Algorithms and Complexity Group | Institute of Logic and Computation | TU Wien, Vienna, Austria

Technical Report AC-TR-2021-009

May 2021

Solving the Longest Common Subsequence Problem Concerning Non-uniform Distributions of Letters in Input Strings

Bojan Nikolic, Aleksandar Kartelj, Mako Djukanovic, Milana Grbic, Christian Blum, and Guenther Raidl

Submitted to the journal Applied Soft Computing

www.ac.tuwien.ac.at/tr

Article

Solving the Longest Common Subsequence Problem Concerning Non-uniform Distributions of Letters in Input Strings

Bojan Nikolic [1,‡], Aleksandar Kartelj [4,‡], Marko Djukanovic [1,2,‡,*], Milana Grbic [1,‡], Christian Blum [3,‡] and Günther Raidl [2,‡]

[1] Faculty of Natural Science and Mathematics: University of Banja Luka, Bosnia and Herzegovina; {bojan.nikolic | milana.grbic}@pmf.unibl.org
[2] Institute of Logic and Computation, TU Wien, Austria; {djukanovic | raidl}@ac.tuwien.ac.at
[3] Artificial Intelligence Research Institute (IIIA-CSIC), Campus UAB, Bellaterra, Spain; christian.blum@iiia.csic.es
[4] Faculty of Mathematics, University of Belgrade, Serbia; aleksandar.kartelj@gmail.com
* Correspondence: djukanovic@ac.tuwien.ac.at; Tel.: +387-65-699-683
‡ These authors contributed equally to this work.

Citation: Nikolic, B.; Kartelj, A.; Djukanovic, M.; Grbic, M; Blum, C.; Raidl, G. Solving the Longest Common Subsequence Problem on Non-uniform Distributions. *Mathematics* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

Abstract: The longest common subsequence (LCS) problem is a prominent $\mathcal{NP}$–hard optimization problem where, given an arbitrary set of input string, the aim is to find a longest subsequence which is common to all input strings. This problem has a variety of applications in bioinformatics, molecular biology, file plagiarism checking, among others. All previous approaches from the literature are dedicated to solving LCS instances sampled from uniform or near-to-uniform probability distributions of letters in the input strings. In this paper we introduce an approach that is able to effectively deal with more general cases, where the occurrance of letters in the input strings follows a non-uniform distribution such as, for example, a multinomial distribution. Texts in any spoken language, for example, are well approximated by multinomial distributions. The proposed approach makes use of beam search, guided by a novel heuristic function named GMPSUM. This heuristic synthesizes two complementary scores in form of a convex combination: the first one performs well in the uniform case and the second one works well in the non-uniform case. Furthermore, we introduce a time-restricted beam search algorithm that is able to adapt the beam size during the algorithm execution in order to achieve a desired target runtime. Apart from benchmark sets from the related literature, in which the distribution of letters is close to uniform, we introduce three new benchmark sets that differ in terms of their statistical properties. One of these benchmark sets concerns a case-study in the context of text analysis. We provide a comprehensive empirical evaluation in two distinctive settings: (1) short-time execution with fixed beam size in order to evaluate the guidance abilities of the compared search heuristics, and (2) long-time executions with fixed target duration times in order to obtain high-quality solutions. In both settings, the newly proposed approach performs comparably to state-of-the-art techniques in the context of close-to-random instances, and outperforms state-of-the-art approaches for non-uniform instances.

Keywords: Longest common subsequence problem; multi-nomial distribution; probability-based search guidance

1. Introduction

In the field of bioinformatics, strings are commonly used to model sequences such as DNA, RNA, and protein molecules or even time series. Strings represent fundamental data structures in many programming languages. Formally, a *string* s is a finite sequence of $|s|$ letters over (usually) a finite alphabet Σ. A *subsequence* of a string s is any sequence obtained by removing arbitrary letters from s. Similarities among several strings can be determined by considering common subsequences, which may serve for deriving relationships and possibly to destil different aspects of the of input strings, such as mutations. More specifically, one such measure of similarity can be defined as follows. Given a set of m input strings $S = \{s_1, \ldots, s_m\}$, the *longest common subsequence* (LCS) problem [1] aims at finding a subsequence of maximum length that is common for all strings from the set of input strings S. The length of the LCS for two or more input strings is a widely used measure in computational biology [2], file plagiarism check, data compression [3,4], text editing [5], detecting road itersections from GPS traces [6], file comparison (e.g., in the `Unix` command *diff*) [7] and revision control systems such as GIT. For a fixed m, polynomial algorithms based on dynamic programming (DP) are known [8] in the literature. These dynamic programming approaches run in $O(n^m)$ time, where n denotes the length of the longest input string. Unfortunately, these approaches become quickly impractical when m and n get large. For an arbitrary large number of input strings, the LCS problem is $\mathcal{NP}$-hard [1]. In practice, heuristic techniques are typically used for larger m and n. Constructive heuristics such as the Expansion algorithm and the Best-Next heuristic [9,10] appeared first in the literature to tackle the LCS problem. Significantly better solutions are obtained by more advanced metaheuristic approaches. Most of these are based on *Beam Search* (BS), see e.g., [11–15]. These approaches differ in various important aspects, which include the heuristic guidance, the branching scheme, and the filtering mechanisms.

Djukanovic et al. (2019) [16] proposed a generalized BS framework for the LCS problem with the purpose of unifying all previous BS–based approaches from the literature. By respective parametrization, each of the previously introduced BS-based approaches from the literature could be expressed, which also enabled a more direct comparison of all of them. Moreover, a heuristic guidance that approximates the expected length of an LCS on uniform random strings was proposed. This way, a new state-of-the-art BS variant that leads on most of the existing random and quasi-random benchmark instances from the literature was obtained.

Concerning exact approaches for the LCS problem, an integer linear programming model was considered in [17]. It turned out not to be competitive enough as it is was not applicable to most of the commonly used benchmark instances from the literature. This was primarily due to the model size – too many binary variables and a huge number of constraints are needed even for small-sized problem instances. Dynamic programming approaches also run out of memory already for small-to-middle sized benchmark instances or typically return only weak solutions, if any. Chen et al. (2016) [18] proposed a parallel FAST_LCS search algorithm that mightingated some of the runtime weaknesses. Wang et al. (2011) in [14] proposed another parallel algorithm called QUICK-DP, which is based on the dominant point approach and employs a quick divide-and-conquer technique to compute the dominant points. Li et al. (2016) in [19] suggested the TOP_MLCS algorithm, which is based on a directed acyclic layered-graph model (called irredundant common subsequence graph) and parallel topological sorting strategies used to filter out paths representing suboptimal solutions. Another parallel and space efficient algorithm based on a graph model, called the LEVELED-DAG, was introduced by Peng and Wang [20]. Recently Djukanovic et al. proposed an A* search that is able to outperform TOP_MLCS and previous exact approaches in terms of memory usage and the number of instances solved to optimality. Nevertheless, the applicability of this exact A* search is still limited to small-sized instances. In the same work, the A* search served as a basis for a hybrid anytime algorithm, which can be stopped at almost any time and then be expected to yield a reasonable heuristic solution. In this approach, classical A* search iterations are intertwined with iterations of *Anytime column search* [21].

The methods so-far proposed in the literature were primarily tested on independent random and quasi-random strings where the number or occurrences of letters in each string is similar for each letter. In fact, we are aware of just one benchmark set with different distributions (`BB`, see section 4), where the input strings are constructed in a way so that they exhibit high similarity, but still the letters' frequencies are similar. In practical applications this assumption of uniform or close-to-uniform distribution of letters does not need to hold. Some letters may occur substantially more frequently than others. For example, if we are concerned of finding motifs in sentences of any spoken language, each letter has its characteristic frequency [22]. Text in natural languages can be modeled by a multinomial distribution over the letters. The required level of model adaptation can vary depending on the distribution assumptions such as letter dependence of a particular language. Also, letter frequencies in a language can differ depending of text types (e.g., poetry, fiction, scientific documents, business documents). For example, it is interesting that the letter 'E' is the most frequent letter in English (12.702%) [22] and German (17.40%) [23], but only the second most common letter in Russian [24]. Moreover, letter 'N' is very frequent in German (9.78%), but not so common in English (6.749%) and Russian (6.8%).

Motivated by this considerations, we develop in the following a new BS-based algorithm which is able to more effectively tackle instances with different string distributions. The novel guidance heuristic applied at the core of this BS can be used as a credible and simplified replacement of the so far leading approximate expected length calculation. Additional advantages are that the novel heuristic is easier to implement than the approximate expected length calculation (which required a Taylor series expansion and a divide-and-conquer approach in an efficient implementation) and that there are no issues with numerical stability.

The main contributions of this article are as follows.

- We propose a novel search guidance for a BS which performs competitively on the standard LCS benchmark sets known from literature and in some cases even produces new state-of-the-art results.
- We introduce two new LCS benchmark sets based on multinomial distributions, whose main property is that letters occur with different frequencies. The proposed new BS variant excels on these instances in comparison to previous solution approaches.
- A new time-restricted BS version is described. It automatically adapts the beam width over BS levels w.r.t. given time restrictions such that the overall running time of BS approximately fits a desired target time limit. A tuning of the beam width to achieve comparable running times among different algorithms is hereby avoided.

In the following we introducing some commonly used notation before giving an overview on the remainder of this article.

1.1. Preliminaries

By S we always refer to the set of m input strings, i.e. $S = \{s_1, \ldots, s_m\}$, $m \geq 1$. The length of a string s is denoted by $|s|$, and its i-th letter, $i \in \{1, \ldots, |s|\}$, is referred to by $s[i]$. Let n refers to the length of a longest string and $n_{\min}$ to the length of a shortest string in S. A continuous subsequence (substring) of string s that starts with the letter at index i and ends with the letter at index j is denoted by $s[i, j]$; if $i > j$, this refers to the empty string ε. The number of occurrences of a letter $a \in \Sigma$ in string s is denoted by $|s|_a$. For a subset of the alphabet $A \subseteq \Sigma$, the number of appearances of each letter from A in s is denoted by $|s|_A$. For an m-dimensional integer vector $\vec{\theta} \in \mathbb{N}^m$ and the set of strings S, we define the set of suffix-strings $S[\vec{\theta}] = \{s_1[\theta_1, |s_1|], \ldots, s_m[\theta_n, |s_n|\}$, which induce a respective LCS subproblem. For each letter $a \in \Sigma$, the position of the first occurrence of a in $s_i[\vec{\theta}_i, |s_i|]$ is denoted by $\vec{\theta}_{i,a}$, $i = 1, \ldots, m$. Last but not least, if a string s is a subsequence of a given string r, we write $s \prec r$.

1.2. Overview

This article is organized as follow. Section 2 provides theoretical aspects concerning the calculation of the probability that a given string is a subsequence of a random string chosen from a multinomial distribution. Section 3 describes the BS framework for solving the LCS problem as well as the novel heuristic guidance. Moreover, the time-restricted BS variant is also proposed. In Section 4, a comprehensive experimental study and comparison is conducted. Section 5 extends the experiments by considering instances derived from a textual corpus. Finally, Section 6 draws conclusions and outlines interesting future work.

