# A*-Based Compilation of Relaxed Decision Diagrams for the Longest Common Subsequence Problem

Matthias Horn(✉) and Günther R. Raidl

Institute of Logic and Computation, TU Wien, Vienna, Austria
{horn,raidl}@ac.tuwien.ac.at

**Abstract.** We consider the longest common subsequence (LCS) problem and propose a new method for obtaining tight upper bounds on the solution length. Our method relies on the compilation of a relaxed multi-valued decision diagram (MDD) in a special way that is based on the principles of A* search. An extensive experimental evaluation on several standard LCS benchmark instance sets shows that the novel construction algorithm clearly outperforms a traditional top-down construction (TDC) of MDDs. We are able to obtain stronger and at the same time more compact relaxed MDDs than TDC and this in shorter time. For several groups of benchmark instances new best known upper bounds are obtained. In comparison to existing simple upper bound procedures, the obtained bounds are on average 14.8% better.

**Keywords:** Longest common subsequence problem · Multi-valued decision diagram · A* search

## 1 Introduction

In the last 10–15 years *decision diagrams* (DDs) have shown to be a powerful tool in combinatorial optimization with which for a wide range of problems new state-of-the-art approaches could be obtained [1,4,14]. This includes prominent problems such as minimum independent set, set covering, maximum cut, maximum 2-satisfiability [3,5] as well as variants of the traveling salesman problem and other sequencing and scheduling problems [14,24]. In particular can DD-based methods be superior where traditional mixed integer linear programming (MIP) or constraint programming (CP) approaches suffer, e.g., from weak dual bounds?

In essence, DDs are data structures that provide graphical representations of the solution space of a combinatorial optimization problem. Restricted DDs represent a subset of feasible solutions and can be used to obtain heuristic solutions

This project is partially funded by the Doctoral Program "Vienna Graduate School on Computational Optimization", Austrian Science Foundation (FWF) Project No. W1260-N35.


P. J. Stuckey (Ed.): CPAIOR 2021, LNCS 12735, pp. 72–88, 2021.
https://doi.org/10.1007/978-3-030-78230-6_5

and primal bounds [3,6] whereas relaxed DDs represent a superset of all feasible solutions in a compact way and can therefore be seen as a discrete relaxation of the problem. Relaxed DDs can be used to obtain dual bounds and provide, for instance, promising new branching schemes [3]. The more general form of a DD in which a node may have more than two outgoing arcs to successor nodes is called multi-valued DD (MDD), and MDDs have proven particularly useful for sequencing and scheduling problems.

Recently, a new A*-based construction (A*C) scheme was presented to compile relaxed MDDs [21]. The authors demonstrate on a prize-collecting scheduling problem that with A*C it is possible to compile in shorter running times relaxed MDDs that provide stronger bounds and are at the same time smaller than relaxed MDDs constructed by traditional top-down or incremental refinement methods. As the name A*C suggests, this method is inspired by A* search [20] and utilizes some fast but not necessarily that strong problem-specific bounding procedure during the construction. However, the prize-collecting scheduling problem from [21] is rather new. The goal of the current work therefore is to investigate the applicability of A*C on the prominent *longest common subsequence* (LCS) problem in order to see if this construction method has the potential to lead to superior results also on this already deeply investigated kind of problem. In our experimental evaluation we will compare the A*C approach for the LCS problem not only to a top-down MDD construction but also to several upper bounding procedures for the LCS from the literature.

The goal of the LCS problem [27] is to find the longest string which is a common subsequence of a set of $m$ input strings $S = \{s_1, s_2, \ldots, s_m\}$ over an alphabet $\Sigma$. We denote the length of a string $s$ by $|s|$, and let $n$ be the maximum length of the input strings, i.e., $n = \max_{i=1,\ldots,m} |s_i|$. A subsequence is a string that can be derived from another string by deleting zero or more characters. A common subsequence can be derived from all input strings. For instance, for the input strings `ABCDBA` and `ACBDBA`, an LCS is `ABDBA`. Determining the length of an LCS is a way to measure the similarity of strings and has a wide range of applications, for example in computational biology where strings often represent segments of RNA or DNA [23,30]. Other applications can be found in text editing, file comparison, data compression, and the production of circuits in field programmable gate arrays, to just name a few [2,12,25]. If $m$ is fixed then the LCS problem can be solved by *dynamic programming* (DP) based algorithms in polynomial time $O(n^m)$ [19]. For an arbitrary number of input strings, however, the problem is known to be NP-hard [27].

In the literature plenty of exact approaches have been proposed for solving the LCS problem. Besides the already mentioned DP based approaches, Blum and Festa [10] investigated a MIP model, which is however not competitive and cannot be practically applied to any of the commonly used benchmark sets in the literature due to its excessive size. Further exact methods are for instance based on dominant point approaches and/or parallelization [13,26,28,31] or on a transformation to the max clique problem [9], but they are still not applicable to practical instances with a large number of long input strings. Solving LCS

instances of practical relevance to proven optimality is still a challenging task in terms of computation time and memory consumption. Therefore, heuristic approaches are used for larger $m$ and $n$. Fast construction heuristics are, e.g., the expansion algorithm [11] or the best next heuristic [18,22]. Among the more advanced search strategies, in particular *beam search* (BS) based approaches have been frequently proposed differing in various details such as the heuristic guidance and filtering. This culminated in a general BS-based framework by Djukanovic at el. [15] which can express essentially all heuristic state-of-the-art approaches from the literature by respective configuration settings. They authors proposed also a novel heuristic guidance function, which approximates the expected length of a LCS for random strings. The BS framework in combination with this novel guidance dominates the other existing approaches on most of the available benchmark instances. The same authors further described novel A* based anytime algorithms by interleaving A* search with BS or anytime column search, respectively [16]. Thereby the novel search guidance from before plays again a crucial role.

Before we proceed let us define further notation. We denote the character at position $j$ in a string $s$ by $s[j]$, and $s[j, j']$, $j \leq j'$, refers to the continuous subsequence of $s$ starting at position $j$ and ending at position $j'$. For $j > j'$, substring $s[j, j']$ is the empty string denoted by $\varepsilon$. Last but not least, let $|s|_a$ be the number of occurrences of character $a \in \Sigma$ in string $s$.

The next section gives a formal definition of MDDs for the LCS problem. Section 3 reviews two known procedures to obtain upper bounds for the length of an LCS and presents a new one that extends one of those. Section 4 explains how relaxed MDDs are compiled for the LCS problem with A*C. Results of computational experiments are discussed in Sect. 5. Finally, Sect. 6 concludes this work.

## 2 Multi-valued Decision Diagrams for the LCS Problem

In the context of the LCS problem a MDD is a directed acyclic multi-graph $\mathcal{M} = (V, A)$ with one root node $\mathbf{r}$. For classical layer-based MDDs, all nodes are partitioned into at most $n + 1$ layers $L_1, \ldots, L_{n+1}$, where $L_1$ is a singleton containing only $\mathbf{r}$ and $L_i$, $i > 0$ contains only nodes that are reachable from $\mathbf{r}$ over exactly $i - 1$ arcs. An arc $\alpha = (u, v) \in A(\mathcal{M})$ in such a MDD is always directed from a source node $u$ in some layer $L_i$ to a target node $v$ in a subsequent layer $L_{i+1}$. In this work we will also construct MDDs that do not follow this layer structure. In all cases, each arc $\alpha$ is associated with a character $c(\alpha) \in \Sigma$ s.t. any directed path originating from $\mathbf{r}$ identifies a sequence of characters and thus a (partial) solution. The length of a path is defined as the number of its arcs and corresponds to the number of characters of the represented sequence. For non-layered MDDs, there is only one node that has no further outgoing arcs which we denote by $\mathbf{t}$.

