Heavy-Tails and Randomized Restarting Beam Search in Goal-Oriented Neural Sequence Decoding

Eldan Cohen and J. Christopher Beck

Department of Mechanical & Industrial Engineering
University of Toronto, Toronto, Canada
{ecohen, jcb}@mie.utoronto.ca

Abstract. Recent work has demonstrated that neural sequence models can successfully solve combinatorial search problems such as program synthesis and routing problems. In these scenarios, the beam search algorithm is typically used to produce a set of high-likelihood candidate sequences that are evaluated to determine if they satisfy the goal criteria. If none of the candidates satisfy the criteria, the beam search can be restarted with a larger beam size until a satisfying solution is found. Inspired by works in combinatorial and heuristic search, we investigate whether heavy-tailed behavior can be observed in the search effort distribution of complete beam search in goal-oriented neural sequence decoding. We analyze four goal-oriented decoding tasks and find that the search effort of beam search exhibits fat- and heavy-tailed behavior. Following previous work on heavy-tailed behavior in search, we propose a randomized restarting variant of beam search. We conduct extensive empirical evaluation, comparing different randomization techniques and restart strategies, and show that the randomized restarting variant solves some of the hardest instances faster and outperforms the baseline.

Keywords: Beam Search · Neural Sequence Models · Randomized Restarts.

1 Introduction

Neural sequence models are commonly used in the modeling of sequential data and are the state-of-the-art approach for tasks such as machine translation [10], text summarization [6], and image captioning [37]. Beam search is the most commonly used algorithm for decoding neural sequence models by (approximately) finding the most likely output sequence conditioned on the input. To do so, beam search generates sequences token-by-token, extending a fixed number of active candidate sequences (beam size) at each step.

Recently, neural sequence models have been successfully applied to different combinatorial search problems such as program synthesis and routing problems. Unlike machine translation and image captioning, such problems often have a goal criteria that can be used to evaluate candidate solutions and require solutions that satisfy the goal criteria. For example, in resource-constrained combinatorial routing problems, we may wish to find a tour that satisfies some resource constraint (e.g., limited fuel or budget). In such scenarios, beam search is used to produce a set of promising (high-likelihood) candidate sequences that are evaluated to determine if they satisfy the goal criteria. If none of them satisfy the criteria, the beam search can be restarted with a larger beam size until a satisfying solution is found.

Previous work on heuristic and combinatorial search algorithms found they tend to exhibit a fat- and heavy-tailed behavior that can be exploited to boost their performance by incorporating randomized restarts in the search (e.g., [12, 8]). In this work, we investigate whether a heavy-tailed behavior can also be observed for goal-oriented beam search. We consider four goal-oriented neural sequence decoding tasks, each with a goal criteria that enforces bounded suboptimality with respect to a chosen evaluation metric. We focus on complete anytime beam search (CAB), a complete variant of beam search commonly used in goal-oriented neural sequence decoding, and perform an extensive empirical study of the heavy-tailed behavior and the impact of randomized restarts. Specifically, we make the following contributions:

1. We show that for goal-oriented neural sequence problems, complete anytime beam search exhibits a fat- or heavy-tailed behavior on ensembles of relaxed problems, similar to the behavior observed for CSPs and SAT.
2. We consider a randomized variant of beam search that is based on noise injection to the inputs of the neural network and show that randomized complete anytime beam search exhibits fat- or heavy-tailed behavior on ensembles of multiple runs on a single instance.
3. Inspired by previous work on heavy-tailed behavior in combinatorial and heuristic search problems, we introduce a randomized restarting variant of complete anytime beam search and show that it outperforms the baseline by solving some of the hardest problems faster.
4. We conduct extensive empirical evaluation and analyze the impact of different parameters including the constrainedness of the goal criteria, the restart policy, and the type of randomization.

2 Background

2.1 Beam Search for Goal-Oriented Neural Sequence Decoding

A neural sequence model learns a probability distribution over sequences by being trained to predict the probability of the next token in a sequence, $p(y_t|x;y_{1:t-1})$, conditioned on the input x and the partial sequence $y_{1:t-1}$ [5]. The total probability of a (partial) sequence $y_{1:t}$ follows from the chain rule of probability:

$$p(y_{1:t}|x) = p(y_t|x;y_{1:t-1}) \cdot p(y_{1:t-1}) = \prod_{t'=1}^{t} p(y_{t'}|x;y_{1:t'-1}). \tag{1}$$

It is common to model $p(y_t|x;y_{1:t-1})$ using a Recurrent Neural Network [16], where the input x and the partial sequence $y_{1:t-1}$ we condition on are expressed by a fixed length representation h_t. This representation is updated each step using a non-linear function f: $h_t = f(h_{t-1}, y_{t-1})$ with h_0 being a representation of the input x and y_0 being a special token that represents the start of the sequence. The conditional probability over the next token y_t can then be computed using the softmax function,

$$p(y_t = v_i|x; y_{1:t-1}) = \frac{\exp(w_i h_t)}{\sum_{j=1}^{|\mathcal{V}|} \exp(w_j h_t)},$$

where $\mathcal{V} = \{v_1, v_2, ...\}$ is the set of all possible tokens and w_i are model weights.

Beam search is a limited-width breadth-first search. In the context of sequence models, it is often used as an approximation to finding the (single) sequence y that maximizes Eq. (1), or as a way to obtain a set of high-probability sequences from the model. At the first step, $t = 0$, we only have one (empty) sequence. At each of the following steps, $t \geq 1$, we consider all one-token extensions of the beam sequences from step $t-1$ and retain (at most) $\mathcal{B}$ partial sequences with the highest probability. In the last step, we return the $\mathcal{B}$ highest probability complete sequences, which we assume to be of equal length (as they can be padded). $\mathcal{B}$ is called the beam width (or, alternatively, beam size) and the probabilities of (partial) sequences are estimated by the neural network.

