Solving Longest Common Subsequence Problems via a Transformation to the Maximum Clique Problem

Christian Blum[1], Marko Djukanovic[2], Alberto Santini[3], Hua Jiang[4], Chu-Min Li[5,6], Felip Manyà[1], and Günter R. Raidl[2]

[1]Artificial Intelligence Research Institute (IIIA-CSIC), Campus UAB, Bellaterra, Spain — *{christian.blum,felip}@iiia.csic.es*
[2]Institute of Logic and Computation, TU Wien, Vienna, Austria — *{djukanovic,raidl}@ac.tuwien.ac.at*
[3]Universitat Pompeu Fabra, Barcelona, Spain — *alberto.santini@upf.edu*
[4]Yunnan University, Kunming, China — *huajiang@ynu.edu.cn*
[5]University of Picardie Jules Verne, Amiens, France — *chu-min.li@u-picardie.fr*
[6]Huazhong University of Science and Technology, Wuhan, China

March 5, 2020

Abstract

Longest common subsequence problems find numerous applications in bioinformatics, data compression and text editing, just to name a few. Even though numerous heuristic approaches were published in the related literature for many of the considered problem variants during the last decades, solving these problems exactly remains an important challenge. This is particularly the case when the number and the length of the input strings grows. In this work we present a way to transform the classical longest common subsequence problem and some of its variants into the maximum clique problem. Moreover, we propose a technique to reduce the size of the resulting graphs. Finally, a comprehensive experimental evaluation of a recent exact maximum clique solver and a recent heuristic maximum clique solver is presented. Numerous problem instances from benchmark sets taken from the literature were solved to optimality in this way.

1 Introduction

One of the common measures when comparing two (or more) strings is the length of their longest common subsequence Iliopoulos and Sohel Rahman (2009); Castelli et al. (2013). A subsequence is a string obtained by possibly deleting characters from another string. For example, AGT is a subsequene of ADDAGTA obtained by deleting the two occurrences of letter D and the last two occurrences of letter A.

The classical *longest common subsequence* (LCS) problem asks to find the longest subsequence common to a given set of strings. The LCS problem is one of the central problems in bioinformatics, often with strings representing segments of RNA or DNA Gusfield (1997); Smith and Waterman (1981); Jiang et al. (2002). Other applications arise in computer science, in the fields of data compression, text editing Kruskal (1983), the production of circuits in field programmable gate arrays Brisk et al. (2004) and file comparison Storer (1988); Aho et al. (1983).

The LCS problem is $\mathcal{NP}$-hard for an arbitrary number of input strings Maier (1978). If the number of strings is a constant, the problem is polynomially solvable by dynamic programming Gusfield (1997). Standard dynamic programming approaches for this problem require a time of $O(n^m)$ where n is the length of the longest input string and m is the number of strings. This complexity requirement makes the LCS problem hard to solve in practice with exact methods.

Real-life applications also require the solution of variants of the LCS problem in which additional constraints are imposed on the solutions. Examples concern the repetition-free longest common subsequence (RFLCS) problem Adi et al. (2010), the constrained longest common subsequence (C-LCS) problem Tsai (2003),

and the generalized constrained longest common subsequence (GC-LCS) problem Chen and Chao (2011). Others are mentioned in survey papers such as Bonizzoni et al. (2010). Henceforth, we refer to the variants of the classical LCS problem, in general, as LCS-type problems.

Despite LCS-type problems being present in the literature for almost forty years, their computational difficulty causes that research is still active on this topic. In particular, in this work we present an approach to solve various LCS-type problems by transforming them into instances of the *maximum clique* (MC) problem Bomze et al. (1999). The central idea of the transformation is to construct, for each instance, a *conflict graph* Lee et al. (2006). Hereby, an independent set in the conflict graph corresponds to a common subsequence concerning the original LCS instance. Moreover, a maximum independent set in the conflict graphs corresponds to a longest common subsequence of the LCS instance. Furthermore, note that finding a *maximum independent set* (MIS) in the conflict graph is equivalent to finding a largest clique on the complement graph of the conflict graph. Therefore, an LCS problem instance can be solved by finding a largest clique in the complement of the conflict graph.

