A Beam Search for the Shortest Common Supersequence Problem Guided by an Approximate Expected Length Calculation

Jonas Mayerhofer[1], Markus Kirchweger[2],
Marc Huber[3], and Günther Raidl[4]

TU Wien, Vienna
`e01633065@student.tuwien.ac.at`[1],
{`mk`[2], `mhuber`[3], `raidl`[4]}`@ac.tuwien.ac.at`

Abstract. The shortest common supersequence problem (SCSP) is a well-known NP-hard problem with many applications, in particular in data compression, computational molecular biology, and text editing. It aims at finding for a given set of input strings a shortest string such that every string from the set is a subsequence of the computed string. Due to its NP-hardness, many approaches have been proposed to tackle the SCSP heuristically. The currently best-performing one is based on beam search (BS). In this paper, we present a novel heuristic (AEL) for guiding a BS, which approximates the expected length of an SCSP of random strings, and embed the proposed heuristic into a multilevel probabilistic beam search (MPBS). To overcome the arising scalability issue of the guidance heuristic, a cut-off approach is presented that reduces large instances to smaller ones. The proposed approaches are tested on two established sets of benchmark instances. MPBS guided by AEL outperforms the so far leading method on average on a set of real instances. For many instances new best solutions could be obtained.

Keywords: Shortest Common Supersequence Problem · Beam Search.

1 Introduction

We define a string s as a finite sequence of letters from a finite alphabet Σ. A subsequence of a string s is a sequence derived by deleting zero or more letters from that string without changing the order of the remaining letters. If s is a subsequence of another string x, then x is a supersequence of s. A common supersequence of a set of n non-empty strings $S = \{s_1, \dots, s_n\}$ is a string x consisting of letters from Σ, which is a supersequence of each $s \in S$.

The *shortest common supersequence problem* (SCSP) asks for a shortest possible string that is a common supersequence of a given set of strings S. For example, a shortest common supersequence of the strings `GAATG`, `AATGG`, and `TAATG` is `GTAATGG`. The problem is not only of theoretical interest but has important applications in many areas, including data compression [21], query optimization in databases [20], AI planning [8], and bioinformatics [14,15]. Hence, there is a vast benefit when finding more effective algorithms for solving the SCSP effectively. For a fixed number of strings n, the SCSP can be solved in time $\mathcal{O}(m^n)$ by dynamic programming, where m is the length of the longest string [12]. As typical values for real instances go up to $n = 500$ and $m = 1000$ [10], this approach is not feasible in practice. For general n, the problem is known to be NP-hard [11], even under a binary alphabet [19] or strings with length two [23]. Although there exists an exact algorithm based on dynamic programming by Irving et al. [12], approximation and heuristic algorithms seem to be unavoidable tackling larger instances in practice. The currently best-performing one amongst them relies on beam search (BS) [10].

Beam search is a well-known incomplete tree search that explores a state graph by only expanding and further pursuing the most promising successor nodes of each level. Besides the SCSP, BS is also able to shine on similar problems like the Longest Common Subsequence Problem [6] and the Longest Common Palindromic Subsequence Problem [7]. For these problems, a BS guided by a theoretically derived function EX that approximates the expected length of the result of random strings from a partial solution achieved exceptional performance and outperformed other approaches on many instances.

Inspired by these approaches, we present a novel function that approximates the expected length of an SCSP of random strings and utilize it as guidance heuristic in the multilevel probabilistic beam search (MPBS) from Gallardo et al. [10]. While our guidance heuristic performs well on smaller instances, numerical issues arise during its calculation on larger instances. To deal with this scalability issue, we present a cut-off approach that effectively reduces large instances to smaller ones. In our experimental evaluation, we consider a standard BS as well as the MPBS, both guided by the new expected value heuristic with and without the cut-off approach, and evaluate them on established benchmark instances. The experimental results show that our approaches can be highly effective and, in many cases, yield better solutions than the so far leading approach. On average, we are able to outperform the leading approach on a test set of large real instances and are competitive on a set of small random instances.

Section 2 reviews related work. In Section 3, we introduce some definitions and notations to describe the SCSP formally. The general BS framework for the SCSP with dominance check and the derivation of the new guidance heuristic as well as the cut-off approach are presented in Section 4. Experimental results of the methods on two established sets of benchmark instances, including some real-world instances, are compared to each other and the best-known results from the literature in Section 5. Finally, we conclude in Section 6, where we also outline promising future work.

2 Related Work

A first definition of the SCSP and an NP-hardness proof for an arbitrary number of sequences over an alphabet of size five were given by Maier in 1976 [16].

Later, in 1994, Irving et al. [12] solved the SCSP for a fixed number of strings in polynomial time by dynamic programming.

In addition to this exact algorithm, many approaches have been proposed to tackle the SCSP heuristically for large instances. A simple greedy heuristic called Majority Merge by Jiang et al. [13] constructs a supersequence by incrementally adding the symbol that occurs most frequently at the beginning of the strings in S and then deleting these symbols from the front of the corresponding strings. However, if the strings in S have different lengths, symbols at the beginning of shorter strings are as likely to be deleted. Therefore, Branke et al. [4,5] suggested a Weighted Majority Merge (WMM) that uses the string lengths as weights. A simple $|\Sigma|$-approximation algorithm called Alphabet was presented by Barone et al. [1]. For an alphabet $\Sigma = \{\mathtt{a}_1, \ldots, \mathtt{a}_\sigma\}$, this algorithm returns as trivial solution $(\mathtt{a}_1 \cdot \mathtt{a}_2 \cdot \ldots \cdot \mathtt{a}_\sigma)^m$. A Deposition and Reduction algorithm was introduced by Ning et al. [18]. It first generates a small set of common supersequences and then tries to shorten these by deleting one or more symbols while preserving the common supersequence property. For generating a small set of common supersequences, a Look Ahead Sum Height (LA-SH) [14] and the Alphabet algorithm [1] are used. The LA-SH algorithm [14] extends the Majority Merge algorithm by a *look-ahead* strategy: Instead of considering only one step for choosing the best letter to add, the LA-SH algorithm looks several steps ahead before a letter is added.

