LARGE LANGUAGE MODEL FOR COMBINATORIAL OPTIMIZATION BY LEVERAGING POLICY ADAPTATION AND HEURISTICS

Anonymous authors
Paper under double-blind review

ABSTRACT

Many techniques have been used to elicit reasoning within Large Language Models (LLMs), and several methods have been developed to solve NP-hard combinatorial problems (CPs). Certain methods relies on prompting strategies to sample solutions from the LLM, potentially finetuned on previous problems to improve the quality of the samples. If the budget allows it, multiple independent samples are obtained and the best solution is retained. Larger model correlate with higher quality solutions, but are also more costly to sample. Meanwhile, heuristics exists for many problems that generate good enough solutions for combinatorial problems very quickly. We present a method to progressively sample a solution from a finetuned model distribution via a series of heuristic solutions, each analyzed by the model to lead to the next heuristic solution. Furthermore, we show how to use the constraints of the problem to improve the samples as well as progressively build a policy from the solutions to improve the future samples. We demonstrate on various combinatorial problems (Binpack, Knapsack, Vehicle Routing Problem and Job-shop scheduling) and various models that the method improves the solutions that can be obtained by using simple heuristics, and that the method is superior to using the LLM simply as a sampling tool in terms of processing time, number of samples and solution quality. We also show further improvement obtainable by constructing a policy progressively from the previous samples and a the constraints of the problem. We show that the computed policy converges eventually to a distribution equivalent to the model with the constraints of the Combinatorial problem.

1 INTRODUCTION

Combinatorial Optimization (CO) problems arise in a wide range of real-world applications, including logistics, manufacturing, scheduling, network design, and resource allocation. Many of these problems are NP-hard, making exact optimization computationally infeasible for large instances. Classical heuristics are often extremely fast and can consistently produce feasible solutions, although they are usually unable to find the best solutions on the hardest problem instances.

Recent advances in Large Language Models (LLMs) have demonstrated that these models can solve a surprisingly broad range of reasoning and planning tasks. This has motivated growing interest in applying LLMs to combinatorial optimization by representing optimization problems as text generation tasks. After fine-tuning on problem-specific datasets, LLMs can generate complete candidate solutions through autoregressive decoding, often achieving competitive performance without explicitly designed search algorithms. Existing approaches generally improve solution quality by sampling multiple independent solutions with prompting techniques or fine-tuning strategies and selecting the best candidate according to the objective function. While larger models typically produce better solutions, they also incur significantly higher inference costs, limiting the number of samples that can be explored under a fixed computational budget.

A key observation is that autoregressive token generation is inefficient for combinatorial optimization. Most tokens generated by an LLM correspond to deterministic structural elements or intermediate calculations rather than actual optimization decisions. Moreover, many combinatorial problems

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

admit efficient heuristics that can rapidly complete a partial solution into a feasible one. Although these heuristic solutions are usually suboptimal, they frequently resemble the solutions that an LLM would generate and therefore contain valuable information about future decoding steps.

In this paper, we propose *Constrained Heuristic Speculative Sampling* (CHSS), a framework that combines heuristic search with speculative decoding to accelerate and improve LLM-based combinatorial optimization. Instead of using a smaller neural model to speculate future tokens, we employ problem-specific heuristics to construct complete candidate solutions from partial solutions. We further incorporate problem constraints directly into the decoding process, ensuring that invalid decisions are never sampled, improving the model's probability distribution over feasible moves.

Beyond Constrained Heuristic Speculative Sampling, we introduce a lightweight policy cache as a tree that store the early choices, skipping duplicate samples. We sample directly from this tree to generate a partial solution to be completed with CHSS. An additional adaptation of the policy progressively builds an approximation of the constrained decoding policy from previously evaluated solutions. This allows late constraints to affect early choices.

We evaluate the proposed framework on four representative combinatorial optimization problems: Traveling Salesman Problem (TSP), Bin Packing, Knapsack, and Job Shop Scheduling (JSSP). Experiments using fine-tuned Llama 3.1 and Qwen3 models demonstrate that CHSS consistently outperforms standard LLM sampling under the same inference budget. Our approach achieves lower optimality gaps, always generates feasible solutions through constrained decoding, substantially reduces the number of model generations required, and benefits further from progressively adapting its sampling policy.

