Learning Beam Search

Training ML-Models for Guiding Beam Search to Solve Longest Common Subsequence Problems

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

Institute of Logic and Computation
TU Wien

December 7, 2020

Idea of Learning Beam Search (LBS) and Related Work

Basic Idea:

- Meta-Algorithm that automatically learns a guidance-heuristic for Beam Search (BS) based on randomly generated instances

Related Work:

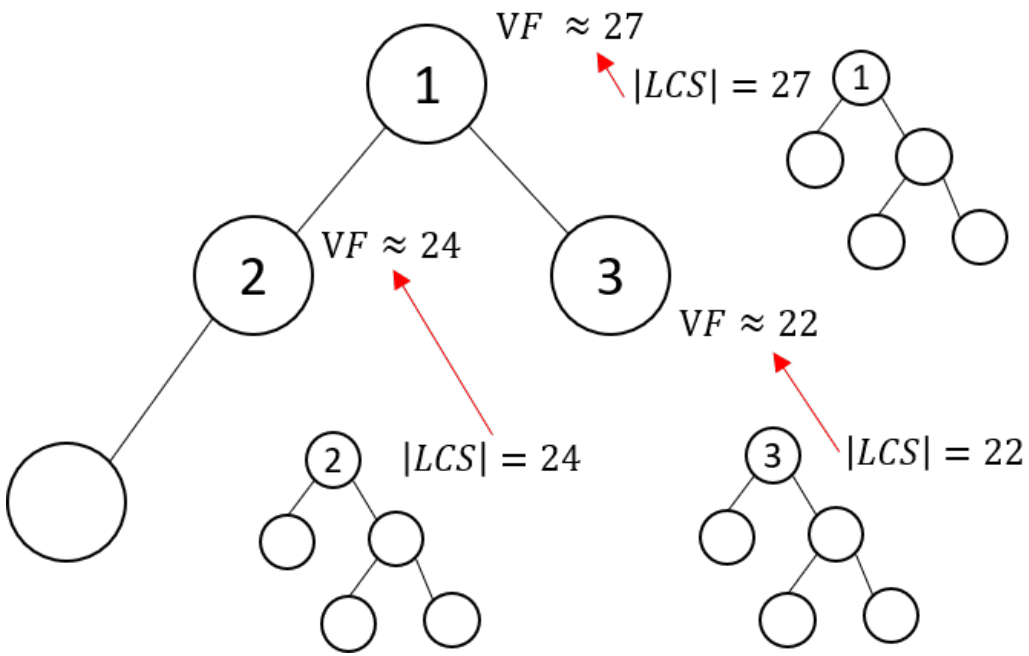
- Learning Beam Search Policies via Imitation Learning (2019)

  by R. Negrinho, M. R. Gormley and G. J. Gordon

  - Unifying Meta-Algorithm for Learning Beam Search policies using imitation learning
  - Meta-Algorithm captures existing learning algorithms and suggests new ones

Basic Idea of Learning Beam Search (LBS)

- Perform a BS, guided by a learnable Value Function (VF)
- Obtain training data by performing independent so-called Embedded Beam Search (EBS)



Procedure

LBS:

1. Initialize VF randomly
2. Create a random instance
3. Perform a BS, guided by the Value Function
4. Store observations and EBS values for each node of a LBS Tree with probability ϕ in a limited size FIFO
5. Train VF on FIFO Data
6. Repeat steps 2-5 until a stopping criterion is fulfilled

Probability ϕ

- Applying an EBS on every node would create too many training samples for a single instance
- Perform for each node an EBS and add samples to the FIFO only with a certain probability ϕ

How to determine a reasonable probability ϕ?

Probability ϕ

Aim: obtaining a constant number of training data per BS run

Consider how many nodes the main BS creates:

- In the expected case $\Theta(\beta \cdot |\Sigma| \cdot \tilde{l}_{max})$ nodes, where
  - β: beam width
  - Σ: set of possible actions
  - $\tilde{l}_{max}$: expected levels calculated by a heuristic

Therefore, choose the probability by which EBS is applied as

$$\phi = \frac{\alpha}{\beta \cdot |\Sigma| \cdot \tilde{l}_{\text{max}}},$$

where $\alpha > 0$ is a strategy parameter

Runtime

Runtime of one LBS iteration:

- One EBS call on average creates $\Theta(\beta' \cdot |\Sigma| \cdot \tilde{l}_{\max})$ nodes
  - β': EBS beam width
- Assume that the heuristic function and the VF of constant size can be calculated in $\Theta(m + |\Sigma|)$ time,
- Then one LBS iteration is performed in time

$$\Theta(\beta \cdot |\Sigma| \cdot \tilde{l}_{\max} \cdot \phi \cdot \beta' \cdot |\Sigma| \cdot \tilde{l}_{\max} \cdot (m + |\Sigma|))$$
$$= \Theta((\beta + \beta') \cdot |\Sigma| \cdot \tilde{l}_{\max} \cdot (m + |\Sigma|))$$