In the context of BS, Gallardo et al. [9] presented a hybrid of a Memetic Algorithm with a BS, and Blum et al. [3] suggested a BS with a probabilistic approach for including elements in the beam, called Probabilistic Beam Search (PBS). More specifically, PBS calculates heuristic values using a look-ahead version of the WMM from [4,5] and computes the probability of a partial solution to be chosen based on these values. However, instead of always selecting a partial solution probabilistically, they employ a mixed strategy in which, at random, either the partial solution with the highest probability value is taken or a solution is chosen by a roulette-wheel selection. Furthermore, for reducing runtime, lower bounds are calculated by adding up the length of the partial solution x and the maximum number of occurrences of each symbol in any of the remaining strings. Later, Mousavi et al. [17] introduced a highly successful Improved BS (IBS) which outperformed all previous algorithms for solving the SCSP in all experiments they performed. This algorithm uses some basic laws of probability theory to calculate the probability that a uniform random string of a certain length is a supersequence of a set of strings S under the assumption that all strings in S are independent and some further simplifying assumptions. This probability is then used as heuristic value to guide the BS.

A multilevel PBS algorithm (MPBS) presented by Gallardo et al. [10] could achieve even better results than IBS and thus constitutes so far the state-of-the-art for the SCSP. The MPBS follows a destroy and repair paradigm. First, an initial solution is generated by the PBS framework from Blum et al. [2] but using the guidance heuristic from the IBS. Afterwards, two processes called PBS-Perturbation and PBS-Reduction, are executed on the current solution until the allowed execution time is reached. PBS-Perturbation replaces a randomly chosen symbol of the current solution by a different one. This replacement can lead to an infeasible solution. Therefore, a repairing mechanism using PBS is employed, where PBS is run for the (now) uncovered part on the right side, and it is tried to find a shorter solution. PBS-Reduction first splits the current solution x into a prefix (x_{L}) and a suffix (x_{R}) and computes the longest suffix for each string in S that is a subsequence of x_{R}. The parts (prefixes) of all strings $s \in S$ that do not belong to the longest suffixes yield a new instance of the problem. Solving this instance using PBS yields a supersequence (x'_{L}) for the prefixes. If $|x'_{\mathrm{L}}| < |x_{\mathrm{L}}|$, then x_{L} is substituted by x'_{L}, i.e., $x = x'_{\mathrm{L}} \cdot x_{\mathrm{R}}$. This process is executed for all possible splits of the current solution and restarted with the enhanced solution as soon as an improvement is made.

3 Preliminaries

We denote the *length* of a string s by $|s|$, and the empty string by ε. If s is a string of length m, then $s[j]$ denotes the j-th letter of s for $1 \leq j \leq m$, and $s[j, j']$ refers to the substring $s[j]s[j+1]\dots s[j']$ for $1 \leq j \leq j' \leq m$ and the empty string else. Let s_1, s_2 be two strings, then $s_1 \cdot s_2$ denotes the concatenation of these. Furthermore, we denote by $s_1 \preceq s_2$ that s_1 is a subsequence of s_2 and s_2 a supersequence of s_1. Let $S = \{s_1, \dots, s_n\}$ be a set of n strings and x another string. Then, by $p_i(x)$ we denote the largest integer such that $s_i \in S$, and $s_i[1, p_i(x)] \preceq x$. We call $p = (p_i)_{i=1,\dots,n}$ *position vector* of a (partial) solution x as it indicates the already covered parts of the input strings. It further follows that x is a common supersequence of S if $|s_i| = p_i(x)$ for all $i \in \{1, \dots, n\}$. Additionally, we define $r_i(x) = s_i[p_i(x) + 1, |s_i|]$ as the remaining part of the string s_i not covered by x. We call a string x a *partial solution* of the problem, if it is not (yet) a common supersequence.

4 Beam Search Framework

We now present the general BS framework with dominance check on which we build and utilize the basic ideas of IBS [17] and EX [6] to calculate the *approximate expected length* of an SCSP on random strings. Moreover, we describe a cut-off approach to make larger instances tractable without running into numerical issues.

Beam search is a prominent graph search algorithm that expands nodes in a breadth-first search manner to find a best path from a root node to a target node. To keep the computational effort within limits, BS evaluates the reached nodes at each level and selects a subset of only up to β most promising nodes to pursue further. The selected subset of nodes is called *beam* B, and parameter β *beam width*. In the context of the SCSP, the state graph is a directed acyclic graph $G = (V, A)$, where a state (node) $v \in V$ contains a position vector p^v, which represents the SCSP sub-instance with input strings $\{r^v_1, \dots, r^v_n\}$ with $r^v_i = s_i[p^v_i + 1, |s_i|],\ i = 1, \dots, n$. The root node $\mathrm{r} \in V$ has position vector

$p^{\mathrm{r}} = (1, \ldots, 1)$ and corresponds to the original SCSP instance. The terminal node $\mathrm{t} \in V$ with $p^{\mathrm{t}} = (|s_i|)_{i=1,\ldots,n}$ represents the instance with all strings being empty, i.e., all letters of the original instance have already been covered. An arc $(u, v) \in A$ refers to transitioning from state u to state v by appending a letter $a \in \Sigma$ to a partial solution, and thus, arc (u, v) is labeled by this letter denoted by $\ell(u, v) = a$. Appending a letter $a \in \Sigma$ is only considered feasible if this letter corresponds with the first letter of at least one remaining string r_i^u, $i = 1, \ldots, n$. Nodes are expanded until the terminal node $\mathrm{t} \in V$ is reached. Then, the BS returns a shortest path from the root node to the terminal node, which represents the final valid solution. In general, there may exist several different paths from the root to some node, corresponding to different (partial) solutions yielding the same position vector. From these, we always only have to keep one shortest path. Thus, it is enough to actually store with each node a single reference to the parent node in order to finally derive the solution in a backward manner. For deciding which nodes are selected into the beam, a heuristic function is needed that expresses how promising each state is. We use for this purpose a new approximation of the expected length an optimal solution to the SCSP sub-instance induced by a state has, for short the Approximate Expected Length (AEL); it will be described in Section 4.3.

