Optimization and Sampling for NLP from a Unified Viewpoint

Marc DYMETMAN[1] Guillaume BOUCHARD[1] Simon CARTER[2]

(1) Xerox Research Centre Europe, Grenoble, France
(2) ISLA, University of Amsterdam, The Netherlands
{marc.dymetman,guillaume.bouchard}@xrce.xerox.com; s.c.carter@uva.nl

Abstract

The OS* algorithm is a unified approach to exact optimization and sampling, based on incremental refinements of a functional upper bound, which combines ideas of adaptive rejection sampling and of A* optimization search. We first give a detailed description of OS*. We then explain how it can be applied to several NLP tasks, giving more details on two such applications: (i) decoding and sampling with a high-order HMM, and (ii) decoding and sampling with the intersection of a PCFG and a high-order LM.

1 Introduction

Optimization and Sampling are usually seen as two completely separate tasks. However, in NLP and many other domains, the primary objects of interest are often probability distributions over discrete or continuous spaces, for which both aspects are natural: in optimization, we are looking for the mode of the distribution, in sampling we would like to produce a statistically representative set of objects from the distribution. The OS* algorithm approaches the two aspects from a unified viewpoint; it is a joint exact Optimization and Sampling algorithm that is inspired both by rejection sampling and by classical A* optimization (O S *).

Common algorithms for sampling high-dimensional distributions are based on MCMC techniques [Andrieu et al., 2003, Robert and Casella, 2004], which are approximate in the sense that they produce valid samples only asymptotically. By contrast, the elementary technique of Rejection Sampling [Robert and Casella, 2004] directly produces exact samples, but, if applied naively to high-dimensional spaces, typically requires unacceptable time before producing a first sample.

By contrast, OS* can be applied to high-dimensional spaces. The main idea is to upper-bound the complex target distribution p by a simpler proposal distribution q, such that a dynamic programming (or another low-complexity) method can be applied to q in order to efficiently sample or maximize from it. In the case of sampling, rejection sampling is then applied to q, and on a reject at point x, q is refined into a slightly more complex q' in an adaptive way. This is done by using the evidence of the reject at x, which implies a gap between $q(x)$ and $p(x)$, to identify a (soft) constraint implicit in p which is not accounted for by q. This constraint is then integrated into q to obtain q'.

The constraint which is integrated tends to be highly relevant and to increase the acceptance rate of the algorithm. By contrast, many constraints that are constitutive of p are never "activated" by sampling from q, because q never explores regions where they would become visible. For example, anticipating our HMM experiments in section 3.1, there is little point in explicitly including in q a 5-gram constraint on a certain latent sequence in the HMM if this sequence is already unlikely at the bigram level: the bigram constraints present in the proposal q will ensure that this sequence will never (or very rarely) be explored by q.

The case of optimization is treated in exactly the same way as sampling. Formally, this consists in moving from assessing proposals in terms of the L_1 norm to assessing them in terms of the L_∞ norm. Typically, when a dynamic programming procedure is available for sampling (L_1 norm) with q, it is also available for maximizing from q (L_∞ norm), and the main difference between the two cases is then in the criteria for selecting among possible refinements.

Related work In order to improve the acceptance rate of rejection sampling, one has to lower the proposal q curve as much as possible while keeping it above the p curve. In order to do that, some authors [Gilks and Wild, 1992, Görür and Teh, 2008], have proposed *Adaptive Rejection Sampling (ARS)* where, based on rejections, the q curve is updated to a lower curve q' with a better acceptance rate. These techniques have predominantly been applied to continuous distributions on the one-dimensional real line, where convexity assumptions on the target distribution can be exploited to progressively approximate it tighter and tighter through upper bounds consisting of piecewise linear envelopes. These sampling techniques have not been connected to optimization.

One can find a larger amount of related work in the optimization domain. In an heuristic optimization context the two interesting, but apparently little known, papers [Kam and Kopec, 1996, Popat et al., 2000], discuss a technique for decoding images based on high-order language models where upper-bounds are constructed in terms of simpler variable-order models. Our application of OS* in section 3.1 to the problem of maximizing a high-order HMM is similar to their (also exact) technique; while this work seems to be the closest to ours, the authors do not attempt to generalize their approach to other optimization problems amenable to dynamic programming. Among NLP applications, [Kaji et al., 2010, Huang et al., 2012] are another recent approach to exact optimization for sequence labelling that also has connections to our experiments in section 3.1, but differs in particular by using a less flexible refinement scheme than our variable-order n-grams. In the NLP community again, there is currently a lot of interest for optimization methods that fall in the general category of "coarse-to-fine" techniques [Petrov, 2009], which tries to guide the inference process towards promising regions that get incrementally tighter and tighter. While most of this line of research concentrates on approximate optimization, some related approaches aim at finding an exact optimum. Thus, [Tromble and Eisner, 2006] attempt to maximize a certain objective while respecting complex hard constraints, which they do by incrementally adding those constraints that are violated by the current optimum, using finite-state techniques. [Riedel and Clarke, 2006] have a similar goal, but address it by incrementally adding ILP (Integer Linear Programming) constraints. Linear Programming techniques are also involved in the recent applications of Dual Decomposition to NLP [Rush et al., 2010, Rush and Collins, 2011], which can be applied to difficult combinations of easy problems, and often are able to find an exact optimum. None of these optimization papers, to our knowledge, attempts to extend these techniques to sampling, in contrast to what we do.

Paper organization The remainder of this paper is structured as follows. In section 2, we describe the OS* algorithm, explain how it can be used for exact optimization and sampling, and show its connection to A*. In section 3, we first describe several NLP applications of the algorithm, then give more details on two such applications. The first one is an application to optimization/sampling with high-order HMMs, where we also provide experimental results; the second one is a high-level description of its application to the generic problem of optimization/sampling with the intersection of a PCFG with a high-order language model, a problem which has recently attracted some attention in the community. We finally conclude and indicate some perspectives.

