On Solving the Multiple Variable Gapped Longest Common Subsequence Problem

Marko Djukanović[1,2,6], Nikola Balaban[2], Christian Blum[3], Aleksandar Kartelj[4], Sašo Džeroski[5], and Žiga Zebec[6]

[1] University of Nova Gorica, Nova Gorica, Slovenia
`marko.dukanovic@ung.si`

[2] Faculty of Natural Sciences and Mathematics, University of Banja Luka, Banja Luka, Bosnia and Herzegovina
`nikola.balaban@student.pmf.unibl.org`, `marko.djukanovic@pmf.unibl.org`

[3] Artificial Intelligence Research Institute (IIIA-CSIC), Barcelona, Spain
`christian.blum@iia.csic.es`

[4] Faculty of Mathematics, University of Belgrade, Belgrade, Serbia
`kartelj@matf.rs`

[5] Jožef Stefan Institute, Ljubljana, Slovenia
`saso.dzeroski@ijs.si`

[6] Institute of Information Sciences (IZUM), Maribor, Slovenia
`ziga.zebec@izum.si`

Abstract. This paper addresses the Variable Gapped Longest Common Subsequence (VGLCS) problem, a generalization of the classical longest common subsequence (LCS) problem to an arbitrarily large set of input strings under flexible gap constraints. The problem arises in applications such as molecular sequence comparison, where structural distance constraints between nucleotides must be respected, and time-series analysis, in which events are required to occur within specified temporal delays. We propose a novel solution framework based on a root-based state graph representation, in which the overall state space is defined as the union of a potentially large number of root-based state graphs. To cope with the resulting combinatorial explosion, we introduce an iterative beam search strategy that dynamically maintains a global pool of promising candidates for root nodes, enabling effective control of diversification across iterations. To further guide the search, several heuristic evaluation functions inspired by the LCS literature are incorporated into the standalone beam search procedure. To the best of our knowledge, this work presents the first comprehensive computational study of the VGLCS problem, using a benchmark of 320 synthetic instances. Experimental results show that the proposed iterative beam search consistently outperforms both a baseline beam search and an iterative greedy approach within comparable computational times. Furthermore, in the special case of two input strings, the proposed method achieves performance comparable to three specialized dynamic programming algorithms on small-to-medium-sized instances, while significantly outperforming two alternative heuristic variants.

Keywords: Longest common subsequences · Beam Search · Bioinformatics · Gap constraints

1 Introduction

The *Longest Common Subsequence Problem* (LCSP) [1, 4] is a well-known combinatorial optimization problem with numerous applications in bioinformatics and computational biology, where it plays a fundamental role in the analysis and comparison of molecular sequences. Given an arbitrarily large set of input sequences

$$S = \{s_1, \ldots, s_m\}, \quad m \in \mathbb{N},$$

defined over an alphabet Σ, the objective is to identify a longest possible subsequence that is common to all sequences $s_i \in S$.

Over the past decades, a variety of practically motivated extensions of the LCSP have been proposed to better capture structural, biological, or application-specific requirements. Notable variants include constrained, arc-preserving, and repetition-free longest common subsequence problems [7, 6, 3], among others.

In this work, we focus on the *Variable Gapped Longest Common Subsequence Problem* (VGLCSP), which was originally introduced for the case of two input strings in [10] and further investigated from a theoretical perspective in [2]. The VGLCSP extends the classical LCSP by incorporating flexible distance constraints (referred to as *gaps*) between consecutive symbols of the resulting subsequence. Unlike fixed-gap models, these gap limits are allowed to vary along the input sequences, thereby offering increased modeling flexibility.

Formally, for each input sequence $s_i \in S$, a gap constraint is defined by a function

$$G_i \colon \{1, \ldots, |s_i|\} \mapsto \mathbb{N}.$$

If the characters of a common subsequence s occur at positions $i_i^1 < \cdots < i_i^{|s|}$ in sequence s_i, then the gap constraint is satisfied if

$$i_i^x - i_i^{x-1} \leq G_i[i_i^x] + 1, \quad x = 2, \ldots, |s|.$$

A common subsequence is considered *feasible* if the corresponding gap constraints are satisfied simultaneously for all sequences in S.

The VGLCSP provides a flexible and realistic model for sequence comparison, particularly suited to applications in DNA and protein analysis where variable structural distances between residues must be respected. Beyond bioinformatics, the problem also arises in time-series analysis, especially in settings where events are required to occur within specified temporal delays [8].

FiXme Note: Give an example of an VGLCS problem instance with solution …

Although several exact dynamic programming approaches exist for the two-sequence case ($m = 2$) [10], their computational complexity becomes prohibitive for larger m. To the best of our knowledge, no effective exact or approximate approaches are currently existing for the generalized VGLCSP with arbitrary large set S, which is clearly an NP-hard problem as a general version of the LCS problem with arbitrary large set of input strings.

