Neural Sequence Generation with Constraints via Beam Search with Cuts: A Case Study on VRP

Pouya Shati[1,2], **Eldan Cohen**[3], **Sheila McIlraith**[1,2]
[1]Department of Computer Science, University of Toronto, Toronto, Canada
[2]Vector Institute for Artificial Intelligence, Toronto, Canada
[3]Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada
pouya@cs.toronto.edu, ecohen@mie.utoronto.ca, sheila@cs.toronto.edu

Abstract

In recent years, neural sequence models have been applied successfully to solve combinatorial optimization problems. Solutions, encoded as sequences, are typically generated from trained models via beam search, a search algorithm that generates sequences token-by-token while keeping a fixed number of promising partial solutions at each step. In this paper, we explore the problem of augmenting beam search generation with the enforcement of requirements—hard constraints that any generated solution must adhere to. We propose a hybrid approach, by encoding the requirements in the form of a constraint satisfaction problem (CSP) and iteratively solving the CSP to cut any partial solution within the beam search that is incapable of satisfying the requirements. We study this problem in the context of vehicle routing problems (VRP) further augmented with capacity-related or temporal requirements. We experimentally show that cuts often allow us to satisfy the requirements with negligible impact on solution quality. Without the use of cuts, beam search is shown to be exponentially less likely to satisfy the requirements as the length of the solution increases and/or the requirements are strengthened.

1 Introduction

Neural sequence models, including recurrent neural networks (Medsker and Jain 2001) and transformers (Vaswani et al. 2017), have been successful in solving combinatorial optimization problems (Bello et al. 2016; Kool, van Hoof, and Welling 2018). While typically applied to natural language tasks, neural sequence models can be trained to generate structured solutions to combinatorial problems either via supervised learning (Vinyals, Fortunato, and Jaitly 2015) or more recently via reinforcement learning (Lafleur, Chandar, and Pesant 2022; Kool, van Hoof, and Welling 2018).

A neural sequence model is commonly decoded through iterative next token prediction, generating the solution one token at a time (Sutskever, Vinyals, and Le 2014; Bahdanau, Cho, and Bengio 2015). A beam search procedure can further expand iterative production to generate a set of solutions rather than one (Freitag and Al-Onaizan 2017; Cohen and Beck 2019). Beam search works well with most sequence generation tasks. Combinatorial tasks, however, often involve strict requirements that are not guaranteed to be satisfied by approaches that purely optimize predictive scores such as beam search (Cohen and Beck 2021). Depending on the application, requirements may reflect safety or liveness constraints, guardrails to ensure appropriate behavior, customizing constraints that reflect laws in different jurisdictions, or they may include physical or temporal constraints that capture properties of the domain. Satisfying such requirements usually needs a global perspective and meticulous reasoning and thus does not suit a statistical iterative approach to generation, even if done in large quantities.

A requirement on the neural sequential generation output can often be formulated as a constraint satisfaction problem (CSP) (Brailsford, Potts, and Smith 1999). A CSP is an NP-hard problem with the goal of finding values for variables to satisfy a finite number of declarative constraints. Examples of CSP paradigms include constraint programming (CP) (Apt 2003), integer programming (IP) (Wolsey 2020), and boolean satisfiability (SAT) (Gong and Zhou 2017).

We present a modular framework to combine neural sequence generation with a CSP requirement. Our approach can be applied to any pre-trained unconstrained model and guarantees requirement satisfaction. Although a new model can be trained or fine-tuned towards satisfying requirements while optimizing the objective, such approach is time-consuming, requires a large amount of specific data for each singular or combination of requirements, and is not guaranteed to satisfy the requirement for each generated solution.

For the purposes of this paper, we focus on Vehicle Routing Problems (VRP) as a case study for our approach. Neural sequence models have been successful in solving VRPs (Kool, van Hoof, and Welling 2018) but only integrate myopic constraints that do not require advanced reasoning. We solve new VRP variants by combining the prediction model of a simpler variant with requirements encoded as CSPs.

The contributions of this paper include the following:

1. We introduce *beam search with cuts* to solve neural sequence generation when solutions must adhere to a requirement. Cuts prevent the expansion of partial solutions that are incapable of satisfying the requirement.
2. We encode the requirement as a constraint satisfaction problem. We introduce bin packing and regular language acceptance as two types of requirements that find application in multiple settings. We further show how they can

be implemented as cuts by enforcing adherence to a given partial solution and solving the instances using IP or SAT.

3. We use beam search with cuts to solve three VRP variants. Specifically, we combine an existing pre-trained neural model with our two requirement types.
4. We experimentally show that beam search with cuts enables satisfaction of a requirement without significant cost to quality in short runtime. Moreover, we show that cuts allow us to satisfy significantly harder requirements until infeasiblity is reached. Furthermore, we show that beam search with cuts scales exponentially better when increasing the solution length or the number of constraints. Lastly, we show that an incremental solving of the CSP significantly lowers the time spent by the solver.

While our contributions are conveyed and evaluated in the context of VRPs (an important application area), the methodology for imposing requirements on solutions is general and can be applied in a diversity of application settings.

2 Preliminaries

A neural sequence model generates solutions using a given alphabet referred to as the tokens Σ. The set Σ^* contains all finite sequences of tokens and the set $\mathcal{P}(\Sigma)$ contains all probability distributions over tokens.

2.1 Sequence Generation

The task of generating a solution sequentially is often carried out by a local endeavour that selects and appends the next token to an existing sequence called the partial solution (Harshvardhan et al. 2020). The local expansion is guided via a function that predicts the appearance of the next token.

Definition 1 (Next Token Prediction Function) *Given a set of tokens Σ and $\mathcal{P}(\Sigma)$ representing the set of all probability distributions over Σ, a next token prediction function $p : \Sigma^* \rightarrow \mathcal{P}(\Sigma)$ assigns probabilities to tokens for occurring next after observing a sequence of existing tokens as the partial solution.*

Next token prediction is often accompanied by a filtering function that maintains solution validity through deciding what subset of tokens can appear next at each step. Only solutions that respect the filtering function are considered throughout the paper. We refer to a solution x as *complete* when a given termination condition is met.