2. Theoretical Aspects of Different String Distributions

Most papers in literature are dedicated to the development and improvement of methods for finding an LCS of instances on strings that come from a uniform distribution. In our work, we propose new methods for the more general case where strings are assumed to come from a multinomial distribution $MN(p_1, \ldots, p_\eta)$ of strings. More precisely, for an alphabet $\Sigma = \{a_1, \ldots, a_\eta\}$, $\eta > 1$, as sample space for the letter of the strings, a multinomial distribution $MN(p_1, \ldots, p_\eta)$ is determined by specifying a (real) number p_i for each letter a_i such that p_i represents the probability of seeing letter a_i and $\sum_{i=1}^{\eta} p_i = 1$. Note that the uniform distribution is a special case of the multinomial distribution $MN(p_1, \ldots, p_\eta)$, with $p_1 = \ldots = p_n = \dfrac{1}{\eta}$.

Assuming that the selection of each letter in a string is independent, each string can be considered a random vector composed of independent random variables, resulting that its probability distribution is being completely determined by a given multinomial distribution. By a random string in this paper, we refer to a string whose letters are chosen randomly in accordance with the given multinomial distribution.

Let r be a given string. We now aim at determining the probability that a random string s, chosen from the same multinomial distribution $MN(p_1, \ldots, p_\eta)$ as string r, is a subsequence of the string r. We denote this probability by $P(s \prec r)$. In the next theorem, we propose a new recurrence relation to calculate this probability.

Theorem 1. *Let r be a given string and s be a random string chosen from the same multinomial distribution. Then,*

$$P(s \prec r) = \begin{cases} 1, & \text{if } |s| = 0; \\ 0, & \text{if } |s| > |r|; \\ P\big(s[1] = r[1]\big) \cdot P\big(s[2, |s|] \prec r[2, |r|]\big) + & \\ \qquad\qquad P\big(s[1] \neq r[1]\big) \cdot P\big(s \prec r[2, |r|]\big), & \text{otherwise.} \end{cases} \tag{1}$$

Proof. It is clear by the definition of a subsequence that the empty string is a subsequence of every string and that a string cannot be a subsequence of a shorter one. Therefore, the cases $|s| = 0$ and $|s| > |r|$ are trivial. In the remaining case ($1 \leqslant |s| \leqslant |r|$),

$$P(s \prec r) = P\big(s[1] = r[1]\big) \cdot P\big(s[2, |s|] \prec r[2, |r|]\big) + P\big(s[1] \neq r[1]\big) \cdot P\big(s \prec r[2, |r|]\big)$$

follows from the law of total probability. □

The probability $P(s \prec r)$ in recurrence relation (1) is dependent not only on the length of string r, but also on the letter distribution of this string. Therefore, it is hard to come up with a closed-form expression for the general case of a multinomial distribution $MN(p_1, \ldots, p_\eta)$. One way to deal with this problem is to consider some special cases of the multinomial distribution, for which closed-form expressions may be obtained.

2.1. Multinomial Distribution – Special Case 1: Uniform Distribution

The most frequently used form of the multinomial distribution considered in the literature is the uniform distribution. Since in this case every letter has the same occurrence probability, probability $P(s \prec r)$ in the recurrence relation (1) depends only on the lengths $k = |s|$ and $l = |r|$ and can be simpler written as $P(k, l)$. This case is covered by Mousavi and Tabataba in [12], where the recurrence relation (1) is reduced as follows:

$$P(k,l) = \begin{cases} 1, & \text{if } k = 0; \\ 0, & \text{if } k > l; \\ \frac{1}{\eta} \cdot P(k-1, l-1) + \frac{\eta-1}{\eta} \cdot P(k, l-1), & \text{otherwise.} \end{cases} \quad (2)$$

Probabilities $P(k, l)$ can be calculated using dynamic programming as described by Mousavi and Tabataba in [12].

2.2. Multinomial Distribution – Special Case 2: Single Letter Exception

Let one letter $a_j \in \Sigma$ have occurrence probability $p \in (0, 1)$, $p \neq 1/\eta$ and each other letter a_i, $i \in \{1, \ldots, \eta\} \setminus \{j\}$ have occurrence probability $(1 - p)/(\eta - 1)$. For this multinomial distribution, recurrence relation (1) reduces to:

$$P(s \prec r) = \begin{cases} 1, & \text{if } |s| = 0; \\ 0, & \text{if } |s| > |r|; \\ q \cdot P\big(s[2, |s|] \prec r[2, |r|]\big) + (1-q) \cdot P\big(s \prec r[2, |r|]\big), & \text{otherwise.} \end{cases} \quad (3)$$

where

$$q := \begin{cases} p, & \text{if } r[1] = a_j; \\ \frac{1-p}{\eta-1}, & \text{otherwise.} \end{cases}$$

Note that, besides lengths $|s|$ and $|r|$, (3) depends only on whether or not a letter in the string r is equal to a_j.

2.3. Multinomial Distribution – Special Case 3: Two Sets of Letters

We now further generalize the previous case. Let $\{\Sigma_1, \Sigma_2\}$ be a partitioning of the alphabet Σ, i.e., let $\Sigma_1, \Sigma_2 \subseteq \Sigma$ be nonempty sets such that $\Sigma = \Sigma_1 \cup \Sigma_2$ and $\Sigma_1 \cap \Sigma_2 = \emptyset$. Let us assume that every letter in Σ_1 has the same occurrence probability and also, that every letter in Σ_2 has the same occurrence probability. We define

$$p_i := \begin{cases} \frac{p}{|\Sigma_1|}, & \text{if } a_i \in \Sigma_1; \\ \frac{1-p}{\eta-|\Sigma_1|}, & \text{if } a_i \in \Sigma_2, \end{cases}$$

where $p \in (0, 1)$ is the probability mass assigned to the set Σ_1. For this multinomial distribution, recurrence relation (1) reduces to

$$P(s \prec r) = \begin{cases} 1, & \text{if } |s| = 0; \\ 0, & \text{if } |s| > |r|; \\ q \cdot P\big(s[2, |s|] \prec r[2, |r|]\big) + (1-q) \cdot P\big(s \prec r[2, |r|]\big), & \text{otherwise.} \end{cases} \quad (4)$$

where

$$q := \begin{cases} \frac{p}{|\Sigma_1|}, & \text{if } r[1] \in \Sigma_1; \\ \frac{1-p}{\eta-|\Sigma_1|}, & \text{if } r[1] \in \Sigma_2. \end{cases}$$

This probability therefore depends on whether or not a letter in r belongs to the set Σ_1 or not.

2.4. The Case of Independent Random Strings

Another approach of calculating the probability that a string s is a subsequence of a string r is based on the assumption that both s and r are random strings chosen from the same multinomial distribution and are independent as a random vectors. Using this setup, we established a recurrence relation for calculating probability $P(s \prec r)$.

Theorem 2. *Let r and s be random independent strings chosen from the same multinomial distribution $MN(p_1, \ldots, p_\eta)$. Then*

$$P(s \prec r) = \begin{cases} 1, & \text{if } |s| = 0; \\ 0, & \text{if } |s| > |r|; \\ \left(\sum_{i=1}^{\eta} p_i^2\right) \cdot P\left(s[2,|s|] \prec r[2,|r|]\right) + \\ \qquad \left(1 - \sum_{i=1}^{\eta} p_i^2\right) \cdot P\left(s \prec r[2,|r|]\right), & \text{otherwise.} \end{cases} \quad (5)$$

Proof. The first two cases are trivial, so it remains to show the last case. Using the law of total probability, we obtain

$$P(s \prec r) = P\big(s[1] = r[1]\big) \cdot P\big(s[2,|s|] \prec r[2,|r|]\big) + P\big(s[1] \neq r[1]\big) \cdot P\big(s \prec r[2,|r|]\big).$$

Probability $P\big(s[1] = r[1]\big)$ can be calculated with another application of the law of total probability, using the assumption that random strings s and r are mutually independent:

$$\begin{aligned} P\big(s[1] = r[1]\big) &= \sum_{i=1}^{\eta} P\big(r[1] = a_i\big) \cdot P\big(s[1] = r[1] \mid r[1] = a_i\big) \\ &= \sum_{i=1}^{\eta} P\big(r[1] = a_i\big) \cdot P\big(s[1] = a_i\big) = \sum_{i=1}^{\eta} p_i^2. \quad \square \end{aligned}$$

Except for the obvious dependency on the multinomial distribution $MN(p_1, \ldots, p_\eta)$, probability $P(s \prec r)$ is determined by the lengths of strings s and r, only. Therefore, as in the case of the uniform distribution, we can abbreviate this probability with $P(k, l)$, where $k = |s|$ and $l = |r|$. This allows us to pre-compute a probability matrix for all relevant values of k and l by means of dynamic programming.

3. Beam Search for Multinomially Distributed LCS Instances

Beam search (BS) is a well-known search heuristic widely applied to many problems from various research fields, such as scheduling [25], speech recognition [26], machine learning tasks [27], packing problems [28], etc. It is a reduced version of breadth-first-search (BFS), where instead of expanding all not-yet-expanded nodes from the same level, only up to a specific number $\beta > 0$ of nodes appearing most promising are selected and considered for expansions. In this way, BS keeps the search tree polynomial in size. The selection of the up to β nodes for further expansion is made according to a problem-specific heuristic guidance function h. The effectivity of the search thus substantially depends on this function. More specifically, BS works as follows. First, an initial beam B is set up with a root node r representing an initial state, in case of the LCS problem the empty partial solution. At each major iteration, all nodes from beam B are expanded in all possible ways by considering all feasible actions. The so obtained child nodes are kept in the set of extensions V_{ext}. Note that for some problems efficient filtering techniques can be applied to discard nodes from V_{ext} that are dominated by other nodes, i.e., nodes that cannot yield better solutions. It is controlled by an internal parameter k_{filter}. This (possibly filtered) set of extensions is then sorted according to the nodes' values obtained from the guidance heuristic h, and the top β nodes (or less if V_{ext} is smaller) then form the beam B of the next level. The whole process is repeated level-by-level until B becomes empty. In general, to solve a combinatorial optimization problem, information about the longest (or shortest) path from the root node to a feasible goal node is kept to finally return a solution that maximizes or minimizes the problem's objective function. The pseudocode of such a general BS is given in Algorithm 1.

Algorithm 1 Beam Search.

Input: A problem instance, heuristic h, $\beta > 0$, k_{filter}
Output: A heuristic solution
$B \leftarrow \{r\}$
while $B \neq \emptyset$ **do**
    $V_{\text{ext}} \leftarrow \emptyset$
    for $v \in B$ **do**
        if v is a goal node **then**
            if node represents new best solution, store it
        else
            add not-yet-visited child nodes of v to V_{ext}
        end if
    end for
    if $k_{\text{filter}} \geq 0$ **then**
        $V_{\text{ext}} \leftarrow \texttt{Filter}(V_{\text{ext}}, k_{\text{filter}})$ // optionally filter dominated nodes
    end if
    $B \leftarrow \texttt{SelectBetaBest}(V_{\text{ext}}, \beta, h)$
end while
return best found solution

3.1. State Graph for the LCS Problem

The state graph for the LCS problem that is used by all BS variants is already well known in the literature, see for example [16,29]. It is defined as a directed acyclic graph $G = (V, A)$, where a node $v = (\vec{\theta}^v, l^v) \in V$ represents the set of partial solutions which

1. have the same length l^v;
2. induce the same subproblem denoted by $S[\vec{\theta}^v]$ w.r.t. the position vector $\vec{\theta}^v$.