A pseudocode for the BS framework is shown in Algorithm 1. The algorithm starts with the root node. Function ExtendAndEvaluate expands each node of the current beam B by creating a new node for each feasible letter $a \in \Sigma$, and an arc, labeled a, between the original node and the new node, and it updates the p vector. Each new node is evaluated by calculating its AEL. In line 5, an optional dominance check is performed, which may discard nodes that are dominated by others; details follow in Section 4.1. Line 6 does the actual selection of the up to β best nodes according to the heuristic values to obtain the new beam.

Procedure ExtendAndEvaluate runs in time $\mathcal{O}(\beta\,|\Sigma|\,T_{\mathrm{AEL}})$, where T_{AEL} is the time of one AEL calculation. The cardinality of the set of new nodes V_{ext} is at most $\beta\,|\Sigma|$. Sorting V_{ext} for selecting the beam takes $\mathcal{O}(\beta\,|\Sigma| \cdot \log(\beta\,|\Sigma|))$ time. The total runtime of the BS without dominance check is therefore

$$\mathcal{O}\left(l\,\beta\,|\Sigma| \cdot (T_{\mathrm{AEL}} + \log(\beta\,|\Sigma|))\right), \tag{1}$$

where l is the length of the solution, i.e., levels of the BS. As we will argue, the dominance check we apply also does not increase this asymptotic time.

4.1 Dominance Check

A dominance check is used to filter out certain nodes that cannot be part of a shortest r–t path. We say a node u *dominates* node v at the current BS level if $p^u \neq p^v$ and $p_i^u \geq p_i^v$ for all $i = 1, \ldots, n$. Nodes that are dominated by other nodes at the current level can be discarded as they cannot lead to better solutions. More specifically, we apply the restricted κ-dominance check in the spirit of [17] to avoid a quadratic effort in the number of nodes. Let

Algorithm 1 BS for the SCSP

```
   Input: instance $(S, \Sigma)$
   Output: a common supersequence of $S$
1: $B \leftarrow \{p^r\}$                                        ▷ beam
2: while true do
3:     $V_\text{ext} \leftarrow$ ExtendAndEvaluate($B$)
4:     if t $\in V_\text{ext}$ then return solution corresponding to r–t path
5:     $V_\text{ext} \leftarrow$ DominanceCheck($V_\text{ext}$)              ▷ optional
6:     B $\leftarrow$ select (up to) $\beta$ best nodes from $V_\text{ext}$
```

$K \subseteq V_\text{ext}$ be the subset of the (up to) κ best-ranked nodes according to the heuristic evaluation; $\kappa \geq 0$ is hereby a strategy parameter. We then do pairwise domination checks only among V_ext and K: First, the expanded set of nodes V_ext is sorted in non-decreasing order by the nodes' heuristic values, and the leftmost κ solutions $K \subseteq V_\text{ext}$ are selected. If $v \in V_\text{ext}$ is dominated by any $u \in K$, then node v is discarded. Note that in contrast to [17], dominance within the leftmost κ solutions is also checked according to their order. A single dominance check of two nodes takes time $\mathcal{O}(n)$, and therefore, the whole κ-dominance check for one level is done in time $\mathcal{O}(\kappa\,\beta\,|\Sigma|\,n)$. If we consider κ to be a constant, the BS's total asymptotic time complexity (1) does not change.

4.2 Heuristic Function from IBS

Before we introduce AEL, we review the heuristic function from Mousavi et al. [17] as it provides a basis for our considerations. Its basic idea is to calculate the probability of a random string of length k being a common supersequence of an independent random string of length q.

Theorem 1. *Let w, y be two uniform random strings with length q and k respectively. The probability of y being a common supersequence of w is*

$$\mathbb{P}(q,k) = \begin{cases} 1 & \text{if } q = 0 \\ 0 & \text{if } q > k \\ \frac{1}{|\Sigma|}\mathbb{P}(q-1,k-1) + \frac{|\Sigma|-1}{|\Sigma|}\mathbb{P}(q,k-1) & \text{otherwise.} \end{cases} \tag{2}$$

By using Theorem 1, the probability that a random string y is a common supersequence of a set of strings S can be calculated by $\prod_{s \in S} \mathbb{P}(|s|,|y|)$. Hence, for a partial solution x, the probability that y is a common supersequence of all $r_i(x)$ is $h(x) = \prod_{i=1}^{n} \mathbb{P}(|r_i(x)|,|y|)$. Mousavi et al. directly use this probability as guidance function with the length of the string y calculated by

$$|y| = \max_{v \in V_\text{ext},\ i \in \{1,\ldots,n\}} |r_i^v| \cdot \log(|\Sigma|), \tag{3}$$

but also mention that selecting the "best" length would need further investigation.

4.3 Approximate Expected Length (AEL)

Now, we present our new guidance heuristic AEL, inspired by the earlier work for the longest common subsequence problem from Djukanovic et al. [6] that approximates the expected length of an SCS on uniform random strings. Let Y be the random variable corresponding to the length of an SCSP of n uniformly randomly generated strings S, and let $Y_k \in \{0,1\}$ be a binary random variable indicating if there is a supersequence of length k for S. The realizations of Y cannot be larger than $u = |\Sigma|\, m$ and smaller than m, where $m = \max_{s \in S} |s|$. The upper bound can be trivially shown by taking an arbitrary permutation of the alphabet and repeating it m times, cf. [22]. These definitions enable us to express $\mathbb{E}(Y)$ in terms of $\mathbb{E}(Y_k)$ by using some basic laws from probability theory:

$$\begin{aligned}\mathbb{E}(Y) &= \sum_{k=m}^{u} k\,\mathbb{P}(Y=k) \\ &= \sum_{k=m}^{u} k \cdot (\mathbb{E}(Y_k) - \mathbb{E}(Y_{k-1})) \\ &= u\,\mathbb{E}(Y_u) - \sum_{k=m}^{u-1} \mathbb{E}(Y_k) - m\,\mathbb{E}(Y_{m-1}) \\ &= u - \sum_{k=m}^{u-1} \mathbb{E}(Y_k).\end{aligned} \tag{4}$$