The advantages of this approach are twofold. First, because of a steady improvement of the solvers for the MC problem, we have high-performing algorithms at our disposal that may make solving an MC problem on the complement of the conflict graph faster than solving the original LCS problem with known exact algorithms. Second, we will show that our transformation—in addition to the classical LCS problem—can be used to tackle other LCS-type problems from the literature, thus providing a unified approach for different LCS-type problems.

In the rest of this section we provide a short review of recent exact methods proposed for LCS-type problems. In Section 2 we provide a description of the LCS-type problems considered in this work, detailing the transformations required to build the conflict graphs and providing further literature references for solution methods tailored to each specific variant. We provide computational evidence of the validity of our approach in Section 4. To this end, we compare the following three techniques with specialised algorithms for each of the considered LCS-type problems: (1) the Integer Linear Programming (ILP) solver Cplex applied to solve the MIS problem in the conflict graph; (2) Lmc Jiang et al. (2016); Li et al. (2017), nowadays one of the best available exact MC solvers; and (3) Lscc-Bms Wang et al. (2016), nowadays one of the best available heuristic MC solvers.

1.1 Literature review

During the last decade, the literature has seen new and efficient heuristic approaches to LCS-type problems Blum and Festa (2016), but it still remains an important challenge to solve these problems to optimality. The dynamic programming approach of Gusfield (1997), which was mentioned above, becomes impractical when the number m of input strings grows. At the same time, real-life applications of LCS-type problems also involve long strings with large values of n, making a runtime of $O(n^m)$ impractical. Another exact approach is to model LCS-type problems as Integer Linear Programms (ILPs), if possible, and then apply general-purpose ILP solvers, such as Cplex or Gurobi, to solve the programms to optimality. Computational experiments by Lee and Gupta (2009) in the context of two ILP models for the classical LCS problem showed that this approach turns impractical already for small values of m.

A recent branch of work on exact techniques is the development of extensions of the classical A* algorithm Hart et al. (1968). One advantage of A* is that it can be hybridized with heuristic algorithms Wang et al. (2010); Djukanovic et al. (2019). Djukanovic et al. (2018) developed two A*-based hybrid variants for the palindromic LCS (see Section 2.3), which belong to the class of *anytime algorithms* (exact algorithms that return a feasible solution of reasonable quality whenever they are terminated Vadlamudi et al. (2012)). Moreover, Djukanovic et al. (2019) present a related study about A*-based anytime algorithms for the classical LCS problem.

2 Considered problems and transformations

We start by describing how to transform an instance of the classical LCS problem into a conflict graph in which a maximum independent set corresponds to a longest common subsequence of the original problem instance. Henceforth, an LCS problem instance is described by a pair (S, Σ) in which $S = \{s_1, \ldots, s_m\}$ is a set of input strings over the finite alphabet Σ. We denote the length of string $s_i \in S$ as $|s_i|$ and the element

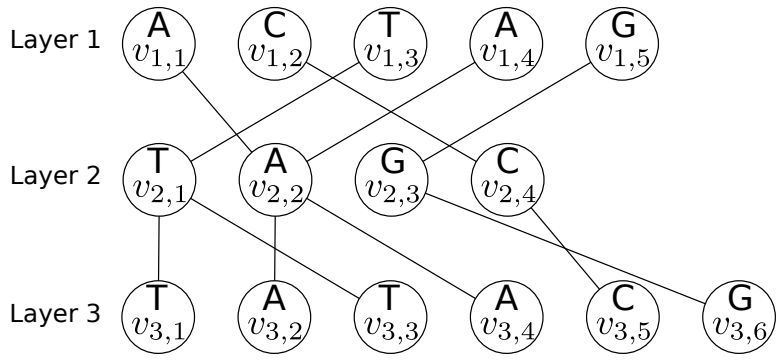

Figure 1: The undirected multi-layered graph G obtained from the LCS instance $\big(S = \{s_1 = \text{ACTAG}, s_2 = \text{TAGC}, s_3 = \text{ATACG}\}, \Sigma = \{\text{A}, \text{C}, \text{T}, \text{G}\}\big)$.