2 The OS* algorithm

OS* is a unified algorithm for optimization and sampling. For simplicity, we first present its sampling version, then move to its optimization version, and finally get to the unified view. We start with some background and notations about rejection sampling.

Note: The OS* approach was previously described in an e-print [Dymetman et al., 2012], where some applications beyond NLP are also provided.

2.1 Background

Let $p : X \to \mathbb{R}_+$ be a measurable L_1 function with respect to a base measure μ on a space X, i.e. $\int_X p(x)d\mu(x) < \infty$. We define $\bar{p}(x) \equiv \frac{p(x)}{\int_X p(x)d\mu(x)}$. The function p can be seen as an unnormalized density over X, and $\bar{p}$ as a normalized density which defines a probability distribution over X, called the *target distribution*, from which we want to sample from[1]. While we may not be able to sample directly from the target distribution $\bar{p}$, let us assume that we can easily compute $p(x)$ for any given x. *Rejection Sampling* (RS) [Robert and Casella, 2004] then works as follows. We define a certain unnormalized *proposal density* q over X, which is such that (i) we know how to directly sample from it (more precisely, from $\bar{q}$), and (ii) q dominates p, i.e. for all $x \in X, p(x) \leq q(x)$. We then repeat the following process: (1) we draw a sample x from q, (2) we compute the ratio $r(x) \equiv p(x)/q(x) \leq 1$, (3) with probability $r(x)$ we accept x, otherwise we reject it, (4) we repeat the process with a new x. It can then be shown that this procedure produces an exact sample from p. Furthermore, the average rate at which it produces these samples, the *acceptance rate*, is equal to $P(X)/Q(X)$ [Robert and Casella, 2004], where for a (measurable) subset A of X, we define $P(A) \equiv \int_A p(x)d\mu(x)$ and $Q(A) \equiv \int_A q(x)d\mu(x)$. In Fig. 1, panel (S1), the acceptance rate is equal to the ratio of the area below the p curve with that below the q curve.

2.2 Sampling with OS*

The way OS* does sampling is illustrated on the top of Fig. 1. In this illustration, we start sampling with an initial proposal density q (see (S1)). Our first attempt produces x_1, for which the ratio $r_q(x_1) = p(x_1)/q(x_1)$ is close to 1; this leads, say, to the acceptance of x_1. Our second attempt produces x_2, for which the ratio $r_q(x_2) = p(x_2)/q(x_2)$ is much lower than 1, leading, say, to a rejection. Although we have a rejection, we have gained some useful information, namely that $p(x_2)$ is much lower than $q(x_2)$, and we are going to use that "evidence" to define a new proposal q' (see (S2)), which has the following properties:

- $p(x) \leq q'(x) \leq q(x)$ everywhere on X;
- $q'(x_2) < q(x_2)$.

One extreme way of obtaining such a q' is to take $q'(x)$ equal to $p(x)$ for $x = x_2$ and to $q(x)$ for $x \neq x_2$, which, when the space X is discrete, has the effect of improving the acceptance rate, but only slightly so, by insuring that any time q' happens to select x_2, it will accept it.

[1]By abuse of language, we will also say that a sample from $\bar{p}$ is a sample from p.

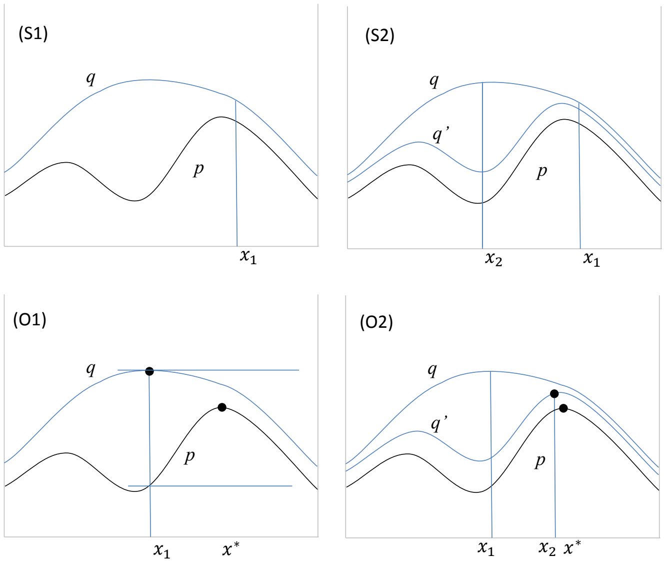

Figure 1: Sampling with OS* (S1, S2), and optimization with OS* (O1, O2).

A better generic way to find a q' is the following. Suppose that we are provided with a small finite set of "one-step refinement actions" a_j, depending on q and x_2, which are able to move from q to a new $q'_j = a_j(q, x_2)$ such that for any such a_j one has $p(x) \leq q'_j(x) \leq q(x)$ everywhere on X and also $q'_j(x_2) < q(x_2)$. Then we will select among these a_j moves the one that is such that the L_1 norm of q'_j is minimal among the possible j's, or in other words, such that $\int_X q'_j(x)d\mu(x)$ is minimal in j. The idea is that, by doing so, we will improve the acceptance rate of q'_j (which depends directly on $\|q'_j\|_1$) as much as possible, while (i) not having to explore a too large space of possible refinements, and (ii) moving to a representation of q'_j that is only slightly more complex than q, rather than to a much more complex representation for a q' that could result from exploring a larger space of possible refinements for q.[2] The intuition behind such one-step refinement actions a_j will become clearer when we consider concrete examples below.