Assume that the probability of a sequence being a common supersequence of all strings in S is independent for distinct sequences. The probability that a string of length k is a common supersequence of S is then given by $\prod_{i=1}^{n} \mathbb{P}(|s_i|, k)$, and the probability that this is not the case by $1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k))$. Under the assumption that these probabilities are independent for all $|\Sigma|^k$ possible strings of length k, the probability that none of them is a common supersequence is $(1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k))^{|\Sigma|^k}$. Hence, $\mathbb{E}(Y_k) = 1 - (1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k))^{|\Sigma|^k}$ holds under these simplifying assumptions. This enables us to utilize the probability function $\mathbb{P}(q, k)$ from the previous Section 4.2, yielding

$$\mathbb{E}(Y) = u - \sum_{k=m}^{u-1} \left(1 - (1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k))^{|\Sigma|^k}\right). \tag{5}$$

To avoid numerical problems and speed up the computation, the expression is not evaluated directly. In practice, most of the terms $1 - (1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k))^{|\Sigma|^k}$ are either quite close to zero or one. Moreover, it is easy to see that $1 - (1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k_1))^{|\Sigma|^{k_1}} \leq 1 - (1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k_2))^{|\Sigma|^{k_2}}$ holds for $k_1 < k_2$. Hence, a divide-and-conquer approach is employed to calculate the sum in Equation 5. More specifically, all values greater than $1-\delta$ or smaller than δ are approximated with 0 or 1, where we chose $\delta = 10^{-20}$ in our experiments.

Since $|\Sigma|^k$ cannot be represented in a standard floating-point arithmetic for large k, the expression $(1 - \prod_{i=1}^{n} \mathbb{P}(|s_i| \preceq k))^{|\Sigma|^k}$ is decomposed into

$$\Bigg(\underbrace{\Big(\cdots(1 - \prod_{i=1}^{n} \mathbb{P}(|s_i|, k))^{|\Sigma|^p}\cdots\Big)^{|\Sigma|^p}}_{\lfloor k/p \rfloor \text{ times}}\Bigg)^{|\Sigma|^{(k \ mod \ p)}} \tag{6}$$

for $p = 10$. Additionally, if the product over the probabilities is very small, numerical issues occur. To tackle this problem, the expression in Equation 6 is estimated by using a Taylor series approximation if the product is smaller than a certain threshold. See [6] for more details on this.

Cut-Off. For larger instances, the heuristic values the above calculation yields are nevertheless often integers due to numerical imprecisions, because either $|\Sigma|^k$ is far too large or the product is far too small. This often results in many equal heuristic values for the nodes, and thus a poor differentiation of how promising the nodes are. To deal with this issue, we shorten the strings for the calculation when they exceed a certain length. More specifically, at each iteration of the while-loop in Algorithm 1, the length m_{ext} of the longest remaining string over all nodes in V_{ext} is taken, i.e., $m_{\text{ext}} = \max_{v \in V_{\text{ext}},\ i=1,\ldots,n} |r_i^v|$, and a *cut-off* $C = \max(0, m_{\text{ext}} - \gamma)$ is determined, where γ is a strategy parameter. Instead of calculating AEL for a node $v \in V_{\text{ext}}$ over the original remaining string lengths $|r_i^v|$, it is now determined over the lengths $\max(0,\ |r_i^v| - C)$, $i = 1, \ldots, n$.

Computational Complexity. In the worst case, for each k, the n probabilities for all n input strings have to be multiplied and the stable power applied. Thus, AEL can be performed in time $\mathcal{O}((q|\Sigma|)^2 n/p)$, where $q = \max_{i=1,\ldots,n}(|r_i^v|)$. In practice, due to the divide-and-conquer approach and approximation of values close to zero and one, only a small fraction of the stable powers and multiplications is needed.

5 Experimental Evaluation

We implemented the BS as well as MPBS, both with AEL as guidance function, in Julia 1.6.2. All tests were performed on an Intel Xeon E5-2640 processor with 2.40 GHz in single-threaded mode and a memory limit of 8GB. We compare these two approaches among each other and to results from IBS and the original MPBS as reported in [10].

5.1 Test Instances

Two benchmark sets already used in [10] are considered to evaluate the approaches. The first set denoted as `Real` consists of real-world instances, and the second one, `Rand`, of random instances. Set `Rand` consists of five instances for each $|\Sigma| \in \{2, 4, 8, 16, 24\}$, each instance having four random strings of length 40 plus four random strings of length 80, i.e., $n = 8$. Set `Real` consists of real DNA and protein instances. There are ten DNA instances for each combination of $n \in \{100, 500\}$ equally long strings of length $m \in \{100, 500, 1000\}$, and $|\Sigma| = 4$, in total thus 60 instances. Moreover, there are 10 protein instances for each $(n, m) \in \{(100, 100), (500, 100), (100, 500)\}$, where again all strings are equally long, and $|\Sigma|$ varies in $\{20, \ldots, 24\}$; in total these are 30 instances.

For both instance sets `Real` and `Rand`, a collection of so far best-known solution lengths l_{best} is provided in [10]. All instances are available online[1].

5.2 Impact of Cut-Off and Comparison to IBS

In this Section, we investigate the impact of the cut-off parameter γ and compare results to the IBS from [17]. Unfortunately, we could not reproduce the results stated by Mousavi et al. in [17] due to some missing details in their paper. An inquiry was without success. Also, we could not obtain the original source codes of former approaches. Moreover, note that the results reported for IBS in the newer publication [10] differ from those in [17]. For us, this does not matter much since for the cases where our approaches performed better they do so for the results reported in both papers. We use the newer values reported in [10] for the comparison here. To enable comparability of our results the same beam width of $\beta = 100$ and the dominance check with $\kappa = 7$ was applied as in [10].

Preliminary tests for the cut-off parameter γ indicated that results are not particularly sensitive as long as $20 \leq \gamma \leq 40$. Higher values lead frequently to the numerical issues we want to avoid, while lower values often result in an over-simplification and poor guidance. We were not able to find a clear relationship of good values for γ, the number of input strings, and the alphabet size. We therefore investigate these two border values as well as calculating AEL without cut-off. The BS variants with these guidance heuristics are denoted in the following by BS_{AEL}, $\text{BS}_{\text{AEL},\gamma=20}$, $\text{BS}_{\text{AEL},\gamma=40}$, respectively.