In goal-oriented neural sequence decoding, we are not looking for the most-likely sequence according to the learned model. Instead, we are looking for a solution that satisfies the goal criteria. In such scenarios, we use beam search to generate a set of $\mathcal{B}$ high-quality candidates that are then evaluated to determine if they satisfy the goal criteria. Once a candidate satisfies the goal criteria, it is returned as the solution of the beam search.

Previous work on goal-oriented neural sequence decoding considered a variant of the complete anytime beam search (CAB) [42] in which failing to find a satisfying solution results in doubling the beam width and re-running the beam search [43, 2, 25]. As the beam width increases, a larger portion of the hypotheses space is explored and the search is guaranteed to find a solution, if one exists. Algorithm 1 shows pseudo-code for this variant of complete anytime beam search.

Algorithm 1 Complete Anytime Beam Search

```
function CAB(goalCriteria)
    $beamWidth \leftarrow 1$
    while not solved do
        $candidates \leftarrow BeamSearch(beamWidth)$
        for $cand \in candidates$ do
            if $Satisfy(cand, goalCriteria)$ then
                return $cand$
        $beamWidth \leftarrow 2 \cdot beamWidth$
```

2.2 Heavy-tailed Behavior and Randomization in Heuristic and Combinatorial Search Algorithms

Analyzing the empirical distribution of search effort over an ensemble of problems, rather than just the mean or median, can often help design better search algorithms. Previous work has found fat- or heavy-tailed behavior in the distribution of search effort for different search algorithms on NP-complete problems, e.g., the number of backtracks in CSPs, on ensembles of random problems [13, 12, 7]. This behavior tends to appear in ensembles of relaxed problems, i.e., problems with high density of solutions. In these ensembles, the median search effort is low, however the hardest instances can require orders-of-magnitude higher effort. Interestingly, Gomes et al. [12] also found heavy-tailed behavior in the search effort distribution of a randomized search procedure on a *single* instance, suggesting that some of the hardest problems can be solved easily by minor changes in the search procedure. This result has motivated significant work on reducing heavy-tailed behavior using randomized restarts, portfolios, etc. [11].

Fat- and heavy-tailed distributions have a long tail containing a considerable concentration of mass. Formally, a random variable X is considered heavy-tailed if it has a Pareto-like decay of its tail above some threshold x_l, i.e., there exists some $x_l{>}0$, $c{>}0$, $\alpha{>}0$ such that $P[X > x] = cx^{-\alpha}$ for $x > x_l$ [32]. An approximately linear behavior over several orders of magnitude in the log-log plot of $1 - CDF(x)$ (i.e., the survival function) is a clear sign of heavy-tailed behavior with the slope providing an estimate of the stability index α [14].

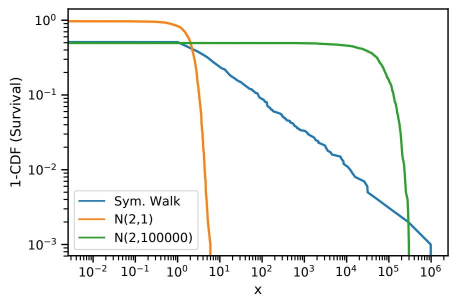


Fig. 1: Heavy and non-heavy tailed behavior [14].

To demonstrate heavy-tailed behavior, we present an example from Gomes et al. [14]. Figure 1 shows the log-log plot of $1 - CDF(x)$ for two normally distributed random variables with a mean of 2 and different standard deviation. It also shows a random variable that represents the number of steps it takes for a symmetric random walk on a line to get back to the starting point. The normal distributions exhibit a fast-decay behavior, while the random walk exhibits a clear heavy-tailed behavior indicated by the approximately linear behavior on the log-log plot.

3 Goal-Oriented Benchmark Problems

In our analysis, we use a set of four goal-oriented benchmark problems. Following is a description of each problem and its goal criteria.

3.1 Combinatorial Routing Problems

Several recent works have demonstrated the potential of using deep learning to solve combinatorial optimzation problems [23, 22, 9, 30]. A recent work [23] proposed an architecture based on attention layers and trained using REINFORCE [41] to generate solutions for combinatorial routing problems that minimize the solution cost. The authors use this architecture to generate solutions to the Travelling Salesman Problem (TSP), two variants of the Vehicle Routing Problem (VRP), the Orienteering Problem (OP), and the Prize Collecting TSP (PCTSP) and show it outperforms a wide range of baselines. Decoding can be done using sampling or beam search, and the best solution among the generated candidates is returned. To eliminate infeasible solutions, e.g., revisiting the same node in TSP, the authors use masking (setting the log-probabilities of infeasible solutions to $-\infty$). In our work, we use Kool et al.'s [23] architecture[1] and problem instances and run experiments on two combinatorial routing problems:

– The Travelling Salesman Problem (TSP) consists of constructing a tour that starts at the depot, visits all nodes exactly once, and returns to the depot.
– The Capacitated Vehicle Routing Problem (CVRP) consists of constructing multiple routes, each starting and ending at the depot, such that the total demand of the nodes in each route does not exceed the vehicle capacity.

The cost of solution in both problems is the sum of pairwise Euclidean distances of consecutive nodes in the solution path (including the depot).