Definition 2 (Sequence Score) *Given a set of tokens Σ and a next token prediction function p, the score $\theta_p(.)$ of a partial solution $x = x_1, x_2, ..., x_k \in \Sigma^*$ is defined as the joint probability of its tokens according to p, i.e., $\theta_p(x) = \prod_{i=0}^{k-1} p(x_1, ..., x_i)[x_{i+1}]$ where $p(x)[x_{i+1}]$ represents the probability assigned to x_{i+1} in the distribution specified by $p(x)$.*

Definition 3 (Sequential Decoder) *Given a set of tokens Σ and a next token prediction function p, a sequential decoder ϕ aims to find a complete solution $x \in \Sigma^*$ with maximum score by iteratively extending partial solutions through next token prediction.*

We hereafter refer to sequential decoders as simply decoders. A decoder ϕ can be implemented by using p to randomly sample or select the maximum probability token at each step to iteratively generate a sequence until a complete solution is found. However, greedily constructing a single solution is highly prone to getting stuck in local minima.

A more comprehensive way of searching the solution space is to consider a set of alternatives simultaneously. The beam search method aims to do so by keeping track of a set of partial solutions at each step and considering their highest scoring expansions for the next step (Cohen and Beck 2019).

Definition 4 (Beam Search) *Given a set of tokens Σ and a next token prediction function p modelled by a neural network, a beam search decoder ϕ^{bs} of width w generates sets of partial solutions S_i at iteration i with $i \leq k$. S_k only contains complete solutions, S_0 is the singular set containing the empty string ϵ, and each S_i with $i > 0$ contains at most w solutions of length i and is recursively constructed using the equation $S_i = argmax_{1:w}(\{\theta_p(x.a) \,|\, x \in S_{i-1}, a \in \Sigma\})$ where $argmax_{1:w}$ selects the members corresponding to the w highest scores.* [1]

Note that the solution length k is not necessarily predetermined and beam search is terminated when all generated solutions are considered to be complete. Once the procedure is finished, the highest-scoring solution from the set S_k is chosen as the output. The sequence score can act as the main objective of the problem or the proxy for another objective that it aims to approximate. In the latter case, the final solution is selected based on the main objective instead.

2.2 Requirement

A requirement is a constraint on the solution space and a complete solution satisfying the requirement is referred to as feasible. A constrained decoder is required to find a solution that is feasible in addition to maximizing the prediction score (Anderson et al. 2017; Cohen and Beck 2021).

Definition 5 (Constrained Sequential Decoder) *Given a set of tokens Σ, a next token prediction function p, and a requirement $R \subseteq \Sigma^*$, a constrained sequential decoder ϕ_c aims to find a complete feasible solution $x \in R$ with maximum score by iteratively extending partial solutions through next token prediction.*

A naive approach to building constrained decoders is to utilize beam search as before and exclude any complete solutions that do not satisfy the requirement at the end. However, relying on beam search alone provides no guarantee to find any feasible solutions.

2.3 Constraint Satisfaction Problems

A requirement R on the solution can be encoded into a Constraint Satisfaction Problem (CSP) on the sequence of tokens. A CSP consists of a finite set of variables with corresponding domains and a finite number of constraints over the variables, with the goal of finding a satisfying assignment (Brailsford, Potts, and Smith 1999). Well-known CSP

[1] The pseudo-code is presented in the supplementary material.

paradigms include Integer Programming (IP) (Wolsey 2020) and Boolean Satisfiability (SAT) (Gong and Zhou 2017), both are NP-hard problems, but come with efficient off-the-shelf solvers. An instance of IP consists of integer variables and linear inequalities. Note that IP commonly involves a linear objective as well. However, for our purposes, the feasibility-only variation suffices. A SAT formula is a conjunction of clauses where each clause is a disjunction of literals (boolean variables or their negation). We consider an extended variant of the SAT paradigm that has direct support for cardinality clauses. A cardinality clause bounds the number of true literals in its given set. Furthermore, we use incremental solving (Eén and Sörensson 2003) for successive calls to a SAT solver. In an incremental approach, learned details are carried over to improve the performance of future steps. Each step in an incremental approach is distinguished through a set of literals, called assumptions, that need to be true in that particular step.

3 Vehicle Routing Problems

In this section, we discuss Vehicle Routing Problems (VRP) and how neural sequence models have been used to solve them in previous work. VRPs are a family of optimization problems aimed at navigating a vehicle through a set of nodes (locations) while potentially interacting with nodes along the way through pickups, deliveries, or time-sensitive tasks (Toth and Vigo 2014). The goal of a VRP is usually to optimize the cost in terms of different aspects such as distance, duration, or the number of vehicles while respecting capacity, time, or vehicular constraints. We focus our attention on an established variant of the Constrained Vehicle Routing Problem (CVRP) alongside two variants based on the Travelling Salesman Problem (TSP) that we introduce.

Problem 1 (CVRP with Maximum Tours (CVRPM)) *Given a finite set of demand nodes as 2D coordinates $N = \{n_i \mid n_i \in \mathbb{R} \times \mathbb{R}\}$, a demand function $D : N \to \mathbb{N}$, a depot node $n_d \in \mathbb{R} \times \mathbb{R}$, a capacity $c \in \mathbb{N}$, and a maximum number of tours $m \in \mathbb{N}$, find a series of tours T partitioning the nodes with $|T| \leq m$ and $\forall x \in T : \sum D(x_i) \leq c$ to minimize the total distance covered starting from the depot $\sum_{x \in T} dis(n_d.x)$ where x_i is the i-th node in x and $dis(x)$ represents the total distance for traversing x from start to finish and back to start.*

Note that our CVRPM definition is commonly referred to as CVRP in the exact methods literature (Uchoa et al. 2017; Augerat et al. 1995). We do not include the maximum number of tours constraint in CVRP to remain consistent with our baseline and make the distinction between the problem that we solve and theirs (Kool, van Hoof, and Welling 2018).