Tables 1 and 2 summarize obtained results for `Real` and `Rand` and those reported in [10] for IBS. Listed are for each instance group average solution lengths $\bar{l}$, the respective standard deviation σ and the average runtime $\bar{t}$ in seconds. In each row, the best result w.r.t. solution length is printed bold.

The results show that BS_{AEL} performs worse than IBS in most cases. This is primarily caused by the many ties the numerical calculating of AEL yields. By using the cut-off approach with $\gamma \in \{20, 40\}$, the performance of our BS with AEL increases significantly, and we obtain better results than IBS on almost all instances on `Real`. The approaches $\text{BS}_{\text{AEL},\gamma\in\{20,40\}}$ each outperform IBS on seven out of nine `Real` instance classes on average, where $\text{BS}_{\text{AEL},\gamma=\{40\}}$ outperforms IBS on all DNA instance classes, and $\text{BS}_{\text{AEL},\gamma=\{20\}}$ on two out of three protein instance classes. On the `Rand` instances, BS_{AEL} is almost on par with IBS, but, the cut-off extension did worsen instead of improving the average solution quality. The reason seems to be the strong differences in the lengths of

[1] Excluded to not give away information about the authors.

	n	m	BS_{AEL}			$\text{BS}_{\text{AEL},\gamma=20}$			$\text{BS}_{\text{AEL},\gamma=40}$			IBS
			$\bar{l}$	σ_l	$\bar{t}$[s]	$\bar{l}$	σ_l	$\bar{t}$[s]	$\bar{l}$	σ_l	$\bar{t}$[s]	$\bar{l}$
DNA	100	100	272.4	2.7	2.2	**270.3**	2.2	2.0	271.6	2.6	2.3	272.3
DNA	500	100	287.9	1.8	6.2	**285.7**	1.5	5.1	287.5	2.4	6.1	288.1
DNA	100	500	1290.9	9.7	6.9	1279.7	5.5	5.3	**1279.2**	5.7	7.2	1284.6
DNA	500	500	1362.4	8.9	26.0	1342.9	7.4	28.7	**1341.2**	7.2	27.1	1351.6
DNA	100	1000	2567.9	26.6	8.6	2545.4	10.9	13.2	**2536.5**	11.8	11.8	2540.1
DNA	500	1000	2682.8	25.2	48.9	2654.2	21.5	46.5	**2641.6**	18.5	52.5	2662.9
PROTEIN	100	100	920.9	16.0	16.7	**896.0**	12.5	21.5	912.9	8.9	16.5	910.6
PROTEIN	500	100	1122.5	26.6	99.9	**1071.4**	16.2	87.2	1092.3	10.7	102.2	1118.1
PROTEIN	100	500	4473.8	86.8	84.8	4405.7	47.5	98.6	4434.0	32.4	85.1	**4374.9**

Table 1: `Real` instances: average solution lengths and runtimes obtained by BS_{AEL}, $\text{BS}_{\text{AEL},\gamma\in\{20,40\}}$, and results of IBS from [10]; $\beta = 100$.

$\lvert\Sigma\rvert$	BS_{AEL}			$\text{BS}_{\text{AEL},\gamma=20}$			$\text{BS}_{\text{AEL},,\gamma=40}$			IBS
	$\bar{l}$	σ_l	$\bar{t}$[s]	$\bar{l}$	σ_l	$\bar{t}$[s]	$\bar{l}$	σ_l	$\bar{t}$[s]	$\bar{l}$
2	**109.4**	1.7	1.4	109.6	2.0	1.3	109.8	1.9	0.8	**109.4**
4	143.0	1.4	1.2	144.0	1.3	1.7	143.2	2.2	1.5	**142.4**
8	**180.4**	2.5	1.9	186.6	1.0	1.8	182.4	2.6	1.8	180.6
16	238.6	4.7	2.0	241.4	6.0	1.9	237.2	4.2	2.0	**235.6**
24	274.4	2.7	1.1	279.0	4.8	2.5	273.8	5.3	1.3	**268.8**

Table 2: `Rand` instances: average solution lengths and runtimes obtained by BS_{AEL}, $\text{BS}_{\text{AEL},\gamma\in\{20,40\}}$ and results of IBS from [10]; $\beta = 100$.

the input strings – remember that in each instance, half of the strings only has half the length of the others. Moreover, string lengths are generally smaller than in the `Real` instances, and therefore fewer ties occur in BS_{AEL} without cut-off. We remark that the runtimes reported in [10] for IBS are in the same order of magnitude, but generally smaller than the ones observed for our approaches. However, these times can hardly be compared due to the different programming languages and hardware used.

5.3 Integrating AEL into MPBS

As MPBS [10] is the approach yielding so far the best results on average, we now equip it with AEL as guidance heuristic. We set the strategy parameters as in [10]: $\beta_{\text{init}} = 100$ for generating the initial solutions, $\beta_{\text{pert}} = 700$ for the perturbation, and $\beta_{\text{redu}} = 200$ for the reduction. The number of generated initial solutions was set to $\zeta_{\text{init}} = 3$, the number of perturbations per iteration to $\zeta_{\text{init}} = 7$, and $\lambda = 0.6$ giving the degree of randomness of PBS (i.e. 60% of all nodes are selected randomly from V_{ext} instead of taking the best node); these values are not provided in [10] and therefore chosen by us following preliminary tests to balance exploration and exploitation.