Goal Criteria. As the current model is trained to minimize the solution cost, we consider the goal-oriented problem of finding a solution with a bounded optimality gap. Assuming a minimization problem with cost function $\mathcal{C}$, our goal criteria for a candidate solution x is $\frac{\mathcal{C}(x)-\mathcal{C}(x^*)}{\mathcal{C}(x^*)} \leq \varepsilon$, where x^* is an optimal solution and ε controls the constrainedness of problems (increasing ε leads to a higher expected number of feasible solutions).[2] Following Kool et al. [23], we compute optimal solutions for TSP using Concorde [1] and approximate optimal solutions for CVRP using KLH3 [17] (Kool et al. [23] note CVRP problems with more than 20 location were intractable for an exact solver).

[1] Obtained from github.com/wouterkool/attention-learn-to-route.
[2] This notion of constrainedness matches the notion of resource-constrainedness previously used to study planning in resource-constrained environments [29].

3.2 Visual Program Synthesis

Several recent works have considered the problem of synthesizing programs for images using deep neural networks [33, 36, 27]. These networks take an image as input and output a program that generates the image. The quality of a candidate program can be evaluated using a metric of projection loss, typically a distance measure between the generated image and the input. In our experiments, we use CSGNet[3] [33], a neural architecture that takes in a 2D or 3D shape image and outputs a program to generate the shape using instructions based on constructive solid geometry (CSG). CSGNet is trained using a combination of supervised learning and reinforcement learning (using REINFORCE [41]) to minimize the visual distance between the generated solutions and the input images.

Goal Criteria. Our goal criteria is based on Chamfer Distance ($\mathcal{CD}$), a measure of visual similarity between two shapes that is used by Sharma et al. [33] to evaluate CSGNet. Let $\mathcal{CD}(a, b)$ denote the (non-negative) Chamfer distance between shape a and shape b. We define our goal criteria for a candidate solution x to be $\mathcal{CD}(x, i) \leq \gamma$ where i is the input shape and the parameter γ controls the constrainedness of problems.

3.3 Conditional Molecular Design

A recent line of work focuses on generating molecules with specific properties [20, 19, 18], such as the molecular weight, the Wildman-Crippen partition coefficient [40], and a quantitative estimation of drug-likeness (QED) [3]. Kang and Cho [20] proposed a semi-supervised variational autoencoder that is trained on a set of existing molecules from the ZINC dataset [35] with only a partial annotation (i.e., only a fraction of the molecules are labelled with the property values).[4]

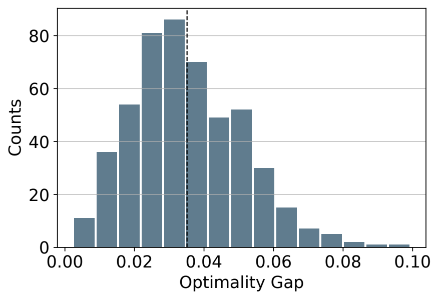
The model represents a generative process in which the input molecule x is generated from the distribution $p(x|z, y)$ that is conditioned on the molecule properties y and a latent variable z. The molecules are represented using SMILES strings [39] and are generated character-by-character. For the conditional generation of molecules with a specific property, we sample z from its prior and y from its prior conditioned on the specific property. A molecule representation $\hat{x}$ is obtained from y and z using the decoder's conditional probability $p(x|y, z)$,

$$\hat{x} = \arg\max p(x|y, z), \tag{2}$$

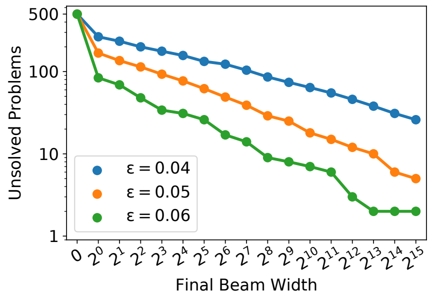
where Eq. (2) is approximated by a beam search.

Goal Criteria. We focus on the QED property [3], a measure of drug-likeness in the range $[0, 1]$ that is based on desirability functions for several molecular properties. We compute QED using RDKit [26] and evaluate the generated

[3] Obtained from github.com/Hippogriff/CSGNet.
[4] Obtained from github.com/nyu-dl/conditional-molecular-design-ssvae.



(a) Histogram of solution quality.



(b) Distribution of beam widths.

Fig. 2: TSP (100 nodes): Results for 500 random instances.

molecules based on the absolute difference between their QED and the desired QED. Formally, we define our goal criteria for a candidate solution x to be $|QED(x) - q| \leq \rho$ where q is the desired value of QED and the parameter ρ represents a bound on the deviation from the desired QED value and controls the constrainedness of the criteria.

4 Fat- and Heavy-tailed Behavior in Goal-Oriented Neural Sequence Decoding

In this section we demonstrate the existence of heavy-tailed behavior in goal-oriented neural sequence decoding. Due to space, we only present results for one benchmark problem, the Travelling Salesman Problem (TSP), however in Appendix A, we present similar results for the other three benchmarks.[5]

We consider a collection of 500 randomly generated TSP problem instances with 100 nodes solved using beam search with a beam width of 10. Figure 2a shows the distribution of solution quality presented as optimality gap ($\frac{\mathcal{C}(x)-\mathcal{C}(x^*)}{\mathcal{C}(x^*)}$) to match our goal criteria. The center of the distribution is around 0.03 with the mean (marked in a dashed line) at approximately 0.034. However, there is a small number of problems for which the optimality gap can be much higher (up to approximately 0.1).

Next, we consider the case of solving the goal-oriented problem where solutions must satisfy a bound on the optimality gap denoted as ε (as discussed in Section 3.1). We use complete anytime beam search (Algorithm 1) to solve the problems with the given bound as goal criteria. We start with a beam width of 1, and double the beam width in each iteration if no solution that satisfies the goal criteria is found. We record the beam width for which a satisfying solution was found representing the required search effort.