Problem 2 (TSP with Demands (TSPD)) *Given a finite set of nodes as 2D coordinates $N = \{n_i \mid n_i \in \mathbb{R} \times \mathbb{R}\}$, a subset of supply nodes $N_d \subseteq N$, demands for non-supply nodes $D : N \setminus N_d \to \mathbb{N}$, and capacity $c \in \mathbb{N}$, find a complete permutation of nodes x with minimum total distance covered $dis(x)$ such that the sum of demands in any continuous sequence of non-supply nodes does not exceed the capacity.*[2]

[2]Note that supply nodes from TSPD and the depot from CVRPM both act as providers. However, TSPD differs from CVRPM given that supply nodes are each to be visited once, where as the depot is a singular node that can be visited in multiple tours.

Problem 3 (TSP with Regular Specification (TSPR)) *Given a finite set of nodes as 2D coordinates $N = \{n_i \mid n_i \in \mathbb{R} \times \mathbb{R}\}$, an alphabet mapping $\sigma : N \to \Sigma_{\mathcal{A}}$, and a regular language $\mathcal{A}$ of alphabet $\Sigma_{\mathcal{A}}$, find a complete permutation of nodes x with minimum total distance covered $dis(x)$ such that $\sigma(x)$ belongs to $\mathcal{A}$, where $\sigma(x)$ denotes applying σ to each x_i and concatenating the results.*

A many-to-one alphabet mapping can indicate a division of nodes based on properties such as containing region, type of interaction, or corresponding party.

3.1 Sequential Solving of VRP

While traditionally solved by heuristic or exact methods, modern deep learning has also enabled us to solve VRPs (Wang and Tang 2021) through methods including iterative improvement (Wu et al. 2021; Lu, Zhang, and Yang 2019) and sequence generation (Kool, van Hoof, and Welling 2018; Nazari et al. 2018). We use the attention-based model presented in Kool, van Hoof, and Welling (2018) both as a non-constrained baseline and as a basis for implementing our constrained sequential decoder.

The baseline utilizes a deep learning model based on attention layers trained using REINFORCE (Williams 1992). Their model uses nodes of a VRP instance as tokens, is trained to minimize the total distance covered through next token prediction, and guarantees solution validity through filtering. They demonstrate support for multiple VRP variants including TSP and CVRP. The filtering function used for TSP guarantees the validity of the solution by excluding the nodes that are already visited. The filtering function used for CVRP guarantees the validity of the solution by excluding the non-depot nodes that are already visited and the nodes whose addition would violate the capacity limit.

4 Beam Search with Cuts

In this section, we introduce a variation of the beam search procedure called Beam Search with Cuts (BSC). Unlike ordinary beam search, BSC allows us to implement a constrained decoder (Definition 5) that is not agnostic of the demanded requirement. Instead, BSC performs explicit checks at each step and cuts any partial solutions that cannot be extended to a complete one satisfying the requirement. As a result, a constrained decoder utilizing BSC is guaranteed to produce feasible solutions if they exist.

Definition 6 (Beam Search with Cuts) *Given a set of tokens Σ, a next token prediction function p modelled by a neural network, and a requirement R, a constrained decoder ϕ^{bsc} of width w using beam search with cuts, generates sets of partial solutions S_i at iteration i with $i \leq k$. S_k is empty or only contains complete solutions, S_0 is the singular set containing the empty string ϵ and each S_i with $i > 0$ contains at most w solutions of length i and is recursively con-*

Algorithm 1: Beam Search with Cuts

Input: Set of tokens Σ, Next token prediction function p modelled by a neural network, Requirement R
Parameter: Width w
Output: Solution $x \in \Sigma^*$ or $\varnothing$

$S_0 = \{\epsilon\}$ {Add the empty string as the initial solution.}
$i = 0$
while S_i contains non-complete solutions **do**
  $S'_i = \{x.a | x \in S_i, a \in \Sigma\}$ {Consider all possible expansions allowed by masking.}
  Sort S'_i based on descending values of θ_p.
  $S_{i+1} = \emptyset$
  for all $x \in S'_i$ **do**
    $feas = \exists x' : x.x' \in R \wedge x.x'$is complete {Check feasibility via solving the CSP.}
    if $feas$ **then**
      $S_{i+1} = S_{i+1} \cup \{x\}$
    end if
    if $|S_{i+1}| == w$ **then**
      Break
    end if
  end for
end while
$S_k = S_i$
return $argmax(\{\theta_p(x) | x \in S_k\})$

structed using the following equation:

$$S_i = argmax_{1:w}(\{\theta_p(x.a) \,|\, x \in S_{i-1}, a \in \Sigma, \exists x' : x.a.x' \in R \wedge [x.a.x' \textit{is complete}]\})$$

where $argmax_{1:w}$ selects the members corresponding to the w highest scores. An expansion $x.a$ is said to be cut if it is removed from contention due to not satisfying $\exists x' : x.a.x' \in R \wedge [x.a.x' \textit{is complete}]$. If S_k is empty the decoding process has failed and no feasible solutions are found.

Algorithm 1 presents the pseudo-code for beam search with cuts. Underlined parts are additions that do not exist in the pseudo-code for ordinary beam search (Definition 4).

CSP cuts. Since we use CSPs to specify solution requirements, we can implement a BSC decoder by extending beam search with calls to a CSP solver acting as cuts. Specifically, at each step we check partial solutions in decreasing order of score, only add them to the next S_i if they can be extended to a feasible solution (i.e., they are not cut), and stop when w solutions are added. We check whether a partial solution needs to be cut by solving the CSP instance with the additional constraint that the assignment must match our current partial solution up to the point that it has been decoded.

Incremental CSP Solving. Given a solution length of k and width of w, at least kw cut decisions are made in a successful run of beam search with cuts. Additionally, all of the calls to the CSP solver share much in common as they are solving the same requirement conditioned on different partial solutions. The multitude and repetitive nature of the CSP calls makes our approach amenable to employing incremental solving, a family of techniques aimed at improving the performance through communication between calls. Previous work has utilized incremental CSP solving for pruning infeasible partial solutions in problems such as preferential subset selection (Brafman et al. 2006; Binshtok et al. 2009). In Section 6, we demonstrate how incremental solving can be applied to our SAT-based requirements.

Timeouts. Deciding a cut via solving a CSP instance is inherently difficult and can potentially take a long time. Thus, we use a timeout limit to disrupt the solver if it does not manage to find a feasible solution in time. A partial solution leading to a timeout is still cut as it is not guaranteed to be extendable to a complete feasible solution.