In conjunction with AEL as guidance heuristic, we found that in the PBS-Reduction approach it seems to be better to not always increase the length of the prefix x_{L} of the current solution by one. Instead, we increase it by 5% of the current solution length. In this way, we had never more than 21 reduction runs

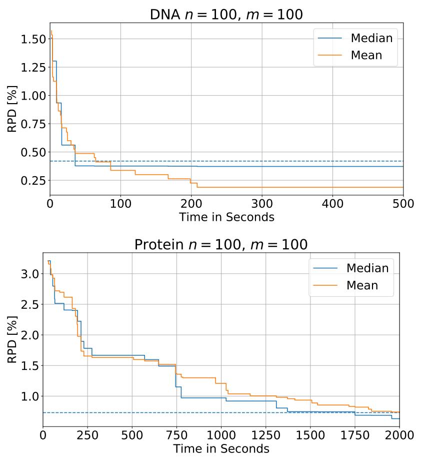

Fig. 1: RPD of $\text{MPBS}_{\text{AEL},\gamma=20}$ over time in seconds. The dashed line marks the average RPD value from MPBS_{IBS} [10].

per iteration, including a reduction run with the total length. This should not have a significant impact on the solution quality as long as the number of letters to be added is sufficiently small. The allowed runtime for MPBS was set to 500 seconds for the DNA instances and to 2000 seconds for the protein instances and 300 seconds for the `Rand` instances.

We denote MPBS with our AEL-heuristic without cut-off by MPBS_{AEL}, with cut-off for $\gamma \in \{20, 40\}$ by $\text{MPBS}_{\text{AEL},\gamma=20}$ and $\text{MPBS}_{\text{AEL},\gamma=40}$, respectively, and with the heuristic function used in IBS by MPBS_{IBS}. Figure 1 shows for $\text{MPBS}_{\text{AEL},\gamma=20}$ how the mean and median solution length over all 10 instances of an instance group change over time. Instead of providing the average solution lengths, we present the relative percentage difference (RPD) which is obtained by $100\% \cdot (l - l_{\text{best}})/l_{\text{best}}$, where l denotes the solution length and l_{best} the best-known length as listed in [10]. It turned out that the reductions and perturbations decrease the solution length frequently at the beginning, but after some time improvements could be achieved only rarely.

Figure 2 shows the RPD for DNA and protein instances obtained by BS_{AEL}, $\text{BS}_{\text{AEL},\gamma\in\{20,40\}}$, MPBS_{AEL}, and $\text{MPBS}_{\text{AEL},\gamma\in\{20,40\}}$ as boxplots. We can observe here once more that $\text{BS}_{\text{AEL},\gamma\in\{20,40\}}$ perform significantly better than the BS_{AEL} without cut-off. Further, turning to MPBS with its reductions and perturbations further improves the results in almost all cases significantly. For some instances, the results are already better than the previously best-known ones.

Figure 3 shows the RPD over all instances of `Real` and `Rand`. For `Real`, $\text{MPBS}_{\text{AEL},\gamma\in\{20,40\}}$ outperforms IBS and is close to MPBS_{IBS}. Furthermore, the figure shows that the average solution length of $\text{MPBS}_{\text{AEL},\gamma\in\{20,40\}}$ lies below the one of MPBS_{IBS} and thus $\text{MPBS}_{\text{AEL},\gamma\in\{20,40\}}$ outperforms MPBS_{IBS}. We remark that a comparison on an instance-based basis is not possible as no values

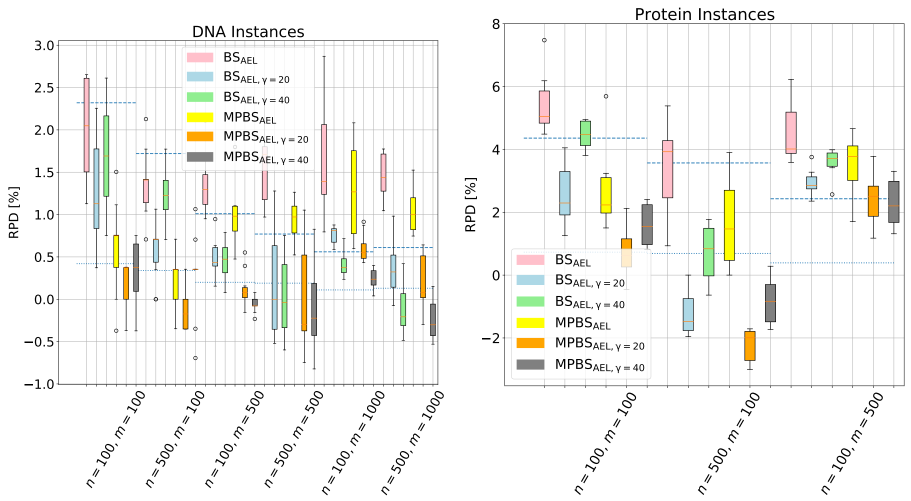

Fig. 2: Boxplots of the RPD for AEL and $MPBS_{AEL}$ on DNA and Protein instances. Dotted lines mark the average RPD from $MPBS_{IBS}$ [10] and dashed lines the average RPD from IBS [10].

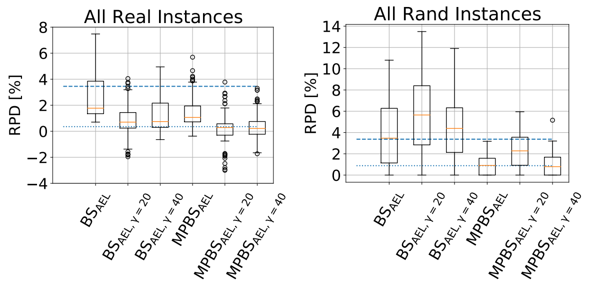

Fig. 3: Boxplots of the RPD for AEL and $MPBS_{AEL}$ over all `Real` and all `Rand` instances. Dotted lines mark the average RPD value from $MPBS_{IBS}$ [10] and dashed lines the average RPD from IBS [10].

per instance are reported for $MPBS_{IBS}$ in [10]. Concerning `Rand`, we observe that BS_{AEL} has almost the same average solution quality as IBS. Also $MPBS_{AEL}$, $MPBS_{AEL,\gamma=40}$ and $MPBS_{IBS}$ have a similar solution quality.

Tables 3 and 4 list average solution lengths and standard deviations of the MPBS variants including the results from [10]. Similarly to Table 1 and 2, we observe that for small random instances our approach does not improve on the previous approach while for almost all larger instances, we could achieve better results. The approaches $MPBS_{AEL,\gamma\in\{20,40\}}$ each outperform $MPBS_{IBS}$ on six out of nine `Real` instance classes on average. In total only one DNA and one protein instance class remain where $MPBS_{IBS}$ was better on average.

