Learning Beam Search: Utilizing Machine Learning to Guide Beam Search for Solving Combinatorial Optimization Problems

Günther R. Raidl

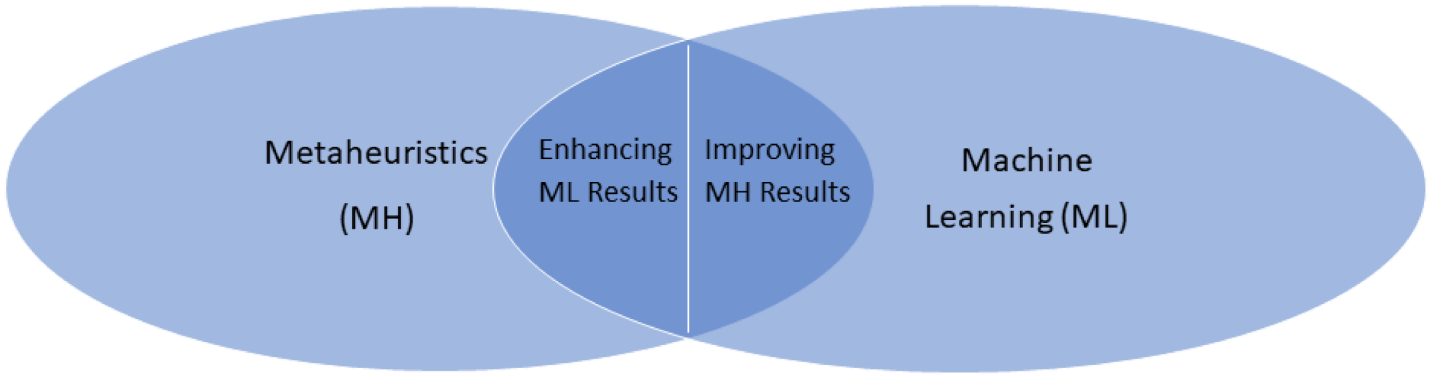
Institute of Logic and Computation, TU Wien, Austria,
raidl@ac.tuwien.ac.at

ELLIIT Hybrid AI Workshop, Linköping
November 1, 2022

TU WIEN Informatics ALGORITHMS AND COMPLEXITY GROUP

Combinatorial Optimization and Learning

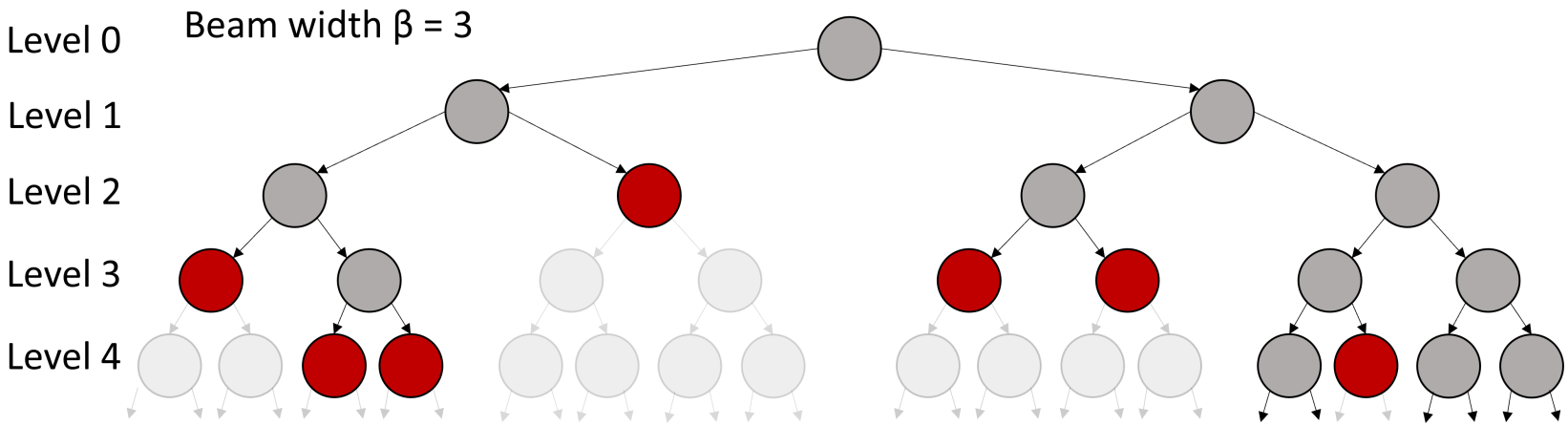
- AI/machine learning boom also hit the area of heuristic methods for combinatorial optimization



