a string ending in \$ and *number* is a scalar.

Algorithm 1 MAIN

```
1: $k \leftarrow 0, F \leftarrow \emptyset, W \leftarrow \{(\epsilon, (1, 0, \ldots, 0))\}$
2: while $W \neq \emptyset$ do
3:     $S \leftarrow$ DOMINATORS($W$)
4:     $k \leftarrow k + 1$
5:     $(W, F) \leftarrow$ FORWARD($S, F$)
6: return $F$
```

On line 1, we initialize F to the empty set, and W to the empty string prefix ϵ together with a vector carrying weight 1 on q_0 and weight 0 on the other states. On line 2, we loop until W is empty. On line 3, we extract a subset S of dominators from W, according to one of the three hull variants we have described. We then increment k, and on line 5, we compute through the FORWARD procedure the next version of W and F corresponding to strings of length $k+1$. Lastly, on line 6, we return the final result F, which is either empty (when A does not recognize any string), or contains a pair (*string*, *number*) where *string* is a[6] max-string for A and *number* its total weight. The loop in line 2 terminates because the automaton is acyclic: all prefixes above a certain length will eventually be mapped to the null D-vector which will result in producing at a certain point an empty W_k.

[4]Our notion of oc-hull is related to that of *anti-blocking polyhedra* in the LP literature (Schrijver, 1998).

[5]*Proof sketch.* Suppose by contradiction that for all $s \in S$, we have $s.z < x.z$. But now (i) if x is in the convex-hull of S, then x can be written as the convex combination $x = \sum_i \alpha_i s_i$ of elements s_i in S; therefore $x.z = \sum_i \alpha_i s_i.z < x.z$, a contradiction; (ii) if x is in the ortho-hull of S, then there exists $s \in S$ s.t. $x \leq s$ coordinate-wise, hence $x.z \leq s.z$ by the non-negativity of z, again a contradiction; (iii) if x is in the ortho-convex-hull of S, then there exists x' s.t. $x \leq x'$ and x' can be written as $x' = \sum_i \alpha_i s_i$, and we can combine the two previous arguments to again reach a contradiction.

[6]The max-string is not always unique: several strings may reach the same maximum.

Algorithm 2 FORWARD(S, F)

```
1: $W \leftarrow \emptyset$
2: **for** $(\textit{prefix}, \textit{vector}) \in S$ **do**
3:     $F \leftarrow \textsc{Max}(F, (\textit{prefix}.\$, \textit{vector}.\$))$
4:     **for** $a \in U$ **and** $\textit{vector}.a \neq 0$ **do**
5:         $W \leftarrow W \cup \{(\textit{prefix}.a, \textit{vector}.a)\}$
6: **return** $(W, F)$
```

Algorithm 2 defines the FORWARD procedure. We start by initializing W to the empty set, then for each pair $(\textit{prefix}, \textit{vector})$ in S we do the following. On line 3, we compute the concatenated string *prefix*.$ along with its weight, given by the scalar product *vector*.$; If *vector*.$ is equal to 0, then MAX does not modify F, if F is empty MAX returns the singleton set $(\textit{prefix}.\$, \textit{vector}.\$)$, and finally if $F = \{(\textit{string}, \textit{number})\}$ it returns either $\{(\textit{string}, \textit{number})\}$ or $\{(\textit{prefix}.\$, \textit{vector}.\$)\}$ according to which of *number* or *vector*.$ is the largest. On line 4, for each symbol a in U such that $\textit{vector}.a$ is not null, we add to W the pair consisting of the prefix string $\textit{prefix}.a$ and of the vector $\textit{vector}.a$. Finally we return the pair (W, F).

Theorem *Whatever the notion of hull used for defining* DOMINATORS *in Algorithm 1, if the result F is empty, then the language of the automaton is empty; otherwise* $F = \{(\textit{string}, \textit{number})\}$*, where 'number' is the maximum weight of a string relative to the automaton A, and where 'string' ends in* $ *and is of weight 'number'.*[7]

We still need to explain how we define the DOMINATORS procedure in Algorithm 1, depending on which notion of hull is chosen. Line 3 of the algorithm consists in pruning the set of prefixes W to get the subset S, and the efficiency of the algorithm as a whole depends on pruning as much as possible at each level k, thus it is in our interest to extract a small set of dominators from W. To simplify the description here, we will pretend that an element $(\textit{prefix}, \textit{vector})$ of W is identified with its second component *vector*, and will identify W to a set of vectors; we can easily recover the prefix associated with each vector at the end of the process. Let us start with the simplest case, that of the ortho-hull. In that case, the minimal set of dominators for W is easily shown to be simply the set of all vectors that survive after eliminating any $w \in W$ s.t. there exists another $w' \in W$ with $w \leq w'$; a straightforward quadratic algorithm in the size of W can be designed for that purpose. If the hull is the convex-hull, the minimal set of dominators is the set of so-called *extreme points* from W, for which there exist several algorithms (Helbling, 2010). Overall, the ortho-convex hull is more effective at pruning than both the ortho- and the convex-hull. Let us therefore give some indications on how to compute a minimal set of dominators relative to the oc-hull.