at position j in string s_i as $s_i[j]$. Given such an instance, we construct an undirected multi-layered graph $G = (V, E)$ whose vertex set V is partitioned into sets $\{V_1, \ldots, V_m\}$. Each V_i is called a *layer* and consists of $|s_i|$ vertices. Note that each layer represents exactly one input string and each vertex of the layer represents a position in the string. More specifically, $V_i = \{v_{i,1}, \ldots, v_{i,|s_i|}\}$, where vertex $v_{i,j}$ represents the j-th position of input string s_i.

We also partition the edge set E of the multi-layered graph G into sets $\{E_1, \ldots, E_{m-1}\}$, where E_i is the set of edges between layers V_i and V_{i+1}. Set E_i contains an edge $e_{j,k}$ connecting vertices $v_{i,j}$ and $v_{i+1,k}$ if and only if $s_i[j] = s_{i+1}[k]$, i.e., if the letter at position j of input string s_i is equal to the letter at position k of input string s_{i+1}. Figure 1 shows an example of this graph construction for three strings over an alphabet of size four.

Any sequence $p = (v_{1,j_1}, v_{2,j_2}, \ldots, v_{m,j_m})$ of m vertices and with the i-th vertex of p being from the i-th layer of G, is called a *complete path* in G if and only if it fulfils the following conditions:

1. The corresponding edge between every pair of consecutive vertices of p exists in G: $e_{j_i,j_{i+1}} \in E_i$ for all $i = 1, \ldots m-1$.

2. The letters at the positions of the input strings corresponding to the $m-1$ vertices are all the same: $s_1[j_1] = s_2[j_2] = \ldots = s_n[j_m]$.

Given a complete path $p = (v_{1,j_1}, v_{2,j_2}, \ldots, v_{m,j_m})$, the common letter at positions $j_1, \ldots, j_m$ of the m input strings is also called the *letter of* p. We denote it by $\ell(p)$.

Two complete paths p and q, with $p = (v_{1,j_1}, v_{2,j_2}, \ldots, v_{m,j_m})$ and $q = (v_{1,k_1}, v_{2,k_2}, \ldots, v_{m,k_m})$, are said to *cross* if and only if there is at least one index $l \in \{1, \ldots, m\}$ such that $j_l \leq k_l$ and at least one index $r \in \{1, \ldots m\}$, $r \neq l$, such that $j_r \geq k_r$. To make the concept of crossing paths clearer, refer to Figure 2 which shows two examples based on the instance depicted in Figure 1. In the left figure, the solid and dashed paths are crossing because they contain crossing edges between layers 1 and 2. In the right figure, they cross because they contain a common vertex in layer 2.

Given these notations, the classical LCS problem can be transformed into the maximum independent set (MIS) problem as follows. First, note that solving the classical LCS problem amounts to finding the largest set of non-crossing paths in the respective multi-layered graph G. Based on G we can create the conflict graph $G^c = (V^c, E^c)$ with a vertex for each complete path of G and an edge between two paths iff they cross. Then, solving the LCS problem is equivalent to solving the MIS problem in G^c which, in turn, is equivalent to solving the MC problem in the complement of graph $\overline{G}^c$.

In the rest of this section we consider three LCS-type problems and show how analogous transformations allow us to reduce each problem to a MC problem on the complement of a conflict graph.