Completeness. A BSC decoder employing timeouts is not guaranteed to produce a feasible solution, even if one exists. This is because cuts due to timeouts might remove all of the partial solutions that can be extended to complete feasible ones. However, if no timeouts occur, a BSC decoder is guaranteed to produce feasible solutions if any exists.

Proposition 1 (BSC Completeness) *For any width w and CSP requirement R such that there exists complete solution $x \in R$, a constrained decoder ϕ^{bsc} that is employing the beam search with cuts procedure and is armed with the CSP solver for R, is guaranteed to find a complete and feasible solution given enough time.*

In the next two sections, we introduce two types of requirements that can be encoded as CSPs to be implemented into a beam search with cuts decoder.

5 Bin Packing Requirements

In this section, we propose bin packing as our first type of requirement. The aim of the bin packing problem is to partition items into a limited number of subsets while satisfying a capacity constraint.

Requirement Type 1 (Bin Packing) *Given a set of items I, their weights $W : I \to \mathbb{N}$, a bin capacity $c \in \mathbb{N}$, and maximum number of bins $m \in \mathbb{N}$, the goal of the bin packing problem is to find a partition $B = \{B_1, B_2, ..., B_m\}$ of items such that each bin respects the capacity constraint $\forall B_i : \sum_{i \in B_i} W(i) \leq c$.*

Bin packing-based requirements can be utilized to solve sequence generation tasks that involve partitioning elements into subgroups. Such tasks include multiple VRP variants, scheduling problems, or spatial reasoning challenges. As two examples, we show how bin packing can be used to solve CVRPM (Problem 1) and TSPD (Problem 2).

To solve CVRPM (TSPD) using beam search with cuts, we combine the CVRP (TSP) model from the baseline using next token prediction function p_{cvrp} (p_{tsp}) with a requirement R_{bin} enforcing that the solution corresponds to a feasible bin packing. In solving CVRPM, the requirement R_{bin} formulates the bin packing problem by seeing nodes as items, tours as bins, demands as weights, and capacities playing the same role. The bin packing is formulated the same for solving TSPD, with the difference of bins being segments of the solution divided by supply nodes.

5.1 Encoding

To have a constrained decoder satisfying R_{bin}, we need to be able to solve a bin packing problem given a partial solution. A partial solution, in CVRPM or TSPD, dictates fixed assignment of a subset of items to bins and a subset of bins to be closed off from accepting more items. In CVRPM (TSPD), if the partial solution has already generated t^F tours (supply visits), we have fixed assignments for bins $[1, t^F+1]$ and bins $[1, t^F]$ have been closed off. We denote the bins with fixed assignments as $B_1^F, B_2^F, ..., B_{t^F+1}^F$.

We present the following IP encoding to solve the problem of bin packing with the added constraint of conforming to a partial solution. We use binary variables $a_{i,j}$ to represent that item i is assigned to bin j. Constraints in Eq. (1) enforce that each item is assigned to exactly one bin. Constraints in Eq. (2) enforce that the sum of weights for items assigned to a bin is less than the capacity. Constraints in Eq. (3) enforce the bin packing to match the given fixed assignments based on the partial solution. Lastly, constraints in Eq. (4) enforce that the items yet to appear in the partial solution, are not assigned to the bins that have been closed off.

$$\forall i \in I : \sum\nolimits_j a_{i,j} = 1 \tag{1}$$

$$\forall j \in [1, m] : \sum\nolimits_i a_{i,j} W(i) \leq c \tag{2}$$

$$\forall j \leq t^F + 1, i \in B_j^F : a_{i,j} = 1 \tag{3}$$

$$\forall j \leq t^F, i \notin \bigcup\nolimits_t B_t^F : a_{i,j} = 0 \tag{4}$$

6 Regular Language Requirements

In this section, we propose regular languages, which have been used in multiple fields such as formal verification, synthesis, and planning, as our second type of requirement. They enable the support of a wide range of specifications including safety, reachability, and grammatical ones. Moreover, modal logic paradigms such as finite linear temporal logic (LTL-f) can be translated to regular languages (De Giacomo and Favorito 2021). In Sections 7.3, 7.4, and 7.5, we use several concrete regular language requirements to solve VRP variants. Any regular language can be represented by a Deterministic Finite Automata (DFA) and vice-versa. Thus, a regular language requirement is equivalent to requiring the solution's acceptance by the corresponding DFA.

Requirement Type 2 (DFA Acceptance) *Let $\mathcal{A}$ be a DFA with alphabet $\Sigma_\mathcal{A}$ and finite set of states $Q_\mathcal{A}$, $q_0 \in Q_\mathcal{A}$ an initial state, $Q_\mathcal{A}^F \subseteq Q_\mathcal{A}$ the accept states, $\delta_\mathcal{A} : Q_\mathcal{A} \times \Sigma_\mathcal{A} \to Q_\mathcal{A}$ a transition function, and $W_\mathcal{A} \subseteq \Sigma_\mathcal{A}^*$ a set of strings corresponding to complete solutions, find $w \in W_\mathcal{A}$ such that $\Delta(q_0, w) \in Q_\mathcal{A}^F$ where $\Delta(.,.)$ is recursively defined as:*

$$\Delta(q, w) = \begin{cases} \Delta(\delta(q, a), w') & a \in \Sigma_\mathcal{A}, w = a.w' \\ q & w = \epsilon \end{cases}$$

Note that we only consider complete solutions as candidates to satisfy the requirement. Therefore, we integrate the termination condition into Requirement Type 2 by setting $W_\mathcal{A}$ to represent the set of all complete solutions. Employing CSPs allows us to encode a wide array of termination conditions represented as the $w \in W_\mathcal{A}$ constraint.

To solve TSPR (Problem 3), we use beam search with cuts by taking the next token prediction function p_{tsp} from the baseline model for TSP and extending it with a regular requirement $R_\mathcal{A}$ enforcing that the final complete solution is accepted by the corresponding DFA. The requirement $R_\mathcal{A}$ uses alphabet mapping σ to construct the set of candidate inputs $W_\mathcal{A}$. Specifically, $W_\mathcal{A}$ contains all $\sigma(x)$ sequences where x is a permutation of all nodes in the problem.