We say that a partial solution s *induces* a subproblem $S[\vec{\theta}^v]$ iff $s_i[1, \vec{\theta}^v_i - 1]$ is the smallest prefix of s_i among all prefixes that has s as a subsequence.

An arc $a = (v_1, v_2) \in A$ exists between two nodes $v_1 \neq v_2 \in V$ and carries label $\ell(a) \in \Sigma$, iff

1. $l^{v_2} = l^{v_1} + 1$;
2. the partial solution that induces v_2 is obtained by appending $\ell(a)$ to the partial solution inducing v_1.

The root node $r = \big((1, \ldots, 1), 0\big)$ of G refers to the original LCS problem on input string set S and can be said to be induced by the empty partial solution ε.

For deriving the successor nodes of a node $v \in V$, we first determine the subset $\Sigma_v \subseteq \Sigma$ of letter that feasibly extend the partial solutions represented by v. The candidates for letter $a \in \Sigma_v$ are therefore all letter $a \in \Sigma$ that appear at least once in each string in the subproblem given by strings $S[\vec{\theta}^v]$. This set Σ_v may be reduced by determining and discarding dominated letters. We say that letter $a \in \Sigma_v$ *dominates* letter $b \in \Sigma_v$ iff

$$\vec{\theta}^v_{i,a} \leq \vec{\theta}^v_{i,b} \quad \forall i \in \{1, \ldots, m\}. \tag{6}$$

Dominated letters can be safely omitted since they lead to suboptimal solutions. Let $\Sigma^{\text{nd}}_v \subseteq \Sigma_v$ be the set of feasible and non-dominated letters. For each letter $a \in \Sigma^{\text{nd}}_v$, graph G contains a successor node $v' = (\vec{\theta}^{v'}, l^v + 1)$ of v, where $\vec{\theta}^{v'}_i = \vec{\theta}^{v'}_{i,a} + 1$, $i \in \{1, \ldots, m\}$ (remember that $\vec{\theta}^{v'}_{i,a}$ denotes the position of the first appearance of letter a in string s_i from position $\vec{\theta}^{v'}_i$ onward). A node v that has no successor node, i.e., when $\Sigma^{\text{nd}}_v = \emptyset$, is called a *non-extensible* node, or *goal* node. Among all goal nodes v we are looking for one representing a longest solution string, i.e., a goal node with largest l^v. Note that any path from the root node r to any node in $v \in V$ represents the feasible partial solution obtained by collecting and concatenating the labels of the traversed arcs. Thus, it is not necessary to store actual partial solutions s in the nodes. In the graph G, any path from

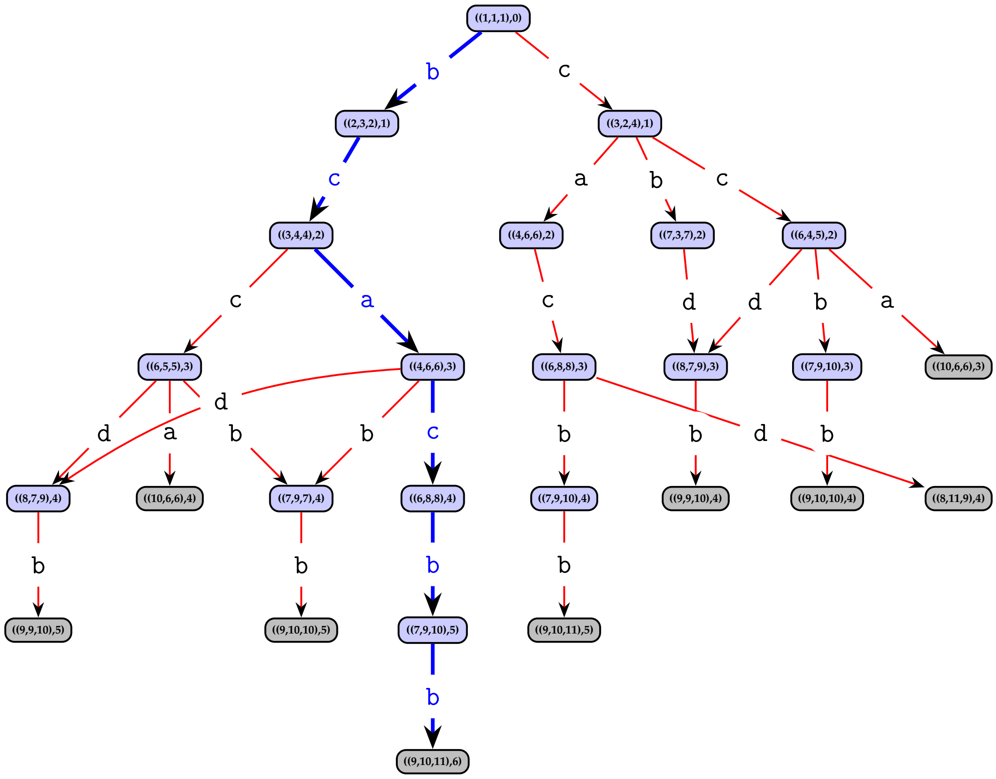


Figure 1. State graph for the LCS problem instance on strings $\{s_1 = \texttt{bcaacbdba}, s_2 = \texttt{cbccadcbbd}, s_3 = \texttt{bbccabcdbba}\}$ and alphabet $\Sigma = \{\texttt{a},\texttt{b},\texttt{c},\texttt{d}\}$). Light-gray nodes are non-extensible goal nodes. The longest path in this state graph is shown in blue, leads from the root to node $((1,10,11),6)$ and corresponds to the solution $s = \texttt{bcacbb}$, having length six.

root r to a non-extensible node represents a common, non-extensible subsequence of S. Any longest path from r to a goal node represents an optimal solution to problem instance S. As an example for a full state graph of an instance, see Figure 1.

Still we have to explain the filtering of dominated nodes from the set V_{ext}, i.e., procedure `Filter` in Algorithm 1. We adopt the efficient *restricted filtering* proposed in [13], which is parameterized by a filter size $k_{\text{filter}} > 0$. The idea is to select only the (up to) k_{filter} best nodes from V_{ext} and to check the dominance relation (6) for this subset of nodes in combination with all other nodes in V_{ext}. If the relation is positively evaluated, the dominated node is removed from V_{ext}. Note that parameter settings $k_{\text{filter}} = 0$ and $k_{\text{filter}} = |V_{\text{ext}}|$ represent the two extreme cases of no filtering and full filtering, respectively. A filter size of $0 < k_{\text{filter}} < |V_{\text{ext}}|$ may be meaningful as full filtering may be to costly in terms of running time for larger beam widths.

3.2. Novel Heuristic Guidance

We now present a new heuristic for evaluating nodes in the BS in order to rank them and to select the beam of the next level. This heuristic, called GMPSUM, in particular aims at unbalanced instances and is a convex combination of the following two scores.

- The GM score is based on the geometric mean and geometric standard deviation of the letters' occurrences across all input strings of the respective subproblem. It is calculated on a per letter basis aggregated into a single numeric value;
- The PSUM score is based on the previously introduced probability matrix $P(k,l)$ for the arbitrary unbalanced multinomial distribution case, see recurrence relation (5), or in the special cases, any of recurrence relations (3)–(4) might be used instead.

More specifically, for a given node v, the GM score is calculated as

$$\textsc{Gm}(v) = \textsc{Gm}(S[\vec{\theta}^v]) = \sum_{a\in\Sigma} \frac{\mu_g(C_a(S[\vec{\theta}^v]))}{\sigma_g(C_a(S[\vec{\theta}^v]))} \cdot \frac{\min_{i=1,\ldots,m} C_a(S[\vec{\theta}^v])_i}{\mathrm{UB}_1(v)} \tag{7}$$

where

$$C_a(S[\vec{\theta}^v]) = (s_1[\vec{\theta}^v_1, |s_1|]_a, \ldots, s_m[\vec{\theta}^v_m, |s_m|]_a)$$

is the vector indicating for each remaining string of the respective subproblem the number of occurrences of letter $a \in \Sigma$, while $\mu_g(\cdot)$ and $\sigma_g(\cdot)$ denote the geometric mean and geometric standard deviation, respectively, which are calculated for $\vec{x} = (x_1, \ldots, x_m) \in \mathbb{R}^m$ by

$$\mu_g(\vec{x}) = \sqrt[m]{x_1 \cdot \ldots \cdot x_m},$$

$$\sigma_g(\vec{x}) = e^{\sqrt{\frac{\sum_{i=1}^{m}\left(\ln \frac{x_i}{\mu_g(\vec{x})}\right)^2}{m}}}.$$

Function $\mathrm{UB}_1(v)$ in expression (7) is the known upper bound on the length of an LCS for the subproblem represented by node v from [30] and calculated as

$$\mathrm{UB}_1(v) = \sum_{a\in\Sigma} \min_{i=1,\ldots,m} C_a(S[\vec{\theta}^v])_i.$$

Overall, the Gm score is thus a weighted average of the adjusted geometric means $(\mu_g(\cdot)/\sigma_g(\cdot))$ of the number of letter occurrences, and the weight of each letter is determined by normalizing the minimal number of the letter occurrences across all strings with the sum of minimal number occurrences across all letters. The motivation behind this calculation is three-fold:

1. Letters with higher average numbers of occurrences across the strings will increase the chance of finding a longer common subsequence (composed of these letters).
2. Higher deviations around the mean naturally reduce this chance.
3. The minimal numbers of occurrences of a letter across all input strings is an upper bound on the length of common subsequences that can be formed by this single letter. Therefore, by normalizing it with the sum of all minimal letter occurrences, an impact of each letter in the overall summation is quantified.

The Gm score is relevant if its underlying sampling geometric mean and standard deviation are based on a sample of sufficient size. In all our experiments, the minimal number of input strings is therefore ten. Working on samples of smaller sizes would make the Gm score likely not that useful.

In addition to the Gm score, we consider the Psum score that is calculated by

$$\textsc{Psum}(v) = \textsc{Psum}(S[\vec{\theta}^v]) = \sum_{k=1}^{l_{\max}(v)} \prod_{i=1}^{m} P(k, |s_i| - \vec{\theta}^v_i + 1) \tag{8}$$

where

$$l_{\max}(v) = \min_{i=1,\ldots,m} (|s_i| - \vec{\theta}^v_i + 1).$$

Unlike the Gm score that considers mostly general aspects of an underlying probability distribution, Psum better captures more specific relations among input strings. It represents the sum of probabilities that a string of length k will be a common subsequence for all remaining input strings relevant for further extensions. Index k goes from one to $l_{\max}(v)$, i.e., the length of the shortest possible non-empty subsequence up to the length of the longest possible one, which corresponds to the size of the shortest input string residual. The motivation behind using a simple (non-weighted) summation across all potential subsequence lengths is three-fold:

1. It is not known in advance the exact length of the resulting subsequence. Note that in the case of the HP heuristic proposed in [12], the authors heuristically determine an appropriate value of k for each level in the BS.
2. The summation across all k provides insight on the overall potential of node v – approximating the integral on the respective continuous function. Note that it is not required for this measure to have an interpretation in absolute terms since throughout the BS it is used strictly to compare different alternative extensions on the same level of the BS tree.
3. A more sophisticated approach that assigns different weights to the different k values would impose the challenge of deciding these specific weights. This would bring us back to the difficult task of an expected length prediction – which would be particularly hard when considering now the arbitrary multinomial distribution.

