On Solving the Multiple Variable Gapped Longest Common Subsequence Problem

Marko Djukanović[1,2,6], Nikola Balaban[1], 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 tackles the variable gapped longest common subsequence problem, a variant of the well-known longest common subsequence (LCS) problem with an arbitrary large set of input strings. The problem has an application in comparing molecular sequences allowing structural distance constraints between nucleotides, and time series analysis where the events occur within specified temporal delays. The proposed methodology first employs root-based state graph definition, and the definition of the full state space given as the union of, in general, a large number of root-based state graphs. To deal with this issue, an iterative beam search approach is employed by utilizing a global, dynamically generated set of root nodes to control the diversification of the approach. To guide the search of standalone beam search, several heuristic guidances from the LCS literature are probed. For the first time, the computational experiments on this general problem are performed by generating 300 synthetic problem instances. Our iterative approach has shown the best performance when compared to the baseline beam search and a greedy approach employing comparable amount of time for the former two. Moreover, for the special case with two input strings, our iterative strategy approach could obtain an optimal solution on more than 80% of problem instances with two input strings, solved by all three dynamic approaches from literature.

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, particularly useful in the analysis of molecular sequences. Given an arbitrary large set of input sequences $S = \{s_1, \ldots, s_m\}, m \in \mathbb{N}$ over an alphabet Σ, we aim to identify the longest possible subsequence common to all $s_i \in S$. In recent decades, many practically motivated variants of the LCSP have been proposed by considering additional structural or biological constraints. Among the well-known variants are the constrained, arc-preserving, repetition-free [7, 6, 3], etc.

In this work, we focus on solving the *Variable Gapped Longest Common Subsequence Problem* (VGLCSP), introduced in [10] for two input strings, and also theoretically examined in [2]. This problem extends the standard LCSP by imposing flexible distance limits ("gaps") between consecutive symbols in the resulting subsequence, allowing these gaps to vary along the input sequences. Formally, for each input sequence $s_i \in S$ and gap function/constraint $G_i \colon \{1, \ldots, |s_i|\} \mapsto \mathbb{N}$ if the characters of a subsequence s appear at the positions $i_i^1 < \ldots < i_i^{|s|}$ in s_i, we say that the gap constraint is satisfied if $i_i^x - i_i^{x-1} \leq G_{s_i}[i_i^x] + 1$ for all $x = 2, \ldots, |s|$. A common subsequence s is considered feasible if the corresponding gap constraints hold across all sequences in S. The VGLCSP represents a flexible and realistic model of sequence alignment, which is particularly useful for comparing DNA or protein sequences with variable structural distances. Another application includes the time series analysis and specially the events that must occur within specified temporal delays [8].

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 presenting the algorithmic concept of the tackled problem, the common notation used throughout this work is introduced.

Let $|s|$ denote the length of a sequence s. The term $s[i]$ refers to the character at the i-th position of sequence s, where the first position is indexed by $i = 1$. The substring of s that begins at index i and ends at index j is denoted by $s[i, j]$; if $i = j$, this corresponds to the single character $s[i]$, whereas 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 by

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

We use $S = \{s_1, \ldots, s_m\}$ to denote the set of input sequences, and unless stated otherwise, n denotes the length of the longest sequence in S, and $m \in \mathbb{N}$ the number of input sequences (corresp. gaps constraints).

The remainder of the paper is organized as follows. Section 2 explains the state graph of the tackled problem. In Section 3 we provided a design of our main methodological contributions through the iterative multi-source beam search approach. Section 4 reports the empirical evaluation between our approach, the baseline standalone beam search and a greedy approach along with the special case of the instances with two input sequences. Finally, Section 5 provides conclusions and an outline for future work.

2 Root-based State Space Formulation

Building on already established LCSP state-space representation from [1], we design a root-based *state graph* model for the VGLCSP. Each state represents (one or more) feasible partial solutions defined by the vector of positions induced by each input sequence and the solution length. Symbolically, a partial subsequence s^v induces the state node $v = (\mathbf{p}^{L,v}, l^v)$ if: (i) $p_i^{L,v} - 1$ represents the smallest index in s_i such that s^v is a subsequence of $s_i[1, s_i[p_i^{L,v} - 1]$; (ii) $l^v = |s^v|$, and (iii) all gap constraints G_{s_i} in the case of s^v are satisfied.