6.1 Encoding

To have a constrained decoder satisfying a regular language requirement $R_\mathcal{A}$, we need to be able solve a DFA acceptance problem given a partial solution. Adherence to the partial solution dictates that the input to $\mathcal{A}$ should start with a sequence w^F of length l^F that corresponds to the output of applying the alphabet mapping to the partial solution. We use the number of nodes $|N|$ as the length of the input as a complete TSP solution contains all nodes exactly once. To guarantee that the input belongs to $W_\mathcal{A}$, for each member of the input alphabet $a \in \Sigma_\mathcal{A}$ we count the number of nodes n such that $\sigma(n) = a$ and denote the number as W_a. Note that it suffices to enforce the number of appearances to make sure that the input corresponds to a permutation of all nodes.

We present the following SAT encoding to solve the problem of DFA acceptance while adhering to a partial solution. We use variables $d_{i,a}$ to represent that $w_i = a$ and variables $s_{i,q}$ to represent that the DFA is transitioned to state q after observing $w_1, w_2, ..., w_i$. Note that adherence to the partial solution needs to be added as assumptions rather than clauses, since it changes for each call to the solver. We specify cardinality clauses using the inequality symbol ($\leq$), disjunctive clauses using parentheses, and assumptions using brackets. Clauses in Eqs. (5) and (6) guarantee that exactly one member of the alphabet is chosen at each step. Clauses in Eqs. (7) and (8) guarantee that exactly W_a appearances exist for each $a \in \Sigma_\mathcal{A}$. Clauses in Eqs. (9) and (10) guarantee that exactly one state is visited at each step. Clauses in Eq. (11) guarantee that the first state visited after the initial one is according to the transition function, and clauses in Eq. (12) enforce the same validity of transition for all of the next steps. Clauses in Eq. (13) guarantee that the last state visited is an accept state. Lastly, the assumptions in Eq. (14) guarantee that the beginning of the solution match w^F and consequently the given partial solution.

$$\forall i : (\bigvee\nolimits_a d_{i,a}) \tag{5}$$

$$\forall i : \sum\nolimits_a d_{i,a} \leq 1 \tag{6}$$

$$\forall a : \sum\nolimits_i d_{i,a} \leq W_a \tag{7}$$

$$\forall a : \sum\nolimits_i \neg d_{i,a} \leq |N| - W_a \tag{8}$$

$$\forall i : (\bigvee\nolimits_q s_{i,q}) \tag{9}$$

$$\forall i : \sum\nolimits_q s_{i,q} \leq 1 \tag{10}$$

$$\forall a : (\neg d_{1,a} \vee s_{1,\delta(q_0,a)}) \tag{11}$$

$$\forall i > 1, q, a : (\neg s_{i-1,q} \vee \neg d_{i,a} \vee s_{i,\delta(q,a)}) \tag{12}$$

$$(\bigvee\nolimits_{q \in Q_\mathcal{A}^F} s_{|N|,q}) \tag{13}$$

$$\forall i \leq l^F : [d_{i,w_i^F}] \tag{14}$$

7 Experimental Results

7.1 Experimental Setup

Through our experiments, we investigate whether beam search with cuts can effectively equip existing neural models with the ability to satisfy hard constraints. We focus our attention on VRPs, a well-known family of problems that involve hard constraints and have been heavily studied in neural approaches to combinatorial problems. We use the attention-based model (Kool, van Hoof, and Welling 2018) with beam search decoding as our baseline. Furthermore, we extend their implementation with requirements encoded as CSPs to run beam search with cuts. We solve IP instances using the Gurobi solver (Gurobi Optimization, LLC 2023) version 10, and SAT instances using PySAT (Ignatiev, Morgado, and Marques-Silva 2018) with Gluecard 4 solver (Eén and Sörensson 2003). Both solvers are allocated a time limit of 10 seconds for each call. Our results are produced on a server with two 12-core Intel E5-2697v2 and 128G of RAM.

To cover a wide range of challenges, we use the dataset proposed in Uchoa et al. (2017) but also synthesize our own instances following the procedure described in the baseline (Kool, van Hoof, and Welling 2018). All of our synthesized instances are randomized using 10 different seeds and the results are averaged over seeds unless noted otherwise.

7.2 Sequence Generation with Requirements

For our first experiment, we solve Problem 1 (CVRPM) for instances in Uchoa et al. by enforcing a maximum number of tours. The requirement is implemented through bin packing-based cuts. We omitted instances with a maximum number of tours larger than 20, as it would be too challenging to solve the bin packing CSP for such instances.

We first ran the ordinary beam search decoder with a comparatively large width (8096) and observed that in 9 out of 27 instances, no generated solutions were able to satisfy the maximum number of tours requirement. Table 1 contains the BS and BSC results for the 9 unsatisfied cases[3]. We ran beam search with cuts for the unsatisfied cases with significantly lower width (4) and were able to find feasible solutions in all cases, indicating that requirement satisfaction in BSC cannot simply be compensated for by using a higher width value in BS. We report the required maximum number of tours m alongside the best value that each approach was able to achieve. Additionally, we include the best total distance amongst the solutions generated by BSC in relative comparison against the best from BS (Δdis. (%)) with a negative value indicating an improvement. Our results show that despite the lower width and the added benefit of satisfying the requirement, BSC was able to achieve solutions of comparable total distance, with even significant improvement in some cases. Furthermore, we observe that ordinary beam search shows higher runtimes in the majority of cases, with BSC taking longer only if a significant number of cuts are needed to prune infeasible solutions. While the runtimes can be further improved with parallelized or GPU-based methods, the results still indicate the intractability of simply increasing the width to satisfy requirements.

[3]Remaining results are presented in the supplementary material.