1.1 Preliminaries and Notation

Before introducing the algorithmic framework for the problem under consideration, we first define the notation used throughout this work.

Let $|s|$ denote the length of a sequence s. The symbol $s[i]$ refers to the character at position i of sequence s, where indexing starts at $i = 1$. The substring of s that begins at position i and ends at position j is denoted by $s[i, j]$. If $i = j$, this substring corresponds to the single character $s[i]$; if $j < i$, then $s[i, j] = \varepsilon$, where ε denotes the empty string.

The number of occurrences of a character $a \in \Sigma$ in a sequence s is denoted by $|s|_a$. For a subset of characters $\Sigma' \subseteq \Sigma$, the number of occurrences of characters from Σ' in s is defined as

$$|s|_{\Sigma'} = \sum_{\sigma \in \Sigma'} |s|_\sigma.$$

We denote by $S = \{s_1, \dots, s_m\}$ the set of input sequences. Unless stated otherwise, $m \in \mathbb{N}$ represents the number of input sequences (and, correspondingly, the number of gap constraints), and n denotes the length of the longest sequence in S.

The remainder of the paper is organized as follows. Section 2 introduces the state graph representation of the problem. In Section 3, we present the design of our main methodological contribution, namely the iterative multi-source beam search approach. Section 4 reports the computational results, comparing the proposed method with a baseline standalone beam search and a greedy heuristic, as well as with specialized approaches for the case of two input sequences. Finally, Section 5 concludes the paper and outlines directions for future research.

2 Root-based State Space Formulation

Building on the state-space representation for the LCSP introduced in [1], we develop a *root-based state graph* model for the Variable Gapped Longest Common Subsequence Problem (VGLCSP). Each state represents one or more feasible partial solutions characterized by a vector of positions induced by the input sequences and by the current subsequence length.

Formally, a partial subsequence s^v induces a state node

$$v = (\mathbf{p}^{L,v},\, l^v),$$

where $\mathbf{p}^{L,v} = (p_1^{L,v}, \dots, p_m^{L,v})$ and the following conditions hold: (i) for each sequence $s_i \in S$, the index $p_i^{L,v} - 1$ is the smallest position such that s^v is a subsequence of $s_i[1, p_i^{L,v} - 1]$; (ii) $l^v = |s^v|$ denotes the length of the partial subsequence; and (iii) all gap constraints G_i are satisfied by s^v with respect to every input sequence s_i.

A directed arc $\alpha = (v_1, v_2)$, labeled with $lett(\alpha) = a \in \Sigma$, exists if

$$l^{v_2} = l^{v_1} + 1 \quad \text{and} \quad s^{v_2} = s^{v_1} \cdot a,$$

that is, state v_2 corresponds to extending a partial solution of v_1 by appending the character a.

To expand a state v, only those characters $a \in \Sigma$ that occur in the suffix of each sequence s_i starting at position $p_i^{L,v}$, for $i = 1, \dots, m$, are considered. For each such character a, we identify the smallest feasible positions $p_i^{L,a} \geq p_i^{L,v}$ such that

$$s_i[p_i^{L,a}] = a \quad \text{and} \quad p_i^{L,a} - p_i^{L,v} \leq G_i(p_i^{L,a}), \qquad i = 1, \dots, m.$$

If such positions exist for all sequences, a child state

$$w = (\mathbf{p}^{L,a} + \mathbf{1},\, l^v + 1)$$

is generated, unless it is dominated by another child state according to the dominance criteria defined later.

To efficiently compute child states and their associated position vectors, we employ a preprocessing data structure denoted by `Succ`. This structure is a three-dimensional integer array indexed by (i, j, a), where i refers to the input sequence s_i, $1 \leq j \leq |s_i|$ denotes a position within that sequence, and $a \in \Sigma$ is a character. The value

$$q = \texttt{Succ}[i, j, a]$$

stores the smallest index $q \geq j$ such that $s_i[q] = a$ and the gap constraint is satisfied at position q, that is,

$$q - j \leq G_i(q).$$

If no such position exists, the value -1 is assigned.

The root state is defined as

$$r = ((1, \dots, 1),\, 0),$$

corresponding to the empty subsequence. Goal states are feasible states that cannot be further expanded. The search space rooted at r is denoted by Space(r). As discussed in the following section, the structure of this space depends on the choice of the root state from which the search is initiated.

Motivation for Multi-Source Beam Search. Unlike the classical LCSP, the VGLCSP may exhibit multiple, potentially exponentially many, *disconnected root components* in its state space. Consequently, a search strategy that expands states from a single initial root may fail to reach feasible and even optimal solutions.