A directed arc $\alpha = (v_1 v_2)$, labeled with $lett(\alpha) = a \in \Sigma$, exists if $l^{v_2} = l^{v_1} + 1 \quad \text{and} \quad s^{v_2} = s^{v_1} \cdot a$, for a partial solution s^{v_1} of v_1 and a partial solution s^{v_2} of v_2.

To expand a state v, only those letters $a \in \Sigma$ that appears in the suffix sequence of s_i starting at position $p_i^{L,v}$, for each $i = 1, \ldots, m$, are considered. For each such letter a, minimal feasible positions $p_i^{L,a} \geq p_i^{L,v}$ so that $s_i[p_i^{L,a}] = a$ are identified which fulfill

$$p_i^{L,a} - p_i^{L,v} \leq G_{s_i}(p_i^{L,a}), i = 1, \ldots, m.$$

A child node is then defined by $w = (p^{L,a} + 1,\ l^v + 1)$, if not dominated by another child nodes. For the purpose of efficient calculation of child nodes and their positions, we make use of the following data structure, labelled `Succ`. It is a 3-dimensional integer list of (i, j, a), where i refers to the sequences s_i, the $1 \leq j \leq |s_i|$ refers to the position inside the sequence s_i and symbol $a \in \Sigma$. The $q =$`Succ`$[i, j, a]$ represents the smallest index $q \leq j$ so that $s_i[q] = a$, and the gap constraint is satisfied at that position, i.e., $q - j \leq G_i(q) + 1$. If no such position, -1 is assigned. The stru

The root state is given by $r = ((1, \ldots, 1), 0)$, representing the empty subsequence, while goal states correspond to non-expandable feasible states. The search space that starts with r as the root node is denoted by $Space(r)$. The space clearly depends on the selected root node from where the search starts, as pointed in the next paragraph.

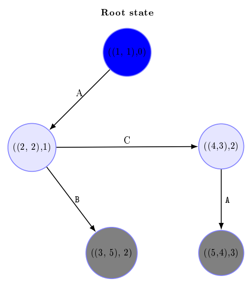


Fig. 1: State graph $Space(((1,1),0))$ for MVGLCSP between the sequences `ABCA` i `ACAB`. Two final/terminal states are displayed in gray.

Motivation for Multi-Source Beam Search. Unlike in the classical LCSP, the VGLCSP may contain multiple (potentially exponentially many) disconnected root components. For instance, in the sequences $S = \{s_1 = \texttt{ATGGAAAA}, s_2 = \texttt{ATCCAAAA}\}$ with gap limits $G_{s_1} = G_{s_2} = 1$ presented in Fig. 1, the initial state $((1,1),0)$ cannot reach any state node with the position vector $\mathbf{p}^L = (5,5)$. As a result, the optimal subsequence `AAA` becomes unreachable under traditional expansions and therefore missed out when starting at the root node $((1,1),0)$. Following this, the full state space is defined as $\bigcup_{q \in \mathcal{R}} Space(q)$, where $\mathcal{R}$ represents a set of all root nodes/states v with its positions $p^{L,v}$ which cannot be reached from any other state through regular directed transitions. Unfortunately, enumerating all root nodes in the search space of an VGLCS problem instance is, in general, a time consuming task, executed in $O(n^m)$ time, thus is hardly doable when m or n is huge. This structural property motivates our *Iterative Multi-Source Beam Search* (IMSBS) approach, specifically designed to explore multiple promising start regions of the state space and overcome disconnection issues caused by the gap constraints, inherent to the VGLCSP.

3 Iterative Multi-source Beam Search Framework

Beam search (BS) is a popular tree-search meta-heuristic approach which performs in a breadth-first-search manner, where the search is controlled by the parameter β referring to as the number of nodes to be further expanded at the considered level, and a guiding function h to make the decisions about which nodes to select for further expansions.

In the *Iterative Multi-source Beam Search* (IMSBS) framework, BS is iteratively executed with an initial set/beam $\mathcal{L}$ of promising root nodes, which are popped from a dynamically maintained global pool $\mathcal{R}$ of root nodes. This set serves to control the diversification of the approach. The single beam search utilized several heuristics to intensify the search towards producing feasible solutions; those are based on letter frequencies, minimal residual substring lengths, or probability-weighted estimates as provided in the next sections.