An *exact* MDD encodes precisely the set of all feasible solutions, i.e., the set of all feasible common subsequences, and a longest path encodes a longest

common subsequence. Due to the NP-hardness of the LCS problem such exact MDDs will in general have exponential size. Therefore we consider more compact relaxed MDDs which encode supersets of all feasible solutions. In such a relaxed MDD nodes of an exact MDD are superimposed (*merged*) s.t. new paths may emerge representing sequences that are no feasible common subsequences. The length of a longest path then represents an upper bound on the LCS length. In contrast, restricted MDDs approximate exact MDDs by removing nodes and arcs from the exact one s.t. a longest path represents a heuristic solution and its length a lower bound on the LCS length. We remark that compiling a restricted MDD to obtain a heuristic solution essentially corresponds to the well-known beam search, and the leading heuristics for the LCS problem are diverse beam search variants as already pointed out in the previous section.

Each node $u \in V(\mathcal{M})$ is associated with a state which is a *position vector* $\mathbf{p}(u)$, with $p_i(u) \in \{1, \ldots, |s_i|\}$, $i = 1, \ldots, m$. On the basis of this position vector it is possible to define a subproblem $S[\mathbf{p}(u)]$ of $S$ by considering the substrings $s_i[p_i(u), |s_i|]$, $i, \ldots, m$. Thus, $S[\mathbf{p}(u)]$ consists of the right part of each string from $S$ starting from the position indicated in position vector $\mathbf{p}(u)$. The root state represents the original problem $S$, indicated by $S[\mathbf{p}(\mathbf{r}) = (1, \ldots, 1)]$. An arc $\alpha = (u, v) \in A(\mathcal{M})$ represents the transition from state $\mathbf{p}(u)$ to state $\mathbf{p}(v)$ by appending character $c(\alpha)$ to the sequences of characters encoded by the paths from $\mathbf{r}$ to $u$. The transition function to obtain successor state $\mathbf{p}(v)$ by considering character $c(\alpha)$ is defined as

$$\tau(\mathbf{p}(u), c(\alpha)) = \begin{cases} (p_{1,c(\alpha)}(u) + 1, \ldots, p_{m,c(\alpha)}(u) + 1) & \text{if } c(\alpha) \in \Sigma^{\mathrm{nd}}(u) \\ (n+1, \ldots, n+1) & \text{else}\,, \end{cases} \tag{1}$$

where $p_{i,a}(u)$, $i = 1, \ldots, m$ denotes for each character $a \in \Sigma$ the position of the first occurrence of $a$ in $s_i[p_i(u), |s_i|]$ and set $\Sigma^{\mathrm{nd}}(u) \subseteq \Sigma$ contains all letters that can be feasibly appended at state $\mathbf{p}(u)$, thus letters that occur at least once in each string in $S[\mathbf{p}(u)]$, and are non-dominated. A character $a \in \Sigma$ dominates character $b \in \Sigma$ iff $p_{i,a}(u) \leq p_{i,b}(u)$ for all $i = 1, \ldots, m$, and therefore it never can be better to append a dominated letter next. States that have no further feasible transition, i.e., where $\Sigma^{\mathrm{nd}} = \emptyset$, are mapped to state $(n+1, \ldots, n+1)$ of target node $\mathbf{t}$.

To create relaxed MDDs we have to define a state merger which computes the state of merged nodes. Let $U$ be a set of nodes that should be merged. An appropriate state merger is

$$\oplus(U) = \left( \min_{u \in U} p_i(u) \right)_{i=1,\ldots,m}. \tag{2}$$

Since we take always the minimum of each position, each feasible solution of any subproblem $S[\mathbf{p}(u)]$, $u \in U$, will also be a feasible solution of the subproblem $S[\mathbf{p}(\oplus(U))]$. Hence, no feasible solution will be lost in the relaxed MDD, but new paths corresponding to infeasible solutions may emerge.

## 3 Independent Upper Bounds

To compile MDDs based on A* search we need a fast-to-calculate independent upper bound on the solution length of LCS subproblems to guide the construction mechanism. We use two well known upper bounds from the literature as well as a third bound which is an adaption of one of the former. The first upper bound from Fraser [18] was tightened by Blum et al. [8] and is based on the number of occurrences of each character. Given a node $u$ and the associated position vector $\mathbf{p}(u)$, this bound calculates the sum of the minimal number of occurrences of each character over all the strings of the corresponding subproblem $S[\mathbf{p}(u)]$, i.e.,

$$\mathrm{UB}_1(u) = \mathrm{UB}_1(\mathbf{p}(u)) = \sum_{a \in \Sigma} \min_{i=1,\dots,m} |s_i\,[p_i(u), |s_i|]|_a\,. \tag{3}$$

By using a suitable data structure prepared in pre-processing, $\mathrm{UB}_1$ can be efficiently computed in $O(m\,|\Sigma|)$ time.

The second upper bound is based on DP and was introduced by Wang et al. [31]. Since the LCS for two input strings can be efficiently computed, for each pair $\{s_i, s_{i+1}\} \subseteq S$, $i = 1, \dots, m-1$ a so-called scoring matrix $\mathbf{M}_i^2$ is computed, where an entry $M_i^2[p, q]$ with $p = 1, \dots, |s_i|$ and $q = 1, \dots, |s_{i+1}|$, stores the length of the LCS of strings $s_i[p, |s_i|]$ and $s_{i+1}[q, |s_{i+1}|]$. The scoring matrices are determined in a pre-processing step. Then

$$\mathrm{UB}_2(u) = \mathrm{UB}_2(\mathbf{p}(u)) = \min_{i=1,\dots,m-1} \mathbf{M}_i^2[p_i(u), p_{i+1}(u)] \tag{4}$$

is an upper bound for the subproblem $S[\mathbf{p}(u)]$ of a given node $u$ and the associated position vector $\mathbf{p}(u)$.

The third upper bound we consider adapts the above one as follows. For $\mathrm{UB}_2$, $m-1$ scoring matrices are computed, one for each pair of input strings $\{s_i, s_{i+1}\}$, $i = 1, \dots, m-1$. However, the pairs of input strings are just chosen according to their natural order given by the instance specification. We are aiming now to choose pairs of input strings in a more controlled and more promising way by utilizing as guidance the version of the first upper bound function for two strings, i.e., $\mathrm{UB}_1(s_i, s_{i'}) = \Sigma_{a \in \Sigma} \min(|s_i|_a, |s_{i'}|_a)$, $s_i, s_{i'} \in S$, $s_i \neq s_{i'}$. Pairs of strings for which this value is small can be expected to typically also have shorter LCSs, possibly leading to an overall tighter bound. The subset of pairs of input strings for which we will compute corresponding scoring matrices, denoted by $P$, is determined as follows. We iterate over all pairs of input strings $\{(s_i, s_{i'}) \in S \times S \mid i < i'\}$ sorted according to $\mathrm{UB}_1(\cdot, \cdot)$ in non-decreasing order and add each string pair for which not both strings already appear in some string pair earlier added to $P$. In this way it is ensured that each input string is used at least once and $|P| = O(m)$. The upper bound of a given node $u$ is then

$$\mathrm{UB}_3(u) = \mathrm{UB}_3(\mathbf{p}(u)) = \min_{(s_i, s_{i'}) \in P} \mathbf{M}^3_{s_i, s_{i'}}[p_i(u), p_{i'}(u)], \tag{5}$$

where $\mathbf{M}^3_{i,i'}$ is the scoring matrix for string pair $(s_i, s_{i'}) \in P$.

Finally, let $\mathrm{UB}(u) = \min\{\mathrm{UB}_1(u), \mathrm{UB}_2(u), \mathrm{UB}_3(u)\}$ be the strongest upper bound we can obtain.