- Focus here:
  - utilize learning to better solve a combinatorial optimization problem
  - with **Beam Search (BS)**
- Let's keep it simple!

Beam Search (BS)

- Limited breadth first search
- Evaluates reached nodes at each level and only expands the β most promising nodes.



- Arcs have associated lengths (costs, rewards,. . .)
- Assume maximization
- Interested in longest path

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 manually 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 an 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 (MCTS) in which a Neural Network (NN) estimates for a current position
    - the expected outcome of the game ($\rightarrow$ "value")
    - and provides a policy (probability distribution) on next moves
  - Learning by self-play, i.e. reinforcement learning in which MCTS acts as a "quality amplifier".

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

  Negrinho et al. (2018)

  - Learn a BS policy for an abstract structured prediction problem to traverse the combinatorial search space of beams.
  - Pure theoretical work.

LBS: High-Level Procedure

ML model $= h(v)$:

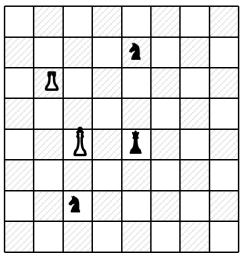
- Input: Problem-specific feature vector representing a node v (state) and possibly some problem instance features.
- Output: estimated further length to go from v.
- Randomly initialized

Repeat until stopping criterion fulfilled:

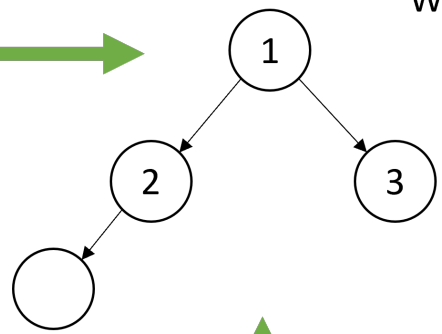
- Create representative random problem instance
- Perform a BS guided by the ML model
  - From each non-terminal node v with small probability:
    - **Perform a Nested BS (NBS)** to obtain approx. length-to-go
    - Store training sample in a limited size FIFO replay buffer
- Perform few training steps with batches of randomly sampled data from replay buffer

LBS: High-Level Procedure

Randomly generated problem instance

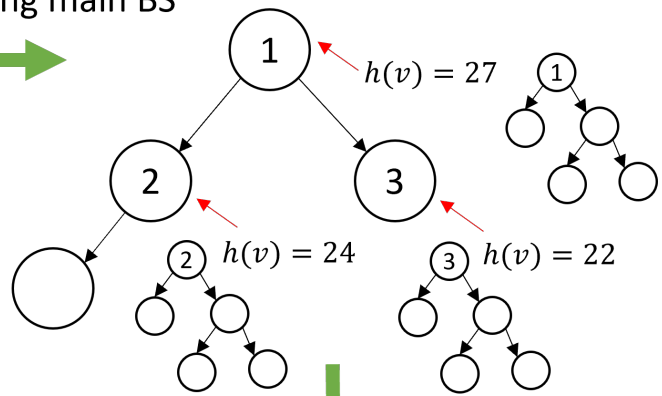


Main BS with beam width β
(solves problem instance)



While performing main BS

NBS calls from selected nodes
(generates α training data)