Instance	Beam Search		Beam Search with Cuts			
($\|N\|$,m)	Time (s)	m	Time (s)	m	Δdis.	Cuts
(134,13)	26.7	14	45.1	13	0.8%	168
(157,13)	35.7	14	14.9	13	-12.2%	17
(190,8)	49.3	11	14.4	8	-23.9%	52
(209,16)	60.7	17	28.3	16	2.0%	19
(214,11)	61.7	12	27.0	11	5.7%	92
(233,16)	73.4	17	384.6	16	20.1%	219
(256,16)	89.3	17	630.4	16	9.0%	408
(367,17)	185.6	18	84.4	17	-0.6%	6
(411,19)	236.7	22	116.7	19	-14.6%	44

Table 1: Beam search (width=8096) against beam search with cuts (width=4) on select instances from Uchoa et al.

To better understand the effects of the width parameter on beam search with cuts, we solve the same 9 instances with BSC in three configurations: a beam width of 4, a larger beam width of 64, and a hybrid approach where we use a variation of our BSC algorithm that only guarantees a fraction of the partial solutions to be extendable to complete feasible ones. Specifically, we consider a *sub-width* ($= 4$) in addition to width ($= 64$) and we solve the CSP problem starting from the highest score as many times as needed in order to fill the sub-width with solutions that can lead to feasibility. The partial solutions shown to be infeasible will then be disregarded and the remaining width will be filled with untested ones. We report the same values as in Table 1 for the three configurations in Table 2. Note that we omit reporting the number of tours achieved as all approaches are able to satisfy the given specification in all instances. As expected, the approach with a width of 64 and the hybrid approach are both able to produce solutions with better distance compared to a width of 4 in all but one instance. Interestingly, we observe that the hybrid approach is closer in runtime to the lower width as opposed to the higher one.

7.3 Tightening Requirements

Next, our goal is to understand the limits of feasibility and the trade-off between challenging requirements and solution quality. Thus, we avoid fixing the requirement and instead use incrementally tighter ones until no feasible solution exists. We use this approach to solve Problem 2 (TSPD) for synthesized instances by enforcing a capacity constraint c implemented through bin packing-based cuts. The demand for each non-supply node is randomly selected from a uniform distribution. Moreover, we solve Problem 3 for synthesized instances by enforcing a regular language requirement $\mathcal{A}_{s_i}$ implemented through DFA acceptance-based cuts. The requirement $\mathcal{A}_{s_i}$ states that in the TSPR solution x, there cannot be i successive visits to nodes that all map to the same member of the alphabet by σ. An equal number of nodes are randomly mapped to each member of the alphabet. We first run an ordinary beam search and observe the lowest c value for TSPD or the lowest i value for TSPR out of all solutions, to use as a starting point for iteratively tightening the corresponding requirement. The TSPD results are presented in Figure 1 (top) and the TSPR results in Figure 1 (bottom).

Instance ($\|N\|$,m)	Width	Beam Search with Cuts Time (s)	Δdis.	Cuts
(134,13)	64	918.5	-1.6%	7609
	64-4	90.1	-0.6%	2397
	4	45.1	0.8%	168
(157,13)	64	180.6	-12.7%	178
	64-4	24.0	-12.7%	147
	4	14.9	-12.2%	17
(190,8)	64	189.3	-24.1%	1136
	64-4	56.4	-23.6%	2186
	4	14.4	-23.9%	52
(209,16)	64	372.7	2.4%	192
	64-4	40.2	0.2%	175
	4	28.3	2.0%	19
(214,11)	64	405.2	0.8%	1126
	64-4	40.5	0.5%	282
	4	27.0	5.7%	92
(233,16)	64	4869.6	18.2%	4172
	64-4	441.5	17.7%	3528
	4	384.6	20.1%	219
(256,16)	64	8652.7	5.7%	6541
	64-4	735.2	6.1%	1814
	4	630.4	9.0%	408
(367,17)	64	1258.4	-1.5%	505
	64-4	123.5	-1.5%	277
	4	84.4	-0.6%	6
(411,19)	64	1700.4	-15.6%	466
	64-4	251.6	-16.8%	1610
	4	116.7	-14.6%	44

Table 2: BSC with different width and sub-width (indicated by -) values on select instances from Uchoa et al.

The horizontal axes indicate how much tighter than the starting point the requirement is. Hence, zero indicates using BS (width=8096) while positive values indicate a tighter constraint (lower c for capacity or i for $\mathcal{A}_i^s$) solved by BSC (width=4). The vertical axes show the total distance and each line represents an instance with a different seed.

The results show that BSC, even with lower width, is able to reduce capacity by a margin of 10 to 30 across instances, and the maximum number of successive visits by up to 5 on some instances, with relatively minor impact on total distance. In both cases, we see that tightening the requirement too much will result in a sudden jump in distance before reaching infeasibility. In all instances, the infeasible runs contain no timeouts, guaranteeing that there cannot exist any solutions. In TSPD, we also see instability in distance for highly challenging requirements that, alongside the decline of quality, can be indicative of the fact that the search is too constrained to explore solutions of high predictive scores.

7.4 Scaling Problem Size

Next, we study the impact of solution length and requirement strength on the ability of BS and BSC to produce feasible solutions. We solve Problem 3 (TSPR) for synthesized

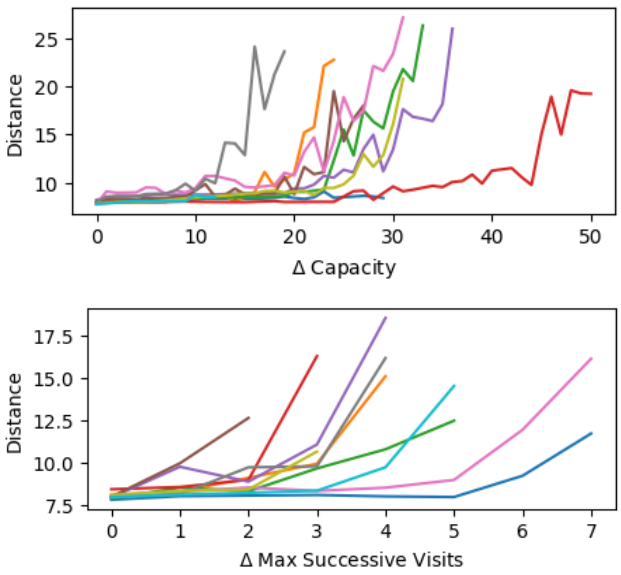

Figure 1: Iterative tightening of the requirement for TSPD (top) and TSPR (bottom) against solution quality, from beam search ($\Delta = 0$, width=8096) to beam search with cuts ($\Delta > 0$, width=4) until infeasibility. Different colors represent different randomly-generated synthetic instances.