## 4 A*-Based Construction of MDDs

To construct relaxed MDDs we essentially follow the A*C approach from [21] using the principles of A* search. We maintain an open list $Q$ of nodes that need to be (re-)expanded. This list is sorted according to a priority function

$$f(u) = Z^{\text{lp}}(u) + \text{UB}(u), \tag{6}$$

where $Z^{\text{lp}}(u)$ denotes the length of the so far best path from the root node $\mathbf{r}$ to node $u$ and $\text{UB}(u)$ is the upper bound described in Sect. 3. To break ties, we prefer the node with higher $Z^{\text{lp}}$-value. Initially, $Q$ contains the root node. At each step, A* search and consequently A*C always take a node $u \in Q$ from the open list that maximizes $f(u)$ and expands $u$ by considering all feasible successor states using transition function $\tau$ from Eq. (1). Newly created nodes are inserted into $Q$; nodes that are reached in better ways via the expanded node are updated and reinserted into $Q$. If $Q$ finally gets empty then A*C has compiled a complete *exact* MDD, since all encountered nodes and arcs corresponding to feasible transitions are stored. However, since UB never underestimates the length of the LCS, the classical A* optimality condition can be applied by terminating early as soon as the target node $\mathbf{t}$ gets selected for expansion the first time. In this case we get in general an incomplete MDD, but due to the optimality condition of A* search at least one optimal path is contained in the MDD.

### 4.1 Relaxed MDDs

To create a relaxed MDD we limit the size of $Q$ by some threshold value $\phi$. As soon as the size of the open list $|Q|$ exceeds $\phi$ the algorithm starts to merge nodes from $Q$. If the MDD construction process is carried out until the open list becomes empty a complete relaxed MDD is obtained. Alternatively, we may again terminate as soon as $\mathbf{t}$ is the first time selected for expansion. Then we obtain in general an incomplete relaxed MDDs where not all feasible solutions may be contained, however, due to the optimality condition of A* search the length of the longest path from $\mathbf{r}$ to $\mathbf{t}$—that is $Z^{\text{lp}}(\mathbf{t})$—is a valid upper bound to the length of the LCS. We denote this upper bound by $Z^{\text{ub}}_{\text{min}}$. Note that this bound cannot further be improved by continuing the MDD construction.

**Merging.** If $|Q|$ exceeds $\phi$ then nodes are selected in a pairwise fashion for merging. This must be done carefully since we have to ensure that no cycles emerge and that the open list gets empty after a finite number of expansions. Furthermore we do not merge nodes which are already expanded since this would require to update all successor states from the expanded node onward. Note that, since nodes are selected from $Q$ for merging, this approach is able to merge nodes across layers, by introducing so called "long arcs" that skip certain layers. Moreover, to compile relaxed MDDs with A*C the recursive problem formulation does not necessarily have to be based on layers at all.

To do the selection for pairing, we label each node $u \in V(\mathcal{M})$ by a labeling function $\mathcal{L}(u)$ that maps the state $\mathbf{p}(u)$ to a simpler label of a restricted finite domain $\mathcal{D}_{\mathcal{L}}$. The idea is that nodes with the same label are considered similar s.t. the merged state is still a reasonable representative for both nodes. Hence, we only merge nodes with the same label. Moreover, labels are chosen in such way that no cycles will emerge through merging and the open list gets empty in a finite number of steps.

To efficiently select partner nodes for merging we use a global set of so-called collector nodes $V^{\mathrm{c}}$ which is realized as a dictionary indexed by the labels and is initially empty. As long as $Q$ is too large, nodes that are not yet expanded are selected from it in increasing $Z^{\mathrm{lp}}$-order. If for a selected node already a collector node $V^{\mathrm{c}}$ with the same label exists then the two nodes get merged s.t. all incoming arcs from the two nodes will be redirected to the new merged node. The two original nodes are removed from $Q$ and $V^{\mathrm{c}}$ and the new merged node is integrated into $V(\mathcal{M})$ and becomes a new collector node in $V^{\mathrm{c}}$. Note that during the whole construction process we never allow multiple nodes for the same state. For more details we refer to [21].

*Static Labeling Function.* For the LCS problem we label all nodes $u \in Q$ by

$$\mathcal{L}_1(u) = (p_{i_{\mathrm{lcs1}}}(u), p_{i_{\mathrm{lcs2}}}(u)) \tag{7}$$

where $i_{\mathrm{lcs1}}$, $i_{\mathrm{lcs2}}$ are two specific indexes that refer to a pair of input strings $\{s_{i_{\mathrm{lcs1}}}, s_{i_{\mathrm{lcs2}}}\} \in S$ with smallest $\mathbf{M}^3_{s_i,s_{i'}}[0,0]$ over all $(s_i, s_{i'}) \in P$. Hence, we merge only nodes whose states have the same positions in strings $s_{i_{\mathrm{lcs1}}}$ and $s_{i_{\mathrm{lcs2}}}$ and thus partially represent the same subproblems. Consequently, the longest path in the relaxed MDD will never be worse than the upper bound obtained from the corresponding scoring matrix and each path originating from $\mathbf{r}$ will be a feasible common subsequence w.r.t. input strings $\{s_{i_{\mathrm{lcs1}}}, s_{i_{\mathrm{lcs2}}}\} \subseteq S$. Since any merged node will have the same values for $p_{i_{\mathrm{lcs1}}}$ and $p_{i_{\mathrm{lcs2}}}$ as the original nodes, and each transition from a state to a corresponding successor state increases the values from $p_{i_{\mathrm{lcs1}}}$ and $p_{i_{\mathrm{lcs2}}}$, the values $p_{i_{\mathrm{lcs1}}}$ and $p_{i_{\mathrm{lcs2}}}$ strictly increase along each path in the relaxed MDD. Consequently, no cycles can occur and the open list gets empty within an finite number of iterations.

*Dynamic Labeling Function.* To derive stronger relaxed MDDs we investigate further the static labeling function

$$\mathcal{L}_2(u) = (p_{i_{\mathrm{lcs1}}}(u), p_{i_{\mathrm{lcs2}}}(u), p_{i_{\mathrm{lcs3}}}(u), p_{i_{\mathrm{lcs4}}}(u)) \tag{8}$$

where $\{s_{i_{\mathrm{lcs3}}}, s_{i_{\mathrm{lcs4}}}\} \in S$ is the additional pair of input strings with smallest $\mathbf{M}^3_{s_i,s_{i'}}[0,0]$ over all $(s_i, s_{i'}) \in P \setminus \{(i_{\mathrm{lcs1}}, i_{\mathrm{lcs2}})\}$. Note that the convergence speed of A*C depends on the size of the domain $|\mathcal{D}_{\mathcal{L}}|$ of the used labeling function $\mathcal{L}$. If the domain size is large then nodes can be grouped into many subgroups and it may be harder to keep the open list size under the desired threshold value $\phi$ since there are fewer possibilities to merge nodes. If the domain size is small then nodes are merged more aggressively, which makes it easier to keep

the open list size under $\phi$. However, the finally compiled relaxed MDD will in general be weaker than a relaxed MDD compiled with a labeling function of a larger domain size. For $\mathcal{L}_1$ the domain size is $|\mathcal{D}_{\mathcal{L}_1}| = \sum_{a \in \Sigma} |s_{i_{\text{lcs1}}}|_a \, |s_{i_{\text{lcs2}}}|_a$. Preliminary results showed that the domain size of $\mathcal{L}_2$ is already too large to let A*C finish in reasonable time on our benchmark instances, but the obtained relaxed MDDs have the potential to be stronger than relaxed MDDs compiled with $\mathcal{L}_1$. Therefore we follow a different strategy: Instead of using a labeling function that is static over the whole compilation process we use a function that adapts its domain size depending on the current situation. We propose the labeling function

$$\mathcal{L}_{2,\Delta}(u) = (p_{i_{\text{lcs1}}}(u),\, p_{i_{\text{lcs2}}}(u),\, \lfloor p_{i_{\text{lcs3}}}(u)/\Delta \rfloor,\, \lfloor p_{i_{\text{lcs4}}}(u)/\Delta \rfloor) \tag{9}$$

which discretizes the values for $p_{i_{\text{lcs3}}}$ and $p_{i_{\text{lcs4}}}$ by discretization factor $\Delta$. A*C starts with $\Delta = 1$ and doubles this parameter after every $k$ consecutive failures of reducing the open list size below $\phi$. If the open list size could be reduced to size $\phi$ then $\Delta$ is reset to one. Each time $\Delta$ is adapted, the set of collector nodes $V^{\text{c}}$ is cleared.