2 PRELIMINARIES

2.1 COMBINATORIAL PROBLEMS

In this section we introduce the definitions of the various Combinatorial Optimization (CO) problems.

In this section we review what are Combinatorial problems, and the definitions of some typical examples. Our approach works on any Combinatorial problems so long as it we can represent the problem to train an LLM to generate solutions, and define a heuristic for the problem.

General Formulation For a set of solution $\mathcal{S}$, a discrete set of feasible solutions $\mathcal{S}^* \subseteq \mathcal{S}$ and a function $f : \mathcal{S} \to \mathcal{R}$ then the general formation of a Combinatorial Optimization Problem is

$$s^* = \arg\min_{s \in \mathcal{S}^*} f(s)$$

We assume a solution S can be written as a composition of several subsequent choices, $S = s_1 s_2 ... s_n$, and each choice is from a finite pool of choices $s_i = o_1 o_2 ... o_K$. This correspond neatly to most Combinatorial Optimization problems.

In this paper, we define a heuristic $\mathcal{H}(\cdot)$ which for a partial solution $S^* = s_1 s_2 ... s_j$ returns (usually quickly) a (usually sub-optimal) complete solution $S^* = s_1 s_2 ... s_j s_{j+1} ... s_n$.

Traveling Salesman Problem (TSP) The Traveling Salesman problem seeks to compute the shortest route going through a list of N cities exactly once (a tour), given a distance $D(i, j)$ between city i and city j, which means computing

$$\min \sum_{i=1}^{N} D(\pi_i, \pi_{i+1}),$$

where π is a permutation of the N cities.

A simple and efficient heuristic for TSP is the to repeatedly select the closest city to the current location (Rego et al., 2011).

Bin Packing Problem (Binpack) The Bin Packing problem seeks to store all N items in the least amount of bins, where each items has an associated weight w_i and each bin B_i can carry at most W weight. Formally, we seek to min K such that

$$\sum_{i \in B_j} w_i \leq W \forall j, \bigcup_{j}^{K} B_j = \{1, 2, ..., N\}, B_j \cap B_{j'} = \varnothing \forall j \neq j'.$$

The Bin Packing problem has many heuristics. A simple one to implement is called First Fit Decreasing (Dósa, 2007). After sorting the items in decreasing order of weight, each item is fit into the first bin that can contain it. If K^* is the minimum number of bin possible, First Fit Decreasing is guaranteed to find a solution at most $\frac{11}{9}K^* + \frac{6}{9}$.

Knapsack Problem (Knapsack) The (0-1) Knapsack problem seeks to maximize the value of the items that can be carried in a weight limited set. Every N item as an associated weight w_i and value v_i, and we can select items with at most W weight total. That is, $\max \sum_i^N v_i x_i$, such that $\sum_i^N w_i x_i \leq K$.

A greedy heuristic for the knapsack problem provides a solution that has at least $\frac{1}{2}$ the value of the best solution (Dantzig, 1957). The *efficiency* of an item o_i is defined by the ratio $eff(o_i) = \frac{v_i}{w_i}$. Sorting the items by decreasing order of efficiency, two solutions are created. The first by repeatedly adding the most efficient item that fits, and the second is constructed uniquely from the first item that did not fit during the precedent procedure. The best one of the two selected as a solution.

Job Shop Scheduling Problem (JSSP) The Job Shop Scheduling Problem involves J jobs and K machines, where each job must be operated on by each machine. This gives $J * M$ operations $O_{j,k}$, and each operation runs for a set time and cannot be stopped, and each job must be operated on by machine i before $i + 1$. The goal is to minimize the completion time, that is the time the last job is completed, $\min C_{\max} = \max_j C_{j,l_j}$, such that:

$$S_{j,k+1} \geq C_{j,k} \text{ and } S_{j,k} \geq C_{j',k'} \text{ or } S_{j',k'} \geq C_{j,k},$$
$$\forall (j,k) \neq (j',k'), M_{j,k} = M_{j',k'}$$

A good heuristic is called Most Work Remaining (Brandimarte, 1993; Montazeri & Wassenhove, 1990). The method is to always select the job with the most remaining total work to do to avoid bottlenecks.

2.2 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.

In this paper we use small LLMs such as Llama 3.1 (8B parameters) and Qwen 3 (8B parameters) to be able to quickly finetune them on new problems, however the method described here works with any model that can be finetuned and for which the token distribution can be obtained.