Finally, the total GMPSUM score is calculated by the linear combination

$$\mathrm{GMPSUM}(v, \lambda) = \lambda \cdot \mathrm{GM}(v) + (1 - \lambda) \cdot \mathrm{PSUM}(v), \tag{9}$$

where $\lambda \in [0, 1]$ is a strategy parameter. Based on an empirical study with different benchmark instances and values for parameter λ, we came up with the following rules of thumb to select λ.

1. Since GM and PSUM have complementary focus, i.e., they capture and award (or implicitly penalize) different aspects of the extension potential, their combined usage is indeed meaningful in most cases, i.e., $0 < \lambda < 1$.
2. GM tends to be a better indicator when instances are more regular, i.e., when each input string better fits the overall string distribution.
3. PSUM tends to perform better when instances are less regular, i.e., when input strings are more dispersed around the overall string distribution.

Regarding the computational costs of the GMPSUM calculation, the GM score calculation requires $O(|\Sigma| \cdot m)$ time. This can be concluded from (7) where the most expensive part is the iteration through all letters from Σ and finding the minimal number of the letter occurrences across all m input strings ($\mu_g(\cdot)$ and $\sigma_g(\cdot)$ have the same time complexity). Note that the number of occurrences of each letter across all possible suffixes of all m input strings positions is calculated in advance, before starting the beam search, and stored in an appropriate three-dimensional array, see [29]. The worst-case computational complexity of this step is $O(|\Sigma| \cdot m \cdot n_{\max})$. This is because the number of occurrences of a given letter across all positions inside the given input string can be determined in a single linear pass. Since this is done only at the start and the expected number of GM calls is much higher than $n_{\max}$, this up-front calculation can be neglected in the overall computational complexity The PSUM score given by (8) takes $O(n_{min} \cdot m)$ time to be calculated due to a definition of $l_{max}(\cdot)$. Similarly as in GM, the calculation of matrix P is performed in pre-processing – its computational complexity corresponds to the number of entries, i.e., $O(n_{max} \cdot n_{max})$, see (5).

Finally, the total computational complexity of GMPSUM can be concluded to be $O((|\Sigma| + n_{min}) \cdot m)$. The total computational complexity of the beam search is therefore a product of the number of calls of GMPSUM $O(n_{min} \cdot \beta \cdot |\Sigma|)$ and the time complexity of GMPSUM. Note that the number of GMPSUM calls equals the number of nodes created within a BS run. Since the LCS length, i.e. the number of BS levels, is unknown, we use here n_{min} as upper bound. In overall, the BS guided by GMPSUM runs in $O(n_{min} \cdot \beta \cdot |\Sigma| \cdot m \cdot (|\Sigma| + n_{min}))$ time if no filtering is performed. In case of filtering, at each level of the BS, $O(\beta \cdot k_{\text{filter}} \cdot m)$ time is required, which gives $O(n_{\min} \cdot \beta \cdot k_{\text{filter}} \cdot m)$ total time for executing the filtering within the BS. According to this, the BS guided by GMPSUM and utilizing (restricted) filtering requires $O(n_{\min} \cdot \beta \cdot m \cdot (k_{\text{filter}} + |\Sigma|^2 + |\Sigma| \cdot n_{\min}))$ time.

3.3. A Time-Restricted BS

In this section we extend the basic BS from Algorithm 1 to a time-restricted beam search (TRBS). This BS variant is motivated by the desire to compare different algorithms with the same time-limit. The core idea we apply is to dynamically adapt the beam width in dependence of the progress over the levels.

Similarly to the standard BS from Algorithm 1, TRBS is parameterized with the problem instance to solve, the guidance heuristic h, and the filtering parameter k_{filter}. Moreover, what was previously the constant beam width β now becomes only the initial value. The goal is to achieve a runtime that comes close to a target time $t_{\max}$ now additionally specified as input. At the end of each major iteration, i.e., level, if $t_{\max} < +\infty$, i.e., the time limit is actually enabled, the beam width for the next level is determined as follows.

1. Let t_{iter} be the time required for the current iteration.
2. We estimate the remaining number of major iterations (levels) by taking the maximum of lower bounds for the subinstances induced by the nodes in V_{ext}. More specifically,
$$LB_{\max}(V_{\text{ext}}) = \max_{(v,a)\in V_{\text{ext}}\times\Sigma} \ \min_{i=1,\ldots,m} C_a(S[\vec{\theta}^v])_i. \tag{10}$$
Thus, for each node $v \in V_{\text{ext}}$ and each letter a we consider the minimal number of occurrences of the letter across all string suffixes $S[\vec{\theta}^v]$ and select the one that is maximal. In other words, this LCS lower bound is based on considering all common subsequences in which a single letter is repeated as often as possible. In the literature, this procedure is known under the name *Long-run* [31] and provides a $|\Sigma|$–approximation.
3. Let t_{rem} be the actual time still remaining in order to finish at time $t_{\max}$.
4. Let $\overline{t_{\text{rem}}} = t_{\text{iter}} \cdot LB_{\max}(V_{\text{ext}})$ be the expected remaining time when we would continue with the current beam width and the time spent at each level would stay the same as it was measured for the current level.
5. Depending on the discrepancy of the actual and expected remaining time, we possibly increase or decrease the beam width for the next level:

$$\beta \leftarrow \begin{cases} \lfloor \beta \cdot 1.2 \rfloor & \text{if } t_{\text{rem}}/\overline{t_{\text{rem}}} > 1.1; \\ \min(100, \lfloor \beta/1.2 \rfloor) & \text{if } t_{\text{rem}}/\overline{t_{\text{rem}}} < 0.9; \\ \beta & \text{otherwise.} \end{cases} \tag{11}$$

In this adaptive scheme, the thresholds for the discrepancy to increase or decrease the beam width, as well as the factor by which the beam width is modified, were determined empirically. Note that there might be better estimates of the LCS length than $LB_{\max}$, however, this estimate is inexpensive to obtain, and even if it underestimate or overestimate the LCS length in early phases, gradually, it converges toward the actual LCS length as the algorithm progresses. This allows TRBS to smoothly adapt its expected remaining runtime to the desired one. Note that we only adapt the beam width and not set it completely anew based on the runtime measured for the current level in order to avoid too erratic changes of the beam width in case of a larger variance of the level's runtimes. Based on preliminary experiments, we conclude that the proposed approach in general works well in achieving the desired time limit, while changing β not dramatically up and down in the course of a whole run. But of course, how close the time limit is met, depends on the actual length of the LCS. For small solutions strings, the approach has less opportunities to adjust β and then tends to overestimate the remaining time, thus, utilizing less time than desired.

4. Experimental Results

In this section we evaluate our algorithms and compare them with the state-of-the-art algorithms from the literature. The proposed algorithms are implemented in C# and executed on machines with Intel i9-9900KF CPUs with @ 3.6GHz and 64 Gb of RAM under Microsoft Windows 10 Pro OS. Each experiment was performed in single-threaded mode. We have conducted two types of experiments:

- **Short runs:** these are limited-time scenarios—that is, BS configurations with $\beta = 600$ are used—executed in order to evaluate the quality of the guidance of each of the heuristics towards promising regions of the search space.
- **Long runs:** these are fixed-duration scenarios (900 seconds) in which we compare the time-restricted BS guided by the GMPSUM heuristic with the state-of-the-art results from the literature. The purpose of these experiments is the identification of new state-of-the-art solutions, if any.

4.1. Benchmark sets

All relevant benchmark sets from the literature were considered in our experiments:

- Benchmark sets RAT, VIRUS and RANDOM, each one consisting of 20 single instances, are well known from the related literature [32]. The first two sets are biologically motivated, originating from the NCBI database. In the case of the third set, instances were randomly generated. The input strings in these sets are 600 characters long. Moreover, they contain instances based on alphabets of size 4 and 20.
- Benchmark set ES, introduced in [33], consists of randomly distributed input strings whose length varies from 1000 to 5000, while alphabet sizes range from 2 to 100. This set consists of 12 groups of instances.
- Benchmark set BB, introduced in [34], is different to the others, because the input strings of each instance are generated in a way that there is a high similarity between them. For this purpose, first, a randomly generated base string was generated. Second, all input strings were generated based on the base string by probabilistically introducing small mutations such as delete/update operations of each letter. This set consists of eight groups (each one containing 10 single instances).
- Benchmark set BACTERIA, introduced in [35], is a real-world benchmark set used in the context of the constrained longest common subsequence problem. We make use of these instances by simply ignoring all pattern strings (constraints). This set consists of 35 single instances.
- Finally, we introduce two new sets of instances:
  - The input strings of the instances of benchmark set POLY are generated in a way such that the number of occurrences of each letter in each input string are determined by a multinomial distribution with known probabilities $p_1, \ldots, p_\eta > 0$, such that $\sum_i p_i = 1$; see [36] for how to sample such distributions. More specifically, we used the multinominal distribution with $p_i = \frac{1}{2^i}, i = 1, \ldots, |\Sigma| - 1$ and $p_\eta = 1 - \sum_{i=1}^{|\Sigma|-1} \frac{1}{2^i}$ for generating the input strings. The number of the occurrences of different letters is very much unbalanced in the obtained input strings. This set consists 10 instances for each combination of the input string length $n \in \{100, 500, 1000\}$ and the number of input strings $m \in \{10, 50\}$, which makes a total of 60 problem instances.
  - Benchmark set ABSTRACT, which will be introduced in Section 5, is a real-world benchmark set whose input strings are characterized by close-to-polynomial distributions of the different letters. The input strings originate from abstracts of scientific papers written in English.

4.2. Considered algorithms

All considered algorithms make use of the state-of-the-art BS component. In order to test the quality of the newly proposed GMPSUM heuristic for the evaluation of the partial solutions at each step of BS, we compare to the other heuristic functions that were

Table 1: Short-run results summary.

Benchmark set		BS-Ex			BS-Pow			BS-Hp			BS-Gmpsum		
Name	#	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$
Random	20	108.9	16	2.7	108.1	6	1.4	108.15	6	1.1	108.95	16	6.7
Rat	20	102.8	13	2.6	101.6	4	1.2	100.95	2	0.9	102.9	14	5.5
Virus	20	115.85	11	2.6	114.1	6	1.5	115.35	6	1.1	116.3	17	7.4
Bb	8	407.13	2	8.5	430.13	6	6.3	422.94	4	3.6	424.86	5	26.9
Es	12	242.18	8	23	241.51	0	15.6	241.14	0	13.8	242.12	4	118.8
Poly	6	232.67	0	5.6	232.27	0	3.3	231.53	0	2.7	233.02	6	6.7
Bacteria	35	809.97	12	14.7	814.86	15	8.2	830.69	22	7.9	832.09	18	29.3
All	121		62			37			40			80	

proposed for this purpose in the literature: Ex [16], Pow [13], and Hp [12]. The four resulting BS variants are labeled BS-Gmpsum, BS-Ex, BS-Pow, and BS-Hp, respectively. These four BS variants were applied with the same parameter settings ($\beta = 600$ and $k_{\text{filter}} = 100$) in the short-run scenario in order to ensure that all of them use the same amount of resources.