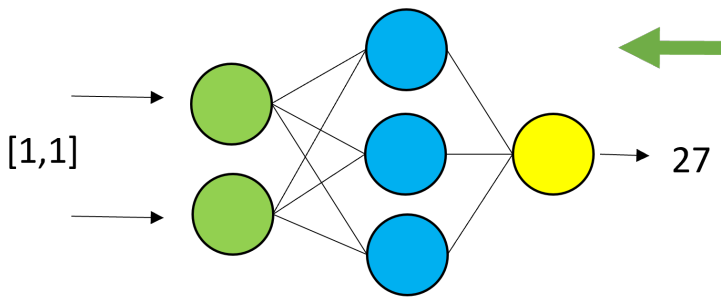


FIFO replay buffer of size γ
(stores training data,
removes older samples)

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

Train ML model

ML model (e.g. NN)
(guides BS)



Experiments on Longest Common Subsequence (LCS) Problem

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

Longest Common Subsequence Problem

- 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 approach for large m and n based on BS with a theoretically derived expected solution length (EX) calculation (Djukanovic et al., 2020b).

Variant: Constrained LCS Problem (CLCS)

Extends LCS problem by a pattern string P that must appear as subsequence in a feasible solution.

Example:

P: CA

s_1: ABCBAB

s_2: CABBA

$\Rightarrow$ LCS: ABBA but CLCS: CAB

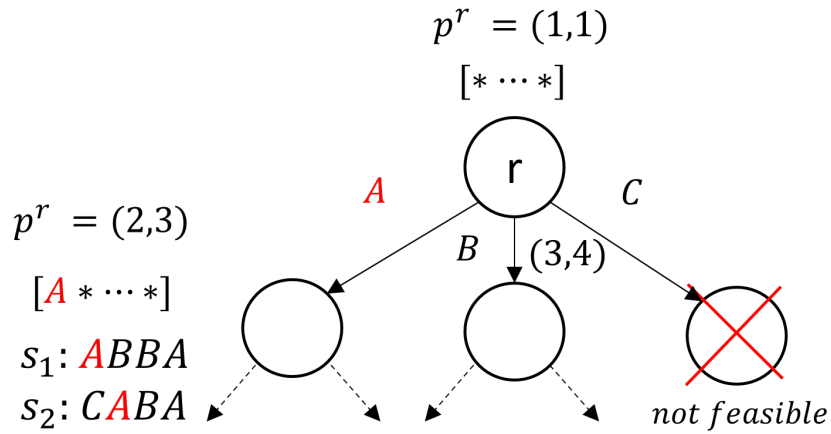
State Graph for LCS Problem

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

- States (nodes) in V are represented by position vectors, where root node r has position $\vec{1}$.

$$\begin{aligned} s_1\colon\ & ABBA \\ s_2\colon\ & CABA \end{aligned} \quad \Rightarrow (p_i^v)_{i=1,2} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}.$$

- Actions (arcs) in A refers to transitioning from one to another state by appending a feasible letter to a partial solution.
- Actions in a state are feasible if the letter to append exists in each remaining string of $s_1, \ldots, s_m$.



State Graph for LCS Problem

Feature vectors (input for ML model):

- remaining string lengths:

$$q_i^v = |s_i| - p_i^v + 1, \quad \forall\, i = 1 \ldots, m.$$

$$\begin{matrix} s_1\colon\ AABA \\ s_2\colon\ CABA \end{matrix}, \ (p_i^v)_{i=1,2} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} \Rightarrow (q_i^v)_{i=1,2} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

sorted non-decreasingly to get rid of symmetries

- minimum letter appearances:

$$o_c^v = \min_{i=1,\ldots,m} |s_i[p_i^v, |s_i|]|_c, \quad c \in \Sigma.$$

$$\begin{matrix} s_1\colon\ AABA \\ s_2\colon\ CABA \end{matrix}, \ o_A^v = \min\{3, 2\} = 2$$

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: LCS Results on rat and BB Benchmark Instances

Compared LBS to the state-of-the-art methods from the literature Djukanovic et al. (2020b).