There is no guarantee that almost every objects or actions in different combinatorial problems need multiple tokens to be represented. In order for the proposed method to work however, we need to represent each move/action/object with a single token. We demonstrate how to properly finetune the models to achieve that later in the paper.

3 Related Works

Various works have introduced techniques to improve the performance of Large Language models.

3.1 SPECULATIVE SAMPLING

By using a smaller model, Speculative Sampling (Stern et al., 2018; Leviathan et al., 2023) shows how to accelerate the inference speed of LLMs. While traditional method of generation involves generating one token at a time, Speculative decoding allows the generation of potentially several token at a time. Because one forward pass over K tokens is less costly than K sequential forward passes, it uses a cheaper model M_q (or any alternative source of prediction token) to predict K token in advance, and then use the main model M_p to check in parallel the tokens. If the first token was successfully predicted, we can immediately check if the second was also correctly predicted without having to pay for the additional generation.

If the cost of sampling K token from M_q is negligible, and the distribution of generation of M_q is similar enough to M_p, then multiple tokens are predicted for the cost of a single forward pass consistently, accelerating the generation.

Selecting K is important, since every subsequent generation is wasted if earlier token were discarded. In the case of typical text generation, $K = 3..5$ show the best performance, while code generation (which is more predictable), $K = 5..7$ give the best results.

In this paper we do not directly speculate on the tokens, and we actually consider a fully complete speculative generation instead of a few token in advance.

3.2 CONSTRAINED GENERATION

Large Language Model produce a distribution over every token of their vocabulary, regardless of the reasonableness of the token in the current context. The training of the model makes the probability of very out-of-context token highly unlikely, however certain tokens can still be attributed erroneously high probability. In the context of Combinatorial Problems, the generation of solutions by the LLM involves training the LLM on correct solutions to the problem, which ensure that the structure of a solution is correctly generated. However, it can happen that the solution generated is structurally correct (e.g. it represent a path in TSP) and still be illegal by containing incorrect moves. It is hard for the model to associate to them extremely low probability. The longer the solutions that need to be generated, the higher the probability that at least one illegal move is generated, leading to an incorrect solution.

A simple solution is to set to zero the probabilities of the tokens that are impossible, leading to a method called Grammar Constrained Decoding (Geng et al., 2024). The Grammar Aligned Decoding paper (Park et al., 2024) shows that this introduces a bias the generation of the model, and propose Generative Aligned Decoding to use multiple samples to progressively generate a policy that will correctly sample from the constrained model.

In this paper the solutions are not directly generated by the model, and therefore there is no need to compute a correct policy over the tokens. Instead, we re-purpose the equation of GAD to compute the correct distribution over the moves of the problems, by propagating the probability given to illegal moves to other move instead of tokens.

3.3 CONSTRAINED DECODING WITH SPECULATIVE LOOKAHEADS

Nakshatri et al. (2024) present a method that uses Speculative decoding to improve the efficiency of the alignment to constraints. In the context of the paper, they seek to align the generation of the llm to human preference, and improve the speed of generation by using speculative decoding, before scoring the generation to accept or reject them. The score is generated by a task specific reward function, which assess the adherence to the constraints, which contained both hard (computable) and soft (judged by a llm) constraints.

This paper differ from ours as its application is quite removed from the generation of solutions to combinatorial problems, and the speculations and verification are made using a draft/verification models, which makes the inference much slower.

3.4 ACCORD REPRESENTATION

Autoregressive Constraint-satisfying Generation for Combinatorial Optimization with Routing and Dynamic attention (ACCORD) (Abgaryan et al., 2025) shows how to dynamically adapt a model to various Combinatorial Problems, and introduce a more effective representation of solutions for LLM.

LLM often perform better at task when they outputs intermediate steps before reaching the solutions, for example in Chain-of-Thought prompting (Wei et al., 2023). A similar principle has been shown in the context of Combinatorial Problems, where augmenting the output representation with extraneous information improves the proposed solutions significantly, for example by having the LLM regularly output the current size of a bin or the remaining time for a job (Abgaryan et al., 2025) (we call such a representation the "ACCORD" representation).

Similarly, incomplete or incorrect intermediary steps, for example assuming that a bin has more space than in actuality, lead the model to output incorrect moves. We use grammar constraint to ensure that the model outputs mathematically correct intermediary steps to improve the final output solutions.

