Learning a Data-Driven Beam Search Heuristic for the Variable Gapped Longest Common Subsequence Problem

1 Overview

This work proposes a learning framework for constructing a data-driven heuristic to guide Beam Search in combinatorial optimization problems, and specifically one sequence problem that arises in bioinformatics, the *Variable Gapped Longest Common Subsequence Problem* (VGLCSP). A feed-forward neural network (FFNN) is used to parameterize the heuristic function. Instead of relying on gradient-based training, the network weights are optimized using a Genetic Algorithm (GA), precisely its popular version called the *Random key generic algorithm* (RKGA).

Each individual in the GA population represents a complete set of neural network weights. The quality of an individual is evaluated by embedding the neural network inside the *Iterative Multi-source Beam Search* (IMSBS) framework (proposed in the EUROCAST paper) to guide the search in the process of forward passes of the Beam search, and measuring the average solution quality obtained on a set of training instances. Validation instances are used exclusively to monitor generalization.

Formally, the learning problem is defined as:

$$\max_{\theta} \frac{1}{|T|} \sum_{I \in T} f(\text{IMS\_BeamSearch}(I, h_\theta)),$$

where T denotes the training set, h_θ is the neural heuristic parameterized by weights θ, and $f(\cdot)$ measures solution quality.

2 Neural Network Model

The heuristic is represented by a *multilayer perceptron* with an arbitrary number of hidden layers and user-defined units per layer. Supported activation functions include `hyperbolic tangent`, `ReLU`, and `sigmoid`.

For each layer l, forward propagation is defined as:

$$\mathbf{z}^{(l+1)} = W^{(l)}\mathbf{a}^{(l)} + b^{(l)}, \quad \mathbf{a}^{(l+1)} = \phi(\mathbf{z}^{(l+1)}),$$

where $W^{(l)}$ and $b^{(l)}$ are the weight matrix and bias vector, respectively, and ϕ denotes the activation function.

All weights and biases are flattened into a single vector, which forms the chromosome of a GA individual.

2.1 Fitness Evaluation

Given a candidate weight vector:

1. The weights are loaded into the FFNN.

2. IMSBS is executed on each training instance using the neural heuristic (guiding the search in the forward iterations as more sensible than the backward iterations).

3. The size of the best sequence obtained for each instance is recorded.

The fitness value is computed as the average solution quality over all training instances. Validation performance is computed analogously on a separate dataset and is used only for reporting.

2.2 Feature Extraction for the Variable-Gap LCS

To guide the search process, each partial solution (node) is represented by a fixed-length numerical feature vector capturing both the current state of the construction and the structure of the remaining search space. These features are designed to be instance-independent and comparable across problem sizes.

Let $S = \{s_1, \ldots, s_k\}$ denote the set of input sequences, and let p_i be the current matched position in sequence s_i at a given node. The vector $\mathbf{p}^L = (p_1, \ldots, p_k)$ is first normalized by the corresponding sequence lengths,

$$p_i \leftarrow \frac{p_i}{|s_i|},$$

so that progress in each sequence lies in $[0, 1]$ and becomes independent of absolute input sizes.

From this normalized position vector, four statistical descriptors are extracted: maximum, minimum, mean, and standard deviation. These quantities characterize, respectively, the most advanced and least advanced sequences, the average depth of the search, and the imbalance of progress among sequences. In addition, *the current length of the partially constructed subsequence* is included as a feature. Together, these values provide a compact description of the current search state.

To capture the feasibility of future extensions, gap-related features are computed from the remaining suffixes of all sequences. For each sequence s_i, all gap bounds associated with positions strictly after the current node position are collected. From this aggregated set of gap values: the maximum, minimum, mean, and standard deviation are calculated. These statistics summarize the remaining flexibility of the instance: large average or maximum gaps indicate greater freedom for future matches, whereas small minimum gaps and large variability typically signal tighter constraints and increased difficulty.

Depending on the selected configuration, additional instance-level characteristics are appended to the feature vector. These may include the alphabet size $|\Sigma|$, the number of sequences (m). Such attributes provide global structural information about the instance and help the learning model adapt to varying problem scales.