In the long-run scenario, we tested the proposed time-restricted BS (TRBS) guided by the novel Gmpsum heuristic, which is henceforth labeled as TRBS-Gmpsum. Our algorithm was compared to the current state-of-the-art approach from the literature: A*+ACS [29]. These two algorithms were compared in the following way:

- Concerning A*+ACS, the results for benchmark sets Random, Virus, Rat, Es and Bb were taken from the original paper [29]. They were obtained with a computation time limit of 900 seconds per run. For the new benchmark sets—that is, Poly and Bacteria—we applied the original implementation of A*+ACS with a time limit of 900 seconds on the above-mentioned machine.
- TRBS-Gmpsum was applied with a computation time limit of 600 seconds per run to all instances of benchmark sets Random, Virus, Rat, Es and Bb. Note that we reduced the computation time limit used in [29] by 50% because the CPU of our computer is faster than the one used in [29]. In contrast, the time limit for the new instances was set to 900 seconds. Regarding restricted-filtering, the same setting ($k_{\text{filter}} = 100$) as for the short-run experiments was used.

Regarding Gmpsum parameter λ, we performed short-run evaluations across a discrete set of possible values: $\lambda \in \{0, 0.25, 0.5, 0.75, 1\}$. The conclusion was that the best performing values are $\lambda = 0$ for Bb, $\lambda = 0.5$ for Virus and Bacteria, $\lambda = 0.75$ for Random, Rat and Poly, and $\lambda = 1$ for ES. The same settings for λ were used in the context of the long-run experiments.

4.3. Summary of the results

Before studying the results for each benchmark set in detail, we present a summary of the results in order to provide the reader with the broad picture of the comparison. More specifically, the results of the short-run scenarios are summarized in Table 1, while the ones for the long-run scenarios are given in Table 2. Table 1 displays the results in a way such that each line corresponds to a single benchmark set. The meaning of columns is as follows: the first column contains the name of the benchmark set, while the second column provides the number of instances—respectively, instance groups—in the set. Then there are four blocks of columns, one for each considered BS variant. The first column of each block shows the obtained average solution quality ($\overline{|s|}$) over all instances of the benchmark set. The second column indicates the number of instances—respectively, instance groups—for which the respective BS variant achieves the best result (#b.). Finally, the third column provides the average running time ($\bar{t}$) in seconds over all instances of the considered benchmark set.

The following conclusions can be drawn:

Table 2: Long-run results summary.

Benchmark set		A*+ACS		TRBS-GMPSUM 600s/900s	
Name	#	$\overline{\lvert s\rvert}$	#b.	$\overline{\lvert s\rvert}$	#b.
Random	20	109.9	20	109.7	16
RAT	20	104.3	17	104.4	18
VIRUS	20	117.0	14	117.3	19
BB	8	412.81	3	430.28	6
ES	12	243.82	9	243.73	4
Poly	6	234.13	4	234.23	5
Bacteria	35	829.26	10	862.63	33
All	121		77		101

- Concerning the fully random benchmark sets RANDOM and ES in which input strings were generated uniformly at random and are independent, it was already well-known before that the heuristic guidance EX performs strongly. Nevertheless, it can be seen that BS-GMPSUM performs nearly as well as BS-EX, and clearly better than the remaining two BS variants.
- In the case of the quasi-random instances of benchmark sets VIRUS and RAT, BS-GMPSUM starts to show its strength by delivering the best solution qualities in 31 out of 40 cases. The second best variant is BS-EX, which is still performing very well, and is able to achieve the best solution qualities in 24 out of 40 cases.
- For the special BB benchmark set, in which input strings were generated in order to be similar to each other, GMPSUM turns out to perform comparably to the best variant BS-POW.
- Concerning the real-world benchmark set BACTERIA, BS-GMPSUM is able to deliver the best results for 18 out of 35 groups, which is slightly inferior to the BS-HP variant with 22 best-performances, and superior to variants BS-EX (12 cases) and BS-POW (15 cases). Concerning the average solution quality obtained for this benchmark set, BS-GMPSUM is able to deliver the best one among all considered approaches.
- Concerning the multinominal non-uniformly distributed benchmark set POLY, BS-GMPSUM clearly outperforms all other considered BS variants. In fact, BS-GMPSUM is able to find the best solutions for all 6 instance groups. Moreover, it beats the other approaches in terms of the average solution quality.
- Overall, BS-GMPSUM finds the best solutions in 80 (out of 121) instances or instance groups, respectively. The second best variant is BS-EX, which is able to achieve best-performance in 62 cases. In contrast, BS-Hp and BS-POW are clearly inferior to the other two approaches. We conclude that BS-GMPSUM performs well in the context of different letter distributions in the input strings, and it is worth to try this variant first when nothing is known about the distribution in the considered instance set.
- Overall the running times of all four BS variants are comparable. The fastest one is BS-HP, while BS-GMPSUM requires somewhat more time compared to the others since it makes use of a heuristic function that combines two functions.

Table 2 provides a summary concerning the long-run scenarios, i.e., it compares the current state-of-the-art algorithm A*+ACS with TRBS-GMPSUM. As the benchmark instances are the same as in the short-run scenarios, the first two table columns are the same as in Table 2. Then there are two blocks of columns, presenting the results of A*+ACS and TRBS-GMPSUM in terms of the average solution quality over all instances of the respective benchmark set ($\overline{|s|}$), and the number of instances (or instance groups) for which the respective algorithm archived the best result (#b.).
The following can be concluded based on the results obtained for the long-run scenarios:

- Concerning RANDOM and ES, A*+ACS is—as expected—slightly better than TRBS-GMPSUM in terms of the number of best results achieved. However, when comparing the average performance, there is hardly any difference between the two approaches: 109.9 vs. 109.7 for the RANDOM benchmark set, and 243.82 vs. 243.73 for the ES benchmark set.

- In the context of benchmark sets RAT and VIRUS, TRBS-GMPSUM improves over the state-of-the-art results by a narrow margin. This holds both for the number of best results achieved and for the average algorithm performance.
- Concerning benchmark set BB, TRBS-GMPSUM significantly outperforms A*+ACS. In six out of eight groups it delivers the best average solution quality, while A*+ACS does so only for three cases.
- The same holds for the real-world benchmark set BACTERIA, that is, TRBS-GMPSUM achieves the best results for 33 out of 35 instances, in contrast to only 10 instances in the case of A*+ACS. Moreover, the average solution quality obtained is much better for TRBS-GMPSUM, namely 862.63 vs. 829.26.
- Finally, the performances of both approaches for benchmark set POLY are very much comparable.
- Overall, we can conclude that TRBS-GMPSUM is able to deliver the best results in 101 out of 121 cases, while A*+ACS does so only in 77 cases. This is because TRBS-GMPSUM provides a consistent solution quality across instances characterized by various kinds of letter distributions. It can therefore be stated that TRBS-GMPSUM is a new state-of-the-art algorithm for the LCS problem.

In summary, for the 32 random instances—respectively, instance groups—from the literature (sets RANDOM and ES) A*+ACS performs quite strong due to the presumed randomness of the instances. However, the new TRBS-GMPSUM approach is not far behind. A weak point of A*+ACS becomes obvious when instances are not generated uniformly at random. In the 40 cases with quasi-random input strings (sets RAT and VIRUS) TRBS-GMPSUM performs best in 37 cases, while A*+ACS does so in 31 case. When input strings are similar to each other—see the 8 instance groups of set BB—A*+ACS performs weak compared to TRBS-GMPSUM. This tendency is reinforced in the context of the instances of set POLY (6 instance groups) for which TRBS-GMPSUM clearly outperforms A*+ACS in all cases. The same holds for the real-world benchmark set BACTERIA. The overall conclusion yields that TRBS-GMPSUM works very well on a wide range of different instances. Moreover, concerning the instances from the previous literature (80 instances/groups) our TRBS-GMPSUM approach is able to obtain new state-of-the-art results in 13 cases. This will be shown in the next section.

4.4. New state-of-the-art results for instances from the literature

Due to space restrictions we provide the complete set of results, for each problem instance, in a document on supplementary material (https://github.com/milanagrbic/LCSonNuD/LCSonNuD_Supplementary_file.pdf). Instead of providing all results we decided to focus on those cases in which new state-of-the-art results are achieved. These cases are presented in Table 3 (short-run scenario) and Table 4 (long-run scenario).

Table 3: New best results for the instances from literature in the short-run scenario.

Instance (group)				Literature Best $\|s\|$		BS-EX		BS-POW		BS-HP		BS-GMPSUM	
Benchmark set	$\|\Sigma\|$	m	n	$\|s\|$	Alg.	$\|s\|$	t	$\|s\|$	t	$\|s\|$	t	$\|s\|$	t
RAT	4	20	600	172	BS-EX	172	2.3	170	0.9	168	0.5	**173**	2.5
RAT	4	40	600	152	BS-EX	152	1.8	150	1	145	0.5	**154**	3.4
RAT	4	200	600	123	BS-EX	123	2.7	123	0.7	122	0.8	**124**	9.9
RAT	20	20	600	54	BS-EX	54	2.5	54	1.7	54	1.2	**55**	3.5
RAT	20	40	600	49	BS-EX	49	3	49	1.1	49	1.2	**50**	4.6
VIRUS	4	25	600	194	BS-EX	194	2.2	192	1.2	194	0.7	**195**	3.1
VIRUS	4	40	600	170	BS-EX	170	2.2	170	1.2	169	0.9	**172**	3.8
VIRUS	4	60	600	166	BS-EX	166	2.4	165	0.8	166	0.7	**168**	5.1
VIRUS	4	100	600	158	BS-EX	158	2.3	155	1.2	158	0.9	**160**	7.8
VIRUS	4	150	600	156	BS-EX	156	2.4	147	1.2	156	0.7	**157**	11
VIRUS	4	200	600	155	BS-HP	154	2.6	148	1.4	155	1.2	**156**	14.8
VIRUS	20	40	600	50	BS-EX	50	2.9	49	1.9	50	0.9	**51**	5.5
BB	2	100	1000	560.7	BS-POW	536.6	6.1	560.7	5.7	558.9	1.9	**560.8**	23.7
ES	2	10	1000	615.06	BS-EX	615.06	4.4	614.2	1.4	612.5	0.9	**615.1**	5.1
ES	10	50	1000	136.32	BS-EX	136.32	3.9	135.52	2.1	135.22	1.4	**136.34**	9.9
ES	25	10	2500	235.22	BS-POW	231.12	19.1	235.22	10.5	233.34	8	**235.58**	29
ES	100	10	5000	144.9	BS-POW	144.18	91.9	144.9	75.9	143.62	71.6	**145.1**	185.4

Table 4: New best results for the instances from literature in the long-run scenario.