4 MAIN CONTRIBUTIONS

Speculative Decoding (Leviathan et al., 2023) predicts the tokens in advance to increase the speed of generation manyfold.

Instead of relying on a smaller version of the model, we can take advantage of heuristics for the combinatorial problems to quickly obtain solutions for the problems. Given a solution, we can obtain the probabilities that the model associates for each tokens, and by extension the probability that the model associates with every move.

Because the logical representation of the solutions for the combinatorial problems translate into a large number of tokens, the potential benefits are manyfold superior to standard speculative decoding. The benefits in speeds depends on how accurately the speculation matches with the LLM (Leviathan et al., 2023). The model is trained to output the correct structure, and therefore will gives a very high probability (almost always 100%) to tokens that are here to modelize the solutions. ACCORD (Abgaryan et al., 2025) uses a more expansive representation for the solution in order to leverage the LLM semantic understanding which demands more token to be generated, many of which are forced.

Based on this, some speedup can be obtained by predicting simply the tokens that are forced with a trivial completion.

However, we can improve the speedup even more by directly predicting full solution with a heuristic, given that the model follows roughly the same heuristic. For many combinatorial problem heuristic exists. Full solutions obtained through heuristics are much faster to obtain than by sampling the models, and have the advantage that we can also check how good the output is as a solution for the problem.

An added bonus is the generation of additional solutions (obtained by the heuristic) that can be tested per full generation.

4.1 HEURISTIC SPECULATIVE SAMPLING FOR COMBINATORIAL PROBLEMS

The natural application of speculative sampling with heuristics is to convert the heuristic solutions into the structures expected by the model (the one it was finetuned to follow), and then use those as the speculative samples. The expectation of performance of speculative sampling can then be analysed by dividing the tokens that needs to be generated by the model in three groups;

1. structural tokens: tokens that are forced by the structure of the solutions;
2. calculation tokens: tokens that represent the results of calculations that needs to be performed;
3. selection tokens: tokens that represent a choice in the solution made by the model.

If the model was properly trained, it will very quickly learn to output the structural tokens with probability 1.0, and therefore speculative sampling will work perfectly for them. The second ones are more problematic; LLMs are known to be able to perform mathematical calculation very well. If the model is wrong, then the speculation will stop, however that is not what we want. The calculation are very important to give local context to the LLM to what selection it can make (e.g. the remaining weights in the knapsack problem), so we will prefer to avoid any such mistake by the model regardless of its preference.

Lastly, the selection tokens can be seen as the crux of the problem, where the probabilities really matters. Again, if the model were to choose an impossible selection (e.g. going twice to the same city in TSP), we would prefer to avoid that.

4.2 Constrained Heuristic Speculative Sampling

Constrain generation (Geng et al., 2024; Park et al., 2024) remove the possibility for the model to generate illegal tokens given a grammar. We propose Constrained Heuristic Speculative Sampling (CHSS), which efficiently use constrained generation with Heuristic Speculative Sampling.

In Constrained Heuristic Speculative Sampling, we modify the probabilities associated with each tokens to force the model to respect the grammar of generation. In our case this means setting the structural and calculation tokens to 1.0. Because they are certain, and the constructed heuristic also follows this grammar, we can always accept any such token during generation. CHSS pseudo-code is equivalent to algorithm 1 without the policy elements.

Naive constrained sampling is unable to properly take into account far away constraints, so an additional adaption can be used to improve the sampling (Park et al., 2024). We use this method later on to improve the policy presented next.

4.3 Model-Based Partial Generation through Policy

Constrained Heuristic Speculative Sampling is able to efficiently sample solution from the grammar constrained LLM, especially when the moves are represented by a unique selection tokens.

When generating highly moves (e.g. adding one last item to a bin in binpack, where only that item would fit), we can expect the model to be almost certain of the move, and the heuristic to also contain that move. In such case, the move will be properly speculated. However if a choice is not certain, the speculation will be incorrect sometime.

We propose a simple and effective method to bypass the problem of aligning the speculation to the LLM distribution by progressively building a simple policy that can be sampled quickly without needing to sample the LLM or use heuristic speculation. The method allows to quickly build a starting partial solution where most early uncertain choices are already made according to the LLM distribution.