As an illustrative example, consider the sequence set

$$S = \{s_1 = \texttt{ATGGAAAA},\ s_2 = \texttt{ATCCAAAA}\},$$

with gap constraints $G_{s_1} = G_{s_2} = 1$, depicted in Fig. 1. In this instance, the initial state $((1,1), 0)$ cannot reach any state with position vector $\mathbf{p}^L = (5, 5)$ via standard directed transitions. As a result, the optimal common subsequence

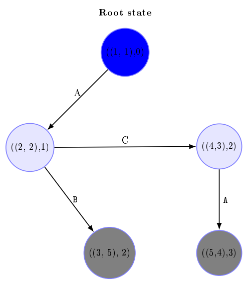


Fig. 1: State graph $Space(((1,1),0))$ for MVGLCSP between the sequences `ABCA` i `ACAB`. Two final/terminal states are displayed in gray.

`AAA` is unreachable when the search is initiated exclusively from the initial root state $((1,1),0)$, and is therefore missed by traditional search strategies.

To address this issue, the full state space of a VGLCSP instance is naturally defined as

$$\bigcup_{q \in \mathcal{R}} \text{Space}(q),$$

where $\mathcal{R}$ denotes the set of all root states, i.e., states with position vectors $\mathbf{p}^{L,q}$ that cannot be reached from any other state through regular directed transitions. However, explicitly enumerating all such root states is computationally prohibitive in general, requiring $O(n^m)$ time in the worst case, and is thus infeasible for instances with large m or n.

This structural characteristic of the VGLCSP motivates the proposed *Iterative Multi-Source Beam Search* (IMSBS) approach, which is specifically designed to dynamically explore multiple promising regions of the state space. By iteratively identifying and expanding a set of candidate root states, IMSBS effectively mitigates the disconnection effects induced by gap constraints and enables the discovery of high-quality solutions that would otherwise remain inaccessible to single-source search methods.

3 Iterative Multi-source Beam Search Framework

Beam search (BS) is a well-established tree-search metaheuristic that operates in a breadth-first-search manner. The search is controlled by a beam width parameter β, which specifies the maximum number of nodes retained for expansion at each level, and by a heuristic evaluation function h, which guides the selection of the most promising nodes for further expansion.

Within the *Iterative Multi-Source Beam Search* (IMSBS) framework, BS is executed iteratively, each time initialized with a beam $\mathcal{L}$ consisting of several promising root nodes. These root nodes are dynamically selected from a global pool $\mathcal{R}$, whose role is to promote diversification across different regions of the state space. Each BS execution thus explores the neighborhood of multiple starting points rather than relying on a single root.

The beam search itself employs several heuristic components to intensify the search toward high-quality feasible solutions. These heuristics include estimates based on letter frequencies, minimal residual substring lengths, and probability-weighted bounds, which are described in detail in the subsequent sections.

The overall procedure is summarized in Algorithm 1. Initially, the pool of root nodes $\mathcal{R}$ is initialized with the root state $r = ((1, \ldots, 1), 0)$, corresponding to the empty partial solution, and the current best solution is set to $s_{\text{best}} = \varepsilon$. The algorithm iterates until either the pool $\mathcal{R}$ becomes empty or a predefined maximum number of beam search calls is reached.

At each iteration, a fixed number of the most promising root nodes from $\mathcal{R}$—controlled by the parameter *number_of_sources_from_R* and ranked according to a heuristic evaluation h'—are extracted and stored in a temporary beam $\mathcal{L}$. This beam serves as the initial point for the current beam search, which is executed using beam width β and heuristic guidance function h.

During the beam search, all completed (i.e., non-expandable) nodes are processed as follows. If a completed node corresponds to a solution that improves upon the current best, the incumbent solution s_{best} is updated accordingly. Each completed node is then independently expanded in an LCS-like manner, ignoring gap constraints, to generate a set of successor nodes. For a generated child node w with position vector $\mathbf{p}^w$, if

$$|s_i| - p_i^w + 1 \leq |s_{\text{best}}| \quad \text{holds for at least one } i \in [m],$$

the node is discarded as it cannot lead to an improvement over the current best solution. Otherwise, a corresponding root candidate $r^w = (\mathbf{p}^w, 0)$ is inserted into the pool $\mathcal{R}$, provided that it has not already been part in the previous iterations.

Once the current beam search terminates, a new iteration is initiated with a freshly selected set of root nodes from $\mathcal{R}$, as long as the termination conditions are not met. The pool $\mathcal{R}$ is maintained as a priority queue implemented via a binary heap, ensuring efficient extraction of the most promising root nodes.

The core advantages of the IMSBS approach over the baseline BS are:

- It provides a heuristic balance between beam search (local exploration of promising paths) and the generation of new sources (global coverage of the search space).
- Filtering suboptimal root nodes reduces unnecessary iterations and enables pruning of dominated paths.

3.1 Heuristic guidances

Several heuristics to guide search are utilized within the IMSBS (as the candidate for h and h').

- *"Look-ahead" for the remaining length:*

$$\text{UB}_1(\mathbf{v}) = \min_{i=1,\ldots,m} (|s_i| - p_i^v + 1) \tag{1}$$

  This is an estimate of the potential extension of the sequence up to the end of the shortest remaining sequence.