### 4.2 Further Details

Similar to [21], in we merge an already expanded node $u \in V(\mathcal{M})$ and a not yet expanded node $v \in Q$ if $\mathbf{p}(v) \oplus \mathbf{p}(u) = \mathbf{p}(u)$, $Z^{\text{lp}}(v) \leq Z^{\text{lp}}(u)$, and $\mathcal{L}_1(u) = \mathcal{L}_1(v)$ holds since we do not need to update the state of node $u$. This is efficiently done by indexing each expanded node by labeling function $\mathcal{L}_1$ and checking the condition after each node expansion for each newly created node.

## 5 Experimental Results

To test our approaches we use six benchmark sets from the literature.

**BL instance set from** [10]**:** 450 instances grouped by different values for $m$, $n$, and $|\Sigma|$. For each combination there are ten uniform random instances.

**Rat, Virus, and Random instance set from** [29]**:** Three benchmark sets consisting of 20 instances each. The Rat and Virus benchmark sets have a biological background whereas instances of the Random benchmark sets are randomly generated.

**ES instance set from** [17]**:** 600 instances grouped by different values for $m$, $n$ and $|\Sigma|$, where each group includes 50 instances.

**BB instance set from** [7]**:** 800 instances that were artificially generated in a way s.t. input strings have a large similarity to each other. There are ten instances for each combination of $m$ and $|\Sigma|$.

We used all of these instances for the experimental evaluation but report here only some due to the lack of space. In particular, the main results table to come in Sect. 5.3 omits data for sets Virus and Random since they are similar to those

obtained for Rat, and from set BL only instances with $n = 100$ are considered as also done in [16]. However, all instances from all mentioned benchmark sets are considered in all the boxplots to come. Complete results over all benchmark instances are available from https://www.ac.tuwien.ac.at/files/resources/results/LCS/cpaior21_mdds.zip. The algorithms were implemented using GNU C++ 7.5.0, and all experiments were performed on a single core of an Intel Xeon E5649 with 2.53 GHz and 32 GB RAM.

To evaluate the A*C algorithm we use the standard *top-down construction* (TDC) as baseline, which compiles a relaxed MDD layer by layer starting with the root node $\mathbf{r}$. All nodes of the current layer $L_i$, $i = 1, \ldots, n$, are expanded and the newly created nodes are inserted into layer $L_{i+1}$. The layer size is limited by parameter $\beta$. If the size of $L_{i+1}$ exceeds $\beta$ then $L_{i+1}$ is reduced, after all nodes of $L_i$ are expanded, by sorting the nodes according to priority function $f$ in non-increasing order and replacing the last nodes with smallest $f$-values from position $\beta$ onward into a single merged node. Note that TDC in general yields an MDD with be multiple target nodes at different layers. In this case the notation $Z^{\text{lp}}(\mathbf{t})$ refers to the length of the longest path from $\mathbf{r}$ to any target node.

### 5.1 Comparison of Independent Upper Bounds

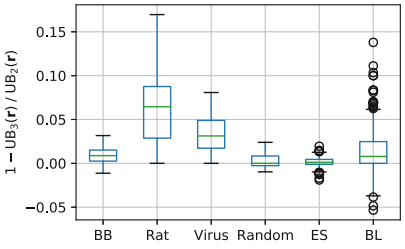


**Fig. 1.** Relative differences of upper bounds $\text{UB}_2(\mathbf{r})$ and $\text{UB}_3(\mathbf{r})$.

We start with a comparison of the upper bounds $\text{UB}_2(\mathbf{r})$ and $\text{UB}_3(\mathbf{r})$ from Sect. 3. Figure 1 shows boxplots for the relative differences $1 - \text{UB}_3(\mathbf{r})/\text{UB}_2(\mathbf{r})$ over the different benchmark sets. Over all instances, tighter upper bounds can be obtained from $\text{UB}_3(\mathbf{r})$ than from $\text{UB}_2(\mathbf{r})$ in 62.2% of the cases, and in these the relative difference is on average 1.6%. Both upper bounds are equal in 17.6% of all instances. Overall, upper bound $\text{UB}_3(\mathbf{r})$ has on average a relative difference to $\text{UB}_2(\mathbf{r})$ of 0.9%. However, differences vary significantly with the type of benchmarks as the figure shows. The largest relative differences could be observed on benchmark sets Rat and Virus. For randomly generated instances, the relative differences seems to be smaller in general. Overall, we conclude that $\text{UB}_3$ provides in general slightly tighter upper bounds than $\text{UB}_2$ but does not dominate it. As both bounding procedures are relatively fast, we conclude that their joint application makes sense.

### 5.2 Impact of Parameters $\phi$ and $\beta$

Next we investigate the impact of parameter $\phi$ as well as the choice of the labeling function on the quality of the obtained relaxed MDDs. For this purpose we compile MDDs for middle size instances from benchmark set BB with $m = 100$, $n = 1000$, and $|\Sigma| = 8$. Figure 2 depicts aggregated characteristics of the relaxed

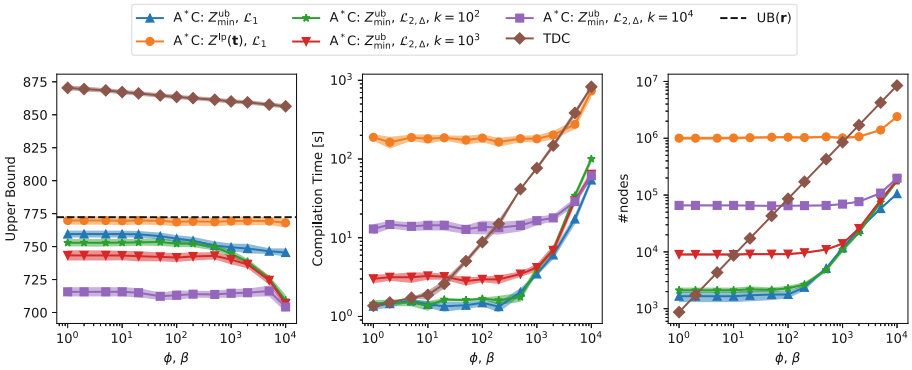


**Fig. 2.** Relaxed MDDs obtained by A*C and TDC for different settings of $\phi$ and $\beta$ for benchmark set BB, $n = 1000$, $m = 100$, $|\Sigma| = 8$

MDDs created by A*C and TDC, respectively. The diagram to the left shows obtained upper bounds, i.e., average lengths of longest **r**-**t** paths, for different values of $\phi$ and $\beta$ in the range of 1 to $10^4$. The different solid lines represent different choices of labeling functions for A*C as well as results obtained from TDC. The small tubes around the lines indicate corresponding standard deviations. For A*C we generally report the upper bound values $Z^{ub}_{min}$ obtained when **t** was selected the first time for expansion, and in case of labeling function $\mathcal{L}_1$ additionally the longest path lengths in the complete relaxed MDDs. The dashed line indicates the combined bound UB(**r**) from Sect. 3. The diagrams in the middle and to the right report the corresponding average computation times in seconds and average numbers of nodes of the relaxed MDDs, respectively.