The policy is built by storing the constrained probability at each move according to the model in a tree. Because moves of CO can represented by multiple tokens in certain cases, imposing constraints to the generation and obtaining the probability distribution over the moves is much less efficient, we finetune the models to output every moves as a single token.

An additional benefit is that we never complete the same partial solution using a heuristic twice. After a solution is generated, sampling it again could only happen through this policy, which is much faster than sampling the LLM. The policy is stored in a tree, where each branch is a choice.

4.4 Integrating Rules into the Cached Policy

As shown in the GAD (Park et al., 2024), to properly take into account the constraints of the grammar we need to rebalance the probabilities when the model predict illegal moves.

The exact rebalancing of GAD can be reused here. Many of the token do not need to be stored however, since they are given probability 1 by the constraints.The final version of the method proposed here, CHSS with Policy Adaptation, is shown in pseudo-code in algorithm 1.

Algorithm 1 CHSS w/ Policy Adaptation

Initialize $L \leftarrow \{root\}, \tilde{c}_S(\cdot) \leftarrow 1$
for $k \leq K$ **do**
  $S_{curr} \leftarrow$ sample leaf from L according to $P^{\theta'}(\cdot)$ ▷ Policy Caching
  while S_{curr} is not complete **do**
    $S_{heur} \leftarrow$ completion of S_{curr} according to a heuristic
    if $value(S_{heur}) < threshold$ **then**
      Terminate
    Obtain the distributions $P^{\theta}(s_1), P^{\theta}(s_2), \ldots, P^{\theta}(s_n)$ in parallel.
    Update L and $\tilde{c}_S(\cdot)$ according to GAD ▷ Policy Adaptation
    $S_{curr} \leftarrow$ Speculative Sampling(S_{curr}, S_{heur})
  if $value(S_{curr}) < threshold$ **then**
    Terminate

5 Experimental Evaluations

5.1 Dataset & Training Details

We test the methods on a the dataset provided in Abgaryan et al. (2025). The datasets consist of input-output pairs for each combinatorial problems (we selected TSP, JSSP, Knapsack and Binpack) of various sizes. We use the ACCORD representation to benefit from the improvements in answer quality. We refer to ACCORD (Abgaryan et al., 2025) as "sampling", since we skip the dynamic selection of the LoRA of ACCORD. Note that this advantages the ACCORD method, as there is no case for the method to use the wrong LoRA for inference.

CHSS works if every move is associated with a single token, since we want to obtain from the LLM the complete distribution over the moves at each step in one forward call. A move could be given a unique id as a number, but every model handles the tokenization of numbers differently. While some tokenizer assign numbers as large as 3 digits to a single token, many LLMs (e.g. qwen3, Gemma4) uses tokenizers that represents numbers digit per digit. We extend the list of tokens to assign a new token per id, for the maximum number of id necessary for the hardest problems We finetune LLama 3.1 (8B) model and Qwen3 (8B) to solve the problems. We increase the size of the embedding to accommodate the 300 new tokens, since the largest problems of the dataset need about 200 of them. We initialize the new tokens embeddings as the mean of the existing vocabulary embeddings (Hewitt, 2021). During training, we randomize the id assignation to train every id equally. Otherwise, we follow the training done by the ACCORD paper (Abgaryan et al., 2025); We split the training dataset into a train and evaluation dataset and we train a Rank-Stabilized Low-Rank Adaptation (RSLoRA) (Kalajdzievski, 2023) with rank $r = 64$ and $\alpha = 64$. We select the parameters with the best performance on the evaluation set after training plateaus.

We evaluate the optimality gap between the best possible value and the best value obtained among the sampled solutions. The optimality gap is calculated as $Gap = \frac{\text{best found} - \text{optimal}}{\text{optimal}}\%$. Each method is given the same sampling budget of 60 samples per problems. For a more general view of LLM methods on Combinatorial problems, Abgaryan et al. (2025) compared ACCORD (the method it introduces) against various techniques and showed a very large improvement over other methods.

6 Results

6.1 Comparisons with the previous best method

Table 1 shows the results comparing CHSS with Policy Adaptation against ACCORD sampling. Overall, CHSS with policy adaptation dominates sampling on every problems type and size.