Pseudocode of the approach is given in Algorithm 1. First, a pool of root nodes $\mathcal{R}$ comprising of the root node $r = ((1, \dots, 1), 0)$ is initialized, referring to the empty partial solution $s_{\text{best}} = \varepsilon$. The algorithm runs until $\mathcal{R}$ is empty or the number of BS calls is exceeded (provided as the algorithm's parameter). At each iteration, the following instructions are executed. First, several most promising root nodes (parameterized by *number_of_sources_from_R*) according to heuristic evaluation h' have been selected from $\mathcal{R}$, subsequently stored into a temporal beam $\mathcal{L}$, serving as the initial beam, used in the current beam search procedure executed with a beam width β and a heuristic guidance h. During the execution, all completed (non-expandable) nodes are processed in the following way: If a new best solution has been found, s_{best} is updated accordingly. Each such complete node is then independently expanded in an LCS-manner (ignoring gaps), generating a set of children. For a generated child with position vector p^w, if $|s_i| - p_i^w + 1 \leq |s_{\text{best}}|$ holds for at least one index $i \in [m]$, it is omitted for further consideration as a proven suboptimal transition; otherwise, $r^w = (p^w, 0)$ has been added to $\mathcal{R}$ as a candidate for root node if not been already generated in the previous iterations (i.e., no duplicate root nodes are or had already been in the pool). When the beam search has completed, a subsequent beam search is started with a new set of (root) nodes taken from $\mathcal{R}$ if none of the conditions for termination is met. Note that the pool $\mathcal{R}$ is organized as a priority queue (binary heap).

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:*

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

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

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

  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].