In general we can observe that tighter upper bounds can be obtained when choosing larger values for $\phi$ or $\beta$. Naturally, this comes at the cost of larger compilation times and lager relaxed MDDs. In comparison to TDC, A*C provides consistently much better results in terms of tightness of obtained upper bounds and for larger values of $\phi$ and $\beta$ also in terms of compilation time and compactness of obtained relaxed MDDs. A*C with the dynamic labeling function $\mathcal{L}_{2,\Delta}$ yields stronger bounds than with $\mathcal{L}_1$, requires, however, more time than $\mathcal{L}_1$. This is not surprising since domain $\mathcal{D}_{\mathcal{L}_{2,\Delta}}$ is larger than $\mathcal{D}_{\mathcal{L}_1}$ and thus leads less frequently to merges. The tightest upper bounds can be obtained with function $\mathcal{L}_{2,\Delta}$ where the discretization factor $\Delta$ is doubled after every $k = 10^4$ consecutive failures of reducing $Q$ below $\phi$. Again this can be explained due to less merges than with other parameter settings. For the same reason these settings need in general more computation time and produce larger relaxed MDDs. Note also that, even for small values of $\phi$, upper bounds obtained from A*C are substantially smaller than UB(**r**).

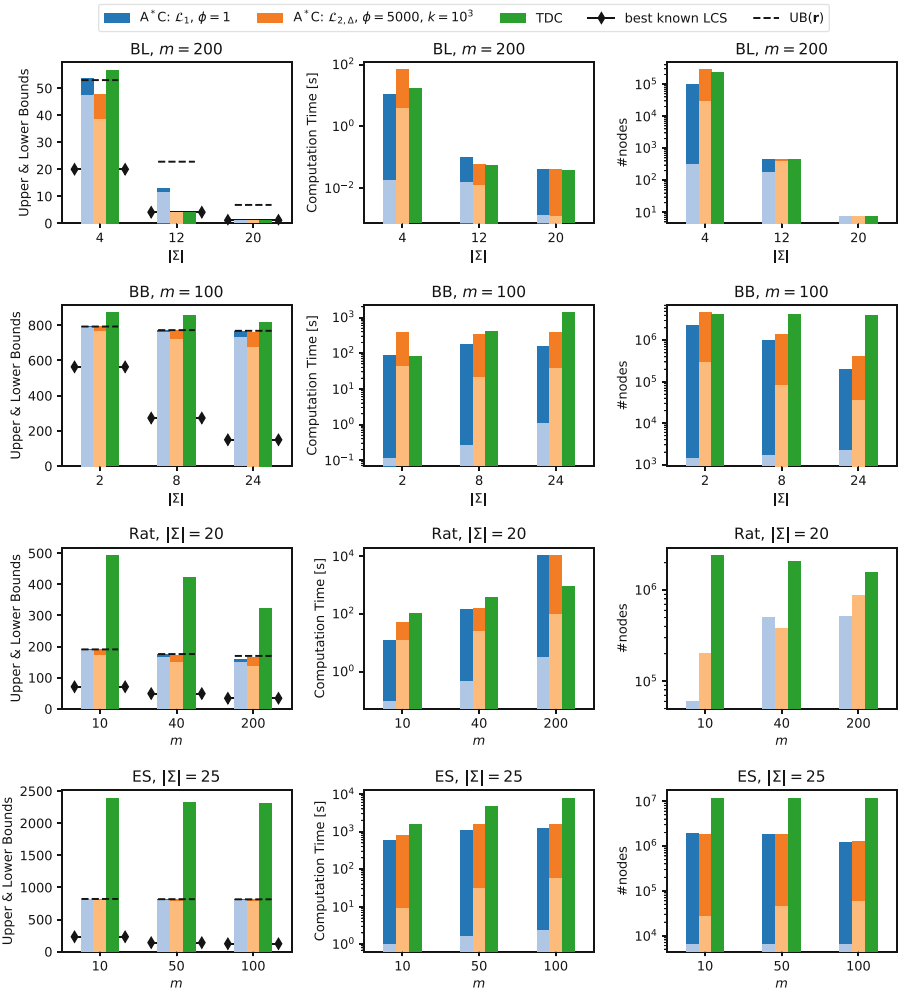


**Fig. 3.** Lower and upper bounds, respective compilation times, and sizes of obtained relaxed MDDs for selected benchmark sets.

### 5.3 Main Comparison of A*C and TDC

We start with a graphical comparison for a selected subset of instance classes in Fig. 3. Shown are upper bounds obtained from relaxed MDDs compiled with A*C and TDC, respectively, corresponding compilation times, and the sizes of the obtained MDDs. Each group of bars corresponds to a specific instance class and shows average results, except for instance class Rat which contains only one instance per instance class. The first two bars from the left to right always correspond to relaxed MDDs obtained from A*C with parameters $\{\mathcal{L}_1, \phi = 1\}$ and $\{\mathcal{L}_{2,\Delta}, \phi = 5000, k = 10^3\}$, respectively. The first parameter setting is the case where A*C merges nodes most aggressively whereas the latter setting

lets A*C select nodes for merging more carefully, but still with a reasonable total compilation time. The third bar corresponds to relaxed MDDs obtained from TDC with $\beta = 5000$. The brighter parts of the bars indicate the results for the in general incomplete relaxed MDDs obtained when A*C terminates as soon as $\mathbf{t}$ is selected for expansion whereas the darker parts show the results for the completed relaxed MDDs. For instance, the brighter part of the bars in the diagrams on the left side show average $Z^{\text{ub}}_{\text{min}}$ values. Diamond markers indicate the average lengths of the best known LCSs from the literature obtained from [16]. The black dashed lines show the independent upper bounds UB($\mathbf{r}$).

We can see that if A*C terminates as soon as $\mathbf{t}$ is selected for expansion then we obtain in all considered cases MDDs yielding significantly tighter bounds than the MDDs obtained from TDC. Moreover, compilation times are shorter and the obtained MDDs are smaller in case of A*C. Note that although these relaxed MDDs are incomplete in the sense that not all feasible solutions are covered, they can still be further used, e.g., for the DD-based branch-and-bound approach as described by Bergman et al. [3]. It is still possible to derive an exact cut set of nodes to branch on by considering nodes that are not expanded yet, too. If we consider complete relaxed MDDs from A*C then the obtained upper bounds are still tighter or equal than those from relaxed MDDs obtained from TDC, however the compilation with A*C is not faster anymore. Note that TDC was not able to compile relaxed MDDs with $\beta = 5000$ within the time limit of three hours for instances from set ES with $n = 5000$. Also the A*C approach could not compile a complete relaxed MDD for instances of set Rat with $m = 200$, $|\Sigma| = 20$, and $n = 600$ within the three hours time limit. However, with the stopping condition of selecting $\mathbf{t}$ for expansion, A*C terminated much earlier. As the length of the longest path of the incomplete relaxed MDD when A*C aborts after three hours is also a feasible upper bound, we show these values in these cases, too.

