Learning Beam Search:

Utilizing Machine Learning to Guide Beam Search for Solving Combinatorial Optimization Problems[1]

Marc Huber and Günther Raidl
{mhuber|raidl}@ac.tuwien.ac.at

Institute of Logic and Computation
TU Wien

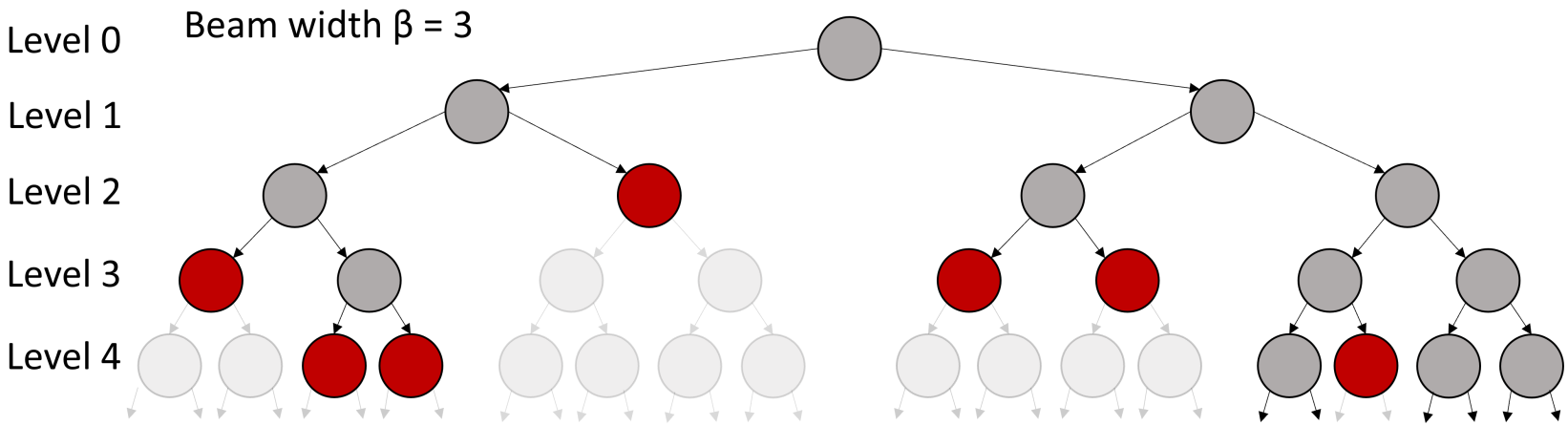
LOD2021, October 4 – 8, 2021

[1]This project is partially funded by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Foundation (FWF), grant W1260-N35.

Motivation

Beam Search (BS):

- Graph search algorithm to heuristically solve combinatorial optimization problems.
- Traverses a state graph from a root node in a breadth-first-search manner to find a best path to a target node.
- Evaluates reached nodes at each level and only expands the β most promising nodes.



Motivation

How to evaluate and select nodes?

- Evaluation function: $f(v) = g(v) + h(v)$, where
  - $g(v)$: length of a best path from the root r to node v.
  - $h(v)$: heuristic guidance function that estimates the further length to go from node v in the best case.

Guidance function:

- Typically developed in a highly problem-specific way.
- Often challenging as the function not only needs to deliver good estimates but also needs to be fast.

Idea:

- Use a machine learning (ML) model as guidance function $h(v)$ in BS.
- Train ML model offline by "self-play" on many representative randomly generated problem instances.

$\rightarrow$ **Learning Beam Search (LBS)**

Related Work

- **A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play:**

  [Silver et al., 2018]

  - Based on Monte Carlo Tree Search in which a Neural Network (NN) estimates the expected outcome of the game from a position.
  - NN also outputs a vector of move probabilities.
  - Learns expected outcome and move probabilities entirely from self-play.

- **Learning Beam Search Policies via Imitation Learning:**

  [Negrinho et al., 2018]

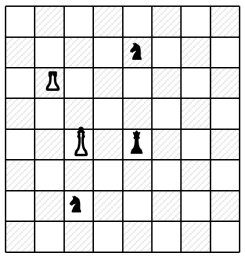
  - Meta-Algorithm that learns a policy for an abstract structured prediction problem to traverse the combinatorial search space of beams.
  - Pure theoretical work.

