Improving Grammar Constraints Generation Alignment by Sampling Highly Probable Playouts

Killian Susini, Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL
killian.susini@dauphine.eu, tristan.cazenave@lamsade.dauphine.fr

Abstract

Large Language Models (LLMs) can generate structured outputs such as program code, mathematical formulas, or well-formed markup language, but some of their outputs may end up non-grammatical, especially when considering very precise and obscure structures. Grammar Constrained Decoding (GCD) is used to force each subsequent token generated to follow a particular grammar. However, simple techniques use a distorted distribution of the LLMs. Grammar Aligned Decoding (GAD) is the problem of aligning sampling with a grammar constraint. Adaptive sampling with approximate expected futures (ASAp) has been proposed to solve GAD but requires storing the change in the probabilities of the LLM for each token in a datastructure. Furthermore, ASAp samples randomly from the aligned distribution as it is computed, which may not be optimal to maximize the alignment speed. We propose Greedy Best First Search with Greedy Sampling (GBFSGS) as a method to obtain high likelihood samples, which is more likely to maximize the alignment speed of the distribution. GBFSGS is able to solve GAD while storing the new probabilities in a datastructure with one element per sample drawn instead of per token changed. We show how the method finds the most probable samples of a distribution, including under a grammar constraint. We show how to implement the algorithm to avoid duplicate samples. We empirically evaluate our method on three sets of problems to evaluate the behavior of the methods. We show GBFSGS is able to select better samples than sampling to improve the alignment of the distribution while reducing the memory needed to store the aligned distribution.

Introduction

The ability to generate the complex and diverse sequences of natural languages by Large Language Models (LLM) may be a downside when trying to generate structured sequences such as program code as such sequences follow a structured grammar. A simple method to guarantee the generation of correctly structured sentences (i.e. *grammatical* sentences) is to reject any non-structured sample (rejection sampling). As long as the distribution of the unstructured generation is similar to the one of the structured output, the method remains efficient, but otherwise a lot of wasted computation occurs. One method to palliate this problem is to reject unstructured tokens immediately during generation, which allows one to always obtain a grammatically correct sequence. This method is described as Grammar-Constrained Decoding (GCD) and, as shown in Grammar-Aligned Decoding by Park et al. (2024), it distorts the distribution over the sequences which does not match the distribution of the structured samples. Generating from the correct distribution (GAD) is formalized in the paper, and while it is achieved by rejection sampling, it is inefficient. However, to resolve the problems highlighted above, the paper introduces a new method, Adaptive Sampling with Approximate Expected Futures (ASAp), which deals with the mismatch of GCD generation by progressively building the correct distribution for GAD through repeated sampling. One large downside of ASAp is the memory required to store the info necessary for the new distribution. Each new sample requires storing an Expected Future Grammaticality (EFG) value for each token along the path for each sample, which quickly grows for long sequences. Furthermore, while using random samples asymptotically computes the true GAD distribution, randomly sampling from the approximate distribution itself may not generate the most informative samples for aligning the distribution. In fact, as the approximate GAD gets closer to the true distribution, the probability of generating an improving sample drops to zero. We propose Greedy Best First Search with Greedy Sampling (GBFSGS), a method that builds an approximation of the GAD distribution like ASAp, while limiting the memory requirements and improving the convergence speed on nearly every problem tested against ASAp. Following (Park et al. 2024), we evaluate the methods on the same two problems of formal program synthesis and constituency parsing. Our experiments show the faster convergence by GBFSGS compared to ASAp on almost every problem.

Definitions

We start with the definitions and important notations in order to present the algorithm that we extend in this paper: ASAp.

Large Language Models An (autoregressive) language model θ defines a probability distribution P^θ on the set of all strings $w \in \Sigma^*$ over a vocabulary of tokens Σ via the left-to-right factorization of the joint probability $P^\theta(w) = P^\theta(w_{1:n}) = P^\theta(w_1...w_n) = \Pi_{i=1}^{n} P^\theta(w_i|w_{1:i-1})$, where n

is the finite length (in tokens) of the sentence w. Although it can be any size, in practice, an upper limit on the length is fixed.

The tokens present in the vocabulary may represent subwords or multiple words, see (OpenAI et al. 2024) for the list used by GPT-4.

The goal of this algorithm is to constrain the default generation of LLMs to generate sentences constrained by a grammar.

Context-Free Grammars *Context-free grammar* (CFG) are useful to logically define various structured outputs such as properly formulated programming languages.

$\mathcal{G} = (\Sigma, \mathcal{N}, S, \mathcal{R})$ is a CFG where Σ is the finite set of tokens (vocabulary, also called set of terminal symbols), $\mathcal{N}$ is the set of non-terminal symbols, $S \in \mathcal{N}$ is the starting non-terminal symbol, and $\mathcal{R}$ is the set of production rules $A \Rightarrow \alpha, A \in \mathcal{N}, \alpha \in (\mathcal{N} \cup \Sigma)^*$.