Figure 2b shows the search effort distribution for three different goal criteria $\varepsilon = 0.04$, $\varepsilon = 0.05$, $\varepsilon = 0.06$. The y-axis represents the number of unsolved

[5] All appendices appear in tidel.mie.utoronto.ca/pubs/rr-beam-appendix.pdf.

problems in log-scale, while the x axis represents the search effort (i.e., beam width) in discrete log_2-scale (i.e., in steps of $2^i, i = 0, 1, ...$) to match the behavior of the complete anytime beam search. We artificially add the step 0 (i.e., no search effort) to denote the total number of problems. For $\varepsilon = 0.05$ and $\varepsilon = 0.06$, there is a clear heavy-tailed behavior with a very low median (beam width of 1) and a slow decay of the tail over multiple orders of magnitude. In fact, not all problems were solved for the maximum beam width of $32,768$. Note that when $\varepsilon = 0.05$, 332 of the 500 problems are solved with a beam width of 1, while five problems could not be solved for a beam width of $32,768$. For a more constrained goal criteria of $\varepsilon = 0.04$, we still observe a fat-tailed behavior, however we see a noticeable increase in the difficulty of problems and the number of problems that could not be solve in the search effort limits is significantly higher. We could not analyze more constrained goal criteria due to the high computational cost, however we hypothesize that problems will become significantly harder and the heavy-tailed behavior will reduce, consistent with previous work [12, 7].

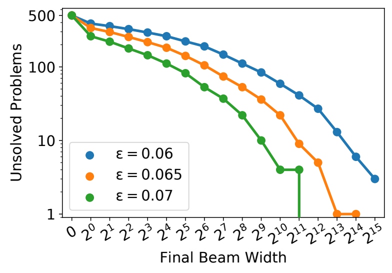
The above results suggest that goal-oriented beam search exhibits a heavy-tailed behavior in ensembles of random problems, similar to the one observed for other combinatorial and heuristic search algorithms. In these algorithms, much of the large variability in the search effort for ensembles of random problems was found to be associated with the algorithm, rather than the problem instances [12]. To isolate the variability of the search algorithm, in the next section we analyze the search effort distribution of a randomized variant of complete anytime beam seach on a single instance.

4.1 Fat- and Heavy-tailed Behavior on a Single Instance

In order to introduce randomization into beam search decoding of neural sequence models, we inject random noise in the inputs of the neural network that is being decoded using beam search. Injecting random noise in the inputs of a neural network is a known technique in the training of neural networks in order to improve their robustness [16].[6] Note that the noise injected to the network's inputs does not impact the goal test that is still based on the original input, i.e., the noise does not change the problem we are solving. The sole purpose of the noise is to introduce some randomness in the network's predicted probabilities and, as a result, in the beam search decoding.

For TSP instances, the inputs to the network consist of the locations of all nodes, expressed as two-dimensional coordinates normalized in the range [0, 1]. We inject noise to the network inputs by *adding* random noise drawn from a uniform distribution, $U(-0.01, 0.01)$. Figure 3a shows the distribution of search effort for 500 randomized runs (i.e., runs with different random injected noise) for different values of ε. We can see a fat-tailed behavior that indicates a significant

[6] Note that we are not aware of any direct connection between noise injection in training to increase robustness and our use of noise injection in testing to introduce randomness in the decoding process. However, it might be interesting to consider whether there is some underlying connection.



(a) TSP (100 nodes): Distribution of beam widths for 500 randomized runs on a single instance.

CVRP	Multiplicative uniform noise on locations and demand
Visual Prog. Synthesis	We flip, with small probability, pixels close to the edges of the shape
Molecule Generation	Additive Gaussian noise on random seeds

(b) Problem-specific noise injection to network's inputs. See Appendix B for detailed description.

variability is associated with the search method. Note that the results in Figure 3a were observed for a single, arbitrarily chosen instance. Experiments with other instances also yielded fat- and heavy-heavy tailed behavior, however we found large differences among instances: different instances exhibited different levels of fat- and heavy-tailedness for different levels of goal criteria constrainedness.

Table 3b briefly summarizes the problem-specific noise injection used for the other three benchmarks. A detailed description of the random noise injection and experimental results for these benchmarks appear in Appendix B.

The above results indicate that significant variability can be associated with the search algorithm itself. Previous works have exploited the large variability associated with the search algorithm to improve problem solving performance by introducing randomized restarts (see Section 2.2). In the next section, we propose a complete variant of beam search that incorporates randomized restarts and evaluate its impact on the distribution of search effort.

5 Randomized Restarting Neural-Guided Beam Search for Goal-Oriented Combinatorial Problems

We present randomized-restarting complete anytime beam search (RR-CAB), a variant of complete anytime beam search (Algorithm 1) that uses randomized beam search and a custom restart strategy. Algorithm 2 presents the pseudocode of RR-CAB, where the goal criteria and the restart strategy are passed as parameters. In each iteration the algorithm runs a randomized beam search (using a random seed) with a beam width that is determined by the restart strategy. The algorithm returns when one of the candidate solutions generated by the beam search satisfies the goal criteria.

In order to randomize the results of a beam search, we consider the following two options.

Beam search with injected input noise. Following the methodology in Section 4.1, we inject random noise to the inputs of the neural networks.