2.3 Optimization with OS*

The optimization version of OS* is illustrated on the bottom of Fig. 1, where (O1) shows on the one hand the function p that we are trying to maximize from, along with its (unknown) maximum p^*, indicated by a black circle on the p curve, and corresponding to x^* in X. It also shows a "proposal" function q which is such — analogously to the sampling case — that (1) the

[2]In particular, even if we could find a refinement q' that would exactly coincide with p, and therefore would have the smallest possible L_1 norm, we might not want to use such a refinement if this involved an overly complex representation for q'.

function q is above p on the whole of the space X and (2) it is easy to directly find the point x_1 in X at which it reaches its maximum q^*, shown as a black circle on the q curve.

A simple, but central, observation is the following one. Suppose that the distance between $q(x_1)$ and $p(x_1)$ is smaller than ϵ, then the distance between $q(x_1)$ and p^* is also smaller than ϵ. This can be checked immediately on the figure, and is due to the fact that on the one hand p^* is higher than $p(x_1)$, and that on the other hand it is below $q(x^*)$, and *a fortiori* below $q(x_1)$. In other words, if the maximum that we have found for q is at a coordinate x_1 and we observe that $q(x_1) - p(x_1) < \epsilon$, then we can conclude that we have found the maximum of p up to ϵ.

In the case of x_1 in the figure, we are still far from the maximum, and so we "reject" x_1, and refine q into q' (see (O2)), using exactly the same approach as in the sampling case, but for one difference: the one-step refinement option a_j that is selected is now chosen on the basis of how much it decreases, not the L_1 norm of q, but the max of q — where, as a reminder, this max can also be notated $||q||_\infty$, using the L_∞ norm notation.[3]

Once this q' has been selected, one can then find its maximum at x_2 and then the process can be repeated with $q_1 = q, q_2 = q', \ldots$ until the difference between $q_k(x_k)$ and $p(x_k)$ is smaller than a certain predefined threshold.

2.4 Sampling L_1 vs. Optimization L_∞

While sampling and optimization are usually seen as two completely distinct tasks, they can actually be viewed as two extremities of a continuous range, when considered in the context of L_p spaces.

If (X, μ) is a measure space, and if f is a real-valued function on this space, one defines the L_p norm $||f||_p$, for $1 \leq p < \infty$ as: $||f||_p \equiv \left(\int_X |f|^p(x) d\mu(x)\right)^{1/p}$ [Rudin, 1987]. One also defines the L_∞ norm $||f||_\infty$ as: $||f||_\infty \equiv \inf\{C \geq 0 : |f(x)| \leq C$ for almost every $x\}$, where the right term is called the *essential supremum* of $|f|$, and can be thought of roughly as the "max" of the function. So, with some abuse of language, we can simply write: $||f||_\infty \equiv \max_{x \in X} |f|$. The space L_p, for $1 \leq p \leq \infty$, is then defined as being the space of all functions f for which $||f||_p < \infty$. Under the simple condition that $||f||_p < \infty$ for some $p < \infty$, we have: $\lim_{p \to \infty} ||f||_p = ||f||_\infty$.

The standard notion of sampling is relative to L_1. However we can introduce the following generalization — where we use the notation L_α instead of L_p in order to avoid confusion with our use of p for denoting the target distribution. We will say that we are performing *sampling of a non-negative function f relative to $L_\alpha(X, \mu)$*, for $1 \leq \alpha < \infty$, if $f \in L_\alpha(X, \mu)$ and if we sample — in the standard sense — according to the normalized density distribution $\bar{f}(x) \equiv \frac{f(x)^\alpha}{\int_X f(x)^\alpha d\mu(x)}$. In the case $\alpha = \infty$, we will say that we are sampling relative to $L_\infty(X, \mu)$, if $f \in L_\infty(X, \mu)$ and if we are performing optimization relative to f, more precisely, if for any $\epsilon > 0$, we are able to find an x such that $|||f||_\infty - f(x)| < \epsilon$.

Informally, sampling relative to L_α "tends" to sampling with L_∞ (i.e. optimization), for α tending to ∞, in the sense that for a large α, an x sampled relative to L_α "tends" to be close to a maximum for f. We will not attempt to give a precise formal meaning to that observation here, but just note the connection with the idea of *simulated annealing* [Kirkpatrick et al.,

[3]A formal definition of that norm is that $||q||_\infty$ is equal to the "essential supremum" of q over (X, μ) (see below), but for all practical purposes here, it is enough to think of this essential supremum as being the max, when it exists.

1983], which we can view as a mix between the MCMC Metropolis-Hastings sampling technique [Robert and Casella, 2004] and the idea of sampling in L_α spaces with larger and larger α's.

In summary, we thus can view optimization as an extreme form of sampling. In the sequel we will often use this generalized sense of sampling in our algorithms.[4]

2.5 OS* as a unified algorithm

The general design of OS* can be described as follows:

- Our goal is to OS-sample from p, where we take the expression "OS-sample" to refer to a generalized sense that covers both sampling (in the standard sense) and optimization.

- We have at our disposal a family $\mathcal{Q}$ of proposal densities over the space (X, μ), such that, for every $q \in \mathcal{Q}$, we are able to OS-sample efficiently from q.

- Given a reject x_1 relative to a proposal q, with $p(x_1) < q(x_1)$, we have at our disposal a (limited) number of possible "one-step" refinement options q', with $p \leq q' \leq q$, and such that $q'(x_1) < q(x_1)$.