Algorithm 1 Iterated Multi-source Beam Search (IMSBS) Framework

```
Require: Sequences $s_1, \ldots, s_m$; Gap functions $G_{s_i}(\cdot)$; beam width $\beta > 0$;
    $number\_sources\_from\_R > 0$; $beam\_iters > 0$, heuristics $h, h'$
Ensure: Approximate common subsequence $s_{\text{best}}$
 1: Initialize $\mathcal{R} \leftarrow \{(\mathbf{1}, 0)\}$                      ▷ A set of root nodes
 2: $iter \leftarrow 0$
 3: $s_{\text{best}} \leftarrow \varepsilon$
 4: while $\mathcal{R} \neq \emptyset \wedge iter < beam\_iters$ do
 5:     Select $\mathcal{L} \subseteq \mathcal{R}$ $number\_sources\_from\_R$ nodes,
 6:     Remove the selected nodes from $\mathcal{R}$          ▷ $\mathcal{R}$ can be a pr. queue ordered by
    heuristic $h'$
 7:     Execute beam search $BS(\mathcal{L}, h)$
 8:     for all complete nodes $v$ at $BS(\mathcal{L}, h)$ do
 9:         for all Symbol $a \in \Sigma$ do
10:             $p^w \leftarrow$ minimum positions in the sequences so that $\geq p^v$ and $s_i[p_i^w] = a$
    (ignore all gap constraints)
11:             if $|s_{\text{best}}| < \min_{i=1,\ldots,m}\{|s_i| - p_i^w + 1\}$ then          ▷ cut-off
12:                 Add $r^w = (p^w, 0)$ in the set $\mathcal{R}$ (if previously have not added)
13:             end if
14:         end for
15:     end for
16:     Update $s_{\text{best}}$ if a new longest subsequence in $BS(\mathcal{L}, h)$ has been found
17:     $iter \leftarrow iter + 1$
18: end while
19: return $s_{\text{best}}$
```

– *Character Frequency Alignment:*

$$\mathrm{UB}_2(\mathbf{v}) = \sum_{\sigma \in \Sigma} \min\left(|s_1[p_1^v, |s_1|]|_\sigma, \ldots, |s_m[p_m^v, |s_m|]|_\sigma\right) \tag{2}$$

In essence, this heuristic counts the maximum number of characters that can still match in the (optimal) solution based on the frequency of each character. This score can be computed efficiently in $O(m)$ time using preprocessing (i.e., constructing suitable data structures), see [1].

– *A probability-based heuristic* guidance by the Mousavi and Tabataba's matrices from [9] in the context of the LCS problem. These probabilities are based on the idea of determining a probability for the event realization that a sequence s of length k is a subsequence of a (random) sequence of length n over an alphabet Σ. The probabilities for each $i = 1, \ldots, k$ and $j = 1, \ldots, n$, are pre-processed by a matrix $\mathcal{P}$ of probabilities of dimension $k \times n$ by a dynamic programming recurrence. Assuming the independence betwen input sequences, at each level of beam search the nodes are evaluated by determining the probabilities for each partial solutions across that level to be expanded by $\overline{k}$ letters (this value is heuristically evaluated, as proposed in [9]). Denote that heuristic guidance by h_{prob}.

– TODO:

FiXme Note: TODO: if we come up with some learning heuristic – should left for the extended version – journal-extendable.

4 Experimental Evaluation

In this section, we compare the performance of heuristic approaches designed for the VGLCSP with an arbitrary number of sequences $m \geq 2$. Specifically, we consider the following methods:

- Bs: a baseline beam search approach that starts from the root node r as the initial beam and performs a single iteration of IMSBS.
- Imsbs: a tuned iterative multi-source beam search, described in Algorithm 1, configured to use a runtime comparable to that of Bs.
- Imsbs-greedy: a variant of IMSBS with a fixed parameter $\beta = 1$, as given in Algorithm 1. This configuration highlights the impact of performing a larger number of beam search iterations on the overall performance of Imsbs.

The Imsbs algorithm is implemented in Python 3.11 and evaluated on the VEGA high-performance computing system hosted at IZUM, Maribor. The cluster consists of 960 compute nodes equipped with AMD EPYC 7H12 CPUs operating at 2.35 GHz. All experiments were conducted in a single-threaded setting.