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 $s_i[p^w_i] = a$
    (ignore all gap constraints)
11:             if $|s_{\text{best}}| < \min_{i=1,\ldots,m}\{|s_i| - p^w_i + 1\}$ 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 $s_{\text{best}}$
```

– *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 results of heuristic approaches designed for the VGLCSP with arbitrary $m \geq 2$, namely:

– The baseline beam search approach, that starts with the root node r as the initial beam and runs a single iteration of IMSBS. We denoted it by Bs.

– The tuned iterative multi-source beam search, given in Algorithm 1, denoted by Imsbs, that uses comparable runtime to those of the Bs.
– The IMSBS for the fixed $\beta = 1$, given in Algorithm 1, denoted by Imsbs-greedy, that show the influence of larger number of BS iterations on the performance of the Imsbs.

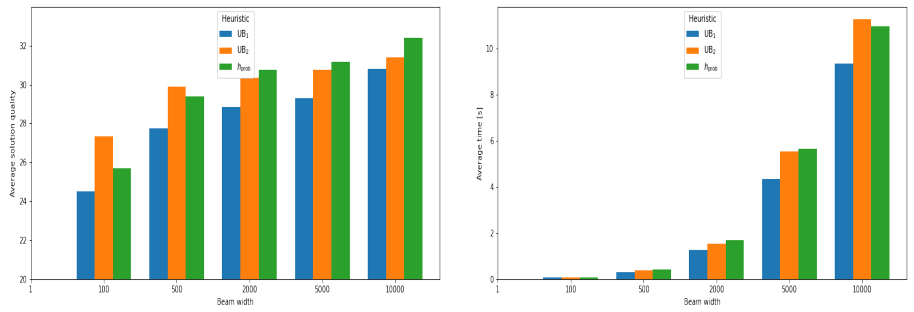
The Imsbs is implemented in Python 3.11, and evaluated on the HPC VEGA Supercomputer placed at IZUM, Maribor. This is a cluster of has 960 nodes, 1920 CPUs with AMD Epyc 7H12, and 2.35GHz. All experiments were executed in a single-threaded setting.

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 equally-size sequences generated uniform-at-random. Then, the gap constraints for each instance (i.e., positions of sequences) are generated so that for each $G_i(j), j = 1, \ldots, |s_i|, i = 1, \ldots, m$, assigned is a value $G_{s_i}(j) \in \mathcal{U}(\{\lfloor 0.5 \cdot |\Sigma| \rfloor, \ldots, \lfloor 1.5 \cdot |\Sigma| \rfloor\})$. 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

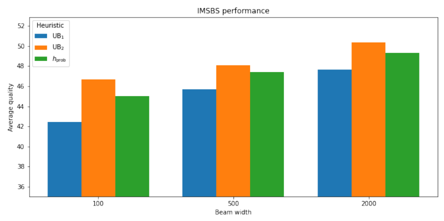
For the Bs approach, a high beam width $\beta = 10,000$ and $h = \mathrm{h}_{prob}$ is utilized as no significant increase in the solution quality of the Bs by further increasing the the size of beam, while h_{prob} provided a slightly better guidance from UB$_2$ but significantly better from UB$_1$; see Fig. 2.



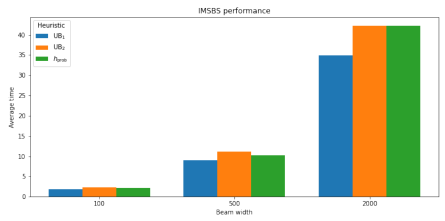
(a) Avg. quality over all benchmark suite Random.
(b) Avg. runtime over all benchmark suite Random.

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

After conducting a series of preliminary experiments, we agreed to fix the parameter $number\_sources\_from\_R = 10$ and $beam\_iters$ is set to 50, as high-quality results have been already achieved with this amount of iterations. With the same reason, root nodes that are kept in the global set of root nodes are prioritized according to $h' = \mathrm{UB}_2$, which allowed the IMSBS to performed consistently better when the UB_1. The remaining parameters of IMSBS approach were tuned utilizing the simple grid search strategy with the following values $h \in \{\mathrm{UB}_1, \mathrm{UB}_2, h_{prob}\}$ and $\beta \in \{100, 500, 2000\}$, see Fig. 3.



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



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

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

In order to ensure a comparable average runtime consumption of IMSBS and the baseline Bs approach, we decided to use the following configuration for the IMSBS in our experimental evaluations: $h = \mathrm{UB}_2$ and $\beta = 500$. As already said, these values were chosen to strike a balance between allowed time resources relative to the baseline BS execution times while keeping reasonable final solution quality of Imsbs. Similarly, due to the same reason, the Imsbs-greedy approach which has a fixed $\beta = 1$, we allowed a budget of 5000 beam search iterations ($beam\_iters = 5000$), while other parameters were fixed to those values as in the case of the Imsbs.

4.3 Numerical results for heuristic approaches

Numerical results for all three heuristic approaches are reported in Table 1, organized into two main parts. The first three columns describe the instance characteristics, namely the number of sequences (m), the sequence length (n), and the alphabet size ($|\Sigma|$). The remaining columns report the performance of 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 $\overline{t}[s]$ provided over 10 instances pera each group. This organization allows a direct comparison of solution quality and computational effort across algorithms for identical instance settings.

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	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 are drawn.

– *Solution quality comparison.* Imsbs approach produces the highest solution quality in 31 out of 32 instance groups, on average exceeding the baseline Bs approach by achieving a relative percentage improvement of about 50%. The baseline BS approach performs equally well or better in just 2 group. The weakest (but also th fastest) in average performing approach is the Imsbs-Greedy approach. It performs fairly (and clearly better from Bs) only for the cases with a larger number of input sequences $m \in \{10\}$ which indicates on how wrong can be a decision of starting the beam search from the root node $((1,1),0)$ due to many more gap constraints (m) to be fulfilled and that exploring other root nodes quite often can be beneficial in improving diversity of the search, this affecting the quality of final solutions.

- *Runtime comparison.* The Bs and the Imsbs approaches have comparable average runtimes. As expected, a significantly smaller average runtime is exhibited for the Imsbs-Greedy approach mainly due to inexpensive beam search iterations utilized ($\beta = 1$) in contrast to high-consuming beam search iterations in the other two approaches.
- *Other conclusions.* The behavior of Bs is explained by the fact that for small m, the choice of taking the root node $r = (1, \ldots, 1)$ is typically favorable, enabling BS to progress deeply and identify high-quality subsequences for higher beam widths. This is in particular the case of instances with $m = 2$ where the difference between solution quality produced by Bs and Imsbs are in favor of the latter approach, but not that dramatically. When comes to larger values of m, one may notice huge gap between the solutions of these two. Overall, IMSBS demonstrates considerably greater robustness and stability with respect to increased number of input sequences.