The repeated application of each rule defines the *language* of $\mathcal{G}$, $L(\mathcal{G}) = \{w \in \Sigma^* | w = Sr_1r_2 \cdots r_n, r_i \in \mathcal{R}\}$.

The *prefix language* of $\mathcal{G}$ is defined as $L_{pfx}(\mathcal{G}) = \{w \in \Sigma^* | wv \in L(\mathcal{G}), v \in \Sigma^*\}$. It describes every sequence that could be completed into a sentence of $L(\mathcal{G})$. Although members of the prefix language are not necessarily members of the language of $\mathcal{G}$, the opposite is true.

For simplicity, given a sequence $w_{1:i}$, we say that the word w' is accepted by $\mathcal{G}$ after $w_{1:i}$ if $w_{1:i} \cdot w' \in L_{pfx}(\mathcal{G})$.

Related works

Grammar Constrained Decoding GCD is the problem of restricting the generation of a LLM to the language of a grammar, such that only sentences that are in the language are generated.

As noted in (Geng et al. 2023), one traditional method to perform GCD is by using rejection sampling. The algorithm discards ungrammatical generation if it occurs and tries again from the beginning, which may incur a significant amount of wasted computation.

Instead of waiting for the end of the generation to check if the sentence generated is in the language, one improvement is to remove the possibility of the LLM generating tokens that would make the partial sentence leave the prefix language of the grammar. This is what the "GCD" algorithm does, setting the probability of ungrammatical tokens to 0 and normalizing the probabilities among the tokens remaining, $P^\theta_{GCD}(w'|w_{1:i})$,

$$\frac{P^\theta(w'|w_{1:i})\mathbb{1}[w_{1:i} \cdot w' \in L_{pfx}(\mathcal{G})]}{\Sigma_{w''} P^\theta(w''|w_{1:i})\mathbb{1}[w_{1:i} \cdot w'' \in L_{pfx}(\mathcal{G})]} \tag{1}$$

Contrary to rejection sampling, GCD distorts the distribution of the sentences by underestimating the effect of the grammar later in the sentence (Park et al. 2024).

It is complex to decide if a token in the vocabulary of a LLM can be generated without leaving the language defined by a CFG, as the tokens do not necessarily match with the symbols of the CFG. This question is explored by various papers (Willard and Louf 2023; Geng et al. 2023; Park, Zhou, and D'Antoni 2025), and techniques have been developed to efficiently obtain the list of the next possible generable tokens by a LLM that are accepted by the language. This paper does not focus on this problem, but improvements on the methods will benefit the speed of the algorithm.

GAD & ASAp Grammar Aligned Decoding is the problem of generating grammar-constrained sentences with probability proportional to that of the LLM,

$$P^\theta_{True}(w) = \frac{P^\theta(w)\mathbb{1}[w \in L(\mathcal{G})]}{\Sigma_{w'} P^\theta(w')\mathbb{1}[w' \in L(\mathcal{G})]}. \tag{2}$$

It is easy to see that rejection sampling generates samples according to equation 2.

The equivalent conditional probability equation is

$$P^\theta_{True}(w'|w_{1:i}) = \frac{P^\theta(w'|w_{1:i})c(w_{1:i} \cdot w')}{\Sigma_{w''} P^\theta(w''|w_{1:i})c(w_{1:i} \cdot w'')}, \tag{3}$$

Where $c(w)$ (named *Expected Future Grammaticality*(Park et al. 2024)) is a constant between 0 and 1 that is equal to the proportion of grammatical sentences that can be obtained starting from w,

$$c(w_{1:i}) = E_{P^\theta(w_{i+1:n}|w_{1:i})}[\mathbb{1}[w_{i+1:n} \in L(\mathcal{G})]]. \tag{4}$$

Note how equation 1 overestimates $c(w)$ to 1 whenever there is even a single grammatical sentence that can be generated after w (Park et al. 2024).

Since $c(w)$ is untractable in general, Adaptive Sampling with Approximate expected futures probability (ASAp)(Park et al. 2024) is a method proposed to solve GAD by progressively building an approximation $\tilde{c}_S(w) \approx c(w)$ through a succession of samples S, defined inductively as

$$\tilde{c}_S(w_{1:i}) = \Sigma_{w_{i+1}} P^\theta(w_{i+1}|w_{1:i})\tilde{c}_S(w_{1:i+1}), \tag{5}$$

with the base case (no samples starting with $w_{1:i}$) defined as

$$\tilde{c}_S(w_{1:i}) = \mathbb{1}[w_{1:i} \in L_{prefix}(\mathcal{G})]. \tag{6}$$

In practice, ASAp initializes its approximation with the same definition as GCD. After each sample $w_{1:n}$, ASAp goes backward to update the approximation using equation 5. The space required to store the approximation $\tilde{c}_S(w)$ for every sentence is roughly equal to that of storing every sample.