4.1 Instance generation

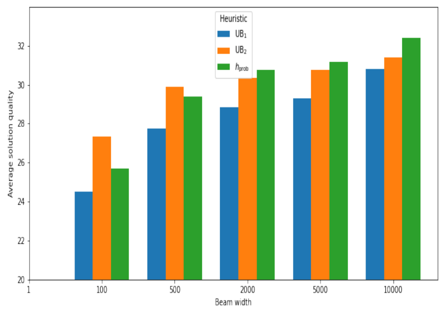
For each combination of instance parameters $n \in \{50, 100, 200, 500\}$, $m \in \{2, 3, 5, 10\}$, and $|\Sigma| \in \{2, 4\}$, we generate 10 random problem instances. First, m sequences of equal length are generated uniformly at random. Then, the gap constraints for each instance (i.e., positions within sequences) are generated such that for each $G_i(j), j = 1, \ldots, |s_i|, i = 1, \ldots, m$, a value $G_{s_i}(j) \in \mathcal{U}(\{\lfloor 0.5 \cdot |\Sigma| \rfloor, \ldots, \lfloor 1.5 \cdot |\Sigma| \rfloor\})$ is assigned. Therefore, a total of 320 problem instances are generated. The benchmark set is denoted as Random. All instances from the Random dataset can be found at `https://github.com/markodjukanovic90/NikolaBalabanDiplomski/tree/main/src/instance_i_generator`.

4.2 Parameter tuning

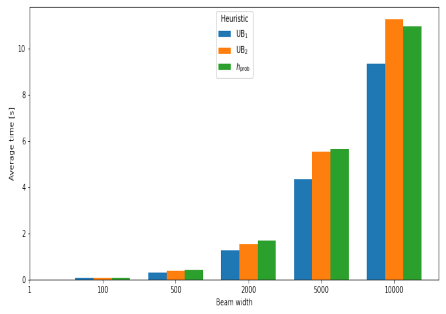
For the Bs approach, a high beam width $\beta = 10,000$ and $h = h_{\text{prob}}$ are used, as further increasing the beam size does not significantly improve solution quality. The heuristic h_{prob} provides slightly better guidance than UB_2 and significantly better than UB_1; see Fig. 2.

After preliminary experiments, we fixed number_sources_from_R $= 10$ and *beam_iters* $= 50$, as high-quality results were already achieved with this number of iterations. Root nodes kept in the global set are prioritized according to $h' = \text{UB}_2$, which consistently improves IMSBS performance compared to using UB_1. The remaining parameters of the IMSBS approach were tuned via a simple grid search over the following values: $h \in \{\text{UB}_1, \text{UB}_2, h_{\text{prob}}\}$ and $\beta \in \{100, 500, 2000\}$, see Fig. 3.

To ensure comparable average runtime between IMSBS and the baseline Bs approach, we use the following IMSBS configuration in our experiments: $h =$

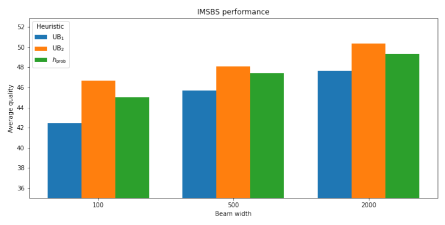


(a) Avg. quality over all instances from the Random benchmark suite.

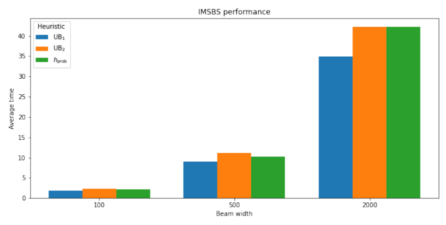


(b) Avg. runtime over all instances from the Random benchmark suite.

Fig. 2: Beam search tuning results on the Random benchmark suite.



(a) Avg. quality for different IMSBS settings over all instances from the Random benchmark suite.



(b) Avg. runtime for different IMSBS settings over all instances from the Random benchmark suite.

Fig. 3: IMSBS parameter tuning results on the Random benchmark suite.

UB$_2$ and $\beta = 500$. These values balance runtime relative to Bs while maintaining reasonable solution quality.

Similarly, for the Imsbs-greedy approach with fixed $\beta = 1$, we allow a budget of 5000 beam search iterations ($beam\_iters = 5000$), while all other parameters remain the same as for Imsbs.

4.3 Numerical results for heuristic approaches

Numerical results for all three heuristic approaches are reported in Table 1, which is organized into two main parts. The first three columns describe the instance characteristics: the number of sequences (m), the sequence length (n), and the alphabet size ($|\Sigma|$). The remaining columns report the performance of the three heuristic approaches: Bs, Imsbs-greedy, and Imsbs.

For each algorithm, two performance indicators are provided: the average objective value $\overline{obj}$ and the average running time in seconds $\overline{t}$, computed over 10 instances per group. This layout allows a direct comparison of solution quality and computational effort across algorithms under identical instance settings.

Table 1: Numerical results of the three heuristic approaches on the Random benchmark suite.

