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 involving flexible gap constraints between consecutive solutions' characters in the input sequences. The problem arises in molecular sequence comparison, where structural distance constraints between residues must be respected, and time-series analysis, in which events are required to occur within specified temporal delays. We propose a novel search framework based on a root-based state graph representation, in which the overall state space is defined as the union of generally large number of root-based state subgraphs. To cope with the resulting combinatorial explosion, an iterative beam search strategy is employed, dynamically maintaining a global pool of promising candidate root nodes, enabling effective control of diversification across iterations. To explore 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, generating a benchmark of 320 synthetic instances with up to 10 input sequences and up to 500 characters. Experimental results show robustness of the iterative beam search over the baseline beam search in a comparable runtime. Furthermore, for the special case with two input strings, the proposed method achieves great performance compared to the specialized dynamic programming algorithms on small-to-medium-sized instances.

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, playing a fundamental role in the analysis and comparison of molecular sequences. Given an arbitrarily large set of input sequences $S = \{s_1, \ldots, s_m\}, m \in \mathbb{N}$, 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 of real-world sequences. 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), originally introduced for 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 at positions of sequence as $G_i \colon \{1, \ldots, |s_i|\} \mapsto \mathbb{N}$. Precisely, if the characters of s occur at positions $i_i^1 < \cdots < i_i^{|s|}$ in s_i, the gap constraint is satisfied iff $i_i^x - i_i^{x-1} \leq G_i[i_i^x] + 1, \quad x = 2, \ldots, |s|$.

A common subsequence s is considered *feasible* if the corresponding gap constraints are satisfied simultaneously for all sequences in S. As an example of a VGLCSP, given is a set of two input sequences $S = \{s_1 = \texttt{ABCA}, s_2 = \texttt{ACAB}\}$ with two gap constraints $G_1(\{1, \ldots, 4\}) \mapsto \mathbf{1} = G_2(\{1, \ldots, 4\})$. The sequence $s = \texttt{ACA}$ is a feasible solution because it is a common subsequence of the both input strings, as its corresponding characters appear at positions 1, 3 and 4 in s_1 and at the positions 1, 2, and 3 in the s_2. As the difference between each consecutive pair of the indices of these positions is less or equal $1 + 1 = 2$, s represents a feasible sequence of the tackled problem. On the other hand, sequence s=AB is a common subsequence as well, but it does not fulfill all gap constraints. In details, the corresponding characters appear at positions 1 and 2 in s_1 adn at positions 1 and 4 in s_2. Obviously, $4 - 1 = 3 > 2 = 1 + 1$ thus violating the second gap constraint.

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 (active) 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].

Although several dynamic programming approaches exist for the two-sequence case ($m = 2$) scenario [10], their computational complexity becomes prohibitive when expanded 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 tackled problem, we first present 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 $S = \{s_1, \ldots, s_m\}$ the set of input sequences, and by $S^{rev} = \{s_1^{rev}, \ldots, s_m^{rev}\}$. 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. Given is a positional vector $\mathbf{p}^L \in \mathbb{N}^m$, subproblem related to these positions is given by $S[\mathbf{p}^L] = \{s_i[\mathbf{p}_i^L] \mid i = 1, \ldots, m\}$. Last but not least, for a gap constraint G_i, its reverse constraint is denoted by G_i^{rev}, and is given as $G_i^{rev}(j) := G_i(|s_i| - j + 1)$, for each $j = 1, \ldots, |s_i|$.

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

2 Rooted State Space Formulation

Building on the state-space representation for the LCSP introduced in [1], we develop a *rooted state graph* model for the 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 $\mathbf{p}^{L,v} = (p_1^{L,v}, \ldots, p_m^{L,v})$ 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 each input sequence s_i. A direct arc $\alpha = (v_1, v_2)$, labeled with $\texttt{let}(\alpha) = a \in \Sigma$, 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 $i = 1, \ldots, m$, are first detected. For each such character $a \in \Sigma$, we identify the smallest feasible positions $p_i^{L,a} \geq p_i^{L,v}$ such that

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

In that way, a child state $w = (\mathbf{p}^{L,a} + \mathbf{1},\ l^v + 1)$ is created, unless it is dominated by another child state according to the dominance criteria defined later in this section. To efficiently compute children 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. As already pointed out, the value $q = \texttt{Succ}[i, j, a]$ stores the smallest index $q \geq j$ such that $s_i[q] = a$ so that 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 returned.