2.1 Repetition-Free Longest Common Subsequence

The repetition-free longest common subsequence (RFLCS) problem Adi et al. (2010) is an LCS variant in which valid solutions are further constrained to contain each possible letter at most once. It was introduced as a comparison measure for sequences of different biological origin. In the related literature, this problem is

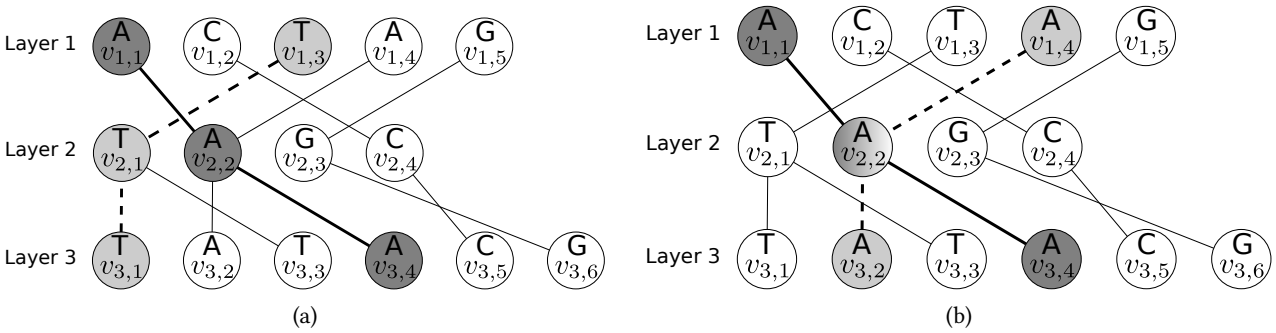

Figure 2: Two examples of complete paths that cross, based on the LCS instance from Figure 1. (a) Paths $p = (v_{1,1}, v_{2,2}, v_{3,4})$ and $q = (v_{1,3}, v_{2,1}, v_{3,1})$ cross because their corresponding edges between layers 1 and 2 cross. (b) Paths $p = (v_{1,1}, v_{2,2}, v_{3,4})$ and $q = (v_{1,4}, v_{2,2}, v_{3,2})$ cross because they both include vertex $v_{2,2}$ from the second layer.

generally considered for the case $m = 2$, that is, for two input strings. Note that even for $m = 2$ the problem is APX-hard (which implies it is $\mathcal{NP}$-hard), as shown by Adi et al. (2010).

Blum and Blesa (2018) proposed the current best specialized algorithm for this problem: a construct, merge, solve and adapt (CMSA) approach in which the authors initialise the reduced sub-instance by beam search. In Blum and Blesa (2018), the authors show how their algorithm outperforms other metaheuristics and the application of Cplex to an ILP model of the problem.

To generate the conflict graph for the RFLCS problem, we first build the multi-layered graph G concerning the two input strings, just like in the case of the classical LCS problem. Note that, due to the two input strings, G will have two layers. Two complete paths p and q of G are in conflict if they fulfil at least one of the following two conditions:

1. p and q cross each other.
2. p and q have the same letter: $\ell(p) = \ell(q)$. Note that this condition ensures that no letter appears more than once in a solution.

2.2 Longest Arc-Preserving Common Subsequence

The second considered LCS variant is known as the longest arc-preserving common subsequence (LAPCS) problem Evans (1999a). As in the case of the RFLCS problem, the LAPCS problem is studied for two input strings/sequences in the literature. Note that, in the case of the LAPCS problem, the input strings are arc-annotated. An *arc annotation* of a string s is a pair of positions in s, say (i_1, i_2) with $i_1, i_2 \in \{1, \ldots, |s|\}$ and, without loss of generality, $i_1 < i_2$. An arc-annotated sequence is a pair (s, P_s) where s is a string over some finite alphabet Σ and P_s is the set of arc annotations of s. The LAPCS problem is then defined for two arc-annotated sequences (s_1, P_1) and (s_2, P_2) as the problem of finding the longest common subsequence between s_1 and s_2 that fulfils the "arc-preservation" condition. This condition states that if there is an arc annotation between two positions in s_1 chosen for the solution, then there must also be an arc annotation between the two corresponding positions in s_2, and vice-versa.

Arc-annotated sequences are useful for the structural comparison of RNA sequences. Figure 3 shows an example of an arc-annotated RNA sequence in which the arc annotations are indicated as solid lines linking the nucleobases ACGT. Evans (1999b,a) introduced the LACPS problem and showed that it is $\mathcal{NP}$-hard already for two strings. Blum and Blesa (2018) proposed the best specialized algorithms for the LAPCS. Depending on the problem instance characteristics, the state-of-the-art algorithm is either a heuristic based on problem reduction, or an iterative probabilistic algorithm, both of which solve reduced ILP models. The authors compared these algorithms with the application of Cplex to solve the MIS problem in the corresponding conflict graphs.