Finally, all features are standardized to zero mean and unit variance prior to being passed to the learning component. Overall, the resulting representation combines information about (i) the current construction progress, (ii) the remaining gap-constraint landscape, and (iii) global instance properties, yielding between 9 and 11 features depending on the configuration. This compact yet expressive encoding enables effective learning-based guidance of the search for high-quality variable-gap longest common subsequences.

3 Genetic Algorithm

The optimization process follows a population-based evolutionary scheme.

3.1 Initialization

Each individual is initialized by sampling weights uniformly from a predefined interval $[-w, w]$ (the value w parametrized, but fixed to -1).

3.2 Elitism

A fixed number of top-performing individuals are copied unchanged into the next generation.

3.3 Mutation

Several individuals are generated completely at random (random immigrants). This mechanism promotes exploration and maintains population diversity.

3.4 Offspring Generation

Three alternative parent-selection strategies are supported:

- **RKGA**: Two parents are selected uniformly at random, and each gene is inherited from either parent with probability 0.5.
- **BRKGA**: One parent is selected from the elite set and one from the non-elite set. Each gene is inherited from the elite parent with a predefined probability.
- **Lexicase Selection**: Training instances are randomly shuffled. Individuals are filtered iteratively by retaining those achieving the best performance on each instance. Parents are selected randomly from the remaining candidates.

We opt to use the *RKGA* as the first option to work well, but other options may also be tried.

3.5 Replacement and Termination

The new population consists of elites, mutants, and offspring. The algorithm terminates once a predefined time limit is reached (in this case we used 2 hours). The best individual encountered during training is continuously tracked and stored.

4 Overall Learning Procedure

The learning process alternates between evolutionary optimization of network weights and evaluation through (iterative multi-source) Beam Search. Importantly, the neural network does not directly predict solutions; instead, it learns how to *guide* the search procedure. This places the method within the class of neuro-evolutionary hyper-heuristics.

5 Algorithm

Algorithm 1 GA-Based Learning of a Beam Search Heuristic

```
1: Initialize population $P$ with random neural network weights
2: Evaluate all individuals on training set $T$
3: Store best solution $\theta^*$
4: $start \leftarrow$ current time
5: while time limit $T_{\max}$ has not exceeded do
6:     Sort $P$ by fitness (descending) // Section 2.1
7:     Initialize empty population $P'$
8:     Copy top $n_{elites}$ individuals from $P$ to $P'$
9:     for $i = 1$ to $n_{mutants}$ do
10:        Create random individual $x$
11:        Evaluate $x$ on $T$
12:        Add $x$ to $P'$
13:        Update $\theta^*$ if improved
14:    end for
15:    for $i = 1$ to $n_{offspring}$ do
16:        Select parents using RKGA, BRKGA, or Lexicase
17:        Generate offspring by crossover
18:        Evaluate offspring on $T$
19:        Add offspring to $P'$
20:        Update $\theta^*$ if improved
21:    end for
22:    $P \leftarrow P'$
23: end while
24: return best weights $\theta^*$
```

6 Discussion

The proposed framework learns heuristics rather than explicit solutions. The neural network is optimized exclusively through downstream search performance, enabling learning in non-differentiable settings. This tightly coupled integration of evolutionary learning and the classical optimization algorithm Beam Search allows the system to adapt its guidance strategy to the structure of the problem instances.