- All training with LBS was done with $\beta = 50$.
- 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 similar to those of BS with EX.

Experimens: CLCS Benchmark Instances and Results

Benchmark set Djukanovic et al. (2020a):

- 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 $\beta = 50$
- For tests on the benchmark instances $\beta = 2000$ was used.

Achieved new best results in 10 out of 36 cases
Scores worse in only 2 out of 36 cases.

ML Model: Simple Multilayer Perceptron

- Make the NN as small as possible.

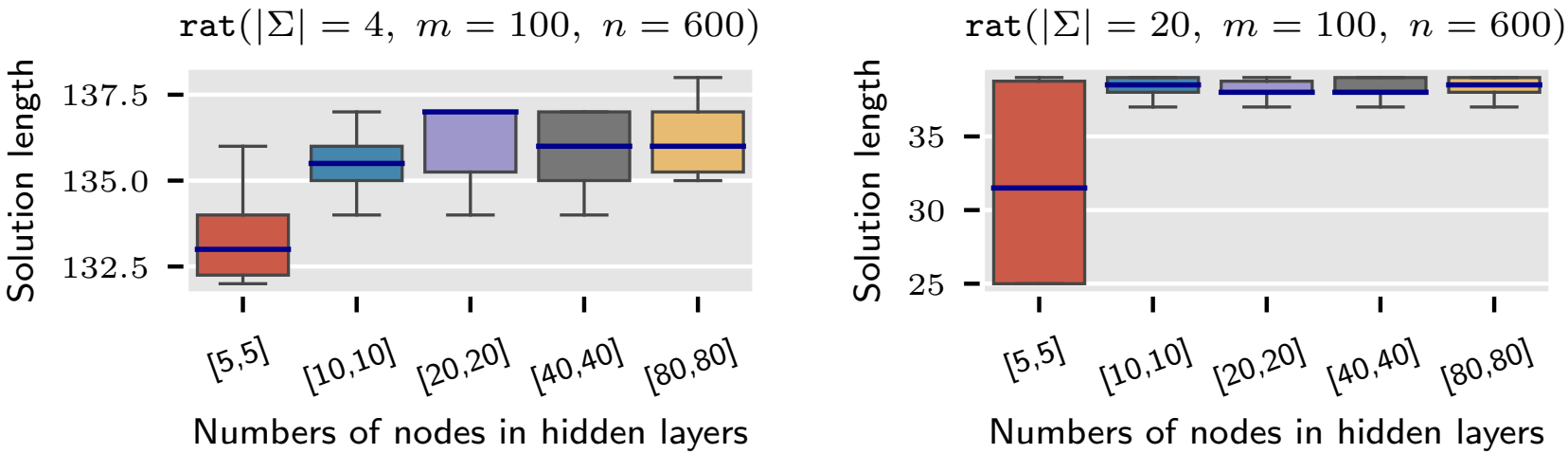


Figure: Impact of the numbers of nodes in the hidden layers on the solution length of LBS on `rat` benchmark instances.

Robust choice: 20 nodes in both hidden layers.