- We then select one such q'. One possible selection criterion is to prefer the q' which has the smallest L_1 norm (sampling case) or L_∞ norm (optimization). In one sense, this is the most natural criterion, as it means we are directly lowering the norm that controls the efficiency of the OS-sampling. For instance, for sampling, if q'_1 and q'_2 are two candidates refinements with $\|q'_1\|_1 < \|q'_2\|_1$, then the acceptance rate of q'_1 is larger than that of q'_2, simply because then $P(X)/Q'_1(X) > P(X)/Q'_2(X)$. Similarly, in optimization, if $\|q'_1\|_\infty < \|q'_2\|_\infty$, then the gap between $\max_x(q'_1(x))$ and p^* is smaller than that between $\max_x(q'_2(x))$ and p^*, simply because then $\max_x(q'_1(x)) < \max_x(q'_2(x))$. However, using this criterion may require the computation of the norm of each of the possible one-step refinements, which can be costly, and one can prefer simpler criteria, for instance simply selecting the q' that minimizes $q'(x_1)$.

- We iterate until we settle on a "good" q: either (in sampling) one which is such that the cumulative acceptance rate until this point is above a certain threshold; or (in optimization) one for which the ratio $p(x_1)/q(x_1)$ is closer to 1 than a certain threshold, with x_1 being the maximum for q.

The following algorithm gives a unified view of OS*, valid for both sampling and optimization. This is a high-level view, with some of the content delegated to the subroutines `OS-Sample`, `Accept-or-Reject`, `Update`, `Refine`, `Stop`, which are described in the text.

[4]Note: While our two experiments in section **??** are based on discrete spaces, the OS* algorithm is more general, and *can be applied to any measurable space (in particular continuous spaces)*; in such cases, p and q have to be measurable functions, and the relation $p \leq q$ should be read as $p(x) \leq q(x)$ a.e. (almost everywhere) relative to the base measure μ.

Algorithm 1 The OS* algorithm

```
1: while not Stop($h$) do
2:     OS-Sample $x \sim q$
3:     $r \leftarrow p(x)/q(x)$
4:     Accept-or-Reject($x, r$)
5:     Update($h, x$)
6:     if Rejected($x$) then
7:         $q \leftarrow$ Refine($q, x$)
8: return $q$ along with accepted $x$'s in $h$
```

On entry into the algorithm, we assume that we are either in sample mode or in optimization mode, and also that we are starting from a proposal q which (1) dominates p and (2) from which we can sample or optimize directly. We use the terminology *OS-Sample* to represent either of these cases, where `OS-Sample` $x \sim q$ refers to sampling an x according to the proposal q or optimizing x on q (namely finding an x which is an argmax of q, a common operation for many NLP tasks), depending on the case. On line (1), h refers to the history of the sampling so far, namely to the set of trials $x_1, x_2, ...$ that have been done so far, each being marked for acceptance or rejection (in the case of sampling, this is the usual notion, in the case of optimization, all but the last proposal will be marked as rejects). The stopping criterion `Stop`(h) will be to stop: (i) *in sampling mode*, if the number of acceptances so far relative to the number of trials is larger than a certain predefined threshold, and in this case will return on line (8), first, the list of accepted x's so far, which is already a valid sample from p, and second, the last refinement q, which can then be used to produce any number of future samples as desired with an acceptance ratio similar to the one observed so far; (ii) *in optimization mode*, if the last element x of the history is an accept, and in this case will return on line (8), first the value x, which in this mode is the only accepted trial in the history, and second, the last refinement q (which can be used for such purposes as providing a "certificate of optimality of x relative to p", but we do not detail this here).

On line (3), we compute the ratio r, and then on line (4) we decide to accept x or not based on this ratio; in optimization mode, we accept x if the ratio is close enough to 1, as determined by a threshold[5]; in sampling mode, we accept x based on a Bernoulli trial of probability r.

On line (5), we update the history by recording the trial x and whether it was accepted or not.

If x was rejected (line (6)), then on line (7), we perform a refinement of q, based on the principles that we have explained.

2.6 A connection with A*

A special case of the OS* algorithm, which we call "OS* with piecewise bounds", shows a deep connection with the classical A* optimization algorithm [Hart et al., 1968] and is interesting in its own right. Let us first focus on sampling, and suppose that q_0 represents an initial proposal density, which upper-bounds the target density p over X. We start by sampling with q_0, and on a first reject somewhere in X, we split the set X into two disjoint subsets X_1,X_2, obtaining a partition of X. By using the more precise context provided by this partition, we may be able

[5]When X is a finite domain, it makes sense to stop on a ratio *equal* to 1, in which case we have found an exact maximum. This is what we do in our experiments in section 3.1.

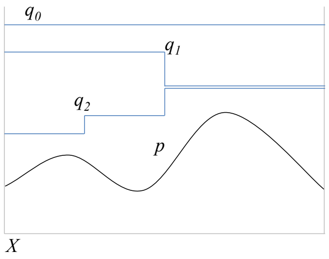

Figure 2: A connection with A*.

to improve our upper bound q_0 over the whole of X into tighter upper bounds on each of X_1 and X_2, resulting then in a new upper bound q_1 over the whole of X. We then sample using q_1, and experience at some later time another reject, say on a point in X_1; at this point we again partition X_1 into two subsets X_{11} and X_{12}, tighten the bounds on these subsets, and obtain a refined proposal q_2 over X; we then iterate the process of building this "hierarchical partition" until we reach a certain acceptance-rate threshold.

If we now move our focus to optimization, we see that the refinements that we have just proposed present an analogy to the technique used by A*. This is illustrated in Fig. 2. In A*, we start with a *constant* optimistic bound — corresponding to our q_0 — for the objective function which is computed at the root of the search tree, which we can assume to be binary. We then expand the two daughters of the root, re-evaluate the optimistic bounds there to new constants, obtaining the piecewise constant proposal q_1, and move to the daughter with the highest bound. We continue by expanding at each step the leaf of the partial search tree with the highest optimistic bound (e.g. moving from q_1 to q_2, etc.).