Instance (group)				Literature best $\lvert s\rvert$		A*+ACS	TRBS-GMPSUM
Benchmark set	$\lvert\Sigma\rvert$	m	n	$\lvert s\rvert$	Alg.	$\lvert s\rvert$	$\lvert s\rvert$
RAT	4	20	600	174	A*+ACS	174	**175**
RAT	4	40	600	154	A*+ACS	154	**156**
RAT	20	25	600	52	A*+ACS	52	**53**
VIRUS	4	10	600	228	A*+ACS	228	**229**
VIRUS	4	15	600	206	A*+ACS	206	**207**
VIRUS	4	60	600	168	A*+ACS	168	**169**
VIRUS	4	80	600	163	A*+ACS	163	**164**
VIRUS	4	100	600	160	A*+ACS	160	**162**
VIRUS	4	150	600	157	A*+ACS	157	**158**
BB	2	100	1000	563.6	APS	547.1	**571.1**
BB	4	100	1000	390.2	APS	344.3	**391.8**
ES	2	10	1000	618.9	A*+ACS	618.9	**619.1**
ES	10	50	1000	137.5	A*+ACS	137.5	**137.6**
ES	25	10	2500	236.6	A*+ACS-DIST	235	**238**

The tables reporting on the new state-of-the-art results are organized as follows. The first column contains the name of the corresponding benchmark set, while the following three columns identify the respective instance (in the case of RAT and VIRUS), respectively the instance group (in the case of BB and ES). Afterwards, there are two columns that provide the best result known from the literature. The first of these columns provides the result, and the second column indicates the algorithm (together with the reference) that was the first one to achieve this result. Next, the tables provide the results of BS-EX, BS-POW, BS-HP and BS-GMPSUM in the case of the short-run scenario, respectively the results of A*+ACS and TRBS-GMPSUM in the case of the long-run scenario. Note that computation times are only given for the short-run scenario, because time served as a limit in the long-run scenario.

Concerning the short-run scenario (Table 3), BS-GMPSUM was able to produce new best results in 17 cases. This includes even four cases of benchmark set ES, which was generated uniformly at random. Remarkable are the four cases of sets VIRUS and RAT in which the currently best-known solution was improved by two letters (see, for example, the case of set RAT and the instance $\lvert\Sigma\rvert = 4$, $m = 40$, and $n = 600$). Concerning the more important long-run scenario, the best-known results so far were improved in 14 cases. Especially remarkable is the case concerning set BB for which an impressive improvement of around 24 letters was achieved.

4.5. Results for benchmark sets POLY *and* BACTERIA

The tables reporting on the results for benchmark set POLY are structured in the same way as those described before in the context of the other benchmark sets. The difference is that instances groups are identified by means of $\lvert\Sigma\rvert$ (first column), m (second column), and n (third column). Best results per instance group—that is, per table row—are displayed in bold font.

The results of the short-run scenario for benchmark set POLY are given in Table 5. According to the obtained results, a clear winner is BS-GMPSUM which obtains the best average solution quality for all six instance groups. This indicates that GMPSUM is clearly better as a search guidance than the other three heuristic functions for this benchmark set. As previously motivated, this is due to the strongly non-uniform nature of the instances, i.e., the intentionally generated imbalance of the number of occurrences of different letters in the input strings. Nevertheless, the absolute differences between the results of BS-GMPSUM and BS-EX are not so high. The results of the long-run executions for benchmark set POLY are provided in Table 6. It can be observed that TRBS-GMPSUM and the state-of-the-art technique A*+ACS perform comparably.

Remember that, as in the case of POLY, the instances of benchmark set BACTERIA are used for the first time in a study concerning the LCS problem. They were initially

Table 5: Short-run results for benchmark set POLY.

Instance group			BS-EX		BS-POW		BS-HP		BS-GMPSUM	
$\|\Sigma\|$	m	n	$\|s\|$	t	$\|s\|$	t	$\|s\|$	t	$\|s\|$	t
4	10	100	43.2	0.5	43.2	0.3	43.1	0.3	**43.3**	0.1
4	10	500	232.5	4.1	232.7	2.6	231.3	2.1	**233**	2.5
4	10	1000	470.7	8.8	470.1	5.4	467.3	4.2	**470.9**	10.3
4	50	100	35.7	0.6	35.6	0.4	35.5	0.3	**35.8**	0.3
4	50	500	201.4	6.1	200.8	3.5	200.4	3	**202.3**	6.2
4	50	1000	412.5	13.2	411.2	7.4	411.6	6.3	**412.8**	20.9

Table 6: Long-run results for benchmark set POLY.

Instance group			A*+ACS	TRBS-GMPSUM	
$\|\Sigma\|$	m	n	$\|s\|$	$\|s\|$	t
4	10	100	**43.4**	**43.4**	580.7
4	10	500	**234.3**	**234.3**	890.5
4	10	1000	**473.9**	473.4	896.2
4	50	100	**35.9**	**35.9**	83.6
4	50	500	203	**203.5**	883.8
4	50	1000	414.3	**414.9**	892.3

proposed in a study concerning the constrained LCS problem [35]. The results are again presented in the same way as described before. This set consists of 35 instances. Therefore, each line in Table 7 (short-run scenario) and Table 8 (long-run scenario) deals with one single instance which is identified by $|\Sigma|$ (always equal to 4), m (varying between 2 and 383), $n_{\min}$ (the length of the shortest input string) and $n_{\max}$ (the length of the longest input string). Best results are indicated in bold font. The results obtained for the short-run scenario allow to observe that BS-HP performs very well for this benchmark set. In fact, it obtains the best solution in 22 out of 35 cases. However, BS-GMPSUM is not far behind with 18 best solutions. Moreover, BS-GMPSUM obtains a slightly better average solution quality than BS-HP. Concerning the long-run scenario, as already observed before, TRBS-GMPSUM clearly outperforms A*+ACS. In fact, the differences are remarkable in some cases such as, for example, instance number 32 (fourth but last line in Table 8) for which TRBS-GMPSUM obtains a solution of value 1241, while A*+ACS finds—in the same computation time—a solution of value 1204.

4.6. Statistical significance of the so-far reported results

In this section we study the results of the short-run and long-run executions from a statistical point of view. In order to do so, Friedman's tests was performed simultaneously considering all four algorithms in the case of the short-run scenario, respectively the two considered algorithms in the case of the long-run scenario.[1]

Given that in all cases the test rejected the hypothesis that the algorithms perform equally, pairwise comparisons were performed using the Nemenyi post-hoc test [38]. The corresponding critical difference (CD) plots considering all benchmark sets together are shown in Figure 2, respectively Figure 3a. Each algorithm is positioned in the segment according to its average ranking w.r.t. average solution quality over all (121) considered instance groups. The critical difference was computed with a significance level of 0.05. The performances of those algorithms whose difference is below the CD are regarded as performing statistically in an equivalent way—that is, no difference of statistical significance can be detected. This is indicated in the figures by bold horizontal bars joining the respective algorithm markers.

Concerning short-run executions, BS-GMPSUM is clearly the overall best-performaing algorithm, with statistical significance. BS-EX is in second position. Moreover, the difference between BS-HP and BS-POW is not statistically significant. Concerning the long-run

[1] All these tests and the resulting plots were generated using R's **scmamp** package [37].

Table 7: Short-run results for benchmark set BACTERIA.

Instance				BS-EX		BS-POW		BS-HP		BS-GMPSUM	
$\|\Sigma\|$	m	$n_{\min}$	$n_{\max}$	$\|s\|$	t	$\|s\|$	t	$\|s\|$	t	$\|s\|$	t
4	383	610	1553	256	31.1	252	13.9	**279**	16.8	271	94.1
4	3	1458	1458	**1365**	2.1	**1365**	1	**1365**	1.8	**1365**	7.7
4	33	1349	1577	610	17.6	605	10.6	**755**	10.8	689	36.5
4	106	1252	1520	503	25.1	483	12	**515**	12.2	514	61.8
4	2	1502	1502	**1499**	0	**1499**	0	**1499**	0	**1499**	0.1
4	12	1274	1413	**659**	13.3	636	8.5	627	6.9	**659**	18.9
4	15	1302	1515	598	13.3	602	8.5	655	7.7	**678**	20.7
4	13	1479	1557	811	15.8	752	10.1	**1061**	10	883	21.7
4	13	1308	1507	1037	17.6	**1039**	11.1	862	8.6	882	25.9
4	44	873	1543	493	16.3	473	9.3	470	7.8	**494**	29.6
4	4	1408	1530	1204	9	**1271**	6.3	**1271**	5.8	**1271**	15.9
4	173	1234	1847	502	34.7	463	15	**541**	18.3	525	97.5
4	13	1446	1551	681	14.5	713	9.5	**794**	8.6	785	22.2
4	88	1360	1545	583	27.3	570	13.8	**667**	15.1	601	67
4	2	1540	1548	**1522**	0.2	**1522**	0.1	**1522**	0.1	**1522**	0.3
4	3	1395	1424	**1141**	11.2	**1141**	6.8	**1141**	6.1	**1141**	15
4	4	1410	1488	886	9.8	**1123**	8	**1123**	6.7	**1123**	17.4
4	51	1266	1522	**681**	25.2	552	12.3	667	12	641	48.7
4	2	1461	1539	**1354**	0.9	**1354**	0.5	**1354**	1.7	**1354**	8.6
4	13	1246	1411	687	13	662	7.4	609	6.7	**699**	19.6
4	4	1434	1478	876	9.6	**1112**	8	**1112**	6.9	**1112**	16.3
4	18	1023	1438	464	11.9	468	7.6	458	5.8	**475**	14.2
4	2	1454	1460	**1431**	0.2	**1431**	0.1	**1431**	0.1	**1431**	0.3
4	8	1401	1533	1024	15.9	**1061**	9.8	858	7.5	864	18.8
4	33	990	1483	410	12.1	**492**	8.8	467	6.9	456	16.4
4	29	1422	1549	587	16.3	581	9.8	**634**	8.9	590	26.3
4	20	571	1394	**438**	9.6	405	5.4	401	4.5	431	11.7
4	96	1270	1565	516	24	467	11	**531**	12.3	522	55.5
4	10	1322	1455	**1026**	16	**1026**	9.7	796	7.1	**1026**	19.5
4	26	1334	1596	617	16.7	584	9.4	**640**	8.6	631	26.2
4	195	1345	1547	503	38.2	448	15.3	**537**	19.2	524	100.9
4	8	1454	1532	1221	16.4	**1241**	10	**1241**	8.6	**1241**	25.5
4	8	1359	1612	555	18.9	555	11.2	600	10.3	**627**	38.4
4	89	455	1587	**251**	11.3	214	4.7	233	4.8	239	18.2
4	2	1465	1469	**1358**	0.7	**1358**	0.4	**1358**	0.5	**1358**	6.8

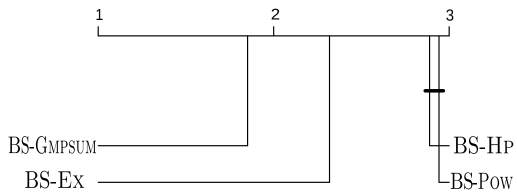
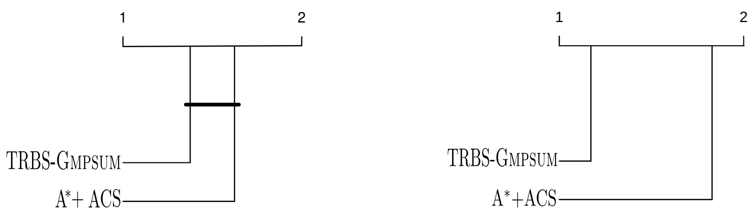


Figure 2. Critical difference (CD) plot over all considered benchmark sets (short-run executions).