Goal states are those that cannot be further expanded. The trivial root state is given by $r = ((1, \ldots, 1),\ 0)$ as in-going edges to this node are not possible, representing an empty partial solution as in the case of LCS problem. However, it is not the only existing root node, in general. The problem can have many (possibly exponentially) root nodes due to the definition of the problem itself allowing a freedom to choose leading letter of partial solutions, induced from a position $u = (i_1^1, \ldots, i_m^1)$ matching the letter, as gap constraints do not apply to the starting letter of each solution.

This motivate us to introduce the state space subgraph rooted with the match u, denoted by RootedSubgraph(u), induced by empty partial solution, which consists of all nodes (and corresponding transitions) that can be feasibly reached via node $(u + \mathbf{1}, 1)$, which is constructed from the root node $(u, 0)$ through its extension by the letter at the positions u. In terms of feasible partial solutions, RootedSubgraph(u) will generate solutions whose leading letter is the letter induced by the match u, recursively repeating the procedure of expansion of solutions by one letter in all possible ways, until no more expansions are possible. As discussed in the following section, the structure of this subspace heavily depends on the choice of the root state from which the search is initiated. As an example, we illustrate the state space of RootedSubgraph$(u = (1, 1), 0)$, depicted in Fig. 1. Note that here the position $(1, 1)$ correspond to the match of letter `A`, where $u = (1, 1), 0)$ indeed represents a root node.

Motivation for Multi-Source Beam Search. Unlike the classical LCSP, the VGLCSP may exhibit multiple, potentially exponentially many, *disconnected root components* in its complete state space. Consequently, a search strategy that expands states from a single initial root may fail to reach the optimal solution. As an illustrative example given in Fig. 2, consider the sequence set $S = \{s_1 = \texttt{ATGGAAA},\ s_2 = \texttt{ATCCAAA}\}$, with gap constraints $G_{s_1} = G_{s_2} = 1$.

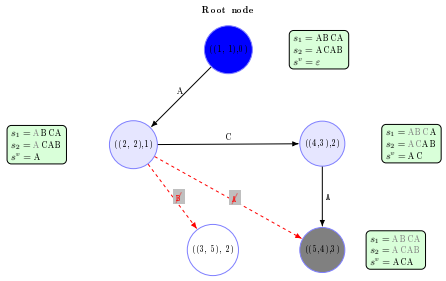


Fig. 1: State space graph RootedSubspace($u = ((1,1),0)$) for MVGLCSP between the sequences A`BCA` and A`CAB`, assuming $G_1 = G_2 = 1$. A final/terminal state is displayed in gray. The white-filled node represents a valid LCS node, but not a feasible VGLCS node, as the second gap constraint has been violated.

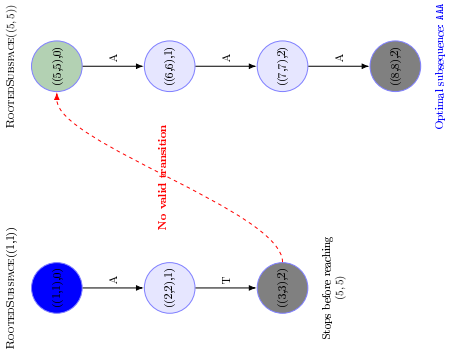


Fig. 2: Root state space subgraphs: disconnected components.

In this instance, any state with position vector $\mathbf{p}^L = (5,5)$ cannot be reached from the initial state $((1,1),0)$ by standard directed transitions. As a result, the optimal common subsequence `AAA` is unreachable when the search is initiated exclusively from the root $((1,1),0)$, and this subgraph is completely explored.

To address this issue, we define the complete state space of an VGLCSP instance naturally as $\bigcup_{v \in \mathcal{R}} \text{RootedSubspace}(\mathbf{q}^{L,v})$, where $\mathcal{R}$ denotes the set of all root states, i.e., states defined by position vectors (matches) $\mathbf{p}^{L,v}$ that cannot be reached from any other state by a series of direct transitions. However, explicitly enumerating all such root states is computationally prohibitive in general, requiring $O(n^m)$ time, and thus infeasible for instances with large m or n.