Finally, Table 1 presents more detailed main results of our computational experiments. Here, A*C is always terminated when $\mathbf{t}$ is selected for expansion. Each row contains aggregated results of one instance class. The characteristics of the instance classes can be seen in the first four columns whereas column UB($\mathbf{r}$) shows the average independent upper bound. The next eight columns belong to results obtained from relaxed MDDs compiled with A*C and TDC, respectively. Hereby, columns $\overline{Z^{\text{ub}}_{\text{min}}}$ and $\overline{Z^{\text{lp}}(\mathbf{t})}$ state the average lengths of the longest paths obtained from the compiled MDDs. Columns $\sigma(\cdot)$ report corresponding standard deviations. Average compilation times in seconds are listed in columns $t$. Finally, columns gap report the remaining optimality gaps (ub−obj)/ub·100% in relation to the objective values of so far best known solutions obtained from [16] and listed in column obj; value ub refers to the upper bound obtained from the considered approach, i.e., $Z^{\text{ub}}_{\text{min}}$ or $Z^{\text{lp}}$. We remark that [16] shows experimental results for two parameter settings, one tailored to obtain as good as possible heuristic solutions, and one targeted towards smallest possible remaining optimality gaps. While we use the better objective values from the former results, the gaps listed in our table for [16] are those of the latter.

**Table 1.** Main results for A*C and TDC and comparison to the anytime A* search from [16].

| | | | | | A*C | | | | TDC | | | | lit. best [16] | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n$ | $\lvert\Sigma\rvert$ | $m$ | UB($\mathbf{r}$) | $\overline{Z^{\text{ub}}_{\min}}$ | $\sigma(\overline{Z^{\text{ub}}_{\min}})$ | $t$[s] | gap[%] | $\overline{Z^{\text{lp}}(\mathbf{t})}$ | $\sigma(\overline{Z^{\text{lp}}(\mathbf{t})})$ | $t$[s] | gap[%] | obj | gap[%] |
| BB | 1000 | 2 | 10 | 807.4 | **781.6** | 9.1 | 8.9 | **13.4** | 882.7 | 4.4 | 18.7 | 23.3 | 676.7 | 16.2 |
| | | | 100 | 792.7 | **767.3** | 4.5 | 43.2 | **26.5** | 871.8 | 4.3 | 74.2 | 35.4 | 563.6 | 30.6 |
| | | 4 | 10 | 796.5 | **759.5** | 6.7 | 6.4 | **28.2** | 879.5 | 4.1 | 27.7 | 38.0 | 545.5 | 29.4 |
| | | | 100 | 779.0 | **739.4** | 8.2 | 22.5 | **47.2** | 868.2 | 4.7 | 181.3 | 55.1 | 390.2 | 50.9 |
| | | 8 | 10 | 794.8 | **732.8** | 11.3 | 7.7 | **36.9** | 874.9 | 5.7 | 45.5 | 47.1 | 462.7 | 38.0 |
| | | | 100 | 772.3 | **708.2** | 5.3 | 23.1 | **61.4** | 857.6 | 3.3 | 386.4 | 68.1 | 273.4 | 65.0 |
| | | 24 | 10 | 786.1 | **689.1** | 14.5 | 12.5 | 44.0 | 846.9 | 3.5 | 131.8 | 54.5 | 385.6 | **40.5** |
| | | | 100 | 768.4 | **669.8** | 9.9 | 42.0 | **77.7** | 818.3 | 1.8 | 1261.4 | 81.7 | 149.5 | 79.5 |
| Rat | 600 | 4 | 10 | 345.0 | **319.0** | – | 4.8 | **35.4** | 570.0 | – | 27.7 | 63.9 | 206.0 | 38.0 |
| | | | 15 | 347.0 | **331.0** | – | 5.2 | **42.9** | 564.0 | – | 20.6 | 66.5 | 189.0 | 44.5 |
| | | | 20 | 293.0 | **277.0** | – | 9.4 | **37.2** | 494.0 | – | 34.4 | 64.8 | 174.0 | 39.5 |
| | | | 25 | 344.0 | **327.0** | – | 5.4 | **47.1** | 557.0 | – | 44.1 | 68.9 | 173.0 | 47.4 |
| | | | 40 | 315.0 | **300.0** | – | 10.6 | 48.7 | 455.0 | – | 26.2 | 66.2 | 154.0 | **48.1** |
| | | | 60 | 343.0 | **323.0** | – | 5.1 | **52.3** | 548.0 | – | 62.5 | 71.9 | 154.0 | 53.1 |
| | | | 80 | 281.0 | **261.0** | – | 12.6 | **44.8** | 466.0 | – | 36.6 | 69.1 | 144.0 | 47.6 |
| | | | 100 | 279.0 | **263.0** | – | 5.5 | **47.1** | 497.0 | – | 118.5 | 72.0 | 139.0 | 49.6 |
| | | | 150 | **222.0** | **222.0** | – | 13.2 | 41.0 | 443.0 | – | 142.3 | 70.4 | 131.0 | **40.2** |
| | | | 200 | 231.0 | **228.0** | – | 40.3 | **44.7** | 436.0 | – | 223.7 | 71.1 | 126.0 | 44.9 |
| | | 20 | 10 | 191.0 | **167.0** | – | 19.6 | **56.9** | 493.0 | – | 101.3 | 85.4 | 72.0 | 58.7 |
| | | | 15 | 198.0 | **169.0** | – | 45.1 | **62.7** | 467.0 | – | 268.2 | 86.5 | 63.0 | 62.9 |
| | | | 20 | 190.0 | **159.0** | – | 101.7 | 65.4 | 456.0 | – | 278.2 | 87.9 | 55.0 | **65.2** |
| | | | 25 | 173.0 | **145.0** | – | 18.5 | **64.1** | 417.0 | – | 158.6 | 87.5 | 52.0 | 68.1 |
| | | | 40 | 176.0 | **143.0** | – | 53.0 | **65.0** | 421.0 | – | 379.6 | 88.1 | 50.0 | 70.3 |
| | | | 60 | 195.0 | **161.0** | – | 439.7 | 70.8 | 431.0 | – | 284.2 | 89.1 | 47.0 | **70.3** |
| | | | 80 | 180.0 | **145.0** | – | 518.7 | 69.7 | 376.0 | – | 269.5 | 88.3 | 44.0 | **69.1** |
| | | | 100 | 173.0 | **138.0** | – | 103.5 | **71.0** | 359.0 | – | 545.3 | 88.9 | 40.0 | 71.8 |
| | | | 150 | 172.0 | **145.0** | – | 128.0 | 73.8 | 323.0 | – | 609.6 | 88.2 | 38.0 | **71.5** |
| | | | 200 | 170.0 | **133.0** | – | 195.7 | 73.7 | 324.0 | – | 897.6 | 89.2 | 35.0 | **70.2** |
| ES | 1000 | 2 | 10 | 795.3 | **783.6** | 4.3 | 5.6 | **21.0** | 987.5 | 1.3 | 19.7 | 37.3 | 618.9 | 21.2 |
| | | | 50 | 791.0 | **779.4** | 3.0 | 12.8 | **30.6** | 982.7 | 1.2 | 40.8 | 45.0 | 540.9 | **30.6** |
| | | | 100 | 788.7 | **777.3** | 3.0 | 18.4 | **32.8** | 980.8 | 0.9 | 77.6 | 46.8 | 522.1 | 32.9 |
| | | 10 | 10 | 477.6 | **462.2** | 2.9 | 4.9 | 55.6 | 951.8 | 2.7 | 138.4 | 78.5 | 205.0 | **54.9** |
| | | | 50 | 473.7 | **455.7** | 1.8 | 15.4 | 69.8 | 928.7 | 2.1 | 339.4 | 85.2 | 137.5 | **69.1** |
| | | | 100 | 472.2 | **454.0** | 2.0 | 28.9 | 72.7 | 919.5 | 2.1 | 591.8 | 86.5 | 124.1 | **71.9** |
| | 2500 | 25 | 10 | 820.1 | **800.1** | 2.4 | 11.5 | 70.4 | 2389.2 | 4.3 | 1453.7 | 90.1 | 236.6 | **70.1** |
| | | | 50 | 816.5 | **791.0** | 1.7 | 39.1 | 82.3 | 2332.4 | 4.5 | 4367.0 | 94.0 | 140.4 | **81.9** |
| | | | 100 | 814.4 | **788.3** | 1.4 | 74.2 | 84.3 | 2309.5 | 3.6 | 7514.3 | 94.7 | 123.4 | **84.0** |
| | 5000 | 100 | 10 | 888.3 | **853.9** | 2.6 | 62.7 | **82.9** | – | – | – | – | 145.7 | **82.9** |
| | | | 50 | 883.5 | **835.9** | 1.7 | 152.1 | 91.4 | – | – | – | – | 72.0 | **91.3** |
| | | | 100 | 882.3 | **829.5** | 1.6 | 373.3 | 92.7 | – | – | – | – | 60.8 | **92.6** |
| BL | 100 | 4 | 10 | 58.8 | **47.5** | 1.6 | 0.5 | 28.2 | 75.6 | 2.0 | 3.0 | 54.9 | 34.1 | **10.8** |
| | | | 50 | 56.2 | **41.7** | 1.4 | 2.1 | 42.0 | 65.0 | 1.2 | 6.1 | 62.8 | 24.2 | **18.7** |
| | | | 100 | 54.7 | **40.6** | 1.1 | 3.2 | 45.8 | 61.0 | 1.8 | 9.6 | 63.9 | 22.0 | **20.4** |
| | | | 150 | 53.8 | **38.7** | 1.2 | 3.9 | 46.8 | 58.0 | 1.4 | 11.6 | 64.5 | 20.6 | **18.1** |
| | | | 200 | 53.0 | **38.3** | 0.8 | 5.0 | 47.8 | 56.8 | 1.8 | 15.9 | 64.8 | 20.0 | **20.2** |
| | | 12 | 10 | 37.4 | **21.2** | 1.7 | 0.2 | 40.1 | 36.3 | 4.1 | 3.7 | 65.0 | 12.7 | **0.0** |
| | | | 50 | 34.4 | **8.7** | 2.1 | 0.2 | 20.7 | 9.6 | 3.0 | 0.3 | 28.1 | 6.9 | **0.0** |
| | | | 100 | 28.8 | **5.2** | 0.4 | <0.1 | **0.0** | **5.2** | 0.4 | <0.1 | **0.0** | 5.2 | **0.0** |
| | | | 150 | 23.8 | **4.7** | 0.5 | <0.1 | **0.0** | **4.7** | 0.5 | <0.1 | **0.0** | 4.7 | **0.0** |
| | | | 200 | 22.8 | **4.1** | 0.3 | <0.1 | **0.0** | **4.1** | 0.3 | <0.1 | **0.0** | 4.1 | **0.0** |
| | | 20 | 10 | 29.2 | **9.5** | 1.0 | <0.1 | 16.8 | 10.5 | 2.2 | 0.3 | 24.8 | 7.9 | **0.0** |
| | | | 50 | 17.5 | **3.0** | 0.0 | <0.1 | **0.0** | **3.0** | 0.0 | <0.1 | **0.0** | 3.0 | **0.0** |
| | | | 100 | 12.1 | **2.1** | 0.3 | <0.1 | **0.0** | **2.1** | 0.3 | <0.1 | **0.0** | 2.1 | **0.0** |
| | | | 150 | 7.2 | **1.9** | 0.3 | <0.1 | **0.0** | **1.9** | 0.3 | <0.1 | **0.0** | 1.9 | **0.0** |
| | | | 200 | 6.8 | **1.1** | 0.3 | <0.1 | **0.0** | **1.1** | 0.3 | <0.1 | **0.0** | 1.1 | **0.0** |