Inst.			Bs		Imsbs-Greedy		Imsbs	
m	n	$\lvert\Sigma\rvert$	$\overline{obj}$	$\overline{t}[s]$	$\overline{obj}$	$\overline{t}[s]$	$\overline{obj}$	$\overline{t}[s]$
2	50	2	33.6	0.02	26.1	0.00	**35.0**	0.04
2	50	4	30.1	0.89	24.8	0.00	**30.1**	0.14
2	100	2	48.9	1.74	41.5	0.00	**65.1**	0.44
2	100	4	**62.1**	9.75	50.5	0.00	61.6	0.67
2	200	2	99.1	14.76	67.7	0.01	**123.3**	3.20
2	200	4	120.5	38.56	74.8	0.01	**120.9**	3.55
2	500	2	65.3	21.05	77.3	0.03	**184.9**	62.69
2	500	4	214.6	155.30	152.4	0.02	**222.8**	35.27
3	50	2	17.5	0.02	17.4	0.00	**25.8**	0.05
3	50	4	**21.7**	0.17	16.1	0.00	**21.7**	0.09
3	100	2	19.7	0.07	25.2	0.00	**46.6**	1.42
3	100	4	34.1	2.50	21.1	0.00	**42.3**	2.49
3	200	2	15.2	0.15	29.2	0.01	**73.6**	9.63
3	200	4	85.3	24.99	38.3	0.01	**91.3**	33.44
3	500	2	12.6	0.10	36.0	0.04	**57.0**	21.62
3	500	4	90.7	80.60	52.6	0.03	**149.2**	171.48
5	50	2	4.8	0.00	9.4	0.00	**15.3**	0.06
5	50	4	8.9	0.01	7.1	0.00	**12.2**	0.11
5	100	2	6.3	0.00	10.3	0.01	**16.2**	0.16
5	100	4	5.3	0.01	7.4	0.01	**18.7**	0.41
5	200	2	5.3	0.01	14.5	0.01	**19.2**	0.31
5	200	4	6.4	0.02	10.6	0.02	**22.6**	0.68
5	500	2	5.9	0.09	15.2	0.08	**21.5**	1.32
5	500	4	6.8	0.10	13.4	0.08	**20.5**	1.15
10	50	2	1.7	0.00	3.7	0.00	**5.9**	0.12
10	50	4	1.9	0.00	2.7	0.00	**4.4**	0.31
10	100	2	1.1	0.00	4.6	0.01	**5.2**	0.22
10	100	4	2.2	0.01	2.8	0.01	**4.1**	0.47
10	200	2	2.5	0.01	5.2	0.04	**6.7**	0.28
10	200	4	2.2	0.01	3.7	0.04	**3.9**	0.43
10	500	2	1.8	0.08	5.9	0.22	**6.3**	1.00
10	500	4	1.9	0.08	4.2	0.23	**4.6**	1.27
Avg.			32.38	10.97	27.24	0.03	**48.08**	11.08

Based on the numerical results presented in Table 1, the following conclusions can be drawn:

– *Solution quality comparison.* The Imsbs approach produces the highest solution quality in 31 out of 32 instance groups, on average surpassing the baseline Bs approach by approximately 50% in relative improvement. The baseline Bs performs equally well or better in only 2 groups. The weakest (but also fastest) approach on average is Imsbs-greedy, which performs competitively (and clearly better than Bs) only for instances with a larger number of input sequences ($m = 10$). This indicates that starting the beam search solely from the root node $(1, 1, \ldots, 1)$ can be suboptimal when many gap constraints must be satisfied. Exploring additional root nodes often improves the diversity of the search and consequently the quality of the final solutions.

- *Runtime comparison.* The Bs and Imsbs approaches exhibit comparable average runtimes. As expected, Imsbs-greedy has a significantly smaller average runtime due to inexpensive beam search iterations ($\beta = 1$), in contrast to the more computationally intensive iterations used by the other two approaches.
- *Other conclusions.* The behavior of Bs can be explained by the fact that for small m, starting from the root node $r = (1, \ldots, 1)$ is generally favorable, allowing the beam search to progress deeply and identify high-quality subsequences, particularly for higher beam widths. This is especially the case for instances with $m = 2$, where the difference in solution quality between Bs and Imsbs is noticeable but not dramatic. For larger m, the gap between the solutions of the two approaches becomes substantial. Overall, Imsbs demonstrates greater robustness and stability as the number of input sequences increases.

4.4 Numerical results for the special case $m = 2$

In this section, we compare the results of approaches from the literature that are specialized for the case $m = 2$. The competitors are listed below and are described in detail in [10].

- Dp: the basic dynamic programming algorithm;
- Dp-1: an advanced dynamic programming approach that uses Incremental Suffix Maximum Queries (ISMQ) with `Col` and `All` matrices to accelerate the basic DP algorithm;
- Dp-2: an enhanced dynamic programming algorithm that handles ISMQ using a *dequeue* data structure for further speedup. All three DP approaches are from [10]. Note that the original paper proposes answering ISMQ using Union-Find operations; however, due to lack of implementation details, we used a dequeue-based approach in our implementation.
- Ilp: an integer linear programming method proposed in this work (see Appendix A), motivated by ILP models for LCS problems [5].