This structural characteristic of the VGLCSP motivates the proposed methodology titled *Iterative Multi-Source Beam Search* (IMSBS) approach, which is specifically designed to dynamically explore multiple promising rooted subregions of the state space systematically moving from one rooted subgraph to another one. 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

Before the main details on the Iterative Multi-source Beam Search (IMSBS) approach are presented, we give a brief overview over a general beam search metaheuristic approach.

Beam search (BS) is a well-established tree-search-based 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 levels, and by a heuristic guidance h, which guides the selection of most promising candidates for further expansion. Initially, the procedure receives a set of nodes as the initial beam, which are all expanded, generating a set of candidate nodes for the subsequence level. After being evaluated, up to β most promising nodes are kept for in the beam of this level. The procedure keeps with it until beam is empty, returning a node with th longest partial solution.

In the IMSBS, BS component is served to intensify the search process. As the search space of an VGLCSP instance is given by the set of root nodes $\mathcal{R}$, BS procedure is given to explore the search space of RootedSubspace() of each match $\mathbf{p}^L$ and corresponding root node $(\mathbf{p}^L, 0)$. Note that one could employ here an exact technique (such as A^* or BFS). However, solving a subproblem itself could be a challenge task, even the same (time) complexity as the original problem itself. Thus, we have decided here for a more robust BS technique.

FiXme Note: Came up to here!!

Within the *Iterative Multi-Source Beam Search* (IMSBS) framework, standalone BS procedure is executed iteratively, each time initialized with a new beam $\mathcal{L}$ consisting of several promising candidates for root nodes. These 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 r.

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 is set to $\mathcal{R} = \{r = ((1, \ldots, 1), 0)\}$, corresponding to the empty partial solution, while the current best solution to $s_{\text{best}} = \varepsilon$. The algorithm iterates until either $\mathcal{R}$ becomes empty, a predefined maximum number of beam search calls is reached, or allowed time limit has exceeded.

At each iteration, a fixed number of the most promising root nodes from $\mathcal{R}$– controlled by the parameter *sources_from_R* and ranked according to heuristic

h'–are taken, extracted and stored in a temporary beam $\mathcal{L}$. These nodes are further refined, executing one effective backward beam search procedure for each node $w \in \mathcal{L}$ as its root node. The procedure works similarly as the baseline BS by expanding nodes with accordance to the gap constraints but in the reversed manner, from the right hand side of each sequence, moving the search towards their leading positions. In essence, for each $w \in \mathcal{L}$, the same RootedBeamSearch is executed on the reversed input sequences S^{rev} and the reversed gap constraints $(G_i^{rev})_{i=1}^m$ with an initial beam consisting of the node $((|s_i| - p^w - 1)_{i=1}^m, 0)$.

The procedure has completed returning the solution s_{bwrd}^w. Note that, in this way, the reverse string of s_{bwrd} is a feasible VGLCS solution. This partial solution is further assigned to the refined node $w \in \mathcal{L}$, which is updated to $w = (p^w, |(s_{bwrd}^w)^{rev}|)$, and passed to the forward beam search procedure.

After the refinement of each $w \in \mathcal{L}$, this set enters in as the initial beam, initiating execution of RootedBeamSearch with the beam width β and the heuristic guidance $h(v)$. During this phase, all completed (i.e., non-expandable) nodes are collected in a list $V_{complete}$ which is then returned along with the found solution s_b. Further, if a complete node from $V_{complete}$ corresponds to a solution that improves upon the current best, it becomes a new incumbent solution. Each such node is then independently expanded in an LCS-like manner, ignoring all gap constraints, in that way generating a set of candidate root nodes. Generated candidate nodes $r^w = (\mathbf{p}^w, 0)$ are inserted into the pool $\mathcal{R}$, provided that it has not already been part during the previous iterations. Once the beam search procedure has been terminated, a new Imsbs iteration is initiated with a freshly selected set of root nodes from $\mathcal{R}$, until one of the termination conditions is met. The pool $\mathcal{R}$ may be maintained as a priority queue implemented via a binary heap, to ensure efficient extraction of the most promising candidates for root nodes.