For many problems, especially the hardest, sampling may fail to produce a valid solution within the budget. This never occurs with CHSS, since the constraints on the problem ensures that a solution is always valid. Even ignoring those problems, CHSS with Policy adaptation shows that it consistently achieves a lower Gap.

Table 1: Comparison of the mean optimality gap (%) achieved by Sampling and CHSS with Policy Adaptation across different problems, averaged by problem size. Lower is better. using the same finetuned Llama 3.1 (8B) model, each with a budget of 60 samples per problems. The parenthesis (p/P) indicates that the algorithm found feasible solutions for only p of the P problems within the allocated budget.

Method	Size	Knapsack	BinPack	JSSP	TSP
Sampling (ACCORD)	5	0.7	**0.0**	0 (56/94)	0.59
	8	2.09	**0.0**	2.75 (78/96)	1.02
	10	2.29 (44/45)	-	2.06 (45/94)	1.51
	12	4.62 (43/45)	0.918 (49/50)	3.16 (56/99)	1.68
	15	4.09 (41/45)	1.15	5.97 (68/97)	2.32
	20	5.27 (37/45)	2.79 (45/50)	3.91 (43/85)	5.3 (95/100)
	50	9.74 (34/45)	0.0 (16/50)	-	-
	100	7.94 (38/45)	0.0 (3/50)	-	-
CHSS w/ Policy Adaptation	5	**0.0**	**0.0**	**0.0**	**0.0**
	8	**0.0**	**0.0**	**0.20**	**0.001**
	10	**0.0**	-	**0.08**	**0.0**
	12	**0.19**	**0.0**	**0.38**	**0.09**
	15	**0.20**	**0.0**	**0.45**	**0.12**
	20	**0.20**	**0.33**	**0.49**	**1.38**
	50	**0.1**	**0.0**	-	-
	100	**0.05**	**0.0**	-	-

6.2 Impact of Constrained Samples and Heuristics Samples

The presented method differs from previous ones by the fact it generates potentially multiple solutions per sampling of the model, and that it constraints the samples. We empirically evaluate the impact of Constraint Generation compared to standard sampling, as well as the additional effect of having the heuristics samples on the results. We compare the methods on the problems of TSP, using the trained Llama3.1 8B model, each with a budget of 60 samples. For comparison, we also show how a single use of the described TSP heuristic performs.

Table 2 Shows the results. Constrained sampling dominates sampling, especially in more complex problems where sampling fail to find a solution for 5 of the 100 problems of size 20. If we count the heuristic samples that are obtained during the speculative sampling of CHSS, we see a very large amelioration for the largest problems over only Constrained Sampling.

Table 3 Shows the average number of heuristics that were generated per constrained samples of the model. The lower it is, the more efficient the sampling is (1 sample at best, #problem size at worst), however we can note that any additional heuristic generated gives a new solutions, potentially improving the optimality gap. It is also important to note that different heuristic could have very different behavior. The worst average ratio of heuristic sample per move is JSSP for size 15, where nearly every 2 choices requires a new generation. CHSS w/ PA is very efficient on Binpack, requiring almost no than a few heuristics for the hardest problems. The consequence is that for the largest problems, solves them on average 156 times faster than simply sampling.

CHSS w/ PA on TSP and JSSP worked relatively well. For knapsack on the largest problems, one heuristic sample could generate 9 moves on average of the model. However that number is very variable; the worst outcome was 84 heuristic samples required to generate one sample of the problem. In that case, straightforward sampling would have been faster. However, the samples in that case are constrained to the grammar of the model, which are of higher quality.

6.3 Impact of Policy Caching and Adaptation

We test the effect of the Policy Adaptation on two different models to see if it generalizes. Table 4 shows the results. Policy Caching theoretically impact on the results is that to sample the same

Table 2: Comparison of the Optimality gap (%) achieved on TSP by a single a single heuristic call, sampling, Constrained Sampling and CHSS. Lower is better.