Greedy Best First Search with Greedy Sampling

We propose a new algorithm to improve upon ASAp, called *Greedy Best First Search with Greedy Sampling* (GBFSGS). This algorithm replaces the sampling used by ASAp to progressively build the approximation of $c(w)$, in order to obtain better convergence properties. We show further

GBFS Greedy Best First Search is a simple algorithm belonging to the family of Best First Search algorithms like A*. It keeps a list of potential next moves, repeatedly expanding the best move according to a metric. For example, to find the path with the least number of moves in a graph, we start with the root, then repeatedly select the sequence

Algorithm 1: GBFSGS

Initialise $L \leftarrow \{root\}, \tilde{c}_S(\cdot) \leftarrow 1$
for $k \leq K$ **do**
  $w_{1:i} \leftarrow$ best leaf from L according to $P^\theta(\cdot)$
  $L \leftarrow L \cup \{w_{1:i} \cdot w' | w_{1:i} \cdot w' \in \mathcal{L}_{pfx}(\mathcal{G})\}$
  $w_{i:n} \leftarrow$ greedy completion of $w_{1:i}$ according to $P^\theta(\cdot)$
  for i in $(n-1 \dots 1)$ **do**
    $\tilde{c}_S(w_{1:i}) \leftarrow \sum_{w'} P^\theta(w'|w_{1:i}) \cdot \tilde{c}_S(w_{1:i} \cdot w')$

with the fewest number of moves to expand. If we are instead interested in the path of least cost, we will instead select the sequence of moves with the least cost to select the next node to expand.

In the context of sentence generation, each token generated is a move. Using the LLM and the grammar as a metric to define the score of a sequence of moves as the likelihood of those moves, and the update rule defined by equation 5, we can use GBFS to progressively build the approximation of $c(w)$ for each token. However, we fail to consider the effect of the grammar deep into the sentences until the algorithm runs for long enough.

GBFSGS We observe that GBFS uses the top part of the sentences (the token closest to the root) to update the approximation. To incorporate additional information further down the tree, we can sample a full sentence from the node selected by GBFS. Since we are interested in maximizing the potential of the improvement, we aim to select a sentence with high likelihood. We propose to use greedy sampling, which differs from standard sampling by shifting the distribution entirely toward the most probable token at each step.

GBFSGS like ASAp progressively builds an approximation of $c(\cdot) \approx \tilde{c}_S(\cdot)$ by accumulating a series of samples S, but improves upon ASAp by selecting the samples most likely to improve the approximation.

Properties and Optimization

A basic implementation of GBFSGS leads to a compute cost and space requirement similar to that of ASAp, since only the nature of the samples accumulated in S are changed. However, GBFSGS is deterministic since it uses the greedy choices according to the current approximation, which is expected to be the choice that improves the approximation the most. We first show that GBFSGS approximation $\tilde{c}_S(\cdot)$ converges properly to the true value of $c(\cdot)$, and we also show several improvements that can be implemented to limit redundant work and the space needed by GBFSGS.

Convergence At every step, GBFS selects a different subsequence from the root, and eventually every complete sequence. As different sequences accumulate in S, eventually, $\tilde{c}_S(\cdot) \to c(\cdot)$, like ASAp. Contrary to ASAp, the greedy nature of the sampling should lead to higher likelihood samples, which should accelerate the speed of convergence over ASAp.

Duplicate greedy samples We can exploit the fact that some samples could change little to $\tilde{c}(\cdot)$, leading to minimal change to the underlying distribution. While the initial search performed with GBFS already increases the exploration of unseen paths (paths closer to the root are naturally more likely than further down the tree), it is possible to detect that the greedy continuation of a sub-sequence selected by GBFS will be the same as one that was previously computed. When we update $\tilde{c}(\cdot)$, we can note if anywhere in the sequence the greedy choice changes because of the change to $\tilde{c}(\cdot)$. If no change in the greedy choice is made, we can conclude that a subsequent greedy sample passing through those tokens will produce the same sample, and therefore we can skip generating the sample entirely and simply expand GBFS. The actual implementation simply adds a binary flag to the actions added to GBFS, which indicates when the greedy sampling step can be skipped.