(a) All instances **(b)** Benchmark set BACTERIA.

Figure 3. Criticial difference (CD) plots concerning the long run scenario.

scenario, the best average rank is obtained by TRBS-GMPSUM. In the case of benchmark set BACTERIA, the difference between TRBS-GMPSUM and A*+ACS is significant, see Figure 3b. For the other benchmark sets the two approaches perform statistically equivalent.

Table 8: Long-run results for benchmark set Bacteria.

Instance				A*+ACS	TRBS-Gmpsum	
$\|\Sigma\|$	m	$n_{\min}$	$n_{\max}$	$\|s\|$	$\|s\|$	t
4	383	610	1553	265	**273**	887.2
4	3	1458	1458	**1365**	**1365**	810.9
4	33	1349	1577	670	**723**	899
4	106	1252	1520	518	**532**	897.1
4	2	1502	1502	**1499**	**1499**	0.1
4	12	1274	1413	665	**694**	899.7
4	15	1302	1515	680	**708**	899.6
4	13	1479	1557	842	**883**	899.5
4	13	1308	1507	870	**1043**	899.7
4	44	873	1543	**514**	501	897.3
4	4	1408	1530	1204	**1271**	898.4
4	173	1234	1847	520	**528**	895.6
4	13	1446	1551	732	**816**	899.6
4	88	1360	1545	557	**634**	897.9
4	2	1540	1548	**1522**	**1522**	0.3
4	3	1395	1424	**1141**	**1141**	899.7
4	4	1410	1488	1059	**1123**	899.4
4	51	1266	1522	659	**871**	898.9
4	2	1461	1539	**1354**	**1354**	851.9
4	13	1246	1411	716	**727**	899.6
4	4	1434	1478	1030	**1112**	899.2
4	18	1023	1438	481	**488**	898.4
4	2	1454	1460	**1431**	**1431**	0.3
4	8	1401	1533	1040	**1063**	899.2
4	33	990	1483	449	**510**	899.1
4	29	1422	1549	643	**661**	899
4	20	571	1394	**439**	432	899.7
4	96	1270	1565	529	**546**	897.2
4	10	1322	1455	**1026**	**1026**	899.7
4	26	1334	1596	654	**676**	899.3
4	195	1345	1547	514	**544**	894.2
4	8	1454	1532	1204	**1241**	898.3
4	8	1359	1612	624	**644**	897.9
4	89	455	1587	250	**252**	898.2
4	2	1465	1469	**1358**	**1358**	829.6

5. Textual Corpus Case Study

In the previous section we showed that the proposed method is highly competitive with state-of-the art methods and generally outperforms them on instances sampled from non-uniform distributions. In order to further investigate the behavior of the proposed method on real-world instances with non-uniform distribution, we performed a case study on a corpus of textual instances originating from abstracts of scientific papers written in English. This set will henceforth be called Abstract. It is known that letters in English language are polynomially distributed [22]. The most frequent letter is e, with a relative frequency of 12.702%. The next most common letter is t (9.056%), followed by a (8.167%), and o (7.507%), etc.

In order to make a meaningful choice of texts we followed [39], where the authors measured the similarity between scientific papers, mainly from the field of artificial intelligence, by making use of various algorithms and metrics. By using tf-idf statistics with cosine similarity, their algorithm identified similar papers from a large paper collection. After that, the similarity between the papers proposed by their algorithm was manually checked and tagged by an expert as either similar (positive) or dissimilar (negative). The results of this research can be found at https://cwi.ugent.be/respapersim.

Keeping in mind that the LCS problem is also a measure of text similarity, we decided to check whether the abstracts of similar papers have longer common subsequences than abstracts of dissimilar papers. Therefore, the purpose of this case study is twofold: (1) to execute the LCS state-of-the art methods along with the method proposed in this paper and to compare their performances on this specific instance set, and (2) to check whether the abstracts of similar papers have a higher LCS than those of dissimilar papers.

Based on these considerations, we formed two groups of twelve papers each, named `POS` and `NEG`. Group `POS` contains twelve papers which have been identified as similar, while group `NEG` contains papers which are not similar to each other. We extracted abstracts from each paper and pre-processed them in order to remove all letters except for those letters from the English alphabet. In addition, each uppercase letter was replaced with its lowercase pair.

For each if the two groups we created a set of test instances as follows. For each $k \in \{10, 11, 12\}$ we generated $\binom{12}{k}$ different instances containing k input strings (considering all possible combinations). This resulted in the following set of instances for both `POS` and `NEG`:

- One instance containing all 12 abstracts as input strings.
- 12 instances containing 11 out of 12 abstracts as input strings.
- 66 instances containing 10 out of 12 abstracts as input strings.

Repeating our experimental setup presented in the previous section, we performed both short and long runs for the described instances. The obtained results for the short–run scenarios are shown in Table 9. The table is organized into five blocks of columns. The first block provides the general information on the instances: `NEG` vs. `POS`, number of input strings (column with heading m), and the total number of instances (column #). The remaining four blocks contain the results of BS-Ex, BS-Pow, BS-Hp and BS-Gmpsum, respectively. For each considered group of instances and each method, the following information about the obtained results is shown:

- $\overline{|s|}$: solution quality of the obtained LCS for the considered group of instances.
- #b.: number of cases in which the method reached the best result for the considered group of instances.
- $\bar{t}$: average execution time in seconds for the considered group of instances.

Table 9: Short-run results for the textual corpus instances (Abstract).

Instance set			BS-Ex			BS-Pow			BS-Hp			BS-Gmpsum		
Name	m	#	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$	$\overline{\lvert s\rvert}$	#b.	$\bar{t}$
NEG	12	1	128	0	14.6	123	0	10.8	126	0	11.8	130	1	11.4
NEG	11	12	132.08	7	15	127	0	11.2	129.42	0	12.2	132.58	8	11.7
NEG	10	66	136.47	29	14.9	132.5	0	11.3	134.82	4	11.7	137.27	50	11.6
POS	12	1	134	1	15.2	128	0	11.9	131	0	11.8	133	0	7.4
POS	11	12	137.67	5	15	131.58	0	11.2	135.92	1	11.5	138.42	11	7.2
POS	10	66	143.33	42	14.5	135.85	0	10.7	141.53	10	11.5	143.14	39	7.2
All Negative		79		36			0			4			59	
All Positive		79		48			0			11			50	
All		158		84			0			15			109	

The results from Table 9 clearly indicate that the best results for instances based on group `NEG` are obtained by BS-Gmpsum. More precisely, BS-Gmpsum works best for the instance with 12 input strings, for eight out of 12 instances with 11 input strings and for 50 out of 66 instances with 10 input strings. In contrast, the second-best approach (BS-Ex) reached the best result for 29 out of 66 instances with 10 input strings and seven out of 12 instances with 11 input strings. The remaining two methods were less successful for this group of instances. For the instances derived from group `POS`, BS-Gmpsum also achieved very good results. More specifically, BS-Gmpsum obtained the best results in almost all instances with 11 input strings. For instances with ten input strings, BS-Ex obtained the best results in 42 out of 66 cases, with BS-Gmpsum performing comparably (best result in 39 out of 66 cases). For the instance with twelve strings, the best solution was found by the BS-Ex. Similarly to the instances from the `NEG` group, BS-Hp and BS-Pow are clearly less successful.

A summary of these results is provided in the last three rows of Table 9. Note that, in total, this table deals with 158 problem instances: 79 regarding group `NEG`, and another 79 regarding group `POS`. The summarized results show that the new GMPSUM guidance is, overall, more successful than its competitors. More precisely, BS-GMPSUM achieved the best results in 59 out of 79 cases concerning `NEG`, and in 50 out of 79 cases concerning `POS`. Moreover, it can be observed that the average LCS length regarding the `POS` instances is greater than the one regarding the `NEG` instances, across all m values.

Table 10: Long-run results for the textual corpus instances (ABSTRACT).

Instance set			A*+ACS		TRBS-GMPSUM		
Name	m	#	$\overline{\lvert s\rvert}$	#best	$\overline{\lvert s\rvert}$	#best	$\bar{t}$
NEG	12	1	129	0	130	1	895.7
NEG	11	12	133.25	2	134.33	11	897.2
NEG	10	66	138.32	31	139.12	60	897.8
POS	12	1	136	1	136	1	896.5
POS	11	12	140.17	6	140.42	9	896.8
POS	10	66	145.33	41	145.52	45	897.4
All Negative		79		33		72	
All Positive		79		48		55	
All		158		81		127	

Table 10 contains information for the long-run executions. The results obtained by A*+ACS and TRBS-GMPSUM are shown. The table is organized in a similar way as Table 9, with the exception that it does not contain information about execution times, since computation time served as the stopping criterion. As it can be seen from the overall results at the bottom of Table, TRBS-GMPSUM obtains more best results than A*+ACS for both groups of instances (`NEG` and `POS`). More precisely, it obtained the best result for the instances with 12 input strings, both in the case of `POS` and `NEG`, while A*+ACS achieved the best result only in the case of the `POS` instance with 12 input strings. Concerning the results for the instances with 11 input strings, it can be noticed that—in the case of the `NEG` instances—TRBS-GMPSUM delivers 11 out of 12 best results, while A*+ACS method does so only in two out of twelve cases. Regarding the `POS` instances with 11 input strings, the difference becomes smaller. More specifically, TRBS-GMPSUM achieves nine out of 12 best results, while A*+ACS achieved six out of 12 best results. A corresponding comparison can be done for the instances with 10 input strings. For the instances concerning group `NEG`, TRBS-GMPSUM delivers the best results for 60 out of 66 instances, while A*+ACS can find the best results only in 31 cases. Finally, in the case of the `POS` instances, the best results were achieved in 45 out of 66 cases by TRBS-GMPSUM, and in 41 out of 66 cases by A*+ACS. The long run results also indicate that abstracts of similar papers are characterized by generally longer LCS measures.