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

In this section, we compare the results of the literature approaches specialized for the fixed $m = 2$ case; the competitors are listed bellow and all are explained in detains in [10].

- The basic dynamic programming algorithm, denoted as Dp;
- Advanced dynamic programming approach, that uses an Incremental Suffix Maximum Queries (ISMQ) by utilizing `Col` and `All` matrices, to speed up the former approach, denoted as Dp-1;
- Advanced dynamic programming algorithm which Handling ISMQ by *dequeue* data structure for further speeding up, denoted as Dp-2; all three DP approaches are from [10]. Note that in the original paper, answering ISMQ with Union-find Operations is proposed. However, due to lack implementation details, we decided to make use of the dequeue data structure in our implementation.
- The integer linear programming method, denoted as Ilp, is proposed in this work, see Appendix A. It is motivated by the ILP model in the context of LCS problems [5].

Each algorithm was given half an hour (30 minutes) of execution time. The Ilp model was solved using the general-purpose solver Cplex version 22.1.

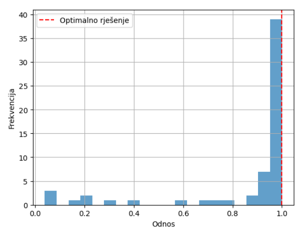
The results are provided in Table 2, which is organized as follows. The first three columns describe characteristics of the instance groups over which the results are averaged (10 instances in each group). The next four blocks report the results of the four algorithms: Dp, Dp-1, Dp-2, and Ilp, respectively.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

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

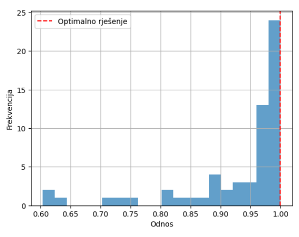
Inst.			DP		DP-1		DP-2		ILP	
m	n	$\|\Sigma\|$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$	$\overline{obj}$	$\bar{t}[s]$
2	50	2	38.1	0.01	38.1	**0.01**	38.1	0.01	38.1	79.42
2	50	4	30.3	**0.01**	30.3	0.02	30.3	0.02	30.3	18.0
2	100	2	77.4	0.1	77.4	**0.03**	77.4	0.047	–	–
2	100	4	62.3	0.07	62.3	**0.06**	62.3	0.09	0.00	1952.0
2	200	2	156.4	0.75	156.4	**0.128**	156.4	0.16	–	–
2	200	4	127.2	0.59	127.2	**0.251**	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, we can draw the following conclusions:

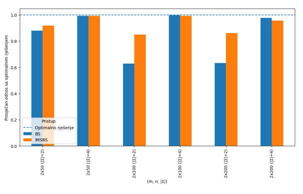
- All three dynamic programming approaches are extremely efficient and are capable of finding optimal solutions for all (80) problem instances.
- The runtimes required by the DP approaches to obtain optimal solutions are reasonably short, mostly remain bellow 10 second.
- The Ilp approach is able to solve only the smallest instances with $n = 50$, but requires execution times which are two orders of magnitude larger than those of dynamic programming. For the remaining instances, and especially for those with $n = 200$, the ILP approach is unable to solve even the initial (root) relaxations.



(a) BS approach



(b) IMSBS approach



(c) Qualities of BS vs. IMSBS w.r.t. optimal solutions (1.0) for instances with $m = 2$

Fig. 4: Number of instances for BS and IMSBS achieving a specific ratio between the quality of obtained (approximate) solutions and the quality of the known optimal solution for $m = 2$.

FiXme Note: TODO: came up to here!!

Concerning efficiency of Bs and Imsbs on those instances solved optimaly by dynamic programming (that is, the instances with $m = 2$), Figure 4 shows the number of instances (on y-axis) achieving specific ratios between the quality of solutions of Bs (a) or Imsbs (b) and the known optimal solutions (on x-axis). As noted, Imsbs for 50 (out of 60) instances an optimal solution, much more than the Bs approach. For the remaining 10 instances where Imsbs cold not reach an optimal solution, it achieved a solution with at least 80% of the quality of optimal solution, which showing its robustness.

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 and comparison of 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 on each other than distant ones. We designed a beam-search-based solution framework, beginning with a baseline beam search over a formally defined state-space graph, and then introduced an enhanced iterative multi-source beam search (IMSBS) that combines local exploration from promising root nodes with a global strategy for selecting these nodes based on previously collected search information. The experimental evaluation compared three heuristic algorithms under several heuristic functions, showing that IMSBS consistently produced the highest-quality solutions, often achieving over 90% of the optimal value 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 designed ILP model, confirming the advanced dynamic programming variant solved all instances in the shortest amount of time in contrast to the ILP unable to solve any larger instances. Future research directions include designing more sophisticated heuristics that directly incorporate gap constraints into scoring functions, optimizing IMSBS to reuse previously explored regions of the state space, generating realistic biological instances for a real-world case study, 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 from the European Regional Development Fund. Co-funded by European Union. Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or 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:

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

   The activation constraint is given by:

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

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:

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

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.