Replay for optimizing memory use The approximation $\tilde{c}_S(\cdot)$ is simple to obtain at the start (when no samples have been obtained, i.e. $S = \{\}$) since it only relies on the grammar. However, each new sample $w_{1:n}$ will change the initial estimation to every token along its path, and in the case of ASAp (Park et al. 2024), these new estimations are stored inside a tree. This is particularly inefficient when we consider tokens much further down the tree, which are unlikely to be resampled. However, it is unavoidable to guarantee correctness. In the case of GBFSGS, however, because the samples are greedy, and therefore deterministic, it is possible to come up with a scheme to only require storing a single value per sample (instead of one per token generated).

First consider the progressive evolution of the set of samples S, denoting $S_0 = \{\}$, $S_1 = \{w^1\}$, $S_2 = \{w^1, w^2\}$,...

The effect of the k^{th} sample w^k is limited to only a few numbers of values of the approximation $\tilde{c}_S(\cdot)$. Denoting $\tilde{c}_{S_k}(\cdot)$ as the state of the approximation after the k^{th} sample, we look back to the equation 5 and derive the update in the value of the approximation due to the new sample,

$$\tilde{c}_{S_k}(w_{1:i}) = \tilde{c}_{S_{k-1}}(w_{1:i}) + P^\theta(w_{i+1}|w_{1:i})\tilde{c}^\Delta_{S_k}(w_{1:i}). \quad (7)$$

Where the "residual value" $\tilde{c}^\Delta_{S_k}(\cdot) = \tilde{c}_{S_k}(\cdot) - \tilde{c}_{S_{k-1}}(\cdot)$ represents the change in the approximation due to sample k at $w_{1:i}$. From eq. 7 we get,

$$\tilde{c}_{S_k}(w_{1:i+1}) = \tilde{c}_{S_{k-1}}(w_{1:i+1}) + \frac{\tilde{c}^\Delta_{S_k}(w_{1:i})}{P^\theta(w_{i+1}|w_{1:i})}, \quad (8)$$

Furthermore, from eq. 8 we derive that,

$$\tilde{c}^\Delta_{S_k}(w_{1:i+1}) = \frac{\tilde{c}^\Delta_{S_k}(w_{1:i})}{P^\theta(w_{i+1}|w_{1:i})} \quad (9)$$

Equations 8 and 9 give us a top-down definition of $\tilde{c}_S$. We can move forward in token from $w_{1:i}$ to $w_{1:i+1}$ and in samples from S_k to S_{k+1} by keeping track of certain values. The values we need are the residual $\tilde{c}^\Delta_{S_k}(w')$ (the change due to the k^{th} sample sentence at position w', chosen wisely) and the sequence of tokens the sentence takes afterward.

For almost every $w_{1:i}$ we have $\tilde{c}^{\Delta}_{S_k}(w_{1:i}) = 0$, except potentially at $\tilde{c}^{\Delta}_{S_k}(w^k_{1:i})$. Furthermore, to use equations 8 and 9 requires determining $w_{1:i+1}$ from $w_{1:i}$ and $\tilde{c}_{S_{k-1}}(\cdot)$; thankfully, GBFSGS uses greedy sampling, and therefore we know that $w_{1:i+1}$ is the greedy choice following $w_{1:i}$.

We therefore need only to keep a record of the residual at the sub-sequence $w^k_{1:i}$ selected by GBFS at step k. The impact of sample w^k on the rest of the tree is derivable from the above equation at runtime, essentially allowing us to replay this sample when we need. Where ASAp needs to keep all the information of the sub-tree in memory, GBFSGS only keeps a single value and computes the sub-tree as needed.

Experimental Evaluation

Using an implementation of GCD (Geng et al. 2023) and of ASAp (Park et al. 2024), we compare GBFSGS against both in terms of convergence to the target distribution. For numerical stability and accuracy, we use log probabilities where possible.

We implement the GBFSGS algorithm by using the ASAp algorithm implementation of the ASAp paper (Park et al. 2024) as a base. While ASAp uses random samples, the update code of $\tilde{c}$ remains the same.

To use the algorithm 1, we need to find the best leaf from set L according to $\tilde{P}^{\theta}_{S_k}$. We represent L as a tree, where a path $w_{1:n}$ in the tree gives the values we stored associated with $w_{1:n}$ (such as $\tilde{c}_S(w_{1:n}), R(w_{1:n})$,...). To get the best leaf, we construct at every step a priority queue of the most probable sequences starting with the root and repeatedly select the top sequence of the priority queue, adding the children to the priority queue. This leads to a search taking at most $O(k \log(k))$ steps, but in practice much less, and is dominated by sampling from the LLM. A smarter way to search (down to $O(k)$ by a simple linear search, or even $O(\log(k))$) may be useful, but the potential impact is too low in our tests. This leads to having a tree containing at most $k * vocab\_size$ nodes at step k, the current number of samples.