Learning Beam Search: Procedure

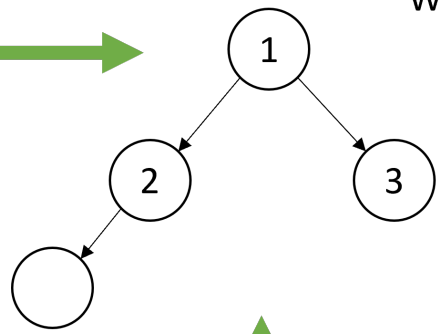
1. Initialize ML model randomly, where
   - Input: Problem-specific feature vector that captures relevant information from the state of v and the problem instance.
   - Output: $h(v)$ = Approximation of estimated further length to go from v.
2. Generate a random problem instance.
3. Perform a BS, guided by the ML model.
   - From each non-terminal node v with small probability:
     - Perform a **nested BS (NBS)** to obtain a training sample.
     - Store training sample in a limited size FIFO **replay buffer**.
4. Train ML model on FIFO replay buffer data.
5. Repeat steps 2-4 until a stopping criterion is fulfilled.

Learning Beam Search: Procedure (steps 2-4)

Randomly generated problem instance

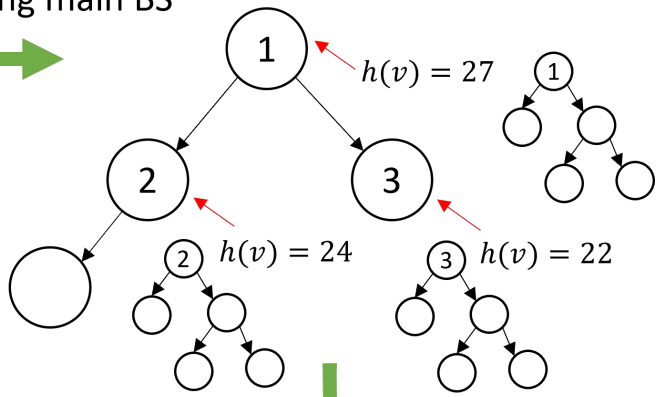


Main BS with beam width β
(solves problem instance)



While performing main BS

NBS with beam width β‘
(generates α training data)

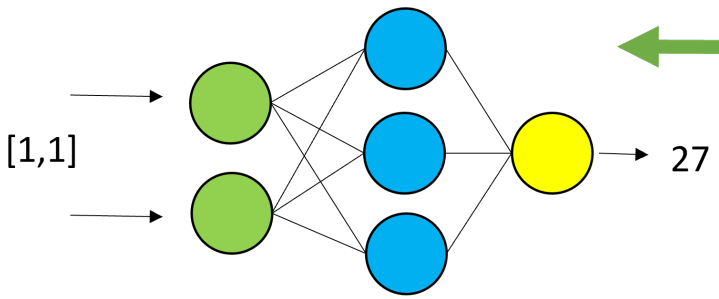


FIFO replay buffer of size γ
(stores training data,
removes older samples)

Feature Vectors	Targets
[1,1]	27
[1,4]	24
[3,5]	22
⋮	⋮

Train ML model

ML model
(guides BS)



Experiments: Considered Problems

Given: set of m input strings $S = \{s_1, \ldots, s_m\}$ over alphabet Σ.

- **Longest Common Subsequence (LCS):** find a longest string that appears as subsequence in any string of S.

  Example: $m = 2$, $|\Sigma| = 3$

  s_1: ABBA
  s_2: CABA
  $\Rightarrow$ ABA.

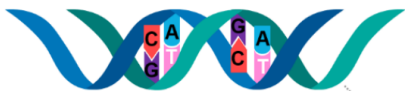
- **Constrained Longest Common Subsequence (CLCS):** extends LCS by a pattern string P that must appear as subsequence in a feasible solution.

  Example: P=AB

  s_1: ABBA
  s_2: CABA
  $\Rightarrow$ ABA.

Experiments: Applications and Complexity

- Many applications in computational biology, text editing, etc.
  - e.g. to compare two DNA or protein sequences to learn how homologous they are.