To generate the conflict graph for a LAPCS problem instance consisting of (s_1, P_1) and (s_2, P_2), we first construct the two-layered multi-graph G based on s_1 and s_2, as done in the classical LCS problem case. Two complete paths p and q are in conflict if and only if they fulfil at least one of the following two conditions:

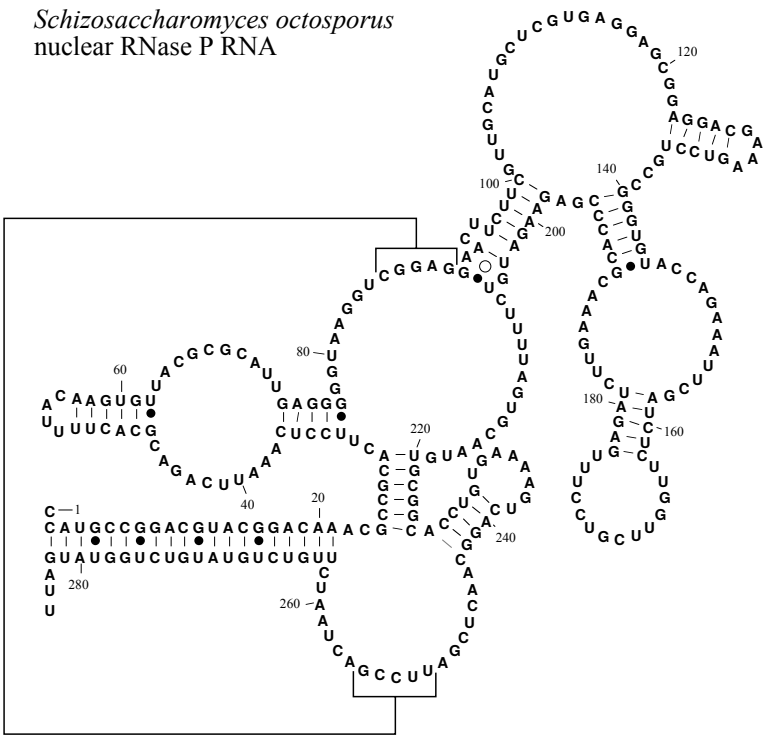

Figure 3: Example of an arc-annotated sequence (RNA of *Schizosaccharomyces octosporus*). The connections between different positions of the RNA sequence, indicated by short lines, are the members of the arc annotation set. Note that this graphic was obtained from the RNase P Database Brown (1999).

1. p and q cross each other.
2. p and q do *n*ot cross each other, but the arc annotations are not preserved: the substring contains two letters coming from positions linked by an arc in one of the two original strings, but not in the other. Formally, this happens if, for some positions j_1, k_1 of s_1 with $j_1 < k_1$ and some positions j_2, k_2 of s_2 with $j_2 < k_2$, it holds that either: $(j_1, k_1) \in P_1$ and $(j_2, k_2) \notin P_2$, or $(j_2, k_2) \in P_2$ and $(j_1, k_1) \notin P_1$.

Figure 4 shows an example LAPCS instance. The solution depicted with dashed lines is infeasible because it matches $v_{1,2}$ and $v_{1,4}$ in s_1 with, respectively, $v_{2,4}$ and $v_{2,5}$ in s_2. An arc annotation links the positions in s_1 but not in s_2, thus violating condition 2 above. The solution depicted with solid lines, instead, is feasible.