Algorithm 2 Randomized Complete Beam Search

```
function RR-CAB(goalCriteria, restartStrategy)
    $iteration \leftarrow 1$
    while not solved do
        $beamWidth \leftarrow restartStrategy(iteration)$
        $seed \leftarrow RandomSeed()$
        $candidates \leftarrow RandomizedBeamSearch(beamWidth, seed)$
        for $cand \in candidates$ do
            if $Satisfy(cand, goalCriteria)$ then
                return $cand$
        $iteration \leftarrow iteration + 1$
```

Stochastic beam search (SBS) [24]. SBS is a stochastic variant of beam search that samples k sequences without replacement from a sequence model and therefore produces randomized output. The level of diversity in SBS is controlled by the softmax temperature that modifies the conditional probability of each token during the decoding process. The probability of token y_t conditioned on the partial sequence $y_{1:t-1}$, is defined as a softmax normalization of the unnormallized log-probabilities, $\phi(y_t|x; y_{1:t-1})$, with a temperature T [24]:

$$p(y_t|x; y_{1:t-1}) = \frac{exp(\phi(y_t|x; y_{1:t-1})/T)}{\sum_{y'} exp(\phi(y'|x; y_{1:t-1})/T)}.$$

The temperature $T > 0$ and higher T leads to higher diversity. The default temperature is $T = 1$, where the predicted probabilities are not modified. In our experiments, we considered two temperature configurations: the default temperature $T = 1.0$ and a higher diversity temperature $T = 1.5$.

Note that we could not perform the analysis in Section 4.1 using SBS since, unlike input noise injection, we cannot guarantee that repeated runs with different beam widths will maintain similar conditional probability distributions (see discussion in Section 7). However, in RR-CAB, we are not interested in maintaining the same probability distributions across runs and therefore SBS can be used as a randomized variant of beam search.

5.1 Restart Strategies

A restart strategy is a sequence $(t_1, t_2, t_3, ...)$ of run lengths after which the search restarts. In goal-oriented neural sequence decoding, the sequence length is either fixed (e.g., in TSP and CVRP) or predicted by the network (e.g., in visual program synthesis or conditional molecule generation). If we want to allocate more search effort, we simply extend the beam width thus allowing more sequences to be tested against the goal criteria.

In each iteration, we run a beam search with a given beam width until a solution if found. In deterministic complete anytime beam search (Algorithm 1), the beam width is increased in each iteration. In RR-CAB, running a search with the same beam width multiple times leads to different results and can sometimes be more efficient than increasing the beam width. We therefore employ a custom restart strategy to determine the beam width in each iteration. We consider two popular restart strategies from the literature.

Fixed-Cutoff Strategy. Fixed-cutoff strategies [15] are simple strategies of the form $(t_c, t_c, ...)$ where t_c is a constant. This strategy is often not robust enough: a small t_c value might not be sufficient to solve all problems, while a larger value will be computationally inefficient.

Geometric Strategy. Geometric strategies [38] take the form $(r^0, r^1, r^2, r^3, ...)$ where the geometric factor r controls how fast the cutoff values grow. When $r = 2$ and randomization is not applied, this strategy has a similar behavior to the complete beam search procedure described in Section 2.1.

6 Empirical Results

In this section, we present empirical analysis of the performance of RR-CAB on the goal-oriented benchmarks. We compare results for the two randomization techniques (input noise injection and SBS) and the two restart strategies (geometric and fixed-cutoff) described in Section 5.

6.1 Results for the Travelling Salesman Problem (TSP)

We consider the same collection of 500 randomly generated TSP problems with 100 nodes used in Section 4. We analyze the results of RR-CAB with random noise injection and the two restart strategies: geometric with $r = 2$ and fixed-cutoff with beam width $\mathcal{B} = 8$. In order to directly compare the performance of a fixed-cutoff strategy and a geometric strategy, we organize the results of fixed-cutoffs beam search in batches of multiple beam searches with a constant beam width, such that they sum to the beam width of the corresponding beam search with geometric restarts. For example, we present results for a geometric restart policy for the beam width thresholds 1, 2, 4, 8, 16, 32, etc. In comparison, for fixed-cutoff restarts, the result for a threshold of 16 represents a batch of two beam searches, each with a constant beam width of 8.

Figure 4 compares the distribution of search effort of standard CAB and RR-CAB in the configurations described above. In general, the randomized variants tend to under perform for the very small beam width: problems that were easily solved without randomization do not benefit, and even suffer, from adding randomization. In particular, since we use a beam width $\mathcal{B} = 8$ for the fixed-cutoff strategy, solutions are only found starting from a threshold of 8. However, as we increased the search effort, we see that the randomized variants outperform standard CAB. For the more constrained problems, we see that the fixed-cutoff strategy significantly outperforms the geometric restarts strategy. This could be due the use of relatively large r chosen for fair comparison with CAB. For $\varepsilon = 0.6$, geometric restarts seem to have similar performance to fixed cut-offs.

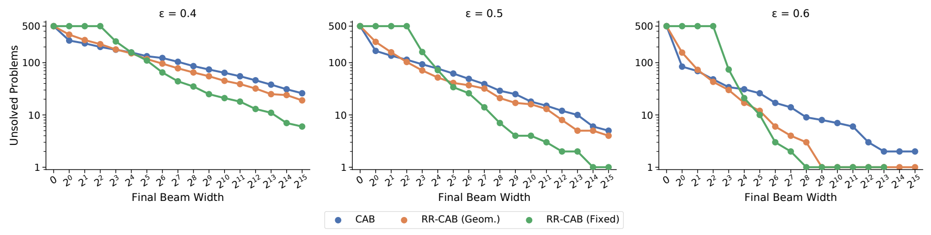


Fig. 4: TSP (100 nodes): Distribution of beam widths for 500 random instance for RR-CAB with input noise injection.