instances by enforcing a regular language requirement $\mathcal{A}_i^P$ implemented through DFA acceptance-based cuts. The requirement $\mathcal{A}_i^p$ states that in the TSPR solution x, there cannot be two successive nodes x_j and x_{j+1} such that $\sigma(x_j) = a_i$ and $\sigma(x_{j+1}) = a_{i+1}$ with a_i and a_{i+1} being the i-th and $i+1$-th members of the alphabet. We use $\mathcal{A}_i^P$ as a shorthand for the combination of $\mathcal{A}_1^p$ to $\mathcal{A}_i^p$. Combining the requirements into one allows us to quantify and control the requirement strength. Note that we can implement the combination of multiple DFA acceptance requirements by simply considering the DFA product. For synthesized instances, an equal number of nodes are randomly mapped to each member of an alphabet of size 6. Firstly, we use beam search to solve instances of size 24 with $\mathcal{A}_1^P$ to $\mathcal{A}_5^P$ as requirements. We start from a width of 1 and double the width every time BS fails to generate a feasible solution. Secondly, we solve instances of size 12, 24, and 36 with $\mathcal{A}_3^P$ as requirement and double the width in case of infeasibility as before. For all cases, we report the average logarithm of the lowest width that was able to satisfy the requirement. For comparison, we also solve each instance using BSC with a width of 4.

The results, provided in Table 3, show that with a linear increase in the number of combined requirements, there is an almost linear increase in the logarithm of the lowest width. When the requirement is fixed, a similar linear relation exists between the length of the solution (number of nodes) and the logarithm of the lowest width. The trends show that the beam search decoder needs to produce an exponentially larger number of solutions in order to satisfy a stronger requirement or satisfy the same requirement for a longer solution. Beam search with cuts however, is able to satisfy com-

$\|N\|$	R	Beam Search $log(w)$	Beam Search with Cuts Cuts	Time (s)
24	P_1	2.7	2.8	1.81
	P_2	4.4	6.4	1.79
	P_3	6.8	8.4	1.80
	P_4	8.9	12.4	1.78
	P_5	10.5	15.4	1.87
12	P_3	1.2	3.5	1.73
24		6.4	8.4	1.80
36		11.1	16.7	1.95

Table 3: Minimum width required in log-scale for BS against the number of cuts and time used by BSC (width=4).

$\|N\|$	Width	Cuts	Solve Time (s) Non-Inc.	Inc.
12	4	1.2	1.59	0.45
	16	4	6.00	1.56
24	4	14.5	13.95	2.45
	16	64.5	57.00	7.62
36	4	40.9	39.58	5.59
	16	143.7	147.59	14.94
48	4	66	89.65	10.94
	16	233.4	345.05	32.09
60	4	94.9	150.17	15.28
	16	332.1	565.47	33.92

Table 4: Solve time of incremental and non-incremental SAT solving for beam search with cuts.

bined requirements and produce longer solutions with an almost linear increase in cuts and negligible cost to runtime.

7.5 Incremental CSP Solving

Lastly, we investigate how incremental SAT solving can enhance the performance of beam search with cuts. Hence, we solve Problem 3 (TSPR) for synthesized instances by enforcing a regular language requirement $\mathcal{A}^W$ implemented via SAT. The requirement $\mathcal{A}^W$ states that within any window of length 8 in the solution, there should be at least one appearance for every member of the alphabet through the mapping σ. For synthesized instances, an equal number of nodes are randomly mapped to each member of an alphabet ($\Sigma_{\mathcal{A}}$) of size 4. We run the BSC algorithm for width values of 4 and 16 and different instance sizes, allowing the solver to carry information forward for the incremental run, and wiping the SAT solver's slate clean after each call in the non-incremental run. We report the total time that is purely spent on solving SAT instances for comparison. Note that we only report the number of cuts once as the two approaches behave exactly the same with regard to cuts and produce the exact same solutions since no timeouts are involved.

The results presented in Table 4 show that the incremental approach is able to improve solve time in all cases. Furthermore, the relative difference in solve time grows monotonically as we increase the size of the instance or the width. The improvement is due to the highly repetitive nature of CSP calls which is further exacerbated in higher widths.

8 Related Works

Recently, Lafleur, Chandar, and Pesant (2022) have attempted to solve neural sequence generation with CSP-based requirements by augmenting the predictive scores with marginal probabilities that approximate the likelihood that assignments participate in a feasible solution. This approach performs well in guiding the search towards feasible solutions. However, there are shortcomings such as a lack of guarantee to find feasible solutions, potential cost to quality due to manipulation of scores, and the need for a CSP solver capable of computing these marginal probabilities.

Another approach to tackle CSP-based requirements for neural sequence generation is to use beam search and double the number of solutions produced at each run until a satisfying solution appears. This approach is agnostic of the requirement and has been shown in recent work to exhibit heavy-tailed behavior which can be successfully mitigated by utilizing randomness (Cohen and Beck 2021).

Constrained beam search with lexical constraints is a common approach to neural sequence generation with requirements. Lexical constraints require a set of phrases to appear in the output and are significantly less expressive than requirements encoded as CSPs. Constrained beam search is commonly achieved by dividing the partial solutions in each iteration based on their number of satisfied constraints (Hokamp and Liu 2017; Post and Vilar 2018), or their state in a finite-state machine encoding the constraints as a whole (Anderson et al. 2017). Alternative approaches to satisfying lexical constraints include augmenting the predictive scores to lean towards the target words (Pascual et al. 2020), and modifying the output through stochastic sampling (Miao et al. 2019) or gradient-guided (Sha 2020) edits.

9 Conclusion

We have presented beam search with cuts that allows us to combine neural sequence models with requirements encoded in CSP towards constrained decoding. We have implemented two types of requirements and used them to solve multiple VRP variants. Through our experiments, we have shown that cuts in beam search enable satisfaction of tight requirements with negligible impact on quality.