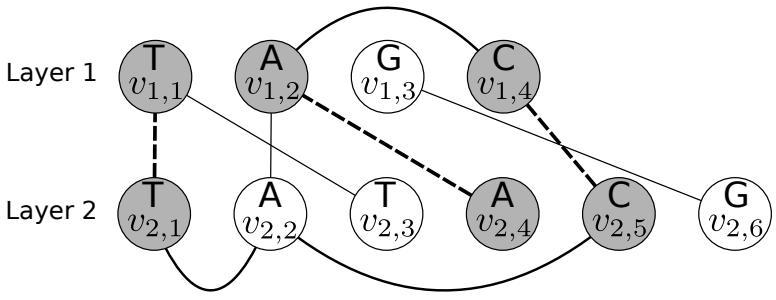

Figure 4: This example shows the undirected multi-layered graph G obtained from the LAPCS instance consisting of ($s_1 = \mathsf{TAGC}, P_1 = \{(2,4)\}$) and ($s_2 = \mathsf{TATACG}, P_2 = \{(1,2),(2,5)\}$). The solution in dashed lines $\{p = (v_{1,1}, v_{2,1}), q = (v_{1,2}, v_{2,4}), r = (v_{1,4}, v_{2,5})\}$ is not valid because arc $(2,4) \in P_1$ connects two chosen positions in s_1, while the corresponding chosen positions in s_2 — that is, positions 4 and 5 — are not connected by an arc from P_2. The solution in solid lines is feasible.

2.3 Longest Common Palindromic Subsequence

Finally, we also consider the so-called longest common palindromic subsequence (LCPS) problem Chowdhury et al. (2014). This is an LCS variant in which we look for a longest common subsequence s^* of m input strings such that s^* is also a palindrome. Note that a string is a *palindrome* if it coincides with its reverse; the reverse string of s is a string s^{rev} such that $s^{\text{rev}}[i] = s[|s| - i + 1]$, for all $1 \leq i \leq \left\lfloor \frac{|s|}{2} \right\rfloor$. For example, KAYAK is a palindrome.

Chowdhury et al. (2014); Hasan et al. (2017); Inenaga and Hyyrö (2018) presented specialized exact algorithms for the LCPS problem on two input strings (2–LCPS). The theoretical lower bound on solving the 2–LCPS is not known, but Abboud et al. (2015) hypothesise it is at least $O(n^4)$; if this were not the case, then the famous strong exponential time hypothesis Calabro et al. (2009) would fail. Djukanovic et al. (2018, 2019) presented the first works on instances with $m > 2$, introducing two A*-based hybrid anytime algorithms.

After generating the layered multi-graph G for the m input strings, in the same way as in the cases outlined before, the conflict graph is built as follows. The set of vertices V^c of the conflict graph G^c consists of two disjoint subsets of vertices: V_{single} and V_{pairs}. More specifically, V_{single} contains a vertex v_p for each complete path $p \in P$, and V_{pairs} contains a vertex $v_{p,q}$ for each pair of complete paths $p \neq q$ with $\ell(p) = \ell(q)$ that do not cross each other. Notice that in the previous cases—that is, the classical LCS problem, the RFLCS problem, and the LAPCS problem—the number of vertices in the conflict graph was equal to the number of complete paths in the multi-layered graph G, say z. In contrast, the number of vertices in the conflict graph of the LCPS problem is of the order $O(z + z^2)$. Finally, we define the edges of the conflict graph by the following conflict relations:

1. Conflicts between vertices from V_{single}: these vertices are all in conflict with each other. This is because the vertices from V_{single} model the possibility to have a singleton letter in the middle of a solution. For example, KAYAK has Y as a singleton letter in the middle. In contrast, KAAK for example, has no singleton letter in the middle. As a solution can have at most one singleton letter in the middle, all vertices from V_{single} are in conflict with each other. As a consequence, all other vertices that form part of a solution are from V_{pairs}. In the case of KAYAK, for example, there would be two such vertices: one representing the two K's and one for the two A's.