- Can be solved efficiently in time $\mathcal{O}(n^2)$ for $m = 2$ strings by dynamic programming (n: string length).
- NP-hard for general m.
- State-of-the-art heuristic approaches for large m and n are based on Beam Search [Djukanovic et al., 2020].

Experiments: State Graph

Directed acyclic graph $G = (V, A)$**:**

- States (nodes) in V are represented by position vectors.
- Actions (arcs) in A refers to transitioning from one to another state by appending a valid letter to a partial solution.
- Actions in a state are valid if the letter to append exists in each remaining string of $s_1, \ldots, s_m$.
  - Additionally, for CLCS, the state that would be obtained by it must allow to cover the remaining pattern string.

Feature Vectors (input for ML model):

- LCS: remaining string length + minimum letter appearances.
- CLCS: remaining string length + remaining string length of the pattern string + minimum letter appearances.

Targets (output of ML model): expected (C)LCS length.

Experiments: LCS Benchmark Instances

- **`rat` instance set [Shyu and Tsai, 2009]:**
  - 20 instances composed of sequences from rat genomes.
  - all differ in their combinations of values for
    - $n = 600$
    - $m \in \{10, 15, 20, 25, 40, 60, 80, 100, 150, 200\}$
    - $|\Sigma| \in \{4, 20\}$.
  - instances are close to independent random strings.
- **`BB` instance set [Blum and Blesa, 2007]:**
  - 80 random instances.
  - ten instances per combination of values for
    - $n = 1000$
    - $m \in \{10, 100\}$
    - $|\Sigma| \in \{2, 4, 8, 24\}$.
  - strings of each instance exhibit large similarities.

Test setting:

- Julia 1.6 using the Flux package for the NN.
- Intel Xeon E5-2640 v4 2.40GHz, all runs single-threaded, memory limit of 20 GB.

Experiments: Calibration of Hyperparameters

Network size has a strong impact on the computation times.

- Make the NN as small as possible.

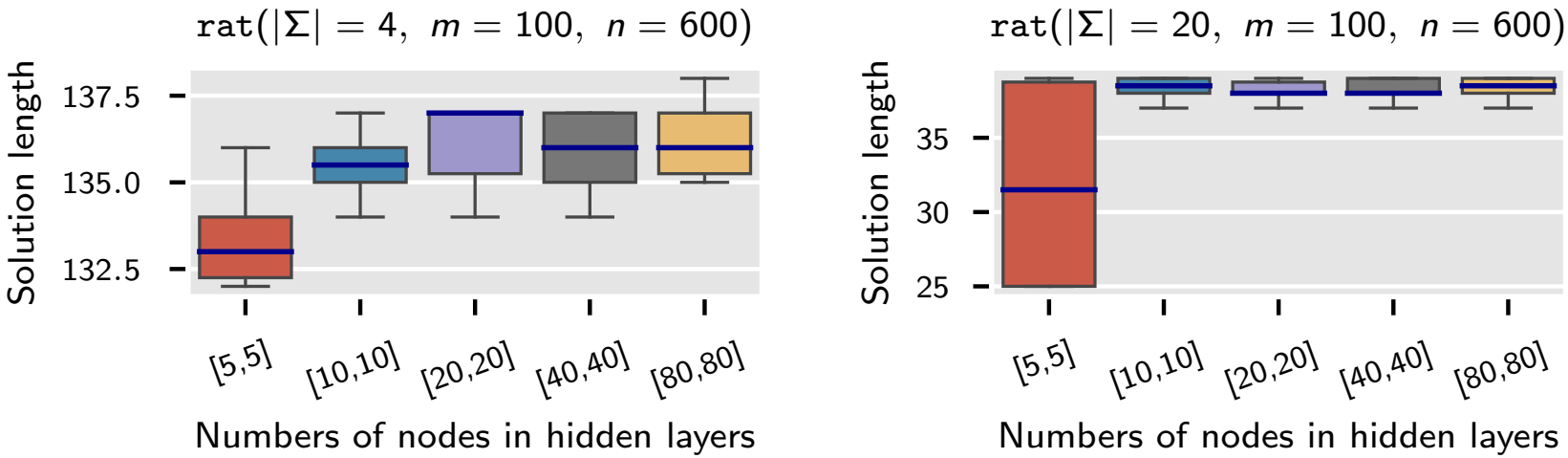


Figure: Impact of the numbers of nodes in the hidden layers on the solution length of LBS on `rat` benchmark instances. Ten runs per configuration.