$\lvert\Sigma\rvert$	MPBS_{AEL}		$\text{MPBS}_{\text{AEL},\gamma=20}$		$\text{MPBS}_{\text{AEL},\gamma=40}$		MPBS_{IBS}
	$\bar{l}$	σ_l	$\bar{l}$	σ_l	$\bar{l}$	σ_l	$\bar{l}$
2	109.0	1.7	109.4	1.6	109.0	1.7	**108.8**
4	139.8	1.6	140.4	1.9	**139.6**	1.5	139.8
8	177.6	1.5	180.6	1.4	177.8	1.9	**177.2**
16	**227.6**	3.9	232.2	3.9	228.0	1.7	227.7
24	257.6	3.0	264.2	2.8	259.6	3.3	**257.3**

Table 3: Average solution lengths and standard deviations of the MPBS variants obtained on the `Rand` instances; results for MPBS_{IBS} from [10]; runtime: 300s.

	n	m	MPBS_{AEL}		$\text{MPBS}_{\text{AEL},\gamma=20}$		$\text{MPBS}_{\text{AEL},\gamma=40}$		MPBS_{IBS}
			$\bar{l}$	σ_l	$\bar{l}$	σ_l	$\bar{l}$	σ_l	$\bar{l}$
DNA	100	100	268.6	1.7	**267.5**	1.6	267.9	1.8	268.1
DNA	500	100	284.6	1.7	**283.7**	1.6	284.8	1.1	285.0
DNA	100	500	1286.1	10.0	1275.1	6.1	**1272.8**	6.2	1276.0
DNA	500	500	1353.1	6.4	1341.1	8.0	**1338.6**	6.6	1343.1
DNA	100	1000	2559.2	25.4	2541.5	10.6	2532.3	11.0	**2529.1**
DNA	500	1000	2672.5	23.1	2649.9	21.1	**2637.9**	20.3	2647.9
PROTEIN	100	100	897.5	17.6	**880.0**	13.0	887.7	8.2	880.2
PROTEIN	500	100	1103.7	26.2	**1060.5**	14.7	1075.9	11.9	1092.2
PROTEIN	100	500	4430.9	70.3	4382.6	51.8	4378.1	41.1	**4296.7**

Table 4: Average solution lengths and standard deviations of the MPBS variants obtained on the `Real` instances; results for MPBS_{IBS} from [10]; runtime: 500s for DNA instances, 2000s for protein instances.

5.4 Improving the Best-Known Solutions

By using a significantly larger beam width than 100, i.e., beam width $\beta \in \{1000, 2000, 5000, 10000\}$ and a timeout of eight hours per instance, we were able to improve several so far best-known solutions. In these experiments we considered BS_{AEL}, and $\text{BS}_{\text{AEL},\gamma\in\{20,40,50\}}$. Also, we consider MPBS_{AEL}, and $\text{MPBS}_{\text{AEL},\gamma\in\{20,40,50\}}$, with the settings given in Section 5.3, but with a timeout of eight hours per instance. Table 5 list for each instance group from `Real` the approach that achieved the best average solution length together with this solution value. In the case of ties, we list the fastest approach for BS_{AEL}, i.e., the one with smaller beam width, or all approaches in case of MPBS. For comparison, the table also lists the so far best-known average solution lengths from [10]. Table 6 list the results for `Rand` with BS_{AEL}, since no other method yields better results. Our new solutions can be found online[2]. f

For all the instance groups, better or the same average results could be found. The table also shows that a higher beam width yields better results. Further, we can see that even for small γ the algorithm performs relatively well and sometimes even returns better results than for higher γ values, especially for the protein instances. This might be due to the larger alphabet size in comparison to the DNA instances. In total, we found new on average best solutions for all instance classes of `Real`, three times with MPBS_{AEL} with different γ values, three

[2] Excluded to not give away information about the authors.

	n	m	method	β	best-found		best-known	
					$\bar{l}$	σ_l	$\bar{l}_{\text{best}}$	σ_l
DNA	100	100	$\text{MPBS}_{\text{AEL},\gamma=20}$	-	**266.0**	1.8	267.0	1.8
DNA	500	100	$\text{MPBS}_{\text{AEL},\gamma=20}$	-	**282.3**	0.9	284.0	1.9
DNA	100	500	$\text{BS}_{\text{AEL},\gamma=50}$	10000	**1269.4**	6.4	1273.5	7.2
DNA	500	500	$\text{MPBS}_{\text{AEL},\gamma\in\{40,50\}}$	-	**1333.1**	6.3	1340.6	4.4
DNA	100	1000	$\text{BS}_{\text{AEL},\gamma=50}$	10000	**2521.4**	11.4	2526.3	12.5
DNA	500	1000	$\text{BS}_{\text{AEL},\gamma=50}$	10000	**2631.1**	18.3	2644.5	22.3
PROTEIN	100	100	$\text{BS}_{\text{AEL},\gamma=20}$	5000	**850.6**	9.3	873.8	8.5
PROTEIN	500	100	$\text{BS}_{\text{AEL},\gamma=20}$	5000	**1040.7**	14.9	1084.7	14.4
PROTEIN	100	500	$\text{BS}_{\text{AEL},\gamma=40}$	5000	**4201.2**	38.2	4280.0	44.5

Table 5: Average solution lengths on `Real` with BS_{AEL}, $\text{BS}_{\text{AEL},\gamma\in\{20,40,50\}}$, MPBS_{AEL}, $\text{MPBS}_{\text{AEL},\gamma\in\{20,40,50\}}$ and high beam widths given in the table in comparison to the previously best-known results.