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

- A provision of balance between complete beam search execution (local exploration of promising paths) and the generation of new sources (global coverage of the search space).
- Preventing from suboptimal candidate root nodes reduced by executing backward beam search procedure.

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:*

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

  This is an estimate of the potential extension of the sequence up to the end of the shortest remaining sequence.
- *Character Frequency Alignment:*

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

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

Require: Sequences $s_1, \ldots, s_m$; Gap functions $G_{s_i}(\cdot)$; beam width $\beta > 0$; $sources\_from\_R > 0$; $beam\_iters > 0$, heuristics h, h', time limit t_{lim}
Ensure: Approximate common subsequence s_{best} which fulfills the gap constraints
1: Initialize $\mathcal{R} \leftarrow \{(\mathbf{1}, 0)\}$ ▷ A set of root nodes
2: **for all** $a \in \Sigma$ **do** ▷ Find closest matching positions for each letter
3: $\mathbf{p}_a^w = (p_{i,a}^w)_{i=1}^m \leftarrow$ minimum position $p_{i,a}^w \geq 1$ and $s_i[p_{i,a}^w] = a$ (ignore all gap constraints, remove dominated root positions)
4: $R \leftarrow R \cup \{(p_a^w, 0)\}$
5: **end for**
6: $iter \leftarrow 0$
7: $s_{\text{best}} \leftarrow \varepsilon$
8: **while** $\mathcal{R} \neq \emptyset \wedge iter < beam\_iters \wedge t_{lim}$ not exceeded **do**
9: Select $\mathcal{L} \subseteq \mathcal{R}$ $sources\_from\_R$ and remove them from $\mathcal{R}$ ▷ $\mathcal{R}$ prioritized by h'
10: **for all** $w \in \mathcal{L}$ **do**
11: $s_{bwrd}^w, \_ \leftarrow$ RootedBeamSearch($(\{(|s_i| - p_i^w - 1)_{i=1}^m, 0\}, S^{rev}, G^{rev}, \beta', h')$ ▷ Refine each $w \in \mathcal{L}$
12: $l^w \leftarrow |(s_{bwrd}^w)^{rev}|$ ▷ Update l^w, assign the reverse of s_{bwrd}^w to w
13: **end for**
14:
15: $s_b, V_{complete} \leftarrow$ RootedBeamSearch($\mathcal{L}, S, G, \beta, h$)
16: **if** $|s_b| > s_{best}$ **then**
17: $s_{best} \leftarrow s_b$
18: **end if**
19: **for all** $v \in V_{complete}$ **do**
20: **for all** $a \in \Sigma$ **do** ▷ Find closest matching positions for each letter
21: $\mathbf{p}_a^w = (p_{i,a}^w)_{i=1}^m \leftarrow$ minimum position $p_{i,a}^w > p_i^v$ and $s_i[p_{i,a}^w] = a$ (ignore all gap constraints)
22: $\mathcal{R} \leftarrow \mathcal{R} \cup \{r^w = (\mathbf{p}_a^w, 0)\}$ (if has not previously been added) ▷ Update $\mathcal{R}$
23: **end for**
24: **end for**
25: $iter \leftarrow iter + 1$
26: **end while**
27: **return** s_{best}

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 the details in [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 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 with dimension $k \times n$ effectively

generated by a dynamic programming. Assuming independence between input sequences, the nodes/partial solutions within the same level of beam search are evaluated by determining the associated probabilities of their expansions by $\overline{k}$ letters (this value is heuristically evaluated, as proposed in [9]). Denote that heuristic guidance by h_{prob}.

4 Experimental Evaluation

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

- 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; it is configured to use an average 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.