Datasets. Following precedent works in Grammar Constraint Decoding and since we are trying to solve the same problems, we test GBFSGS on the same problems that ASAp was tested on (Park et al. 2024). Two sets of benchmarks come from Syntax-Guided Synthesis (SyGuS) (Alur et al. 2019) and another from the constituency parsing (CP) task (Geng et al. 2023). SyGuS involves generating a term that respects a logical specification and a context-free grammar (given to each algorithm). For CP, a context-free grammar is given to the model to generate a correctly parenthesized parse tree of English sentences. The specifications and grammars may be implicitly or explicitly defined. For more info on the specific problems, see ASAp (Park et al. 2024). Due to limited computing resources, we limit ourselves to the same randomly chosen problems from the benchmarks, 15 from SLIA (strings with linear integer arithmetic), 15 from INV-BV (loop invariant generation with bitvector arithmetic) and 6 CP problems. The evident tendency seen in the results will allow us to infer conclusions beyond those sets of problems. We use the open-source model Mistral 7B [12] on an NVIDIA RTX A6000 for all the tests.

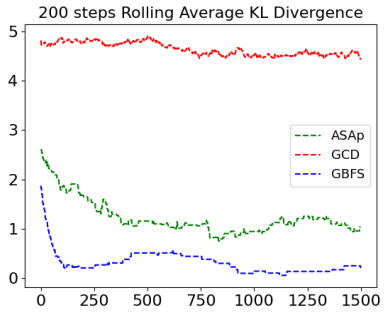

Figure 1: 200 steps rolling average KL Divergence on SLIA/name-combine-4-long

Measures. We use the same measures as were used for ASAp (Park et al. 2024) for comparison. We give each algorithm on each problem 2000 generations to estimate the target distribution. To assess the convergence to the distribution, two metrics are used, the KL divergence and the empirical expectation. We use the Kullback–Leibler (KL) divergence to estimate the distance between GCD, ASAp and GBFSGS to the target distribution at each iteration. The samples generated by each method can be used to make an estimate of the true target distribution, since the distribution of the grammar-constrained samples by the LLM is linearly proportional to the ideal GAD distribution. Then the quantity $KL(Q\|P)$ differs only by a constant from $KL(Q\|Q_{true})$. See (Park et al. 2024) for the derivation. A 200-rolling average is used for the KL-divergence plot. While the samples of GCD and ASAp are randomly taken based on the distribution at iteration k, GBFSGS samples cannot be used to estimate the distribution of the distribution obtained at step k. To get a sample, we generate an additional sample using GCD on the distribution at every step; note that this sample is for measurement only and does not affect the distribution being computed.

We also want to evaluate the behavior of the sampling done by the two algorithms. We plot a 200-rolling average of the log likelihood of the samples made by each algorithm, and also add a line to show what random samples made by GBFSGS would look like if they were made instead of the greedy samples.

To evaluate the effect of the two optimizations proposed, we record for each problem the full tree that would be generated by GBFSGS if we do not use the technique presented, as well as record the number of times a greedy sampling step is skipped.

Results. We show the convergence of each algorithm on different selected problems that exemplify the general behavior. The full list of figures is available in the supplementary materials. Figures 1, 2, 3 and respectively figures 5, 6, 7 are example problems where both ASAp and GBFS converge as the number of iterations increases. In each case, we can see that GCD exhibits variable levels of initial "fit-

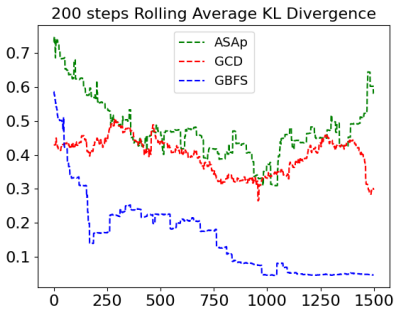

Figure 2: 200 steps rolling average KL Divergence on INV-BV/find-inv-eq-bvand-4bit

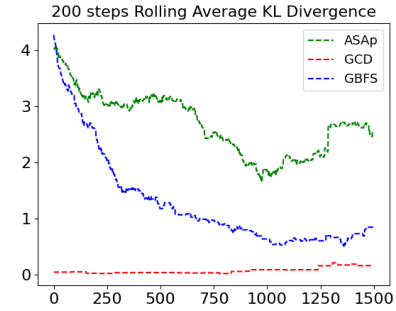

Figure 3: 200 steps rolling average KL Divergence on SLIA/firstname

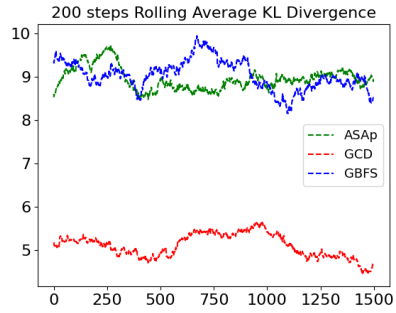

Figure 4: 200 steps rolling average KL Divergence on INV-BV/find-inv-bvugt-bvurem0-4bit