Impact of Beam Width on Solution Quality

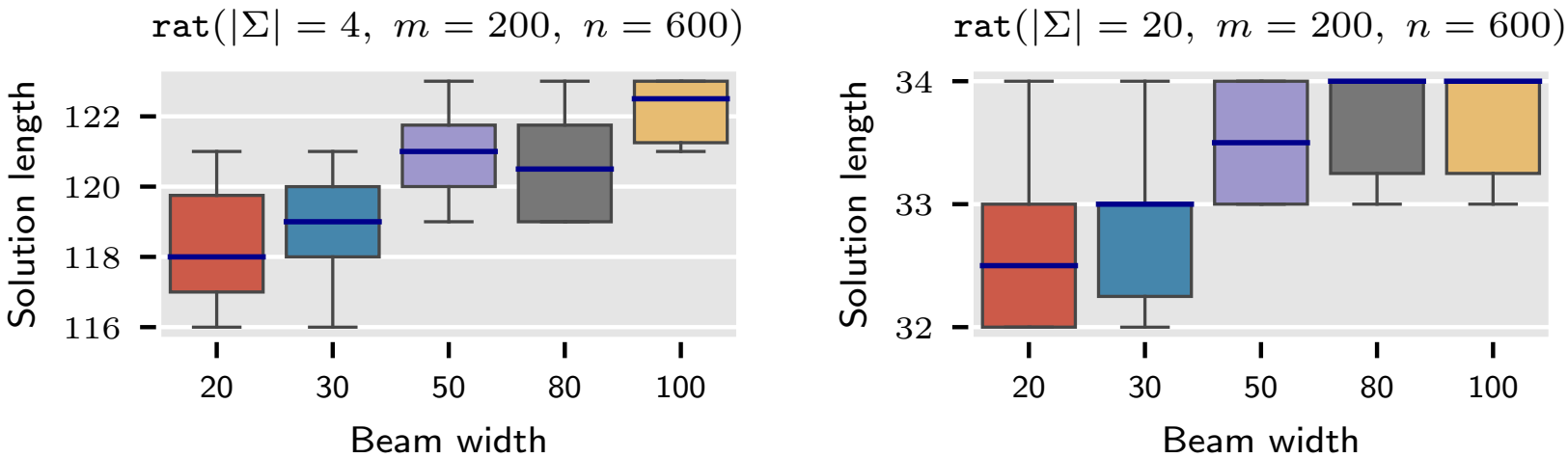


Figure: Impact of beam width β' in training and testing on `rat` instances.

Reasonable choice: $\beta = 50$.

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. (2020b).

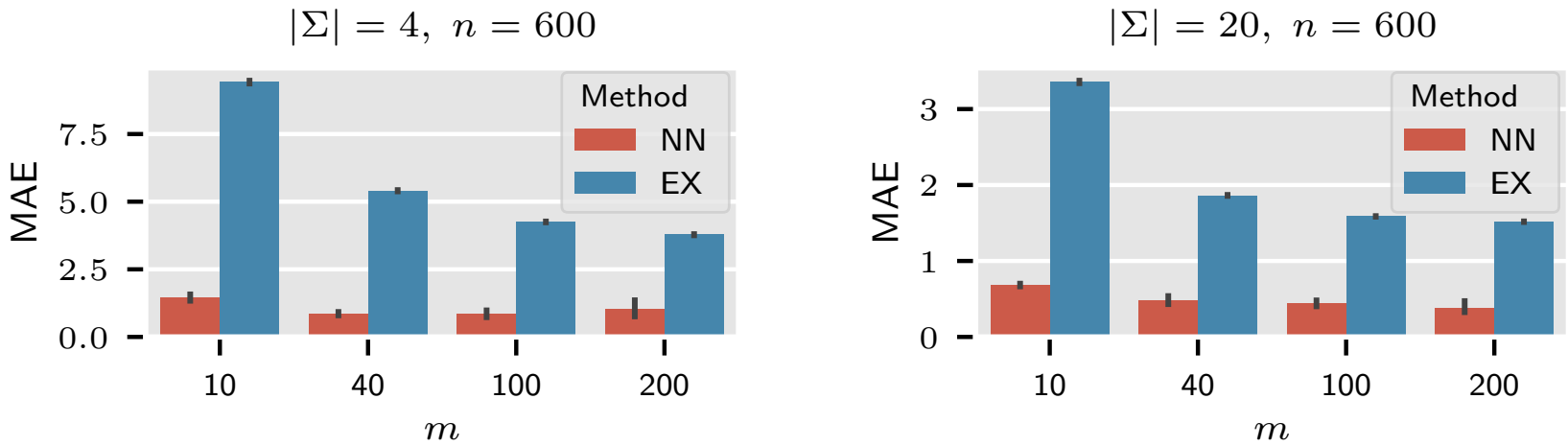


Figure: Mean absolute error of the trained NNs and EX on 10000 test samples, created by a BS with EX guidance function.