6. Conclusions and Future Work

In this paper we considered the prominent longest common subsequence problem with an arbitrary set of input strings. We proposed a novel search guidance, named GMPSUM, for tree search algorithms. This new guidance function was defined as a convex combination of two complementary heuristics: (1) the first one is suited for instances in which the distribution of letters is close to uniform-at-random, and (2) the second one is convenient for all cases in which letters are non-uniformly distributed. The combined score produced by these two heuristics provides a guidance function which navigates the search towards promising regions of the search space, on a wide range of instances with different distributions. We ran short-run experiments in which beam search makes use of a comparable number of iterations under different guidance heuristics. The conclusion was that the novel guidance heuristic performs statistically equivalent to the best-so-far heuristic from the literature on close-to-random instances. Moreover, it was shown that it significantly outperforms the known search guidance functions on instances with a non-uniform letter frequency per input string. This capability of the proposed heuristic to deal with a non-uniform scenario was validated on two newly introduced benchmark sets: (1) POLY, whose input strings are generated from a multinomial distribution, and (2) ABTRACT, which are real-world instances whose input strings follow a multinomial distribution and originate from abstracts of scientific papers written in English. In a second part of the experimentation we performed long-run executions. For this purpose we combined the GMPSUM guidance function with a time-restricted BS that dynamically adapts its beam width during execution such that the overall running time is very close to the desired time limit. This algorithm was able to outperform the best approach from the literature (A^*+ACS) significantly. More specifically, the best-known results from the literature were at least matched for 63 out of 80 considered instance groups. Moreover, regarding the two new benchmark sets (POLY and BACTERIA), the time-restricted BS guided by GMPSUM was able to deliver equally good, and in most cases better, solutions than A^*+ACS in 38 out of 41 instance groups.

In future work we plan to adapt GMPSUM to other LCS-related problems such as the constrained longest common subsequence problem [40], the repetition-free longest common sunsequene problem [41], the LCS problem with a substring exclusion constraint [42], and the longest common palindromic subsequence problem [43]. Also, it would be interesting to incorporate this new guidance function into the leading hybrid approach A^*+ACS to possibly further boost the obtained solution quality.

Author Contributions: B.N. was responsible for conceptualization, methodology, and writing. A.K. was responsible for software implementation, writing and methodology. M.D. was responsible for conceptualization, witting and visualization. M.G. was responsible for resources and writing. C.B. was responsible for writing, editing and supervision. G.R. was responsible for concept polishing, supervision, and funding acquisition.

Data Availability Statement: The reported results can be found at https://github.com/milanagrbic/LCSonNuD.

Acknowledgments: This research was partially supported by Ministry for Scientific and Technological Development, Higher Education and Information Society, Government of Republic of Srpska, B&H under the Project “Development of artificial intelligence methods for solving computer biology problems", project no. 19.032/-961-24/19. Marko Djukanovic was funded by the Doctoral Program Vienna Graduate School on Computational Optimization (VGSCO), Austrian Science Foundation, project no. W1260-N35. Christian Blum was funded by project CI-SUSTAIN of the Spanish Ministry of Science and Innovation (PID2019-104156GB-I00).

Abbreviations

The following abbreviations are used in this manuscript:

LCS	Longest Common Subsequence
BS	Beam Search
ACS	Anytime column search
APS	Anytime pack search

References

1. Maier, D. The Complexity of Some Problems on Subsequences and Supersequences. *Journal of the ACM* **1978**, *25*, 322–336.
2. Minkiewicz, P.; Darewicz, M.; Iwaniak, A.; Sokołowska, J.; Starowicz, P.; Bucholska, J.; Hrynkiewicz, M. Common Amino Acid Subsequences in a Universal Proteome—Relevance for Food Science. *International Journal of Molecular Sciences* **2015**, *16*, 20748–20773.
3. Storer, J. *Data Compression: Methods and Theory*; Computer Science Press: MD, USA, 1988.
4. Beal, R.; Afrin, T.; Farheen, A.; Adjeroh, D. A new algorithm for “the LCS problem” with application in compressing genome resequencing data. *BMC Genomics* **2016**, *17*, 544. doi:10.1186/s12864-016-2793-0.
5. Kruskal, J.B. An overview of sequence comparison: Time warps, string edits, and macromolecules. *SIAM review* **1983**, *25*, 201–237.
6. Xie, X.; Liao, W.; Aghajan, H.; Veelaert, P.; Philips, W. Detecting Road Intersections from GPS Traces Using Longest Common Subsequence Algorithm. *ISPRS International Journal of Geo-Information* **2017**, *6*.
7. Bergroth, L.; Hakonen, H.; Raita, T. A survey of longest common subsequence algorithms. In Proceedings of SPIRE 2000 – The 7th International Symposium on String Processing and Information Retrieval. IEEE, 2000, pp. 39–48.

8. Gusfield, D. *Algorithms on Strings, Trees, and Sequences*; Computer Science and Computational Biology, Cambridge University Press, 1997.
9. Fraser, C.B. Subsequences and Supersequences of Strings. PhD thesis, University of Glasgow, Glasgow, UK, 1995.
10. Huang, K.; Yang, C.; Tseng, K. Fast Algorithms for Finding the Common Subsequences of Multiple Sequences. Proceedings of ICS 2004 – The 9th International Computer Symposium. IEEE Press, 2004.
11. Blum, C.; Blesa, M.J.; López-Ibáñez, M. Beam search for the longest common subsequence problem. *Computers & Operations Research* **2009**, *36*, 3178–3186.
12. Mousavi, S.R.; Tabataba, F. An improved algorithm for the longest common subsequence problem. *Computers & Operations Research* **2012**, *39*, 512–520.
13. Tabataba, F.S.; Mousavi, S.R. A hyper-heuristic for the longest common subsequence problem. *Computational Biology and Chemistry* **2012**, *36*, 42–54.
14. Wang, Q.; Korkin, D.; Shang, Y. A fast multiple longest common subsequence (MLCS) algorithm. *IEEE Transactions on Knowledge and Data Engineering* **2011**, *23*, 321–334.
15. Djukanovic, M.; Raidl, G.R.; Blum, C. Anytime algorithms for the longest common palindromic subsequence problem. *Computers & Operations Research* **2020**, *114*, 104827. doi:https://doi.org/10.1016/j.cor.2019.104827.
16. Djukanovic, M.; Raidl, G.; Blum, C. A Beam Search for the Longest Common Subsequence Problem Guided by a Novel Approximate Expected Length Calculation. Proceedings of LOD 2019 – The 5th International Conference on Machine Learning, Optimization, and Data Science. Springer, 2019, LNCS. to appear.
17. Blum, C.; Festa, P. Longest Common Subsequence Problems. In *Metaheuristics for String Problems in Bioinformatics*; Wiley, 2016; chapter 3, pp. 45–60.
18. Chan, H.T.; Yang, C.B.; Peng, Y.H. The Generalized Definitions of the Two-Dimensional Largest Common Substructure Problems. Proceedings of the 33rd Workshop on Combinatorial Mathematics and Computation Theory. National Taiwan University, Department of Mathematics, 2016, pp. 1–12.
19. Li, Y.; Wang, Y.; Zhang, Z.; Wang, Y.; Ma, D.; Huang, J. A novel fast and memory efficient parallel MLCS algorithm for long and large-scale sequences alignments. IEEE 32nd International Conference on Data Engineering, 2016, pp. 1170–1181.
20. Peng, Z.; Wang, Y. A Novel Efficient Graph Model for the Multiple Longest Common Subsequences (MLCS) Problem. *Frontiers in Genetics* **2017**, *8*, 104. doi:10.3389/fgene.2017.00104.
21. Vadlamudi, S.G.; Gaurav, P.; Aine, S.; Chakrabarti, P.P. Anytime column search. Proceedings of AI'12 – The 25th Australasian Joint Conference on Artificial Intelligence. Springer, 2012, pp. 254–265.
22. Lewand, R.E. *Cryptological mathematics*; Vol. 16, American Mathematical Soc., 2000.
23. Beutelspacher, A. *Kryptologie*; Vol. 7, Springer, 1996.
24. Bakaev, M. Impact of familiarity on information complexity in human-computer interfaces. MATEC Web of Conferences. EDP Sciences, 2016, Vol. 75, p. 08003.
25. Ow, P.S.; Morton, T.E. Filtered beam search in scheduling. *The International Journal Of Production Research* **1988**, *26*, 35–62.
26. Ney, H.; Haeb-Umbach, R.; Tran, B.H.; Oerder, M. Improvements in beam search for 10000-word continuous speech recognition. [Proceedings] ICASSP-92: 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing. IEEE, 1992, Vol. 1, pp. 9–12.
27. Kumar, A.; Vembu, S.; Menon, A.K.; Elkan, C. Beam search algorithms for multilabel learning. *Machine learning* **2013**, *92*, 65–89.
28. Araya, I.; Riff, M.C. A beam search approach to the container loading problem. *Computers & Operations Research* **2014**, *43*, 100–107.
29. Djukanovic, M.; Raidl, G.R.; Blum, C. Finding Longest Common Subsequences: New anytime A search results. *Applied Soft Computing* **2020**, *95*, 106499. doi:https://doi.org/10.1016/j.asoc.2020.106499.
30. Blum, C.; Blesa, M.J.; Lopez-Ibanez, M. Beam search for the longest common subsequence problem. *Computers & Operations Research* **2009**, *36*, 3178–3186.
31. Jiang, T.; Li, M. On the approximation of shortest common supersequences and longest common subsequences. *SIAM Journal on Computing* **1995**, *24*, 1122–1139.
32. Shyu, S.J.; Tsai, C.Y. Finding the longest common subsequence for multiple biological sequences by ant colony optimization. *Computers & Operations Research* **2009**, *36*, 73–91.
33. Easton, T.; Singireddy, A. A large neighborhood search heuristic for the longest common subsequence problem. *Journal of Heuristics* **2008**, *14*, 271–283.
34. Blum, C.; Blesa, M.J. Probabilistic beam search for the longest common subsequence problem. International Workshop on Engineering Stochastic Local Search Algorithms. Springer, 2007, pp. 150–161.
35. Djukanovic, M.; Kartelj, A.; Matic, D.; Grbic, M.; Blum, C.; Raidl, G. Solving the Generalized Constrained Longest Common Subsequence Problem with Many Pattern Strings. Technical Report AC-TR-21-008, AC, 2021.
36. Kesten, H.; Morse, N. A property of the multinomial distribution. *The Annals of Mathematical Statistics* **1959**, *30*, 120–127.
37. Calvo, B.; Santafé Rodrigo, G. scmamp: Statistical comparison of multiple algorithms in multiple problems. *The R Journal, Vol. 8/1, Aug. 2016* **2016**.
38. Pohlert, T. The pairwise multiple comparison of mean ranks package (PMCMR). *R package* **2014**, *27*, 9.
39. Magara, M.B.; Ojo, S.O.; Zuva, T. A comparative analysis of text similarity measures and algorithms in research paper recommender systems. 2018 conference on information communications technology and society (ICTAS). IEEE, 2018, pp. 1–5.

40. Gotthilf, Z.; Hermelin, D.; Lewenstein, M. Constrained LCS: Hardness and Approximation. Proceedings of CPM 2008 – The 19th Annual Symposium on Combinatorial Pattern Matching. Springer, 2008, Vol. 5029, *LNCS*, pp. 255–262.
41. 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* **2010**, *158*, 1315–1324. doi:https://doi.org/10.1016/j.dam.2009.04.023.
42. Zhu, D.; Wang, X. A Simple Algorithm for Solving for the Generalized Longest Common Subsequence (LCS) Problem with a Substring Exclusion Constraint. *Algorithms* **2013**, *6*, 485–493.
43. Chowdhury, S.R.; Hasan, M.M.; Iqbal, S.; Rahman, M.S. Computing a Longest Common Palindromic Subsequence. *Fundamenta Informaticae* **2014**, *129*, 329–340, [arXiv:1110.5296v1]. doi:10.3233/FI-2014-974.