It is important to note that OS*, when used in optimization mode, is in fact *strictly more general than A*,* for two reasons: (i) it does not assume a piecewise refinement strategy, namely that the refinements follow a hierarchical partition of the space, where a given refinement is limited to a leaf of the current partition, and (ii) even if such a stategy is followed, it does not assume that the piecewise upper-bounds are constant. Both points will become clearer in the HMM experiments of section 3.1, where including an higher-order n-gram in q has impact on several regions of X simultaneously, possibly overlapping in complex ways with regions touched by previous refinements; in addition, the impact of a single n-gram is non-constant even in the regions it touches, because it depends of the multiplicity of the n-gram, not only on its presence or absence.

3 Some NLP applications of OS*

The OS* framework appears to have many applications to NLP problems where we need to optimize or sample from a complex objective function p. Let us give a partial list of such situations:

- Efficient decoding and sampling with high-order HMM's.
- Combination of a probabilistic context free grammar (PCFG) with a complex finite-state automaton (FSA):

  - Tagging by joint usage of a PCFG and a HMM tagger;
  - Hierarchical translation with a complex target language model

- Parsing in the presence of non-local features.
- PCFG's with transversal constraints, probabilistic unification grammars.

We will not explore all of these situations here, but will concentrate on (i) decoding and sampling with high-order HMM's, for which we provide details and experimental results, and (ii) combining a PCFG with a complex finite-state language model, which we only describe at a high-level. We hope these two illustrations will suffice to give a feeling of the power of the technique.

3.1 High-Order HMMs

Note: An extended and much more detailed version of these HMM experiments is provided in [Carter et al., 2012].

The objective in our HMM experiments is to sample a word sequence with density $\bar{p}(x)$ proportional to $p(x) = p_{\text{lm}}(x)\ p_{\text{obs}}(o|x)$, where p_{lm} is the probability of the sequence x under an n-gram model and $p_{\text{obs}}(o|x)$ is the probability of observing the noisy sequence of observations o given that the word sequence is x. Assuming that the observations depend only on the current state, this probability can be written:

$$p(x) \quad = \quad \prod_{i=1}^{\ell} p_{\text{lm}}(x_i|x_{i-n+1}^{i-1})\ p_{\text{obs}}(o_i|x_i)\ . \tag{1}$$

Approach Taking a tri-gram language model for simplicity, let us define $w_3(x_i|x_{i-2}x_{i-1}) = p_{\text{lm}}(x_i|x_{i-2}x_{i-1})\ p_{\text{obs}}(o_i|x_i)$. Then consider the observation o be fixed, and write $p(x) = \prod_i w_3(x_i|x_{i-2}x_{i-1})$. In optimization/decoding, we want to find the argmax of $p(x)$, and in sampling, to sample from $p(x)$. Note that the state space associated with p can be huge, as we need to represent explicitly all contexts (x_{i-2}, x_{i-1}) in the case of a trigram model, and even more contexts for higher-order models.

We define $w_2(x_i|x_{i-1}) = \max_{x_{i-2}} w_3(x_i|x_{i-2}x_{i-1})$, along with $w_1(x_i) = \max_{x_{i-1}} w_2(x_i|x_{i-1})$, where the maxima are taken over all possible context words in the vocabulary. These quantities, which can be precomputed efficiently, can be seen as optimistic "max-backoffs" of the trigram x_{i-2}^{i}, where we have forgotten some part of the context. Our initial proposal is then $q_0(x) = \prod_i w_1(x_i)$. Clearly, for any sequence x of words, we have $p(x) \leq q_0(x)$. The state space of q_0 is much less costly to represent than that of $p(x)$.

The proposals q_t, which incorporate n-grams of variable orders, can be represented efficiently through weighted FSAs (WFSAs). In Fig. 3(a), we show a WFSA representing the initial proposal q_0 corresponding to an example with four observations, which we take to be the acoustic realizations of the words 'the, two, dogs, barked'. The weights on edges correspond only to unigram max-backoffs, and thus each state corresponds to a NULL-context. Over this WFSA, both optimization and sampling can be done efficiently by the standard dynamic programming techniques (Viterbi [Rabiner, 1989] and "backward filtering-forward sampling" [Scott, 2002]), where the forward weights associated to states are computed similarly, either in the max-product or in the sum-product semiring.

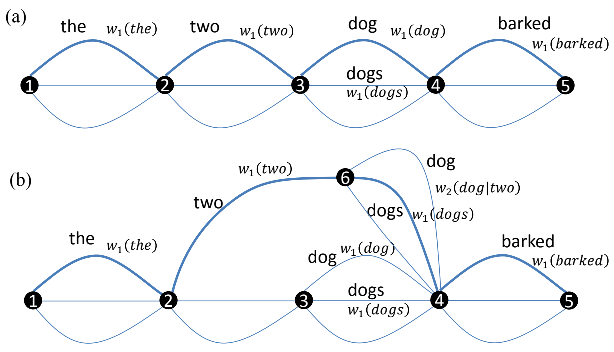

Figure 3: *An example of an initial q-automaton (a), and its refinement (b).*

Consider first sampling, and suppose that the first sample from q_0 produces $x_1 =$ *the two dog barked*, marked with bold edges in the drawing. Now, computing the ratio $p(x_1)/q_0(x_1)$ gives a result much smaller than 1, in part because from the viewpoint of the full model p, the trigram *the two dog* is very unlikely; i.e. the ratio $w_3(dog|the\ two)/w_1(dog)$ is very low. Thus, with high probability, x_1 is rejected. When this is the case, we produce a refined proposal q_1, represented by the WFSA in Fig. 3(b), which takes into account the more realistic bigram weight $w_2(dog|two)$ by adding a node (node 6) for the context *two*. We then perform a sampling trial with q_1, which this time tends to avoid producing *dog* in the context of *two*; if we experience a reject later on some sequence x_2, we refine again, meaning that we identify an n-gram in x_2, which, if we extend its context by one (e.g from a unigram to a bigram or from a bigram to a trigram), accounts for some significant part of the gap between $q_1(x_2)$ and $p(x_2)$. We stop the refinement process when we start observing acceptance rates above a certain fixed threshold.