NNs approximate the LCS lengths much better than EX.

Nice!. . . But There are Still Weaknesses

- One has to train one model for each combination of m and Σ
- Long training times (hours per model) for large string lengths
- Weaker results for large string lengths n
  - Expected lengths-to-go differ only slightly within one beam

  $\rightarrow$ approximation must be very precise!

Observation:

- We do not need to approximate the length-to-go.
- Instead, we just need a scoring function to rank the nodes in the beam.

Relative Value Based Approach (RV-LBS)

Huber and Raidl (2022)

New Input Features:

Let $V_{\text{ext}} \subseteq V$ be the expanded set of nodes in the BS.

Cut-off value for remaining strings q_i^v, for $i = 1, \ldots, m$:

$$b^{\text{sl}} = \max(0, \overline{q_{\text{ext}}} - \lambda|\Sigma|), \tag{1}$$

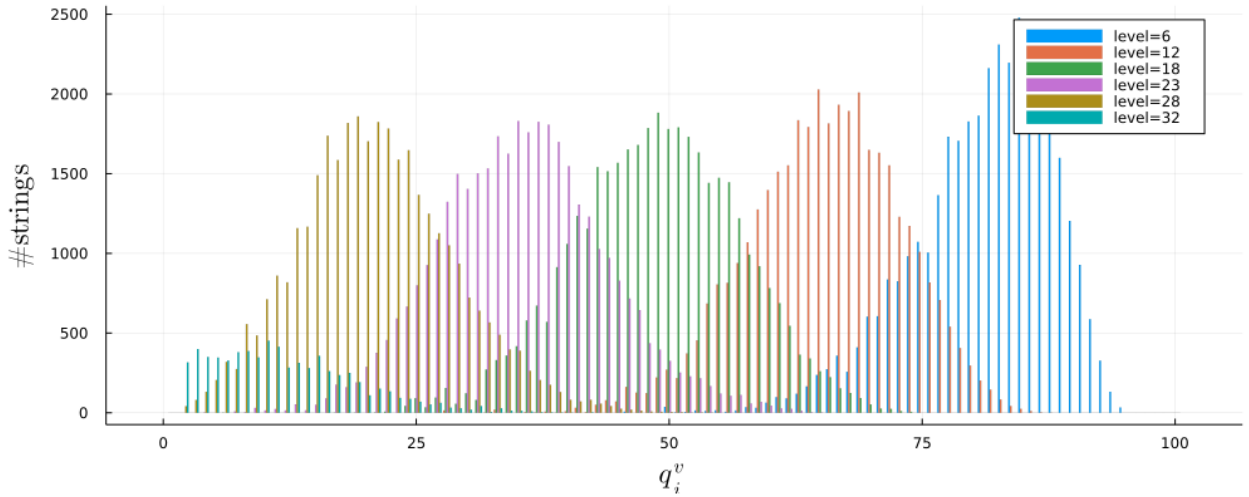
where

$$\overline{q_{\text{ext}}} = \frac{1}{m\,|V_{\text{ext}}|} \sum_{i=1}^{m} \sum_{v \in V_{\text{ext}}} q_i^v, \tag{2}$$