7 Preliminary results

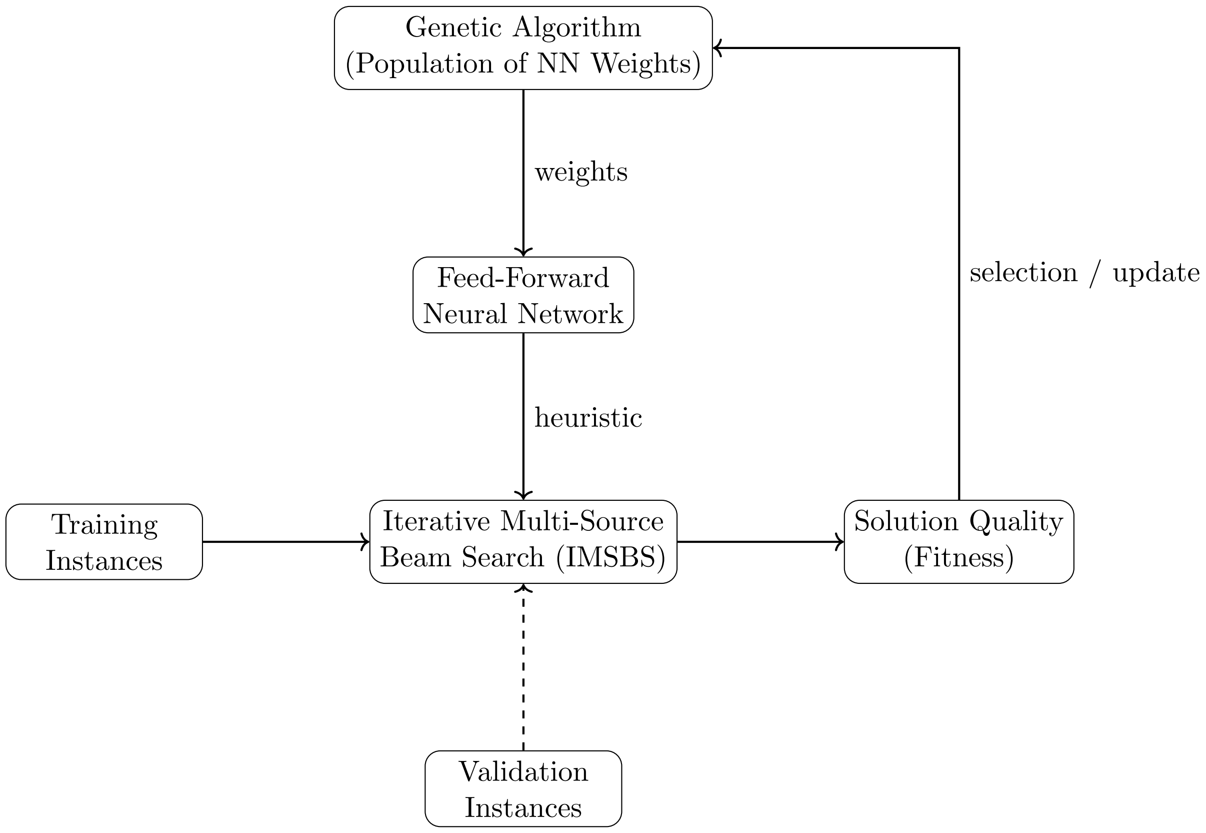

Figure 1: Learning framework integrating a Genetic Algorithm with Iterative Multi-Source Beam Search (IMSBS). The GA evolves neural network weights. For each individual, the FFNN provides a heuristic used by IMSBS on training instances. The resulting solution quality defines fitness, which drives evolutionary selection. Validation instances are used only for monitoring generalization.

Table 1: Comparison of objective values between Learning-based Beam Search and IMSBS. Best results per instance category are shown in bold (lower is better).

Instance group	Learning IMSBS (U5_5_5_A1_F3)	Basic IMSBS-500 (h5, iters=100, roots=10)
10_100_2	**9.1**	8.6
10_100_4	**7.2**	6.2
10_200_2	**10.5**	10.3
10_200_4	**6.3**	6.2
10_500_2	**10.2**	9.5
10_500_4	**6.6**	6.1
10_50_2	**9.5**	9.1
10_50_4	5.9	**6.0**
2_100_2	72.6	**72.8**
2_100_4	**61.7**	**61.7**
2_200_2	**137.8**	136.9
2_200_4	120.7	**125.1**
2_500_2	**276.1**	272.1
2_500_4	295.5	**299.3**
2_50_2	**37.7**	**37.7**
2_50_4	30.0	**30.1**
3_100_2	58.2	**58.5**
3_100_4	48.3	**48.5**
3_200_2	**98.6**	89.7
3_200_4	**97.6**	**97.6**
3_500_2	**113.5**	105.4
3_500_4	**215.3**	195.5
3_50_2	**31.2**	**31.2**
3_50_4	**22.9**	**22.9**
5_100_2	**28.7**	22.3
5_100_4	**22.8**	22.1
5_200_2	**32.3**	26.4
5_200_4	**31.1**	25.7
5_500_2	**29.9**	27.9
5_500_4	**35.0**	27.1
5_50_2	**20.5**	20.0
5_50_4	**15.3**	**15.3**
Avg.	**62.45**	60.43