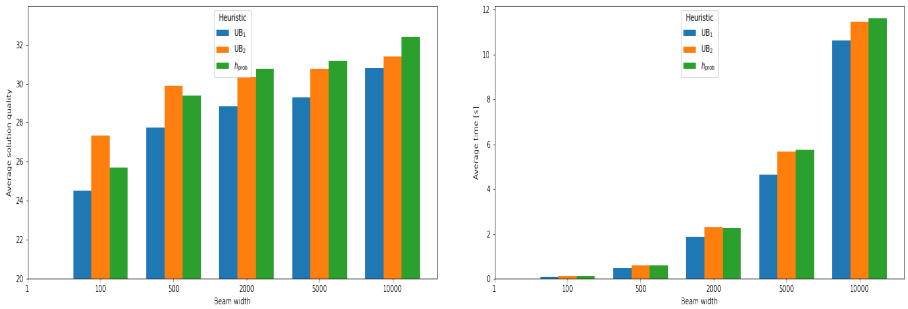
4.1 Instance generation

For each combination of instance parameters $n \in \{50, 100, 200, 500\}$, $m \in \{2, 3, 5, 10\}$, and $|\Sigma| \in \{2, 4\}$, 10 random problem instances are generated. 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. 3.

First, we fixed $t_{lim} = 1800$ CPU seconds. By conducting a series of preliminary experiments, we decided to fix $sources\_from\_R = 10$ and $beam\_iters =$

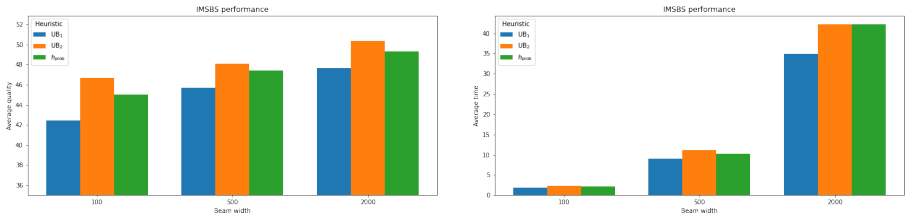


(a) Avg. quality over all instances from the RANDOM benchmark suite.

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

Fig. 3: Beam search tuning results on the RANDOM benchmark suite.

100 as good results were already achieved employing these numbers. Root nodes kept in the global set are prioritized according to $h' = \mathrm{UB}_2$, which consistently improves IMSBS performance compared to using UB_1. Remaining parameters of the IMSBS approach were tuned via a simple grid search over the following values: $h \in \{\mathrm{UB}_1, \mathrm{UB}_2, h_{\mathrm{prob}}\}$ and $\beta \in \{100, 500, 2000\}$, see Fig. 4. As a well-performing, effective setting of the backward beam search $\mathrm{BS}^{\uparrow}$ used to refine the generated root nodes in $\mathcal{R}$, we fixed $h = \mathrm{UB}_2$ and $\beta = 10$.



(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. 4: IMSBS parameter tuning results on the RANDOM benchmark suite.

To keep comparable average runtime between IMSBS and the baseline BS approach, we use the following IMSBS configuration in our experiments: $h = \mathrm{UB}_2$ and $\beta = 500$. These values balance runtime relative to BS while maintaining reasonable solution quality. Similarly, for the IMSBS-GREEDY approach with fixed $\beta = 1$, while all other parameters hold as in the IMSBS.

4.3 Comparison between the 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, each in its own block. For each algorithm, two performance indicators are provided: the average objective value $\overline{obj}$ and the average running time in seconds $\bar{t}$, computed over 10 instances per group.