Each algorithm was allocated 30 minutes of execution time. The Ilp model was solved using the general-purpose solver Cplex version 22.1.

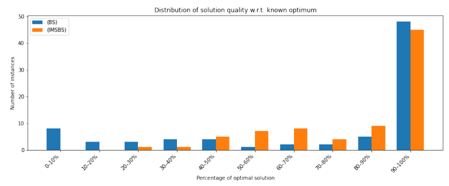
The results are reported in Table 2, which is organized as follows. The first three columns describe the characteristics of the instance groups, over which the results are averaged (10 instances per group). The remaining columns report the performance of the four algorithms: Dp, Dp-1, Dp-2, and Ilp. For each approach, both the average solution quality and the average execution time are provided across the 10 instances.

Table 2: Results on the Random benchmark set for $m = 2$: the exact approaches from the literature.

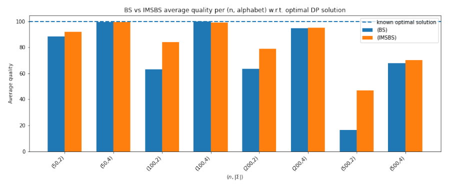
Inst.			Dp		Dp-1		Dp-2		Ilp	
m	n	$\|\Sigma\|$	$\overline{obj}$	$\overline{t}[s]$	$\overline{obj}$	$\overline{t}[s]$	$\overline{obj}$	$\overline{t}[s]$	$\overline{obj}$	$\overline{t}[s]$
2	50	2	38.1	0.01	38.1	**0.01**	38.1	0.01	38.1	89.43
2	50	4	30.3	**0.01**	30.3	0.02	30.3	0.02	30.3	18.0
2	100	2	77.4	0.1	77.4	**0.03**	77.4	0.05	–	–
2	100	4	62.3	0.07	62.3	**0.06**	62.3	0.09	0.00	1800.0
2	200	2	156.4	0.75	156.4	**0.13**	156.4	0.16	–	–
2	200	4	127.2	0.59	127.2	**0.25**	127.2	0.32	–	–
2	500	2	395.9	13.57	395.9	**0.84**	395.9	1.05	–	–
2	500	4	317.2	10.18	317.2	**1.70**	317.2	2.1	–	–

Based on the numerical results presented in Table 2, the following conclusions can be drawn:

– All three dynamic programming approaches are highly efficient and are able to find optimal solutions for all 80 problem instances.
– The runtimes required by the DP approaches to obtain optimal solutions are relatively short, mostly remaining below 10 seconds.
– The Ilp approach is able to solve only the smallest instances with $n = 50$, and its execution times are roughly two orders of magnitude higher than those of the DP approaches. For larger instances, especially those with $n = 200$, the ILP approach is unable to solve even the initial (root) relaxations.



(a) Distribution of solution quality for Bs vs. Imsbs with respect to the optimal solutions.



(b) Relative solution quality of Bs and Imsbs compared to the optimal solutions.

Fig. 4: Number of instances for which Bs and Imsbs achieve a specific ratio of the obtained (approximate) solution quality to the known optimal solution for $m = 2$.

Regarding the efficiency of Bs and Imsbs on instances solved optimally by dynamic programming ($m = 2$), Figure 4 illustrates the distribution of instances (on the y-axis) achieving specific ratios of solution quality relative to the known optimal solutions (on the x-axis). In particular, Imsbs reaches a near-optimal solution for 49 out of 80 instances. For 75% of the problem instances, the solution obtained by IMSBS is at least 80% of the optimal solution, whereas these numbers are significantly lower for the BS approach. These results indicate that while IMSBS performs considerably better than BS, there remains room for improvement in designing a more robust guiding function especially for small alphabet size, as discussed in the future work section.

5 Conclusions and Future Work

In this work, we studied a generalized version of the Longest Common Subsequence (LCS) problem for an arbitrary number of sequences, a problem with broad applications in bioinformatics, particularly for comparing DNA and RNA molecules structurally. The Variable Gapped LCS problem considered here extends the classical LCS formulation by constraining the allowable distances between matched characters, reflecting biological scenarios in which spatially closer nucleotides interact more strongly than more distant ones.

We proposed a beam-search-based solution framework, starting with a baseline beam search over a formally defined state-space graph and extending it to an enhanced Iterative Multi-Source Beam Search (IMSBS) approach. IMSBS combines local exploration from promising root nodes with a global strategy for selecting these nodes based on previously collected search information. Experimental results show that IMSBS consistently produces the highest-quality solutions, often achieving 80–90% of the known optimal solutions for two input sequences, while requiring comparable or less computation time than the baseline beam search. For the special case with $m = 2$, we additionally compared three exact dynamic programming approaches from the literature with a newly designed ILP model. The results confirm that the advanced DP variant solved all instances in the shortest amount of time, while the ILP approach was limited to smaller instances.