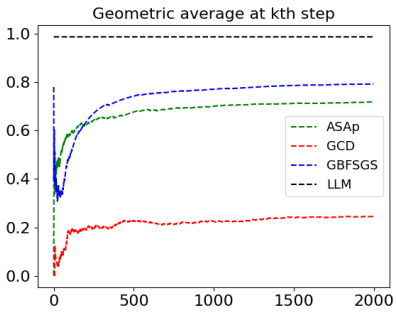

Figure 5: 200 steps rolling average KL Divergence on SLIA/name-combine-4-long

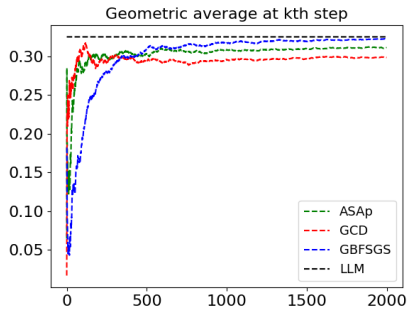

Figure 6: 200 steps rolling average KL Divergence on INV-BV/find-inv-eq-bvand-4bit

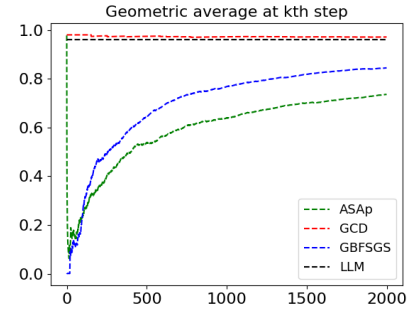

Figure 7: 200 steps rolling average KL Divergence on SLIA/firstname

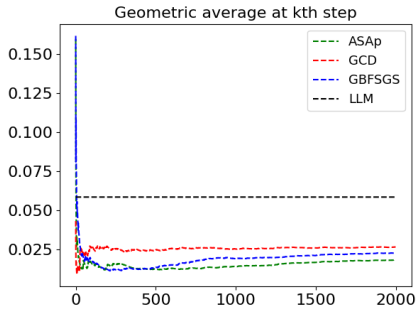

Figure 8: 200 steps rolling average KL Divergence on INV-BV/find-inv-bvugt-bvurem0-4bit

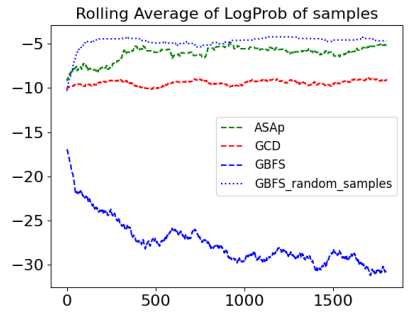

Figure 9: 200 Rolling average of samples log likelihood @t, plus a pseudo random sample of GBFSGS on SLIA/name-combine-4-long.

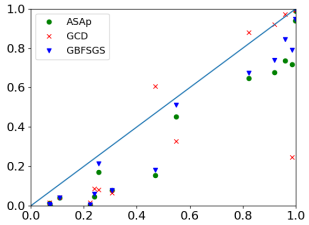

Figure 10: Final Expectation against theoretical expectation on SLAI problems

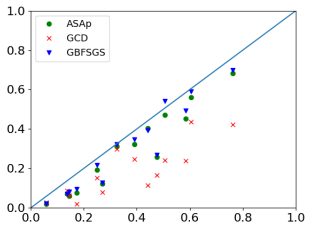

Figure 11: Final Expectation against theoretical expectation on INV-BV problems

ness" to the distribution, reflected by both the stagnating KL difference and empirical average. On almost every problem, when comparing ASAp to GBFSGS, GBFSGS shows a faster initial improvement, and converges to a better value. This behavior occurs on nearly every problems.

Figures 10, 11 and 12 show the behavior of each algorithm's samples, plus the behavior fully random samples for GBFSGS instead of greedy sampling would look like. As can be seen, in every case the behavior is very different between GBFSGS and ASAp. As the number of samples increases, GBFSGS samples get more surprising with respect to the LLM; the opposite behavior occurs for ASAp. Random samples that would be made by GBFSGS (in a similar fashion as ASAp) show the same behavior as ASAp, and in fact go higher even faster, slowing down when the KL divergence has shown convergence (see figure 9 and corresponding figure 1).

Discussions

As we aimed to do, GBFSGS improves upon the convergence rate of ASAp in the vast majority of the cases with the same amount of compute.