Similar to the o-hull case, we want to eliminate any w from W which is in the oc-hull of the remaining vectors $w_1, ..., w_n$ of W. Such a vector is determined by the condition that one can find a convex combination $\sum_i \alpha_i w_i$ such that $w \leq \sum_i \alpha_i w_i$. We can directly map this problem into a Linear Programming format, for which a large number of solvers are available: the solver is asked to decide whether the following LP, in the variables $\alpha_1, ..., \alpha_n$, is *feasible*:[8]

$$\begin{aligned} \sum_i \alpha_i &= 1, \\ \sum_i \alpha_i w_i - w &\geq 0, \\ \alpha_i &\geq 0, \forall i. \end{aligned}$$

The complexity of this oc-hull algorithm is on the order of $n+1$ times the complexity for solving one instance of the LP above, which cannot be characterized simply and depends on the type of solver used (simplex vs. interior-point based). However, the preliminary experiments we have conducted indicate that it is more efficient to first prune W relative to the o-hull notion, which already eliminates many points and only then prune this intermediary result using the LP formulation above (this can be shown to preserve the notion of oc-dominators).

[7] *Proof sketch.* Let T be the set of strings (often, this set is actually a singleton) ending in $ which do reach the actual max-string weight relative to the automaton. Call "rank" of a string $t \in T$ the largest number k such that a prefix of t appears in S_k, and let us focus on a t which has maximal rank k relative to the other elements of T, and on its prefix x of length k. In case $t = x.\$$, then t "makes it" to F in line 5 of Algorithm 1, and we are done. Otherwise t can be written as $t = x.a.z$, with $a \in U$, and with $x.a \notin S_{k+1}$. But then, by the Lemma, there exists $s \in S_{k+1}$ s.t., in vectorial terms, $x.a.z \leq s.z$, and therefore, because of the definition of T, the string $s.z$ also belongs to T; but $s.z$ has rank $k+1$, a contradiction.

[8] This LP is our adaptation to the ortho-convex-hull case of a similar program described by (Helbling, 2010) for the convex-hull case, of which more sophisticated versions are also proposed.

4 Conclusion

The procedure that we have described can be seen as a generalization of the standard Viterbi technique. Viterbi (even in the case of an original non-deterministic automaton) can be formulated in terms of a max-string problem over a certain deterministic automaton. For such a deterministic automaton, our procedure only produces vectors that are each placed on a *single* axis of $\mathbb{R}^D$, corresponding to the single state reached by the corresponding prefix. In this case it can be checked that o-dominators and oc-dominators lead to the same result, namely to *keeping the maximum point on each axis separately*, which exactly corresponds to the Viterbi procedure, which keeps the maximum on each state independently of other states.

It should also be noted that pruning the space of prefixes using the oc-hull construction appears to be the best we can hope to achieve if we are not allowed to use heuristics that look forward in the automaton: it can be shown that by appropriately choosing the weights of transitions not yet seen at level k, the max-string can be made to "select" *any* of the oc-dominators from W_k — this is however not true for the c-dominators or the o-dominators.

We believe the method to have potential applications to such domains as speech recognition or phrase-based statistical machine translation; the latter in particular tends to produce large word lattices where many paths can correspond to the same string; there the main object of interest is the max-string, to which the Viterbi best-path is only an approximation. More generally, the method could be of interest for doing inference with Hidden Markov Models, when the objects of real interest are not the hidden paths in the HMM, but rather the projections of these paths onto sequences of directly interpretable labels.

Acknowledgments

Thanks to Salah Aït-Mokhtar, Nicola Cancedda and András Sebö for discussions on a preliminary version of this paper, and to the anonymous reviewers for pointing to some important references that I had not identified.

References

B. Aminof, O. Kupferman, and R. Lampert. 2011. Rigorous approximated determinization of weighted automata. In *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*, pages 345 –354, june.

Adam L. Buchsbaum, Raffaele Giancarlo, and Jeffery Westbrook. 1998. On the determinization of weighted finite automata. In *ICALP'98*, pages 482–493.

Francisco Casacuberta and Colin de la Higuera. 2000. Computational complexity of problems on probabilistic grammars and transducers. In *ICGI*, pages 15–24.

Christian Helbling. 2010. Extreme Points in Medium and High Dimensions. Master's thesis, Dpt of Computer Science, ETH Zrich.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL*. The Association for Computer Linguistics.

Jonathan May and Kevin Knight. 2006. A better n-best list: Practical determinization of weighted finite tree automata. In *HLT-NAACL*.

Mehryar Mohri and Michael Riley. 2002. An efficient algorithm for the n-best-strings problem. In *In Proceedings of the International Conference on Spoken Language Processing 2002 (ICSLP 02*.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Comput. Linguist.*, 23(2):269–311.

Mehryar Mohri. 2009. Weighted automata algorithms. In Manfred Droste, Werner Kuich, and Heiko Vogler, editors, *Handbook of Weighted Automata*, pages 213–254. Springer.

Alexander Schrijver. 1998. *Theory of Linear and Integer Programming*. Wiley.

Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, IT-13(2):260–269, April.