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

Inst.			BS		IMSBS-GREEDY		IMSBS	
m	n	$\|\Sigma\|$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$
2	50	2	33.6	0.02	33.1	0.00	**37.7**	0.06
2	50	4	30.1	0.98	27.7	0.00	**30.1**	0.16
2	100	2	48.9	2.07	64.5	0.01	**72.8**	0.94
2	100	4	**62.1**	11.19	56.9	0.01	61.6	0.91
2	200	2	99.1	18.56	95.5	0.02	**136.4**	6.21
2	200	4	120.5	38.15	116.1	0.05	**124.9**	6.58
2	500	2	65.3	23.27	153.6	0.07	**265.7**	119.75
2	500	4	214.6	163.69	227.7	0.12	**294.4**	60.49
3	50	2	17.5	0.03	27.2	0.00	**31.2**	0.14
3	50	4	21.7	0.18	21.5	0.00	**22.9**	0.19
3	100	2	19.7	0.06	41.8	0.01	**58.5**	3.15
3	100	4	34.1	2.35	43.4	0.03	**48.4**	5.58
3	200	2	15.2	0.16	63.6	0.02	**90.0**	22.48
3	200	4	85.3	23.45	77.2	0.08	**97.1**	72.25
3	500	2	12.6	0.10	69.9	0.07	**102.9**	53.56
3	500	4	90.7	86.35	104.2	0.29	**187.7**	412.27
5	50	2	4.8	0.00	14.9	0.00	**20.0**	0.36
5	50	4	8.9	0.01	13.6	0.06	**15.3**	0.79
5	100	2	6.3	0.01	17.7	0.01	**22.4**	0.57
5	100	4	5.3	0.01	**23.2**	10.85	22.1	1.44
5	200	2	5.3	0.01	21.6	0.03	**26.6**	1.07
5	200	4	6.4	0.02	**32.5**	604.11	25.7	2.10
5	500	2	5.9	0.10	25.5	0.14	**27.9**	3.22
5	500	4	6.8	0.10	**43.6**	1341.25	26.9	3.52
10	50	2	1.7	0.00	8.8	2.28	**9.1**	0.47
10	50	4	1.9	0.00	**7.0**	508.36	6.1	1.46
10	100	2	1.1	0.00	**14.0**	1421.10	8.6	0.54
10	100	4	2.2	0.01	**8.9**	1800.45	6.3	1.51
10	200	2	2.5	0.01	**13.2**	1710.49	10.3	0.77
10	200	4	2.2	0.02	**7.9**	1800.54	6.1	1.74
10	500	2	1.8	0.08	**13.8**	1611.70	9.5	1.53
10	500	4	1.9	0.09	**8.2**	1800.46	6.1	2.37
Avg.			32.38	10.97	46.82	394.14	**59.73**	24.63

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

- *Solution quality comparison.* The IMSBS approach produces the highest average solution quality in 21 out of 32 instance groups, surpassing the baseline BS approach dramatically, indicating that selecting a right root node is essential in achieving high-quality solutions for this problem. The baseline BS performs equally well or better in just one specific case. Concerning the IMSBS-GREEDY, it surpasses the other two approaches in 10 (out of 32) cases, mostly for the instances with larger $m = 10$, having shorter final solutions where it turns out that frequently jumping from one to another part of the search space is strategically better than strengthening intensification around a solution. Exploring additional root nodes serves to improve the diversity of the search process. This is exactly where IMSBS-GREEDY fits well.
- *Runtime comparison.* The BS and IMSBS approaches are tuned to exhibit comparable average runtimes. IMSBS-GREEDY has a significantly larger average runtime due to a higher number of executed iterations ($beam\_iters = 10,000$), in contrast to more computationally intensive iterations used by the other two approaches (limited to $beam\_iters = 100$).
- *Other conclusions.* Overall, IMSBS demonstrates greater robustness and stability as the number of input sequences increases in comparison to the baseline BS approach which performance is gradually deteriorated.

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 decided to employ a dequeue-based approach in our implementation.
- ILP: an integer linear programming method proposed in this work (see Appendix A), motivated by the ILP model for LCS problem [5].

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

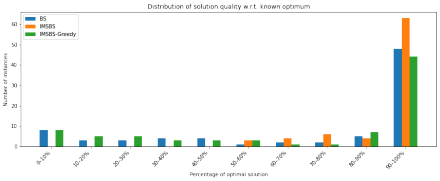
The results are reported in Table 2, which is organized as follows. The first three columns describe the characteristics of the instance groups, over which the results are averaged (10 instances per group). The remaining columns report the performance of the four algorithms: DP, DP-1, DP-2, and ILP. 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.

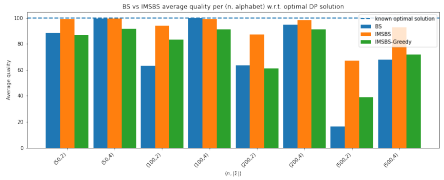
Inst.			DP		DP-1		DP-2		ILP	
m	n	$\|\Sigma\|$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$
2	50	2	38.1	0.01	38.1	**0.01**	38.1	0.01	38.1	168.3
2	50	4	30.3	**0.01**	30.3	0.02	30.3	0.02	30.3	28.0
2	100	2	77.4	0.1	77.4	**0.03**	77.4	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, solving all 80 problem instances.
- The runtimes of DPs required to obtain optimal solutions are relatively short, mostly remaining below 10 seconds.
- The ILP approach is able to solve only the smallest instances with $n = 50$; 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) Relative solution quality percentage distribution for each heuristic approach with respect to the optimal solutions.