The inherent differences between the two restart strategies result in an apparent inferiority of fixed-cutoffs in smaller beam widths: in addition to having no solutions for beam widths smaller than 8, even for a beam width of 8 it underperforms since RR-CAB with geometric restarts has already made three randomized runs (for beam width 1, 2, and 4) that can lead to solutions. In practice, this is easily mitigated by using a restart policy that starts with geometric restarts before changing to fixed-cutoffs: $1, 2, 4, 8, 8, \ldots$ To maintain simple and clear comparison we do not adopt this enhancement in our evaluation.

Figure 5 shows similar comparison to Figure 4 where the beam search is randomized using SBS with a softmax temperature of $T = 1$ (top) and $T = 1.5$ (bottom). Again, we see that introducing randomization to CAB leads to better performance. Using softmax temperature of $T = 1.5$ exhibits better performance and manages to solve more hard instances faster. Interestingly, for SBS we find that geometric restarts are approximately as good as fixed-cutoff strategy.

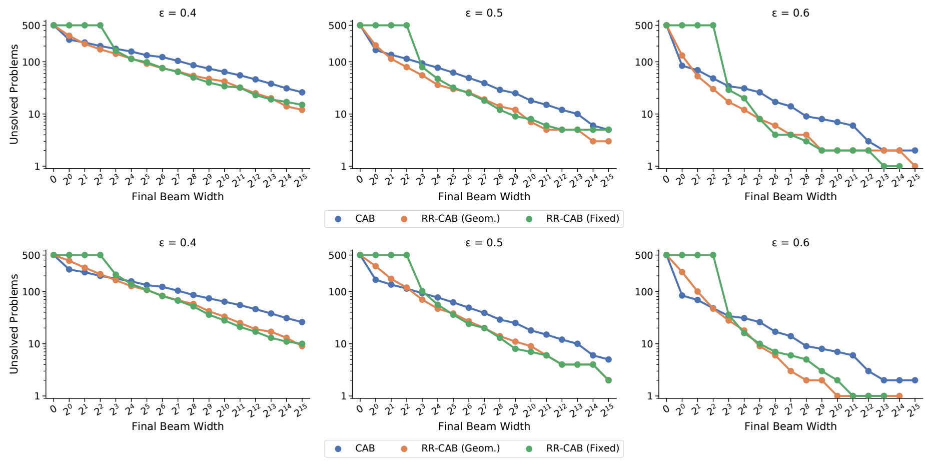


Fig. 5: TSP (100 nodes): Distribution of beam widths for 500 random instances for RR-CAB with SBS using $T = 1.0$ (top) and $T = 1.5$ (bottom).

The above results show that introducing randomization in the search can help solve some of the hardest instances faster. Consistent with previous work on CSPs and SAT, the impact on more relaxed instances tends to be more significant [12]. However, note that we cannot analyze the impact of RR-CAB on more constrained instances due to computational limitations and even for $\varepsilon = 0.4$, using randomization seems to have positive impact on the performance.

6.2 Results for the Other Benchmarks

Figure 6, Figure 7, and Figure 8 show the results for CVRP, visual program synthesis and conditional molecule generation, respectively. For CVRP and molecule generation, we found that, similar to TSP, a temperature of $T = 1.5$ yields better results when using SBS. In visual program synthesis, higher temperature did not lead to better results and we present results for $T = 1$.

In the visual program synthesis problem, the number of potential expansions of each of the beam candidates is much higher than the other problems (approximately 400, compared to 36-100 in the other problems). Therefore, when using SBS for this problem, we only consider the top 50 extensions of each candidate. Practically, it is unlikely that an extension of partial hypothesis that is not in the most likely 50 extensions will lead to a hypothesis that will be returned by the beam search. However, when applying randomization it may have the undesired outcome of promoting very low-ranked hypotheses and we therefore consider only the top 50 hypotheses.

Consistent with our results for TSP, we find that RR-CAB solves some the hardest problems faster and outperforms the baseline. As in TSP, when using random noise injection, the fixed cut-offs strategy tends to outperforms the geometric strategy.

7 Discussion and Future Work

Our empirical results suggest that RR-CAB exploits the variability associated with the search procedure and significantly outperforms the baseline by solving some of the hardest problems faster. In this section, we discuss different aspects related to RR-CAB and directions for future work.

Randomization Techniques. We consider two techniques that can randomize the results of a beam search: input noise injection and SBS. While both techniques introduce randomization to the predicted probabilities, there are some important differences between them. A key limitation of the noise injection technique is that it needs to be tailored for each problem. In our work, we had to manually try different randomization approaches in order to find one that would generate sufficient variability on a single instance without making the problem significantly harder across different runs. Alternatively, an inherent limitation of SBS is that we are unable to guaranteed that repeated runs with different beam

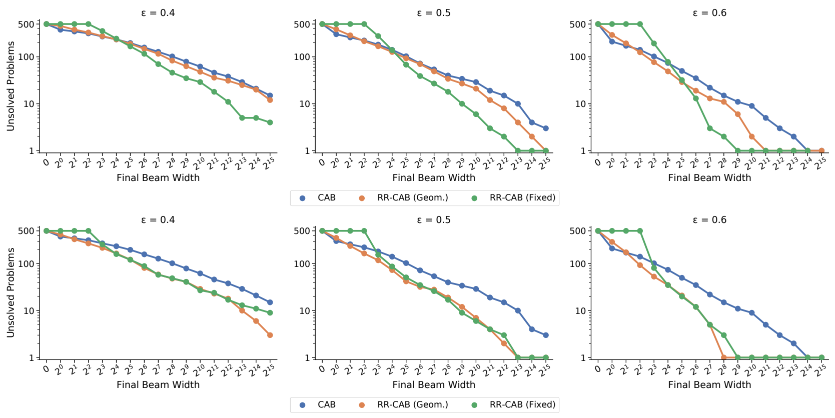


Fig. 6: CVRP (50 nodes): RR-CAB with noise injection (top), SBS (bottom).