The case of optimization is similar. Suppose that with q_0 the maximum is $x_1 =$ *the two dog barked*, then we observe that $p(x_1)$ is lower than $q_0(x_1)$, reject x_1 and refine q_0 into q_1. We stop the process at the point where the value of q_t, at its maximum x_{q_t}, is equal to the value of p at x_{q_t}, which implies that we have found the maximum for p.

Setup We evaluate our approach on an SMS-message retrieval task. Let N be the number of possible words in the vocabulary. A latent variable $x \in \{1, \cdots, N\}^\ell$ represents a sentence defined as a sequence of ℓ words. Each word is converted into a sequence of numbers based on a mobile phone numeric keypad, assuming some level of random noise in the conversion. The task is then to recover the original message.

We use the English side of the Europarl corpus [Koehn, 2005] for training and test data (1.3 million sentences). A 5-gram language model is trained using SRILM [Stolcke, 2002] on 90% of the sentences. On the remaining 10%, we randomly select 100 sequences for lengths 1 to 10 to obtain 1000 sequences from which we remove the ones containing numbers, obtaining a test set of size 926.

Optimization We limit the average number of latent tokens in our decoding experiments to 1000. In the plot (d1) of Fig. 4 we show the average number of iterations (running Viterbi then

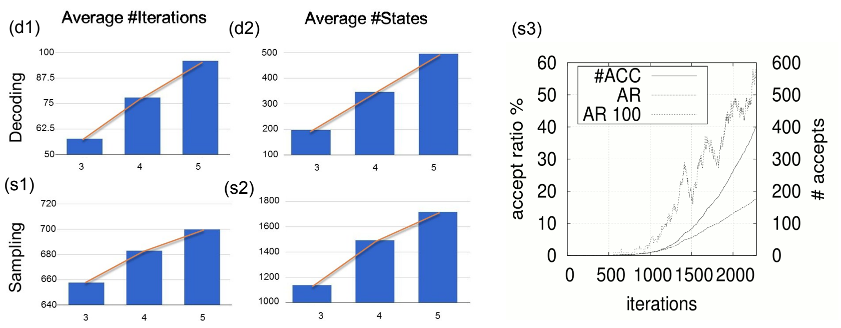

Figure 4: SMS experiments.

updating q) that the different n-gram models of size 3, 4 and 5 take to do exact decoding of the sentences of fixed length (10 words) in the test-set. We can see that decoding with larger n-gram models tends to show a linear increase w.r.t. n in the number of iterations taken. Plot (d2) shows the average number of states in the final automaton q for the same sentences, also showing a linear increase with n (rather than the exponential growth expected if all the possible states were represented). We also display in Table 1 the distribution of n-grams in the final model for one specific sentence of length 10. Note that the total number of n-grams in the full model would be $\sim 3.0 \times 10^{15}$; exact decoding here is not tractable using existing techniques. By comparison, our HMM has only 118 five-grams and 9008 n-grams in total.

n:	1	2	3	4	5
q:	7868	615	231	176	118

Table 1: *# of n-grams in our variable-order HMM.*

Sampling For the sampling experiments, we limit the number of latent tokens to 100. We refine our q automaton until we reach a certain fixed cumulative acceptance rate (AR). We also compute a rate based only on the last 100 trials (AR-100), as this tends to better reflect the current acceptance rate. In plot (s3) of Fig. 4, we show a single sampling run using a 5-gram model for an example input, and the cumulative # of accepts (middle curve). It took 500 iterations before the first successful sample from p.

We noted that there is a trade-off between the time needed to compute the forward probability weights needed for sampling, and the time it takes to adapt the variable-order HMM. To resolve this, we use batch-updates: making B trials from the same q-automaton, and then updating our model in one step. By doing this, we noted significant speed-ups in sampling times. Empirically, we found $B = 100$ to be a good value, leading to an order of magnitude improvement in speed. In plots (s1) and (s2), we show the average # of iterations and states in our models until refinements are finished (for an AR-100 of 20%), for different orders n over sentences of length 10 in the test set. We note linear trends in the number of iterations and states when moving to higher n; for length=10, and for $n = 3,4,5$, average number of iterations: 3-658, 4-683, 5-701; averager number of states: 3-1139, 4-1494, 5-1718. In particular the number of states is again much lower than the exponential increase we would expect if using the full underlying state space.

3.2 OS* for intersecting PCFG's with high-order LM's

We now move to a high-level description of the application of OS* to an important class of problems (including hierarchical SMT) which involve the intersection of PCFG's with high-order language models. The study of similar "agreement-based" problems involving optimization (but not sampling) over a combination of two individual tasks have recently been the focus of a lot of attention in the NLP community, in particular with the application of dual decomposition methods [Rush et al., 2010, Rush and Collins, 2011, Chang and Collins, 2011]. We only sketch the main ideas of our approach.

A standard result of formal language theory is that the intersection of a CFG with a FSA is a CFG [Bar-Hillel et al., 1961]. This construct can be generalized to the intersection of a Weighted CFG (WCFG) with a WFSA (see e.g. [Nederhof and Satta, 2003]), resulting in a WCFG. In our case, we will be interested in optimizing and sampling from the intersection p of a PCFG[6] G with a complex WFSA A representing a high-order language model. For illustration purposes here, we will suppose that A is a trigram language model, but the description can be easily transposed to higher-order cases.