Every final expectation of $\tilde{P}_S^{\theta}(\cdot)$ (except for one problem) of GBFSGS is higher than ASAp. There are a few cases where we see GBFSGS perform better in alignment compared to GCD when ASAp performs worse than GCD, but it is pretty rare. If ASAp manages to converge, then GBFSGS will converge even faster. Furthermore, the convergence of GBFSGS seems more stable than that of ASAp.

One limitation to our approach is that if ASAp fails to converge at all early on (or very slowly), then neither will GBFSGS converge. In those cases, however, neither GCD nor any of the proposed algorithms will perform well (this was already noted by (Park et al. 2024)).

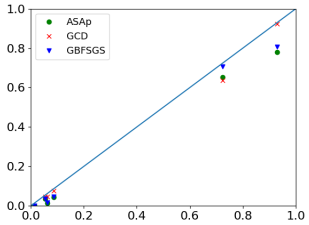

Figure 12: Final Expectation against theoretical expectation on CP problems

ASAp and GBFSGS samples' likelihood have very different behavior; the samples generated by ASAp get more likely as ASAp gets closer to the distribution, whereas GBFSGS generates samples with lower and lower likelihood. The fact that the original distribution ($P^{theta}(\cdot)$ scores the random samples higher as we align with the grammar indicates that most of the weight of the sentences with correct grammar is scored highly by the distribution. We suppose that this explains the slow down in convergence of ASAp; the closer the distribution is to the true distribution, the less likely randomly sampling finds an informative sample. We suppose that limiting the repetition of samples helps GBFSGS too.

It would be worth exploring other methods used to generate high-quality samples, such as beam search instead of greedy sampling, together with GBFS. The apparatus needed to efficiently store the samples may be too complex, but worth it if the convergence is improved. The compres sed representation is not needed for the improved convergence, so a hybrid method using random sampling and some form of greedy sampling may get an even better convergence.

Investigating why certain problems show very little improvement when others behave nicely may lead to a better approach for those cases. Some works (Melcer et al. 2024) point to problems where the probability of error (ungrammatical) is greater towards the end of the samples; this makes computing the distribution by only relying on precisely recording erroneous samples (like GBFSGS and ASAp) to be less practical in those cases than their proposed solution. Their solution, however, does not truly sample from the distribution, but exploring how GBFSGS and Approximately aligned decoding (AprAD) (Melcer et al. 2024) can be used together may be a potentially interesting next exploration.

Conclusion

We proposed a new algorithm to compute an approximation of the ideal distribution of the Grammar-Aligned Decoding (GAD) problem. We showed that it is guaranteed to converge eventually to the true distribution. We introduced a method that leverages the deterministic nature of greedy sampling to have a compact representation of the distribution being approximated that can still be used as is to sample with any techniques with low overhead. We demonstrated its ability to converge faster and smoother, as well as reach a lower difference with the target grammar-aligned distribution.

References

Alur, R.; Fisman, D.; Padhi, S.; Singh, R.; and Udupa, A. 2019. SyGuS-Comp 2018: Results and Analysis. arXiv:1904.07146.

Geng, S.; Josifoski, M.; Peyrard, M.; and West, R. 2023. Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning. In Bouamor, H.; Pino, J.; and Bali, K., eds., *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 10932–10952. Singapore: Association for Computational Linguistics.

Melcer, D.; Gonugondla, S.; Perera, P.; Qian, H.; Chiang, W.-H.; Wang, Y.; Jain, N.; Garg, P.; Ma, X.; and Deoras, A. 2024. Approximately Aligned Decoding. arXiv:2410.01103.

OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; Avila, R.; Babuschkin, I.; Balaji, S.; Balcom, V.; Baltescu, P.; Bao, H.; Bavarian, M.; Belgum, J.; Bello, I.; Berdine, J.; Bernadett-Shapiro, G.; Berner, C.; Bogdonoff, L.; Boiko, O.; Boyd, M.; Brakman, A.-L.; Brockman, G.; Brooks, T.; Brundage, M.; Button, K.; Cai, T.; Campbell, R.; Cann, A.; Carey, B.; Carlson, C.; Carmichael, R.; Chan, B.; Chang, C.; Chantzis, F.; Chen, D.; Chen, S.; Chen, R.; Chen, J.; Chen, M.; Chess, B.; Cho, C.; Chu, C.; Chung, H. W.; Cummings, D.; Currier, J.; Dai, Y.; Decareaux, C.; Degry, T.; Deutsch, N.; Deville, D.; Dhar, A.; Dohan, D.; Dowling, S.; Dunning, S.; Ecoffet, A.; Eleti, A.; Eloundou, T.; Farhi, D.; Fedus, L.; Felix, N.; Fishman, S. P.; Forte, J.; Fulford, I.; Gao, L.; Georges, E.; Gibson, C.; Goel, V.; Gogineni, T.; Goh, G.; Gontijo-Lopes, R.; Gordon, J.; Grafstein, M.; Gray, S.; Greene, R.; Gross, J.; Gu, S. S.; Guo, Y.; Hallacy, C.; Han, J.; Harris, J.; He, Y.; Heaton, M.; Heidecke, J.; Hesse, C.; Hickey, A.; Hickey, W.; Hoeschele, P.; Houghton, B.; Hsu, K.; Hu, S.; Hu, X.; Huizinga, J.; Jain, S.; Jain, S.; Jang, J.; Jiang, A.; Jiang, R.; Jin, H.; Jin, D.; Jomoto, S.; Jonn, B.; Jun, H.; Kaftan, T.; Łukasz Kaiser; Kamali, A.; Kanitscheider, I.; Keskar, N. S.; Khan, T.; Kilpatrick, L.; Kim, J. W.; Kim, C.; Kim, Y.; Kirchner, J. H.; Kiros, J.; Knight, M.; Kokotajlo, D.; Łukasz Kondraciuk; Kondrich, A.; Konstantinidis, A.; Kosic, K.; Krueger, G.; Kuo, V.; Lampe, M.; Lan, I.; Lee, T.; Leike, J.; Leung, J.; Levy, D.; Li, C. M.; Lim, R.; Lin, M.; Lin, S.; Litwin, M.; Lopez, T.; Lowe, R.; Lue, P.; Makanju, A.; Malfacini, K.; Manning, S.; Markov, T.; Markovski, Y.; Martin, B.; Mayer, K.; Mayne, A.; McGrew, B.; McKinney, S. M.; McLeavey, C.; McMillan, P.; McNeil, J.; Medina, D.; Mehta, A.; Menick, J.; Metz, L.; Mishchenko, A.; Mishkin, P.; Monaco, V.; Morikawa, E.; Mossing, D.; Mu, T.; Murati, M.; Murk, O.; Mély, D.; Nair, A.; Nakano, R.; Nayak, R.; Neelakantan, A.; Ngo, R.; Noh, H.; Ouyang, L.; O'Keefe, C.; Pachocki, J.; Paino, A.; Palermo, J.; Pantuliano, A.; Parascandolo, G.; Parish, J.; Parparita, E.; Passos, A.; Pavlov, M.; Peng, A.; Perelman, A.; de Avila Belbute Peres, F.; Petrov, M.; de Oliveira Pinto, H. P.; Michael; Pokorny; Pokrass, M.; Pong, V. H.; Powell, T.; Power, A.; Power, B.; Proehl, E.; Puri, R.; Radford, A.; Rae, J.; Ramesh, A.; Raymond, C.; Real, F.; Rimbach, K.; Ross, C.; Rotsted, B.; Roussez, H.; Ryder, N.; Saltarelli, M.; Sanders, T.; Santurkar, S.; Sastry, G.; Schmidt, H.; Schnurr, D.; Schulman, J.; Selsam, D.; Sheppard, K.; Sherbakov, T.; Shieh, J.; Shoker, S.; Shyam, P.; Sidor, S.; Sigler, E.; Simens, M.; Sitkin, J.; Slama, K.; Sohl, I.; Sokolowsky, B.; Song, Y.; Staudacher, N.; Such, F. P.; Summers, N.; Sutskever, I.; Tang, J.; Tezak, N.; Thompson, M. B.; Tillet, P.; Tootoonchian, A.; Tseng, E.; Tuggle, P.; Turley, N.; Tworek, J.; Uribe, J. F. C.; Vallone, A.; Vijayvergiya, A.; Voss, C.; Wainwright, C.; Wang, J. J.; Wang, A.; Wang, B.; Ward, J.; Wei, J.; Weinmann, C.; Welihinda, A.; Welinder, P.; Weng, J.; Weng, L.; Wiethoff, M.; Willner, D.; Winter, C.; Wolrich, S.; Wong, H.; Workman, L.; Wu, S.; Wu, J.; Wu, M.; Xiao, K.; Xu, T.; Yoo, S.; Yu, K.; Yuan, Q.; Zaremba, W.; Zellers, R.; Zhang, C.; Zhang, M.; Zhao, S.; Zheng, T.; Zhuang, J.; Zhuk, W.; and Zoph, B. 2024. GPT-4 Technical Report. arXiv:2303.08774.

Park, K.; Wang, J.; Berg-Kirkpatrick, T.; Polikarpova, N.; and D' Antoni, L. 2024. Grammar-Aligned Decoding. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 24547–24568. Curran Associates, Inc.

Park, K.; Zhou, T.; and D'Antoni, L. 2025. Flexible and Efficient Grammar-Constrained Decoding. arXiv:2502.05111.

Willard, B. T.; and Louf, R. 2023. Efficient Guided Generation for Large Language Models. arXiv:2307.09702.