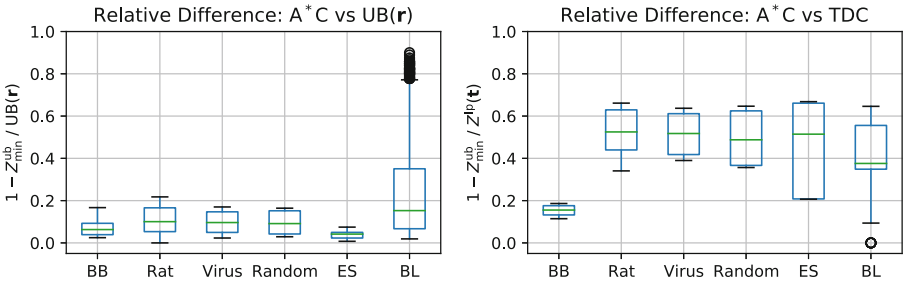


**Fig. 4.** Relative differences of upper bounds between $Z^{\text{ub}}_{\min}$ and $\text{UB}(\mathbf{r})$ as well as between $Z^{\text{ub}}_{\min}$ and $Z^{\text{lp}}(\mathbf{t})$.

For the compilation of MDDs we set $\beta = 5000$ for TDC and $\phi = 5000$ for A*C with labeling function $\mathcal{L}_{2,\Delta}$ and $k = 10^4$ for all instance except for benchmark set ES where $k$ is set to $10^3$.

We observe that in all considered cases the obtained upper bounds $Z^{\text{ub}}_{\min}$ are tighter than $\text{UB}(\mathbf{r})$ as well as the upper bounds obtained from relaxed MDDs compiled with TDC. Only in one single case, for benchmark set Rat with $|\Sigma| = 4$, $m = 150$, $n = 600$, the upper bound $\text{UB}(\mathbf{r})$ is equal to $Z^{\text{ub}}_{\min}$. We notice an average relative difference between $Z^{\text{ub}}_{\min}$ and $\text{UB}(\mathbf{r})$ of 14.8% over all instances. Considering $Z^{\text{ub}}_{\min}$ and $Z^{\text{lp}}(\mathbf{t})$ from relaxed MDDs compiled with TDC we get an average relative difference of 43.7%. The boxplots shown in Fig. 4 give deeper insight on the relative differences between $Z^{\text{ub}}_{\min}$ and $\text{UB}(\mathbf{r})$ as well as the differences between $Z^{\text{ub}}_{\min}$ and $Z^{\text{lp}}(\mathbf{t})$. The largest relative difference between upper bounds obtained from relaxed MDDs compiled by A*C and TDC occurs for instance sets Rat, Virus, Random, and ES. For these benchmark sets the median of the obtained relative differences is about 50%. Regarding instances of the BB benchmark set, substantially smaller relative differences are obtained. The fact that BB instances are created in a way s.t. input strings have a large similarity to each other seems to be an explanation for this discrepancy. The median of the relative differences between upper bounds $Z^{\text{ub}}_{\min}$ and $\text{UB}(\mathbf{r})$ is about 10% for all benchmark sets. Only results from benchmark set ES exhibit a median relative difference of about 4%, which can be explained by the longer input strings of ES instances, e.g., $n = 5000$. Finally, BL instances exhibit some outliers, e.g., instances with a relative difference between $Z^{\text{ub}}_{\min}$ and $\text{UB}(\mathbf{r})$ of 80% and differences between $Z^{\text{ub}}_{\min}$ and $Z^{\text{lp}}(\mathbf{t})$ (TDC) of 0%. This is not surprising, since benchmark set BL contains small instances that could be solved to proven optimality by exact methods, and both construction methods, A*C as well as TDC, are able to compile relaxed MDDs that yield the optimal solution values as upper bounds. This is also documented in Table 1 for instance classes of set BL with $n = 100$ and $|\Sigma| \in \{12, 20\}$ where the average optimality gap is 0%. In comparison to [16], we can observe that A*C is able to obtain even smaller optimality gaps in 315 cases and equal optimality gaps in 73 cases. Most of the gaps from

[16] were only obtained after a time limit of 15 min, while A*C created the MDDs in much shorter time.