Hyperparameters

Parameter	Description
iter	number of LBS iterations
min_obs_for_learn	minimum number of observations to start with learning
β	beam width of the main BS
β'	beam width of the EBS
α	factor for controlling number of generated samples per LBS iteration
replay_capacity	maximum size of the FIFO
batch_size	minimum batch size for learning
ML model	ML model used as VF

The Longest Common Subsequence (LCS) Problem

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

Task: find a longest string that appears as subsequence in any string of S

Example: $\Sigma = \{\text{A,C,G,T}\}$, $s_1 = \text{AGCATGA}$, $s_2 = \text{ACGTGGA}$

Common subsequence (CS):

s_1: AGCATGA
s_2: ACGTGGA
$\Rightarrow$ AGTGA

The Longest Common Subsequence (LCS) Problem

- 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
- Exact methods only suitable for small m or n
- State-of-the-art heuristic approaches for large m and n are based on Beam Search

Beam Search for the LCS Problem (Djukanovic et al., 2019)

- Leading on many public benchmark instances – which are mostly uniformly random distributed
- Ranking of nodes based on a theoretically derived heuristic
  - which approximates the expected LCS length
  - for uniformly distributed random strings
- Dominance checking and filtering
- Not so good on instances with similar strings for example

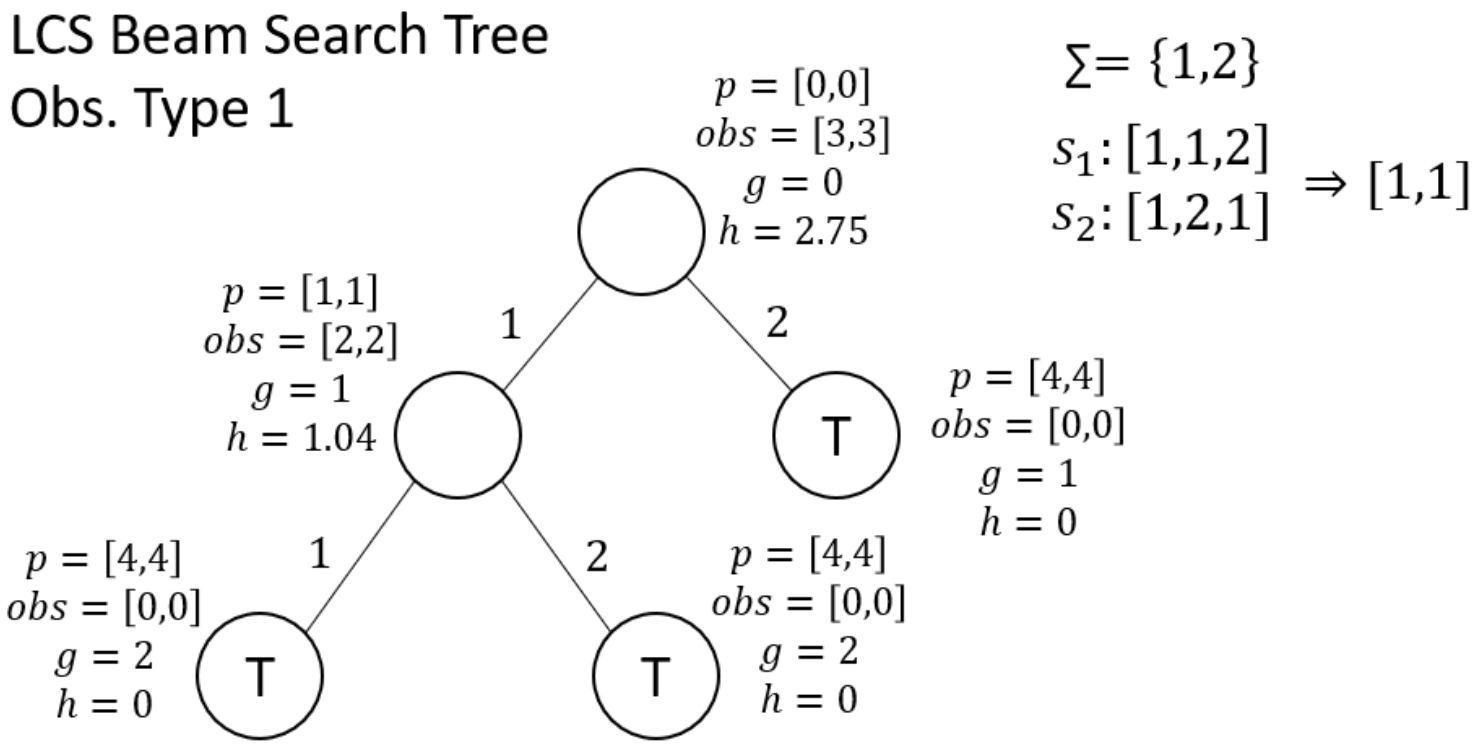
LCS State Space

For LCS Problem, define:

- State space: directed acyclic graph $G = (V, A)$, in which
  - states (nodes) $v \in V$ are represented by position vectors $p^v = (p_i^v)_{i=1,\ldots,m}$ with $p_i^v \in 1, \ldots, |s_i| + 1$
  - the root node $r \in V$ has position vector $p^r = (0, \ldots, 0)$
  - Arcs in A represent actions (a valid letter to a partial solution)
  - Terminal states are represented by the state $t \in V$ with $p^t = (|s_i| + 1)_{i=1,\ldots,m}$
- Input for VF (=Observations):
  - Type 1: Remaining string lengths $|s_i| - p_i^v + 1$, $i = 1, \ldots, m$, sorted according to non-decreasing values
  - Type 2: Type 1 + minimum letter appearances $\min_{i=1,\ldots,m} |s_i[p_i^v, |s_i|]|_c$, $c \in \Sigma$

LCS State Space and Environment

- Example:



Early Results

Parameter	Value
iter	1500
min_obs_for_learn	3000
β	10
β'	50
α	40
replay_capacity	5000
batch_size	32
ML model	NN with $30 + 30$ hidden nodes, ReLU activation or Boosted Trees (BT), default settings

Early Results

Instance: $n = 50$, $m = 3$, $|\Sigma| = 4$

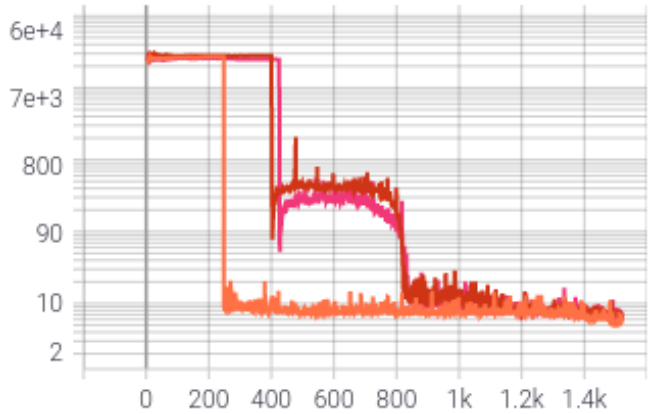
Testing over 50.000 random instances, beam width 10

	Avg. length	time
NN	24.35576	201s
BT	24.34664	10776s
Marko	24.34618	167s

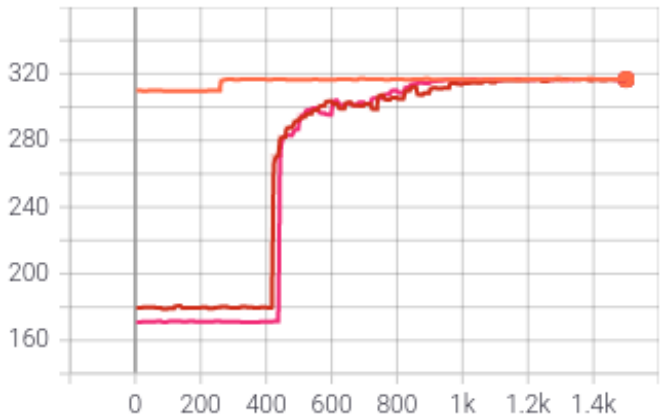
Early Results (NN)

Instance: $n = 600$, $m = 3$, $|\Sigma| = 4$

Loss of VF on FIFO (Training Data) over LBS itertions



Independent interleaved test: average solution lenght of BS with VF over LBS iterations



Early Results (NN)

Avg. Training time: 2.4h

Avg. Metrics of VF on last FIFO (Test Data)

mse	rmse	mae	r2	std
7.1556	2.6745	2.0547	0.9991	2.6226

Avg. Metrics of Markos approximation on last FIFO (Test Data)

mse	rmse	mae	r2	std
504.6056	22.4633	19.8667	0.9386	10.4858

Early Results (NN)

Testing over 50.000 random instances, beam width 10

	Avg. length	Avg. time
VF	316.6916	2984s
Marko	316.9269	2925s

Conclusions

- LBS predicts the length on average well, and more precisely than Marko's heuristic, but his heuristic guides the BS more successfully
- Loss of VF and average lengths converge well, self-learning works

Future Work

- The actual values are not important, but just the resulting ranking of the nodes. Can we exploit this freedom by some better suited target values for the learning?
- Learn a function that receives as input two observations from two states and outputs a probability value indicating which of the two states is more promising to lead to an optimal solution
- Interesting comparison: apply the MCTS agent and use the trained value subnet within a BS

[1] Renato Negrinho and Matthew R. Gormley and Geoffrey J. Gordon, "Learning Beam Search Policies via Imitation Learning", 2019

[2] L. Bergroth, H. Hakonen and T. Raita, "A survey of longest common subsequence algorithms," Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000, A Curuna, Spain, 2000, pp. 39-48, doi: 10.1109/SPIRE.2000.878178.

[3] M. Djukanovic, G. R. Raidl, and C. Blum. A beam search for the longest common subsequence problem guided by a novel approximate expected length calculation. In G. Nicosia, P. Pardalos, G. Giuffrida, R. Umeton, and V. Sciacca, editors, Proceedings of LOD 2019 – The 5th International Conference on Machine Learning, Optimization and Data Science, volume 11943 of LNCS, pages 154–167. Springer, 2020.