TSP, # cities	5	8	10	12	15	20
Heuristic only	4.33	7.71	10.8	13.4	13.6	16.1
Sampling	0.591	1.02	1.51	1.68	2.32	5.3 (95/100)
Constrained Sampling	0.006	**0.**	0.01	0.121	0.277	4.21
CHSS	**0.**	0.001	**0.**	**0.09**	**0.12**	**1.38**

Table 3: Average number of heuristic speculated per true sample of the model by CHSS w/ PA. The maximum possible for each problem is the associated size

Size	Knapsack	BinPack	JSSP	TSP
5	1.02	1	1.1	1.74
8	1.27	1.03	2.46	2.49
10	1.13	-	3.33	2.91
12	1.8	1.2	4.69	3.73
15	1.61	1.11	7.38	4.13
20	3.24	1.61	8.33	7.12
50	3.43	1.07	-	-
100	12.9	1.15	-	-

solutions twice (heuristic or otherwise), and the results shows no to a small improvements. The other problems the results are not certain.

Table 4: Comparison of the mean optimality gap (%) achieved by Sampling and CHSS and the two version of policy caching. Lower is better.

Llama 3.1 (8B)	Knapsack	BinPack	JSSP	TSP
CHSS	0.105	**0.047**	0.615	0.313
CHSS w/ Policy Caching	0.0881	**0.047**	0.606	0.298
CHSS w/ Policy Adaptation	**0.0735**	**0.047**	**0.534**	**0.265**
Qwen3 (8B)				
CHSS	**0.0501**	**0.047**	0.804	0.285
CHSS w/ Policy Caching	0.0548	**0.047**	**0.718**	**0.242**
CHSS w/ Policy Adaptation	0.0518	**0.047**	0.739	0.266

7 Conclusion

We introduced Constrained Heuristic Speculative Sampling, as well as an associated policy construction. We empirically evaluated the performance on the technique on various problem compared to previous works, and showed that CHSS w/ PA performs extremely well. We evaluated the effect of the different part introduced in this paper, and showed that constrained generation and heuristic speculation had large effects, and the policy construction had much smaller but probably positive effects.

8 Limitations and Future Work

The need to train new tokens may restrict the kind and difficulty of problems that can be efficiently solved. In future work we want to explore solutions that efficiently uses speculative sampling together with constrained generation, without requiring to train new bespoke tokens.

REFERENCES

Henrik Abgaryan, Tristan Cazenave, and Ararat Harutyunyan. Accord: Autoregressive constraint-satisfying generation for combinatorial optimization with routing and dynamic attention, 2025. URL https://arxiv.org/abs/2506.11052.

Paolo Brandimarte. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993. URL https://api.semanticscholar.org/CorpusID:33443952.

George B. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–277, 1957. ISSN 0030364X, 15265463. URL http://www.jstor.org/stable/167356.

György Dósa. The tight bound of first fit decreasing bin-packing algorithm is $\text{ffd}(i) \leq 11/9\text{opt}(i) + 6/9$. In Bo Chen, Mike Paterson, and Guochuan Zhang (eds.), *Combinatorics, Algorithms, Probabilistic and Experimental Methodologies*, pp. 1–11, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-74450-4.

Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning, 2024. URL https://arxiv.org/abs/2305.13971.

John Hewitt. Initializing new word embeddings for pretrained language models. https:/nlp.stanford.edu/~johnhew//vocab-expansion.html, 2021.

Damjan Kalajdzievski. A rank stabilization scaling factor for fine-tuning with lora, 2023. URL https://arxiv.org/abs/2312.03732.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

M. Montazeri and L. N. Van Wassenhove. Analysis of scheduling rules for an fms. *International Journal of Production Research*, 28(4):785–802, 1990. doi: 10.1080/00207549008942754. URL https://doi.org/10.1080/00207549008942754.

Nishanth Nakshatri, Shamik Roy, Rajarshi Das, Suthee Chaidaroon, Leonid Boytsov, and Rashmi Gangadharaiah. Constrained decoding with speculative lookaheads. *arXiv preprint arXiv:2412.10418*, 2024.

Kanghee Park, Jiayu Wang, Taylor Berg-Kirkpatrick, Nadia Polikarpova, and Loris D' Antoni. Grammar-aligned decoding. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 24547–24568. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/2bdc2267c3d7d01523e2e17ac0a754f3-Paper-Conference.pdf.

César Rego, Dorabela Gamboa, Fred Glover, and Colin Osterman. Traveling salesman problem heuristics: Leading methods, implementations and latest advances. *European Journal of Operational Research*, 211(3):427–441, 2011. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2010.09.010. URL https://www.sciencedirect.com/science/article/pii/S0377221710006065.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise parallel decoding for deep autoregressive models, 2018. URL https://arxiv.org/abs/1811.03115.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.