## 6 Conclusions

In this work we compiled relaxed MDDs for the LCS problem to obtain upper bounds. The proposed construction algorithm A*C is not layer-oriented such as TDC and utilizes fast independent upper bounds on subproblems for guidance in the spirit of A* search. As independent upper bound we suggested using a combination of two fast-to-calculate bounds from the literature and the new variant $UB_3$ that is approximately equally fast to compute but occasionally stronger than the former bounds. To control the size of the relaxed MDD, A*C merges nodes when the list of open nodes exceeds a certain size. To determine suitable partner nodes for merging, we investigated different LCS-specific labeling functions. The better performing dynamic labeling function adapts the domain dynamically during the compilation process s.t. depending on the current situation nodes are merged more or less aggressively. When rigorously comparing A*C with a classical TDC on several benchmark instance sets from the literature, we observed that A*C is able to provide more compact relaxed MDDs that are significantly stronger than relaxed MDDs obtained from TDC in shorter time. For several instance classes relaxed MDDs compiled with A*C yielded stronger bounds than the best known upper bounds from the literature.

For future work it seems promising to embed the compilation of relaxed MDDs into a branch-and-bound approach that branches over exact nodes of the relaxed MDD, as already done in the literature for other kinds of problems, where, however, the classical TDC was used instead of A*C to compile relaxed MDDs. To obtain also high quality heuristic solutions for subproblems within such a branch-and-bound approach, ideas from the leading beam search methods can further be adopted. Further interesting research directions will be to investigate different strategies for the novel dynamic labeling functions mechanism and to perform more detailed analysis of the ability of A*C to reduce the size of relaxed MDDs, e.g. comparing for small instances the size of exact reduced MDDs with relaxed MDDs compiled with A*C.

## References

1. Andersen, H.R., Hadzic, T., Hooker, J.N., Tiedemann, P.: A constraint store based on multivalued decision diagrams. In: Bessière, C. (ed.) CP 2007. LNCS, vol. 4741, pp. 118–132. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74970-7_11
2. Beal, R., Afrin, T., Farheen, A., Adjeroh, D.: A new algorithm for "the LCS problem" with application in compressing genome resequencing data. BMC Genom. **17**(4), 544 (2016). https://doi.org/10.1186/s12864-016-2793-0
3. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.N.: Discrete optimization with decision diagrams. INFORMS J. Comput. **28**(1), 47–66 (2016)

4. Bergman, D., Cire, A.A., von Hoeve, W.J., Hooker, J.N.: Optimization bounds from binary decision diagrams. INFORMS J. Comput. **26**(2), 253–268 (2014)
5. Bergman, D., Cire, A.A., van Hoeve, W.J., Hooker, J.N.: Decision Diagrams for Optimization. Artificial Intelligence: Foundations, Theory, and Algorithms. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-319-42849-9
6. Bergman, D., Cire, A.A., van Hoeve, W.-J., Yunes, T.: BDD-based heuristics for binary optimization. J. Heuristics **20**(2), 211–234 (2014). https://doi.org/10.1007/s10732-014-9238-1
7. Blum, C., Blesa, M.J.: Probabilistic beam search for the longest common subsequence problem. In: Stützle, T., Birattari, M., Hoos, H.H. (eds.) SLS 2007. LNCS, vol. 4638, pp. 150–161. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74446-7_11
8. Blum, C., Blesa, M.J., López-Ibáñez, M.: Beam search for the longest common subsequence problem. Comput. Oper. Res. **36**(12), 3178–3186 (2009)
9. Blum, C., et al.: Solving longest common subsequence problems via a transformation to the maximum clique problem. Comput. Oper. Res. **125**, 105089 (2021). https://doi.org/10.1016/j.cor.2020.105089
10. Blum, C., Festa, P.: Longest common subsequence problems. In: Metaheuristics for String Problems in Bioinformatics, chap. 3, pp. 45–60. Wiley (2016)
11. Bonizzoni, P., Della Vedova, G., Mauri, G.: Experimenting an approximation algorithm for the LCS. Discret. Appl. Math. **110**(1), 13–24 (2001)
12. Brisk, P., Kaplan, A., Sarrafzadeh, M.: Area-efficient instruction set synthesis for reconfigurable system-on-chip designs. In: Proceedings of DAC 2004 - the 41st Annual Design Automation Conference, pp. 395–400. IEEE Press (2004)
13. Chan, H.T., Yang, C.B., Peng, Y.H.: The generalized definitions of the two-dimensional largest common substructure problems. In: Proceedings of the 33rd Workshop on Combinatorial Mathematics and Computation Theory, pp. 1–12. National Taiwan University (2016)
14. Cire, A.A., van Hoeve, W.J.: Multivalued decision diagrams for sequencing problems. Oper. Res. **61**(6), 1411–1428 (2013)
15. 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., Pardalos, P., Umeton, R., Giuffrida, G., Sciacca, V. (eds.) LOD 2019. LNCS, vol. 11943, pp. 154–167. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-37599-7_14
16. Djukanovic, M., Raidl, G.R., Blum, C.: Finding longest common subsequences: new anytime A* search results. Appl. Soft Comput. **95**, 106499 (2020). https://doi.org/10.1016/j.asoc.2020.106499
17. Easton, T., Singireddy, A.: A large neighborhood search heuristic for the longest common subsequence problem. J. Heuristics **14**(3), 271–283 (2008). https://doi.org/10.1007/s10732-007-9038-y
18. Fraser, C.B.: Subsequences and supersequences of strings. Ph.D. thesis, University of Glasgow, UK (1995)
19. Gusfield, D.: Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, Cambridge (1997)
20. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci. Cybern. **4**(2), 100–107 (1968)
21. Horn, M., Maschler, J., Raidl, G.R., Rönnberg, E.: A*-based construction of decision diagrams for a prize-collecting scheduling problem. Comput. Oper. Res. **126**, 105125 (2021). https://doi.org/10.1016/j.cor.2020.105125

22. Huang, K., Yang, C., Tseng, K.: Fast algorithms for finding the common subsequences of multiple sequences. In: Proceedings of the IEEE International Computer Symposium, pp. 1006–1011. IEEE Press (2004)
23. Jiang, T., Lin, G., Ma, B., Zhang, K.: A general edit distance between RNA structures. J. Comput. Biol. **9**(2), 371–388 (2002)
24. Kinable, J., Cire, A.A., van Hoeve, W.J.: Hybrid optimization methods for time-dependent sequencing problems. Eur. J. Oper. Res. **259**(3), 887–897 (2017)
25. Kruskal, J.B.: An overview of sequence comparison: time warps, string edits, and macromolecules. SIAM Rev. **25**(2), 201–237 (1983)
26. 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. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pp. 1170–1181. IEEE Press (2016)
27. Maier, D.: The complexity of some problems on subsequences and supersequences. J. ACM **25**(2), 322–336 (1978)
28. Peng, Z., Wang, Y.: A novel efficient graph model for the multiple longest common subsequences (MLCS) problem. Front. Genet. **8**, 104 (2017)
29. Shyu, S.J., Tsai, C.Y.: Finding the longest common subsequence for multiple biological sequences by ant colony optimization. Comput. Oper. Res. **36**(1), 73–91 (2009)
30. Smith, T., Waterman, M.: Identification of common molecular subsequences. J. Mol. Biol. **147**(1), 195–197 (1981)
31. Wang, Q., Korkin, D., Shang, Y.: A fast multiple longest common subsequence (MLCS) algorithm. IEEE Trans. Knowl. Data Eng. **23**(3), 321–334 (2011)