(b) Relative solution quality achieved by the heuristic approaches compared to the optimal solutions.

Fig. 5: Number of instances for which BS and IMSBS achieve a specific ratio of the obtained (approximate) solution quality to the known optimal solution for $m = 2$.

Concerning efficiency of BS, IMSBS-GREEDY, and IMSBS on the instances solved optimally by dynamic programming ($m = 2$), Figure 5 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 in 63 out of 80 instances. These numbers are significantly lower in the case of other two heuristic approaches. While in the case of instances with $|\Sigma| = 4$ (and $m = 2$), difference in the solution qualities between these two approaches is slightly in favor of IMSBS, in the case of instance with small-sized alphabet ($|\Sigma| = 2$) the IMSBS approach clearly outperforms the other two approaches. Obviously, for the group of instances with $n = 500$ and $|\Sigma| = 2$ there is a room for further improvement. This is due to the fact that the LCS heuristics perform weakly on the instances with smaller alphabet size and larger sequence length [1] thus one has to look for other alternatives, 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 with an arbitrary number of sequences, 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 distant ones, but also to time-series analyses when events occur within specified time delays.

We proposed a beam-search-based solution framework, starting with a baseline beam search over a formally defined state-space graph. It is further extended to an enhanced Iterative Multi-Source Beam Search (IMSBS) approach. IMSBS combines local exploration from promising root nodes with a global strategy of selecting candidates for root nodes based on previously collected search information. Experimental results indicate the IMSBS approach consistently produces highest-quality solutions over the baseline beam search strategy. In the case of instances with shorter feasible sequences, the strategy of frequently jumping from one feasible region to another one in the search space appears efficient, which is done by setting the beam width to one in IMSBS approach. Concerning exact solving, we compared three dynamic programming approaches from the literature with a newly designed ILP model. The results confirmed the DP variants solved all instances in the shortest amount of time, while the ILP approach was limited to smaller instances. Additionally, IMSBS approach has frequently achieved close-to-optimum results for the two-input-sequence problem case, while requiring comparable or less computation time than the baseline beam search.

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, testing designed approaches on real-world biological instances, and scaling experiments up 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:

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

   The activation constraint is given by:

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

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

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

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

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

4. Auxiliary constraints: $s_{i,j} = 1 \Rightarrow x_{i',j'} = 0, \forall(i' \leq i-1, j' \leq j-1)$. The constraints are expressed in the following form:

$$\sum_{(i',j')|i' \leq i \wedge j' \leq j} x_{i'j'} + s_{i,j} \leq 1, \forall (i,j) \in M$$

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.

B Pseudocodes

Algorithm 2 Rooted Beam Search

function RootedBeamSearch($R, S, \{G_i\}_{i=1}^m, \beta, h$) ▷ r: root node, $S = \{s_1, \ldots, s_m\}$, gap constraints G_i, beam width β, heuristic h
  $s_{\text{best}} \leftarrow \varepsilon$
  $B \leftarrow R$ ▷ Starting beam
  $V_{\text{complete}} \leftarrow \emptyset$
  while $B \neq \emptyset$ **do**
    $Children \leftarrow \emptyset$
    for all $v \in B$ **do**
      if v is complete **then**
        $V_{\text{complete}} \leftarrow V_{\text{complete}} \cup \{v\}$
        if $v.l^v > |s_{\text{best}}|$ **then**
          $s^v \leftarrow$ Extract partial solution associated with v
          $s_{\text{best}} \leftarrow s^v$
        end if
        continue
      end if
      $C_v \leftarrow$ Expand($v, S, \{G_i\}_{i=1}^m$)
      $Children \leftarrow Children \cup C_v$
    end for
    Sort($Children, h$) ▷ decreasing order
    $B \leftarrow Children[:\beta]$
  end while
  return $s_{\text{best}}, V_{\text{complete}}$
end function