2. Conflicts between vertices from V_{pairs}: to describe a conflict between two such vertices, it is actually easier to state when they are *not* in conflict with each other. Consider two vertices $v_{p,q}, v_{p',q'} \in V_{\text{pairs}}$, with

$$\begin{aligned} p &= (v_{1,j_1}, \ldots, v_{m,j_m}) \\ q &= (v_{1,k_1}, \ldots, v_{m,k_m}) \\ p' &= (v_{1,j'_1}, \ldots, v_{m,j'_m}) \\ q' &= (v_{1,k'_1}, \ldots, v_{m,k'_m}) \end{aligned}$$

   and assume wlog that $j_1 < k_1$ and that $j'_1 < k'_1$. Then $v_{p,q}$ and $v_{p',q'}$ are *not* in conflict if either $j_i < j'_i < k'_i < k_i$ for all $i = 1, \ldots, m$, or $j'_i < j_i < k_i < k'_i$ for all $i = 1, \ldots, m$.

3. Conflicts between vertices from V_{single} and vertices from V_{pairs}: again, we state when there is *no* conflict between two such vertices. Consider vertex $v_{p'} \in V_{\text{single}}$ and vertex $v_{p,q} \in V_{\text{pairs}}$, with

$$\begin{aligned} p &= (v_{1,j_1}, \ldots, v_{m,j_m}) \\ q &= (v_{1,k_1}, \ldots, v_{m,k_m}) \\ p' &= (v_{1,j'_1}, \ldots, v_{m,j'_m}) \end{aligned}$$

   and assume wlog that $j_1 < k_1$. Then $v_{p''}$ and $v_{p,p'}$ are not in conflict if $j_i < j'_i < k_i$ for all $i = 1, \ldots, m$.

Notice that all vertices from V_{pairs} have weight 2 and, if chosen in the final clique, they will contribute for two letters in the respective solution.

Figure 5 shows the multi-layerd graph for input strings TAGCAT and TATACG. Complete paths are shown by lines and, in particular, we use dashed and dotted lines to highlight relevant paths concerning letters T and

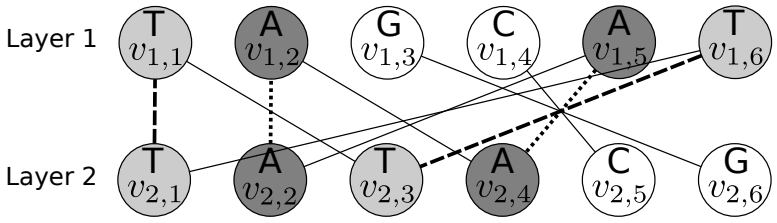

Figure 5: The multi-layered graph G obtained from the LCPS instance on the two input strings s_1 = TAGCAT and s_2 = TATACG. This graph contains 10 complete paths, corresponding to the 10 vertices of the conflict graph (V_{single}). Two pairs of non-crossing paths have the same letters: the first pair (with letter T) is indicated in light grey and dashed lines, the second one (with letter A) is indicated in dark grey and dotted lines.

A. Note how the rightmost highlighted paths for T and A are crossing. Therefore, the potential solution TAAT cannot be constructed. This string is only a substring of the first input string, but not of the second one. The optimal solution in this example is, in fact, TAT.

3 Conflict graph reduction

The size of the conflict graphs (in terms of the number of vertices) mainly depends on the length and on the number of input strings. Let $n_{\max} := \max_{i=1,\dots,m}\{|s_i|\}$. Then, the sizes of the conflict graphs can be expressed as follows: $O(n_{\max}^m)$ in the case of the classical LCS problem, $O(n_{\max}^2)$ in the case of the RFLCS and LAPCS problems, and $O(n_{\max}^m + n_{\max}^{2m})$ in the case of the LCPS problem. In fact, during preliminary experiments we realized that the conflict graphs are too large, even for rather small problem instances from the literature, in the cases of the classical LCS problem and the LCPS problem. Therefore, we henceforth focus exclusively on the RFLCS and LAPCS problems. However, even for these two problems, the conflict graphs are very large when large-scale problem instances are concerned. Therefore, we decided to investigate into techniques for reducing the size of the conflict graphs. Note that there are basically two potential strategies for reducing the size of a given conflict graph G^c: (1) making use of problem-specific information relative to the respective LCS problems, and (2) analysing and reducing G^c from the point of view of the MC problem. However, the latter strategy has proven ineffective in preliminary computational experiments. This is because solver LMC (the state-of-the-art exact MC problem solver that we used Jiang et al. (2016); Li et al. (2017)) already implements powerful graph reduction procedures which were not able to reduce G^c. Therefore, we focused on reducing the conflict graphs by making use of LCS specific information.