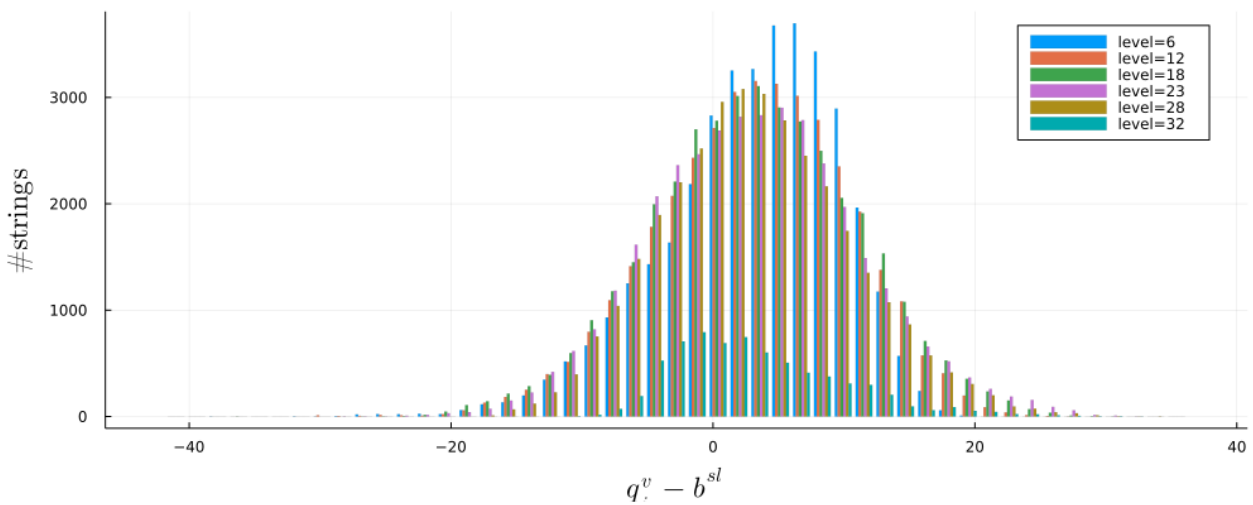
denotes the average length of all remaining input strings for all nodes in V_{ext}, and λ is a control parameter.

Relative Value Function Based Approach (RV-LBS)

Original remaining string lengths: $m = 10$, $n = 100$, $|\Sigma| = 4$



Cut remaining string lengths: $m = 10$, $n = 100$, $|\Sigma| = 4$, $\lambda = 1$



Relative Value Function Based Approach (RV-LBS)

Target Values:

Evaluate solutions in relation to other states at a current BS level.

- Approximation Goal:

$$h(v) \approx \mathrm{LCS}_{\mathrm{exp}}(v) - \frac{1}{|V_{\mathrm{ext}}|} \sum_{v' \in V_{\mathrm{ext}}} \mathrm{LCS}_{\mathrm{exp}}(v'), \tag{3}$$

$\mathrm{LCS}_{\mathrm{exp}}(v)$: expected solution length from node v.

Relative Value Function Based Approach (RV-LBS)

Obtaining Training Samples:

- Select whole levels of each BS run to create training data
- Utilize NBS to get a approximation of the expected solution length from a node v onward.
- Target value:

$$y^v = \mathrm{NBS}(v) - \frac{1}{|V_{\mathrm{ext}}|} \sum_{v' \in V_{\mathrm{ext}}} \mathrm{NBS}(v'), \tag{4}$$

Relative Value Function Based Approach (RV-LBS)

To get rid of number of the specific number of input strings: m

- Compress information of a remaining string lengths vector $q = (q_1, \ldots, q_m)$ into smaller vector of constant size $m' < m$ by downsampling through binning