Let us denote by x a derivation in G, and by $y = y(x)$ the string of terminal leaves associated with x (the "yield" of the derivation x). The weighted intersection p of G and A is defined as: $p(x) \equiv G(x).A(x)$, where $A(x)$ is a shorthand for $A(y(x))$.

Due to the intersection result, p can in principle be represented by a WCFG, but for a trigram model A, this grammar can become very large. Our approach will then be the following: we will start by taking the proposal $q^{(0)}$ equal to G, and then gradually refine this proposal by incorporating more and more accurate approximations to the full automaton A, themselves expressed as weighted automata of small complexity. We will stop refining based on having found a good enough approximation in optimization, or a sampler with sufficient acceptance rate in sampling.

To be precise, let us consider a PCFG G, with $\sum_x G(x) = 1$, where x varies among all the finite derivations relative to G.[7] Let us first note that it is very simple to sample from G: just expand derivations top-down using the conditional probabilities of the rules. It is also very easy to find the derivation x of maximum value $G(x)$ by a standard dynamic programming procedure.

We are going to introduce a sequence of proposals $q^{(0)} = G, q^{(1)} = q^{(0)}.B^{(1)}, ..., q^{(i+1)} = q^{(i)}.B^{(i+1)}, ...$, where each $B^{(i)}$ is a small automaton including some additional knowledge about the language model represented by A. Each $q^{(i)}$ will thus be a WCFG (not normalized) and refining $q^{(i)}$ into $q^{(i+1)}$ will then consist of a "local" update of the grammar $q^{(i)}$, ensuring a desirable incrementality of the refinements.

Analogous to the HMM case, we define: $w_3(x_5|x_3x_4) = \max_{x_2} w_4(x_5|x_2x_3x_4)$, $w_2(x_5|x_4) = \max_{x_3} w_4(x_5|x_3x_4)$, and $w_1(x_5) = \max_{x_4} w_4(x_5|x_4)$.

[6]A Probabilistic CFG (PCFG) is a special case of a WCFG.

[7]Such a PCFG is said to be consistent, that is, it is such that the total mass of infinite derivations is null [Wetherell, 1980]. We do not go into the details of this (standard) assumption here.

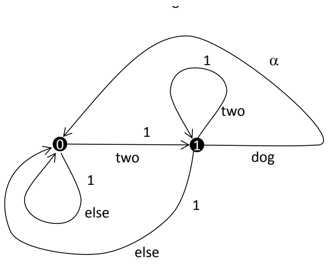

Figure 5: The "TwoDog" automaton.

Let's first consider the optimization case, and suppose that, at a certain stage, the grammar $q^{(i)}$ has already "incorporated" knowledge of $w_1(dog)$. Then suppose that the maximum derivation $x^{(i)} = \text{argmax}^{(i)}(x)$ has the yield: *the two dog barked*, where $w_1(dog)$ is much larger than the more accurate $w_2(dog|two)$.

We then decide to update $q^{(i)}$ into $q^{(i+1)} = q^{(i)}.B^{(i+1)}$, where $B^{(i+1)}$ represents the additional knowledge corresponding to $w_2(dog|two)$. More precisely, let us define:

$$\alpha \equiv \frac{w_2(dog|two)}{w_1(dog)}.$$

Clearly $\alpha \leq 1$ by the definition of w_1, w_2. Now consider the following two-state automaton $B^{(i+1)}$, which we will call the "TwoDog" automaton:

In this automaton, the state (0) is both initial and final, and (1) is only final. The state (1) can be interpreted as meaning "I have just produced the word *two*". All edges carry a weight of 1, apart from the edge labelled *dog*, which carries a weight of α. The "else" label on the arrow from (0) to itself is a shorthand that means: "any word other than *two*", and the "else" label on the arrow from (1) to (0) has the meaning: "any word other than *dog* or *two*".

It can be checked that this automaton has the following behavior: it gives a word sequence the weight α^k, where k is the number of times the sequence contains the subsequence *two dog*.

Thus, when intersected with the grammar $q^{(i)}$, this automaton produces the grammar $q^{(i+1)}$, which is such that $q^{(i+1)}(x) = q^{(i)}(x)$ if the yield of x does not contain the subsequence *two dog*, but such that $q^{(i+1)}(x) = \alpha^k.q^{(i)}(x)$ if it contains this subsequence k times. Note that because $q^{(i)}$ had already "recorded" the knowledge about $w_1(dog)$, it now assigns to the word *dog* in the context of *two* the weight $w_2(dog|two) = w_1(dog).\alpha$, while it still assigns to it in all other contexts the weight $w_1(dog)$, as desired.[8]

We will not go here into the detailed mechanics of intersecting the automaton "TwoDog" with the WCFG $q^{(i)}$, (see [Nederhof and Satta, 2008] for one technique), but just note that, because this automaton only contains two states and only one edge whose weight is not 1, this intersection can be carried efficiently at each step, and will typically result in very local updates of the grammar.

[8]Note: Our refinements of section 3.1 might also have been seen as intersections between a weighted FSA (rather than a weighted CFG) and a "local" automaton similar to "TwoDog", but their implementation was more direct.

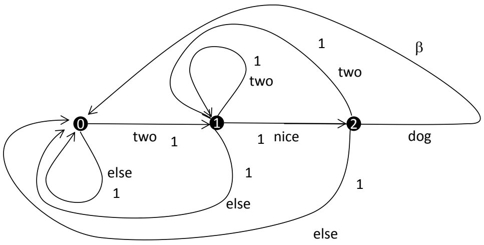

Figure 6: The "TwoNiceDog" automaton.