Robust choice: 20 nodes in both hidden layers.

Experiments: Calibration of Hyperparameters

Impact of beam width on the solution quality.

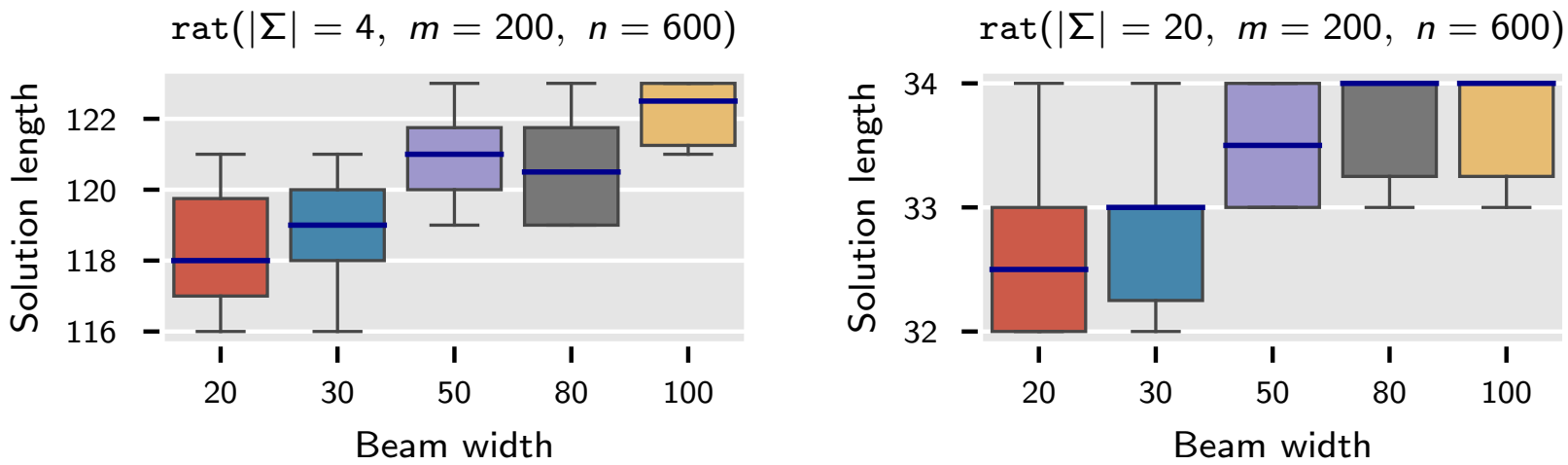


Figure: Impact of beam width $\beta = \beta'$ in training and testing on `rat` instances. Ten runs per configuration.

Good choice: $\beta = \beta' = 50$.

Experiments: Approximation of the real LCS length

How well does a NN trained by LBS predict the real LCS length?

- Approximate exact LCS lengths by applying the so far leading BS with EX guidance function [Djukanovic et al., 2020].