Future research directions include designing more sophisticated heuristics that directly incorporate gap constraints into scoring functions, optimizing IMSBS to efficiently reuse previously explored regions of the state space, generating realistic biological instances for real-world case studies, and scaling experiments to longer sequences.

Acknowledgments. This publication is co-funded by the European Union's Horizon Europe research and innovation program under the Marie Sklodowska-Curie COFUND Postdoctoral Programme (grant agreement No. 101081355 – SMASH), and by the Republic of Slovenia and the European Union through the European Regional Development Fund. Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Research Executive Agency (REA). Neither the European Union nor the REA can be held responsible for them.

References

1. Finding longest common subsequences: New anytime A* search results. Applied Soft Computing **95**, 106499 (2020)
2. Adamson, D., Kosche, M., Koß, T., Manea, F., Siemer, S.: Longest common subsequence with gap constraints. In: International Conference on Combinatorics on Words. pp. 60–76. Springer (2023)
3. Adi, S.S., Braga, M.D., Fernandes, C.G., Ferreira, C.E., Martinez, F.V., Sagot, M.F., Stefanes, M.A., Tjandraatmadja, C., Wakabayashi, Y.: Repetition-free longest common subsequence. Discrete Applied Mathematics **158**(12), 1315–1324 (2010)
4. Bergroth, L., Hakonen, H., Raita, T.: A survey of longest common subsequence algorithms. In: Proceedings Seventh International Symposium on String Processing and Information Retrieval. SPIRE 2000. pp. 39–48. IEEE (2000)
5. Blum, C., Festa, P.: Metaheuristics for String Problems in Bio-informatics, vol. 6. John Wiley & Sons (2016)
6. Farhana, E., Rahman, M.S.: Constrained sequence analysis algorithms in computational biology. Information Sciences **295**, 247–257 (2015)
7. Jiang, T., Lin, G., Ma, B., Zhang, K.: The longest common subsequence problem for arc-annotated sequences. Journal of Discrete Algorithms **2**(2), 257–270 (2004)
8. Lainscsek, C., Sejnowski, T.J.: Delay differential analysis of time series. Neural computation **27**(3), 594–614 (2015)
9. Mousavi, S.R., Tabataba, F.: An improved algorithm for the longest common subsequence problem. Computers & Operations Research **39**(3), 512–520 (2012)
10. Penga, Y.H., Yangb, C.B.: The longest common subsequence problem with variable gapped constraints. In: Proceedings of the 28th Workshop on Combinatorial Mathematics and Computation Theory, Penghu, Taiwan. pp. 17–23 (2011)

A ILP model for the fixed $m = 2$ VGLCS Problem

When considering the VGLCS problem with two input sequences, in this section we propose an *integer programming model*, representing a methodological approach that differs from most existing approaches in the literature, which are predominantly based on dynamic programming (DP). This is a proof-of-concept showing structural difficulty of this approach and a baseline modeling contribution when solving the fixed version of the tackled problem. Shortly, the purpose of the ILP model is not computational competitiveness but structural modeling and benchmarking.

To this end, let:

- $M = \{(i, j) \mid s_1[i] = s_2[j]\}$ denote the set of all positions where the characters match.

For each $(i, j) \in M$, we define a binary variable:

$$x_{i,j} = \begin{cases} 1, & \text{if the pair } (i, j) \text{ is selected as part of the solution,} \\ 0, & \text{otherwise.} \end{cases}$$

We introduce additional binary variables $s_{i,j}$ indicating whether the pair (i, j) represents the *starting position of the subsequence*.

Objective function. We maximize the number of selected admissible matches:

$$\max \sum_{(i,j)\in M} x_{i,j}$$

Constraints. The following constraints are imposed:

1. Predecessors (variable gap). For each $(i, j) \in M$, we define the set of valid predecessors:

   $\text{Pred}(i, j) = \{(i', j') \in M \mid i' < i,\ j' < j,\ i - i' \le G_{s_1}[i]+1,\ j - j' \le G_{s_2}[j]+1\}$

   The activation constraint is given by:

   $$x_{i,j} \le \sum_{(i',j')\in\text{Pred}(i,j)} x_{i',j'} + s_{i,j}$$

2. Starting position. At most one starting pair is allowed:

   $$\sum_{(i,j)\in M} s_{i,j} \le 1$$

3. Conflict constraints (character ordering). Two distinct pairs (i, j) and (i', j') are in conflict if they violate the character order:

   $$\text{If } (i \le i' \text{ and } j \ge j') \text{ or } (i \ge i' \text{ and } j \le j'), \text{ then: } \quad x_{i,j} + x_{i',j'} \le 1$$

The domain of the variables is given by

$$x_{i,j},\ s_{i,j} \in \{0,1\}, \quad \forall (i,j) \in M.$$

The model is formulated to maximize the number of valid matches forming a common subsequence, while respecting the ordering of elements and the maximum allowed gaps between consecutive elements in both sequences.