$\lvert\Sigma\rvert$	β	best-found		best-known	
		$\bar{l}$	σ_l	$\bar{l}_{\text{best}}$	σ_l
2	1000	**108.8**	1.3	**108.8**	1.3
4	2000	**139.2**	1.2	**139.2**	1.2
8	5000	**176.0**	1.6	**176.0**	1.7
16	5000	**222.4**	3.3	224.6	3.4
24	5000	**249.4**	1.9	252.4	1.6

Table 6: Average solution lengths on `Rand` with BS_{AEL} and high beam widths given in the table in comparison to the previously best-known results.

times with $\text{BS}_{\text{AEL},\gamma=50}$, two times with $\text{BS}_{\text{AEL},\gamma=20}$ and once with $\text{BS}_{\text{AEL},\gamma=40}$. For the `Rand` instance classes, we found the same average solutions for three out of five instance classes and new on average best solutions for the other two instance classes with a higher alphabet size of $|\Sigma| \in \{16, 24\}$.

6 Conclusions and Future Work

Inspired by previous work for the longest common subsequence problem, we developed a novel approximate expected length calculation for the SCSP and used it to guide a respective BS heuristic as well as the more advanced MPBS from [10]. To reduce ties that arise from numerical imprecisions in case of larger instances and impact performance, an effective cut-off approach was introduced. Our experiments show that by applying AEL instead of the heuristic used in IBS [17], we were able to achieve better results on most of the benchmark instances and to outperform MPBS_{IBS}, the so-far leading approach for the SCSP, on average. While AEL works particularly well on benchmark set `Real`, `Rand` instances turned out to be more challenging for AEL due to their particularity that the input strings differ heavily in their lengths. By using AEL in conjunction with higher beam widths and allowing longer runtimes, we could ultimately find new best-known solutions for almost all benchmark instances.

In future work, it would be interesting to adapt AEL for different variants of the SCSP, such as the constrained SCSP, or to apply machine learning techniques, in particular, reinforcement learning based, in order to learn a possibly even more appropriate guidance heuristic.

References

1. Barone, P., Bonizzoni, P., Vedova, G.D., Mauri, G.: An approximation algorithm for the shortest common supersequence problem: An experimental analysis. In: Proc. of the 2001 ACM Symposium on Applied Computing. p. 56–60. ACM (2001)
2. Blum, C., Blesa, M.J.: Probabilistic beam search for the longest common subsequence problem. In: Stützle, T., et al. (eds.) Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics, International Workshop. LNCS, vol. 4638, pp. 150–161. Springer (2007)
3. Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E.: A Probabilistic Beam Search Approach to the Shortest Common Supersequence Problem. In: Cotta, C., van Hemert, J. (eds.) Evolutionary Computation in Combinatorial Optimization. LNCS, vol. 4446, pp. 36–47. Springer (2007)
4. Branke, J., Middendorf, M.: Searching for shortest common supersequences by means of a heuristic-based genetic algorithm. In: Proceedings of the Second Nordic Workshop on Genetic Algorithms. pp. 105–113. University of Vaasa, Finland (1996)
5. Cotta, C.: A comparison of evolutionary approaches to the shortest common supersequence problem. In: Cabestany, J., et al. (eds.) Computational Intelligence and Bioinspired Systems, 8th International Work-Conference on Artificial Neural Networks. LNCS, vol. 3512, pp. 50–58. Springer (2005)
6. Djukanovic, M., Raidl, G.R., Blum, C.: A beam search for the longest common subsequence problem guided by a novel approximate expected length calculation. In: Nicosia, G., et al. (eds.) Proc. of the 5th Int. Conf. on Machine Learning, Optimization, and Data Science. LNCS, vol. 11943, pp. 154–167. Springer (2019)
7. Djukanovic, M., Raidl, G.R., Blum, C.: Anytime algorithms for the longest common palindromic subsequence problem. Computers & Operations Research **114**, 104827 (2020)
8. Foulser, D.E., Li, M., Yang, Q.: Theory and algorithms for plan merging. Artificial Intelligence **57**(2-3), 143–181 (1992)
9. Gallardo, J.E., Cotta, C., Fernandez, A.J.: On the Hybridization of Memetic Algorithms With Branch-and-Bound Techniques. IEEE Transactions on Systems, Man and Cybernetics, Part B **37**(1), 77–83 (2007)
10. Gallardo, J.E.: A Multilevel Probabilistic Beam Search Algorithm for the Shortest Common Supersequence Problem. PLOS ONE **7**(12), e52427 (2012), public Library of Science
11. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., USA (1990)
12. Irving, R.W., Fraser, C.B.: Maximal common subsequences and minimal common supersequences. In: Crochemore, M., Gusfield, D. (eds.) Combinatorial Pattern Matching. vol. 807, pp. 173–183. Springer (1994)
13. Jiang, T., Li, M.: On the approximation of shortest common supersequences and longest common subsequences. SIAM J. Comput. **24**(5), 1122–1139 (1995)
14. Kang, N., Pui, C.K., Wai, L.H., Louxin, Z.: A post-processing method for optimizing synthesis strategy for oligonucleotide microarrays. Nucleic Acids Research **33**(17), e144–e144 (2005)

15. Kasif, S., Weng, Z., Derti, A., Beigel, R., DeLisi, C.: A computational framework for optimal masking in the synthesis of oligonucleotide microarrays. Nucleic Acids Research **30**(20), e106 (2002)
16. Maier, D.: The complexity of some problems on subsequences and supersequences. Journal of the ACM **25**(2), 322–336 (1978)
17. Mousavi, S.R., Bahri, F., Tabataba, F.: An enhanced beam search algorithm for the shortest common supersequence problem. Eng. Appl. Artif. Intell. **25**(3), 457–467 (2012)
18. Ning, K., Leong, H.W.: Towards a better solution to the shortest common supersequence problem: the deposition and reduction algorithm. BMC Bioinformatics **7**(S4), S12 (2006)
19. Räihä, K., Ukkonen, E.: The shortest common supersequence problem over binary alphabet is NP-complete. Theor. Comput. Sci. **16**, 187–198 (1981)
20. Sellis, T.K.: Multiple-query optimization. ACM Trans. Database Syst. **13**(1), 23–52 (1988)
21. Storer, J.A.: Data Compression: Methods and Theory. Comp. Sci. Press (1988)
22. Sven, R.: The shortest common supersequence problem in a microarray production setting. Bioinformatics **19**, ii156–ii161 (2003)
23. Timkovskii, V.G.: Complexity of common subsequence and supersequence problems and related problems. Cybernetics **25**(5), 565–580 (1990)