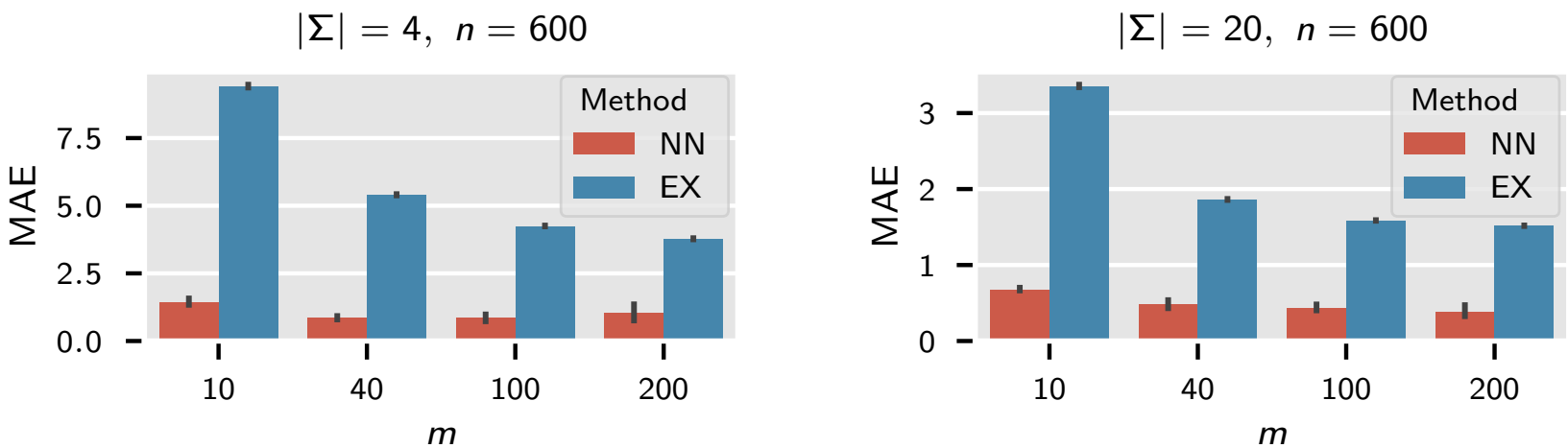


Figure: Mean absolute error of the trained NNs and EX on 10000 test samples, created by a BS with EX guidance function.

NNs approximate the LCS lengths much better than EX.

Experiments: LCS Results on `rat` and BB Benchmark Instances

Compared LBS to the state-of-the-art methods from the literature [Djukanovic et al., 2020].

- All training with LBS was done with $\beta = \beta' = 50$.
- Tests on the benchmark instances were performed with two different beam widths:
  - Low (computation) time with $\beta = 50$.
  - High quality with $\beta = 600$.

Achieved new best results for

- low time experiments: **13 out of 28**.
- high quality experiments: **7 out of 28**.

Runtimes are very similar to those of reimplemented BS-EX.

Experimens: CLCS Benchmark Instances and Results

Benchmark set [Djukanovic et al., 2020]:

- Ten instances for each combination of
  - $n \in \{100, 500, 1000\}$
  - $m \in \{10, 50, 100\}$
  - $|\Sigma| \in \{4, 20\}$
  - $\frac{n}{|P|} \in \{4, 10\}$, where P denotes the pattern string.

CLCS Results:

- All training with LBS was done with $\beta = \beta' = 50$
- For tests on the benchmark instances $\beta = 2000$ was used.

Achieved new best results in ten out of 36 cases
Scores worse in only two out of 36 cases.

Conclusions and Future Work

- Presented a general learning beam search framework to solve combinatorial optimization problems.
- Experiments on the LCS and the CLCS problems clearly show that this learning approach can be highly effective.

Specifically for the LCS and CLCS problems:

- Aim at relying on different features that just describe the distribution of remaining input string lengths.

General improvement potential for LBS:

- Come up with alternative optimization targets and loss functions for the training.
- Parallelization and the utilization of GPUs.

Thank you for your attention!

Questions?

References I

Blum, C. and Blesa, M. J. (2007).
Probabilistic beam search for the longest common subsequence problem.
In Stützle, T. et al., editors, *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics*, pages 150–161. Springer.

Djukanovic, M., Berger, C., Raidl, G. R., and Blum, C. (2020).

On solving a generalized constrained longest common subsequence problem.
In Olenev, N. et al., editors, *Optimization and Applications*, volume 12422 of *LNCS*, pages 55–70. Springer.

References II


Djukanovic, M., Raidl, G. R., and Blum, C. (2020).
A beam search for the longest common subsequence problem guided by a novel approximate expected length calculation.
In Nicosia, G. et al., editors, *Proc. of the 5th Int. Conf. on Machine Learning, Optimization and Data Science*, volume 11943 of *LNCS*, pages 154–167. Springer.


Negrinho, R., Gormley, M., and Gordon, G. J. (2018).
Learning beam search policies via imitation learning.
In Bengio, S. et al., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 10652–10661. Curran Associates, Inc.


Shyu, S. J. and Tsai, C.-Y. (2009).
Finding the longest common subsequence for multiple biological sequences by ant colony optimization.
Computers & Operations Research, 36(1):73–91.

References III


Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. (2018).
A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play.
Science, 362(6419):1140–1144.