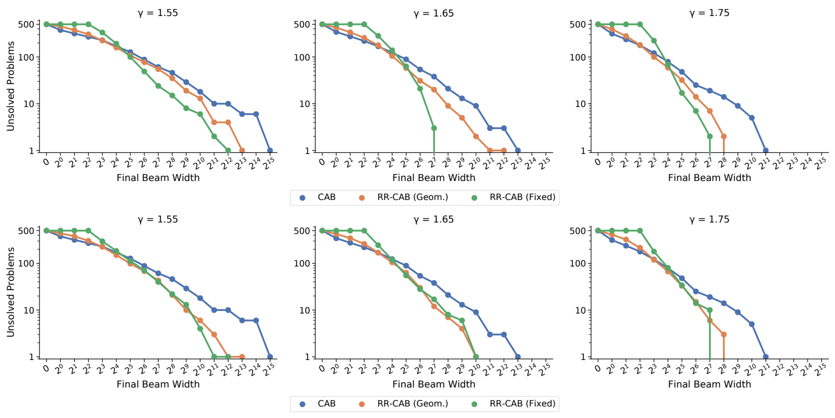


Fig. 7: Visual Program Synthesis: RR-CAB with noise injection (top), SBS (bottom).

widths will maintain similar conditional probability distributions. The implication of this limitation is that we cannot analyze the search effort distribution of SBS on a single problem instance, as we do for beam search with noise injection in Section 4.1. As future work, it is interesting to investigate other generic ways of introducing noise into the decoding process. Potential directions include applying noise to hidden units [31, 4] or using dropout [34] in inference.

Restart Strategies and Parallelization. We focused on two well known restart strategies: fixed-cutoff and geometric restarts. Previous works in combinatorial optimization has considered more advanced restart strategies such as Luby's universal strategy [28] and dynamic and learning restart strategies (e.g., [21]). Investigating ways to incorporate such strategies in RR-CAB is an interesting direction for future work.

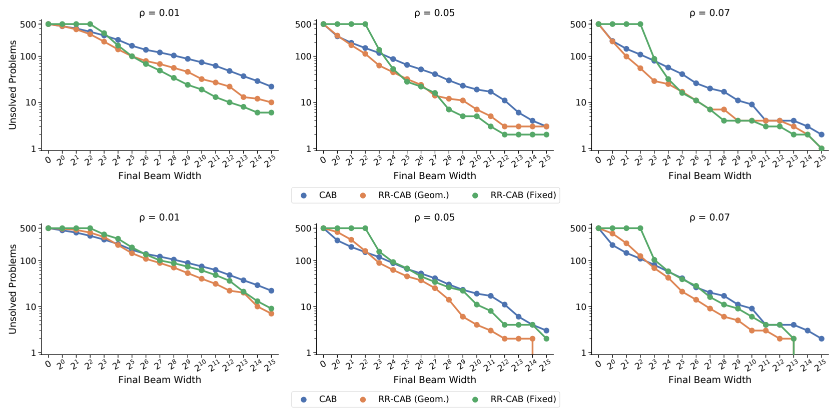


Fig. 8: Molecule Generation: RR-CAB with noise injection (top), SBS (bottom).

A key challenge in designing restart strategies for beam search is their ability to be parallelized on a GPU. In our experiments, we present results for the fixed-cutoff restart strategy by batching together beam searches and comparing these results to the corresponding final beam width of a geometric strategy. As we start investigating more complicated restart strategies, such as Luby's universal strategy [28], we will not be able to batch the results together to maintain comparability. Furthermore, even in our comparison, it is not clear that a set of four beam search instances, each with a beam width of 8 and executed together on a GPU, is comparable to one beam search with a beam width of 32. Our work, therefore, raises the need for well-defined evaluation metrics that can be used to compare the results of parallelized complete beam searches with different restart strategies, even when it not possible to batch together runs as we currently do.

8 Conclusion

In this work we show that fat- and heavy-tailed behavior, that was previously observed for several combinatorial and heuristic search algorithms, can be observed for complete anytime beam search in goal-oriented neural sequence decoding. We perform an extensive empirical analysis, across four goal-oriented benchmarks, and find fat- and heavy-tailed behavior in the distribution of search efforts of beam search. Inspired by previous work on combinatorial and heuristic search, we propose a randomized restarting variant of complete anytime beam search, RR-CAB, and study the impact of different randomization techniques and restart strategies. Our experiments show that RR-CAB solves some of the hardest problems faster and outperforms the baseline. Our work raises interesting questions on the impact of parallelization on the development and evaluation of randomized restarting beam search algorithms and highlights directions for future work.

Acknowledgements

We thank the anonymous reviewers for their valuable feedback. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