Higher-order n-grams can be registered in the grammar in a similar way. For instance, let's suppose that we have already incorporated in the grammar the knowledge of $w_2(dog|nice)$ and that we now wish to incorporate the knowledge of $w_3(dog|two\ nice)$, we can now use the following automaton ("TwoNiceDog" automaton), where $\beta = w_3(dog|two\ nice)/w_2(dog|nice)$:

For optimization, we thus find the maximum $x^{(i)}$ of each grammar $q^{(i)}$, check the ratio $p(x^{(i)})/q^{(i)}(x^{(i)})$, and stop if this ratio is close enough to 1. Else, we choose an n-gram in $x^{(i)}$ which is not yet registered and would significantly lower the proposal $q^{(i)}(x^{(i)})$ if added, build the automaton corresponding to this n-gram, and intersect this automaton with $q^{(i)}$.

Sampling is done in exactly the same way, the difference being that we now use dynamic programming to compute the *sum* of weights bottom-up in the grammars $q^{(i)}$, which is really a matter of using the sum-product semiring instead of the max-product semiring.

4 Conclusion

In this paper, we have argued for a unified viewpoint for heuristic optimization and rejection sampling, by using functional upper-bounds for both, and by using rejects as the basis for reducing the gap between the upper-bound and the objective. Bringing together Optimization and Sampling in this way permits to draw inspirations (A* Optimization, Rejection Sampling) from both domains to produce the joint algorithm OS*.

In particular, the optimization mode of OS*, which is directly inspired by rejection sampling, provides a generic exact optimization technique which appears to be more powerful than A* (as argued in section 2.6), and to have many potential applications to NLP distributions based on complex features, of which we detailed only two: high-order HMMs, and an agreement-based model for the combination of a weighted CFG with a language model. As these examples illustrate, often the same dynamic programming techniques can be used for optimization and sampling, the underlying representations being identical, the difference being only a change of semiring.

Both optimization and sampling are prominent questions in NLP generally, in the contexts of inference as well as of learning. In particular, we are currently working on applying these techniques to decoding and training tasks for phrase-based and also hierarchical statistical machine translation.

References

Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1-2):5–43, 2003.

Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift für Phonetik, Sprachwissenschaft und Kommunicationsforschung*, 14: 143–172, 1961.

Simon Carter, Marc Dymetman, and Guillaume Bouchard. Exact Sampling and Decoding in High-Order Hidden Markov Models. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1125–1134, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D12-1103`.

Yin-Wen Chang and Michael Collins. Exact decoding of phrase-based translation models through lagrangian relaxation. In *EMNLP*, pages 26–37, 2011.

M. Dymetman, G. Bouchard, and S. Carter. The OS* Algorithm: a Joint Approach to Exact Optimization and Sampling. *ArXiv e-prints*, July 2012.

W. R. Gilks and P. Wild. Adaptive rejection sampling for gibbs sampling. *Applied Statistics*, pages 337–348, 1992.

D. Görür and Y.W. Teh. Concave convex adaptive rejection sampling. Technical report, Gatsby Computational Neuroscience Unit, 2008.

P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions On Systems Science And Cybernetics*, 4(2): 100–107, 1968. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4082128`.

Zhiheng Huang, Yi Chang, Bo Long, Jean-Francois Crespo, Anlei Dong, Sathiya Keerthi, and Su-Lin Wu. Iterative viterbi a* algorithm for k-best sequential decoding. In *ACL (1)*, pages 611–619, 2012.

Nobuhiro Kaji, Yasuhiro Fujiwara, Naoki Yoshinaga, and Masaru Kitsuregawa. Efficient Staggered Decoding for Sequence Labeling. In *Meeting of the Association for Computational Linguistics*, pages 485–494, 2010.

Anthony C. Kam and Gary E. Kopec. Document image decoding by heuristic search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:945–950, 1996.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *Proceedings of Machine Translation Summit*, pages 79–86, 2005.

Mark-Jan Nederhof and Giorgio Satta. Probabilistic parsing. In *New Developments in Formal Languages and Applications*, pages 229–258, 2008.

M.J. Nederhof and G. Satta. Probabilistic parsing as intersection. In *Proc. 8th International Workshop on Parsing Technologies*, 2003.

Slav Petrov. *Coarse-to-Fine Natural Language Processing*. PhD thesis, University of California at Berkeley, 2009.

K. Popat, D. Bloomberg, and D. Greene. Adding linguistic constraints to document image decoding. In *Proc. 4th International Workshop on Document Analysis Systems*. International Association of Pattern Recognition, December 2000.

Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

Sebastian Riedel and James Clarke. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 129–137, Sydney, Australia, July 2006. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W/W06/W06-1616`.

Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2004. ISBN 0387212396.

Walter Rudin. *Real and Complex Analysis*. McGraw-Hill, 1987.

Alexander M. Rush and Michael Collins. Exact decoding of syntactic translation models through lagrangian relaxation. In *ACL*, pages 72–82, 2011.

Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings EMNLP*, 2010.

Steven L. Scott. Bayesian methods for hidden markov models: Recursive computing in the 21st century. *Journal of the American Statistical Association*, 97:337–351, 2002. URL `http://EconPapers.repec.org/RePEc:bes:jnlasa:v:97:y:2002:m:march:p:337-351`.

Andreas Stolcke. Srilm - an extensible language modeling toolkit. In *Proceedings of the International Conference of Spoken Language Processing (INTERSPEECH 2002)*, pages 257–286, 2002.

Roy W. Tromble and Jason Eisner. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Proceedings of the Human Language Technology Conference of the North American Association for Computational Linguistics (HLT-NAACL)*, pages 423–430, New York, June 2006. URL `http://cs.jhu.edu/~jason/papers/#tromble-eisner-2006`.

C. S. Wetherell. Probabilistic languages: A review and some open questions. *ACM Comput. Surv.*, 12(4):361–379, 1980. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/356827.356829.