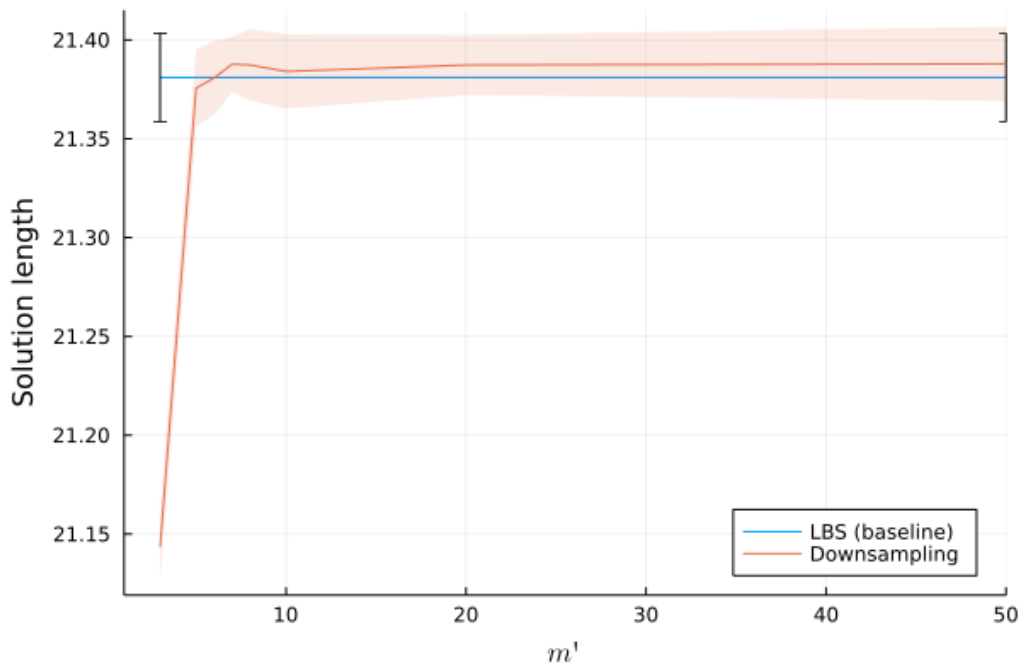


Figure: Previous LBS vs. downsampling approach $m = 100$, $n = 100$, $|\Sigma| = 4$.

Results

Training: $\lambda = 1$, $m' = 7$

m	n	Σ	s_RelVal	s_std
10	100	4	33.8	0.1054

Generalization: $\lambda = 1$, $m' = 7$

set	m	n	Σ	s_RelVal	s_std	s_LBS	s_Lit
rat	10	600	4	197.6	2.118	199	201
rat	100	600	4	132.7	2.869	135	133
BL	10	500	4	180.35	1.490	-	182.0
BL	10	1000	4	365.830	1.389	-	368.5

$\rightarrow$ RL-LBS trained on a small instance generalizes reasonably well to larger instances.

rat inst. set Shyu and Tsai (2009), BL inst. set Blum and Festa (2016)

Policy-Based Learning Beam Search

- Follows point-of-view from Negrinho et al. (2018):
  - ML model gets whole beam V_{ext} as input
  - Learn a scoring function for ranking the nodes, select β best nodes
- Special NN architecture
- Different loss functions considered and compared
- We again utilize Nested BS for obtaining training data
- Results comparable to LBS

Comparison to AlphaZero-like MCTS-based Approach

- Similar to Silver et al. (2018),
  but adapted to single-player scenario, normalization of values
- We did not obtain competitive results
- BUT: Learning works, because plugging trained NN into a BS yields almost competitive results!

Conclusions

- Built on BS because **simple is beautiful!**
- Learning with Nested BS works surprisingly well
- Only exploited a very limited number of features here
- Demonstrated on LCS/CLCS problem, but also nice results on
  - no-wait flowshop problem (Mayerhofer, 2022)
  - shortest common supersequence problem (ongoing work)
  - a quantum circuit design problem (ongoing work)
- Bootstrapping as in temporal difference learning also works here
- Evaluation on a wider-range of benchmark problems remains
- For some problems special NN structures (e.g., GNNs) required
- Well suited for parallelization

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.

Blum, C. and Festa, P. (2016). *Longest Common Subsequence Problems*, pages 55–85.

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

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

Huber, M. and Raidl, G. R. (2022). A relative value function based learning beam search for the longest common subsequence problem. In Moreno-Díaz et al., editors, *Proceedings of EUROCAST 2022 – 18th International Conference on Computer Aided Systems Theory*, LNCS. Springer.

References II

Mayerhofer, J. (2022). Minimizing makespan in flow shops with a reinforcement learning like approach. Master's thesis, TU Wien, Vienna, Austria.

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.

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.