References

1. Applegate, D., Bixby, R., Chvatal, V., Cook, W.: Concorde TSP solver (2006)
2. Balog, M., Gaunt, A., Brockschmidt, M., Nowozin, S., Tarlow, D.: Deepcoder: Learning to write programs. In: International Conference on Learning Representations (ICLR) (2017)
3. Bickerton, G.R., Paolini, G.V., Besnard, J., Muresan, S., Hopkins, A.L.: Quantifying the chemical beauty of drugs. Nature chemistry **4**(2), 90 (2012)
4. Cho, K.: Noisy parallel approximate decoding for conditional recurrent language model. arXiv preprint arXiv:1605.03835 (2016)
5. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: EMNLP (2014)
6. Chopra, S., Auli, M., Rush, A.M.: Abstractive sentence summarization with attentive recurrent neural networks. In: North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT). pp. 93–98 (2016)
7. Cohen, E., Beck, J.C.: Fat- and heavy-tailed behavior in satisficing planning. In: AAAI Conference on Artificial Intelligence (AAAI). pp. 6136–6143 (2018)
8. Cohen, E., Beck, J.C.: Local minima, heavy tails, and search effort for GBFS. In: International Joint Conferences on Artificial Intelligence (IJCAI). pp. 4708–4714 (2018)
9. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.M.: Learning heuristics for the tsp by policy gradient. In: International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR). pp. 170–181 (2018)
10. Gehring, J., Auli, M., Grangier, D., Yarats, D., Dauphin, Y.N.: Convolutional sequence to sequence learning. In: International Conference on Machine Learning (ICML). pp. 1243–1252 (2017)
11. Gomes, C.: Randomized backtrack search. In: Milano, M. (ed.) Constraint and integer programming: Toward a unified methodology, pp. 233–291. Springer Science & Business Media (2003)
12. Gomes, C.P., Fernández, C., Selman, B., Bessière, C.: Statistical regimes across constrainedness regions. Constraints **10**(4), 317–337 (2005)
13. Gomes, C.P., Selman, B., Crato, N.: Heavy-tailed distributions in combinatorial search. In: International Conference on Principles and Practice of Constraint Programming (CP). pp. 121–135. Springer (1997)
14. Gomes, C.P., Selman, B., Crato, N., Kautz, H.: Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. Journal of automated reasoning **24**(1), 67–100 (2000)
15. Gomes, C.P., Selman, B., Kautz, H., et al.: Boosting combinatorial search through randomization. National Conference on Artificial Intelligence (AAAI) **98**, 431–437 (1998)

16. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
17. Helsgaun, K.: An extension of the lin-kernighan-helsgaun TSP solver for constrained traveling salesman and vehicle routing problems. Roskilde: Roskilde University (2017)
18. Jin, W., Barzilay, R., Jaakkola, T.: Junction tree variational autoencoder for molecular graph generation. In: International Conference on Machine Learning (ICML). pp. 2323–2332 (2018)
19. Jin, W., Yang, K., Barzilay, R., Jaakkola, T.: Learning multimodal graph-to-graph translation for molecule optimization. In: International Conference on Learning Representations (ICLR) (2018)
20. Kang, S., Cho, K.: Conditional molecular design with deep generative models. Journal of chemical information and modeling **59**(1), 43–52 (2018)
21. Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., Selman, B.: Dynamic restart policies. National Conference on Artificial Intelligence (AAAI) pp. 674–681 (2002)
22. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. In: Conference on Neural Information Processing Systems (NeurIPS). pp. 6348–6358 (2017)
23. Kool, W., van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (ICLR) (2019)
24. Kool, W., Van Hoof, H., Welling, M.: Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In: International Conference on Machine Learning (ICML). pp. 3499–3508 (2019)
25. Lample, G., Charton, F.: Deep learning for symbolic mathematics. In: International Conference on Learning Representations (2019)
26. Landrum, G.: Rdkit: Open-source cheminformatics, http://www.rdkit.org
27. Liu, Y., Wu, Z., Ritchie, D., Freeman, W.T., Tenenbaum, J.B., Wu, J.: Learning to describe scenes with programs. In: International Conference on Learning Representations (ICLR) (2018)
28. Luby, M., Sinclair, A., Zuckerman, D.: Optimal speedup of las vegas algorithms. Information Processing Letters **47**(4), 173–180 (1993)
29. Nakhost, H., Hoffmann, J., Müller, M.: Resource-constrained planning: A monte carlo random walk approach. In: International Conference on Automated Planning and Scheduling (ICAPS) (2012)
30. Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. In: Conference on Neural Information Processing Systems (NeurIPS). pp. 9839–9849 (2018)
31. Poole, B., Sohl-Dickstein, J., Ganguli, S.: Analyzing noise in autoencoders and deep networks. arXiv preprint arXiv:1406.1831 (2014)
32. Resnick, S.I.: Heavy-tail phenomena: probabilistic and statistical modeling. Springer Science & Business Media (2007)
33. Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., Maji, S.: Csgnet: Neural shape parser for constructive solid geometry. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5515–5523 (2018)
34. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1), 1929–1958 (2014)
35. Sterling, T., Irwin, J.J.: Zinc 15–ligand discovery for everyone. Journal of chemical information and modeling **55**(11), 2324–2337 (2015)

36. Tian, Y., Luo, A., Sun, X., Ellis, K., Freeman, W.T., Tenenbaum, J.B., Wu, J.: Learning to infer and execute 3d shape programs. In: International Conference on Learning Representations (ICLR) (2019)
37. Vinyals, O., Toshev, A., Bengio, S., Erhan, D.: Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. IEEE Transactions on Pattern Analysis and Machine Intelligence **39**(4), 652–663 (2017)
38. Walsh, T.: Search in a small world. In: International Joint Conference on Artificial Intelligence (IJCAI). pp. 1172–1177 (1999)
39. Weininger, D.: Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. Journal of chemical information and computer sciences **28**(1), 31–36 (1988)
40. Wildman, S.A., Crippen, G.M.: Prediction of physicochemical parameters by atomic contributions. Journal of chemical information and computer sciences **39**(5), 868–873 (1999)
41. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning **8**(3-4), 229–256 (1992)
42. Zhang, W.: Complete anytime beam search. In: National Conference on Artificial Intelligence (AAAI). pp. 425–430 (1998)
43. Zohar, A., Wolf, L.: Automatic program synthesis of long programs with a learned garbage collector. In: Conference on Neural Information Processing Systems (NeurIPS). pp. 2094–2103 (2018)