Our approach can be developed in multiple interesting directions. Feasibility results can be cached and queried using equivalency, e.g., two partial TSPD solutions that have visited the same nodes in different orders are equivalent. Equivalency checks can also cut solutions that are of lower quality in order to improve the search or increase the diversity of the results. Furthermore, our approach can be extended with new types of requirements or application to other neural sequence models in areas such as planning or program synthesis. Lastly, cuts can be added to the Beam-Stack search procedure (Zhou and Hansen 2005) to mitigate the impact of requirements guiding the search to a sub-optimal space.

Acknowledgements

The authors gratefully acknowledge funding from NSERC and the Canada CIFAR AI Chairs program (Vector Institute for Artificial Intelligence).

References

Anderson, P.; Fernando, B.; Johnson, M.; and Gould, S. 2017. Guided Open Vocabulary Image Captioning with Constrained Beam Search. In *EMNLP*, 936–945.

Apt, K. 2003. *Principles of constraint programming*. Cambridge university press.

Augerat, P.; Belenguer, J.; Benavent, E.; Corberán, A.; Naddef, D.; and Rinaldi, G. 1995. Computational results with a branch and cut code for the capacitated vehicle routing problem research. *Universite Joseph Fourier, Grenoble, France*, 949–M.

Bahdanau, D.; Cho, K. H.; and Bengio, Y. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

Bello, I.; Pham, H.; Le, Q. V.; Norouzi, M.; and Bengio, S. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Binshtok, M.; Brafman, R. I.; Domshlak, C.; and Shiomony, S. 2009. Generic preferences over subsets of structured objects. *JAIR*, 34: 133–164.

Brafman, R. I.; Domshlak, C.; Shimony, S. E.; and Silver, Y. 2006. Preferences over sets. In *AAAI*, volume 6, 1101–1106.

Brailsford, S. C.; Potts, C. N.; and Smith, B. M. 1999. Constraint satisfaction problems: Algorithms and applications. *EJOR*, 119(3): 557–581.

Cohen, E.; and Beck, C. 2019. Empirical analysis of beam search performance degradation in neural sequence models. In *ICML*, 1290–1299. PMLR.

Cohen, E.; and Beck, J. C. 2021. Heavy-Tails and Randomized Restarting Beam Search in Goal-Oriented Neural Sequence Decoding. In *CPAIOR*, 115–132. Springer.

De Giacomo, G.; and Favorito, M. 2021. Compositional approach to translate LTLf/LDLf into deterministic finite automata. In *ICAPS*, volume 31, 122–130.

Eén, N.; and Sörensson, N. 2003. An extensible SAT-solver. In *SAT*, 502–518. Springer.

Freitag, M.; and Al-Onaizan, Y. 2017. Beam Search Strategies for Neural Machine Translation. In Luong, T.; Birch, A.; Neubig, G.; and Finch, A., eds., *WNMT*, 56–60.

Gong, W.; and Zhou, X. 2017. A survey of SAT solver. In *AIP Conference Proceedings*, volume 1836. AIP Publishing.

Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual.

Harshvardhan, G.; Gourisaria, M. K.; Pandey, M.; and Rautaray, S. S. 2020. A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 38: 100285.

Hokamp, C.; and Liu, Q. 2017. Lexically Constrained Decoding for Sequence Generation Using Grid Beam Search. In *ACL*, volume 1, 1535–1546.

Ignatiev, A.; Morgado, A.; and Marques-Silva, J. 2018. PySAT: A Python Toolkit for Prototyping with SAT Oracles. In *SAT*, 428–437.

Kool, W.; van Hoof, H.; and Welling, M. 2018. Attention, Learn to Solve Routing Problems! In *ICLR*.

Lafleur, D.; Chandar, S.; and Pesant, G. 2022. Combining Reinforcement Learning and Constraint Programming for Sequence-Generation Tasks with Hard Constraints. In *CP*, 30:1–30:16.

Lu, H.; Zhang, X.; and Yang, S. 2019. A learning-based iterative method for solving vehicle routing problems. In *ICLR*.

Medsker, L. R.; and Jain, L. 2001. Recurrent neural networks. *Design and Applications*, 5(64-67): 2.

Miao, N.; Zhou, H.; Mou, L.; Yan, R.; and Li, L. 2019. Cgmh: Constrained sentence generation by metropolis-hastings sampling. In *AAAI*, volume 33, 6834–6842.

Nazari, M.; Oroojlooy, A.; Snyder, L. V.; and Takác, M. 2018. Reinforcement Learning for Solving the Vehicle Routing Problem. In *NeurIPS*, 9861–9871.

Pascual, D.; Egressy, B.; Bolli, F.; and Wattenhofer, R. 2020. Directed beam search: Plug-and-play lexically constrained language generation. *arXiv preprint arXiv:2012.15416*.

Post, M.; and Vilar, D. 2018. Fast Lexically Constrained Decoding with Dynamic Beam Allocation for Neural Machine Translation. In *NAACL*, 1314–1324.

Sha, L. 2020. Gradient-guided unsupervised lexically constrained text generation. In *EMNLP*, 8692–8703.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In *NeurIPS*, 3104–3112.

Toth, P.; and Vigo, D. 2014. *Vehicle routing: problems, methods, and applications*. SIAM.

Uchoa, E.; Pecin, D.; Pessoa, A.; Poggi, M.; Vidal, T.; and Subramanian, A. 2017. New benchmark instances for the capacitated vehicle routing problem. *EJOR*, 257(3): 845–858.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All you Need. In *NeurIPS*, 5998–6008.

Vinyals, O.; Fortunato, M.; and Jaitly, N. 2015. Pointer Networks. In *NeurIPS*, 2692–2700.

Wang, Q.; and Tang, C. 2021. Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems*, 233: 107526.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8: 229–256.

Wolsey, L. A. 2020. *Integer programming*. John Wiley & Sons.

Wu, Y.; Song, W.; Cao, Z.; Zhang, J.; and Lim, A. 2021. Learning improvement heuristics for solving routing problems. *IEEE transactions on neural networks and learning systems*, 33(9): 5057–5069.

Zhou, R.; and Hansen, E. A. 2005. Beam-Stack Search: Integrating Backtracking with Beam Search. In *ICAPS*, 90–98.