Our main idea for the reduction of the conflict graphs is based on having at our disposal a high-quality primal (lower) bound value lb for the tackled problem, that is, the value of a high-quality solution. The value of the best-known solution from the literature can be taken for this purpose, for example. Before we proceed, the following notation is required: given a string t and two indices $l, r \in \{1, \dots, |t|\}$ with $l \leq r$, $t[l, r]$ denotes the substring of t starting at position l and ending at position r. Now, on the basis of the primal bound lb, it can be decided for every complete path $p = v_{1,j_1}, \dots, v_{m,j_m}$ of the multi-layered graph, if the corresponding vertex v_p can be removed from the conflict graph G^c without loosing an optimal solution.[1] This is done as follows. First, note that the complete path under consideration splits each input string s_i into two parts: $s_i[1, j_i - 1]$ (the left-hand side) and $s_i[j_i + 1, |s_i|]$ (the right-hand side). Henceforth we denote the set of left-hand sides corresponding to a complete path p by S_p^{L}, and the set of right-hand sides by S_p^{R}. More formally:

$$S_p^{\text{L}} = \left\{ s_i[1, j_i - 1] \mid i = 1, \dots, m \right\}$$
$$S_p^{\text{R}} = \left\{ s_i[j_i + 1, |s_i|] \mid i = 1, \dots, m \right\}$$

Note that both S_p^{L} and S_p^{R} are subinstances of the original problem instance. Therefore, any upper bound function UB() known for the problem (RFLCS, respectively LAPCS) can be used for (over)-estimating the

[1] Note that the conflict graph reduction will be described for a general case of n input strings, even though we only have two input strings in the cases of the RFLCS and LAPCS problems.

quality of the length of an optimal solution in S_p^{L} and S_p^{R}. Given such an upper bound function UB(), vertex v_p and all corresponding edges can be deleted from the conflict graph G^{c} iff

$$\mathrm{UB}(S_p^{\mathrm{L}}) + 1 + \mathrm{UB}(S_p^{\mathrm{R}}) < lb\,. \tag{1}$$

For the following discussion, bear in mind that any upper bound for the classical LCS problem is also an upper bound for the RFLCS and LAPCS problems. This is, because these two problems correspond to classical LCS problems with additional constraints. In other words, the set of valid solutions of a RFLCS problem instance, respectively a LAPCS problem instance, is a subset of the set of valid solutions of the instance if solved as a classical LCS problem. Therefore, upper bound functions developed for the classical LCS problem are candidates to be used for UB() in Equation (1).

Blum et al. (2009), for example, introduced an upper bound function henceforth labelled $\mathrm{UB}_1^{\mathrm{LCS}}()$ for the classical LCS problem (which is a tightened version of a bound originally introduced by Fraser (1995)). Given a problem instance (S, Σ), for each input string $s_i \in S$ and each letter $a \in \Sigma$, let $|s_i|_a$ be the number of occurrences of a in s_i and let $c_a(S) = \min_{s_i \in S} |s|_a$. Then, $\mathrm{UB}_1^{\mathrm{LCS}}()$ is defined as follows:

$$\mathrm{UB}_1^{\mathrm{LCS}}(S) = \sum_{a \in \Sigma} c_a(S)$$

Let $\delta(a, S)$ for $a \in \Sigma$ evaluate to one, if letter a appears at least once in each input string from S, and otherwise to zero. As each letter from Σ can mostly appear once in a valid RFLCS solution, $\mathrm{UB}_1^{\mathrm{LCS}}()$ from above reduces to the following upper bound function in the context of the RFLCS problem:

$$\mathrm{UB}_1^{\mathrm{RFLCS}}(S) = \sum_{a \in \Sigma} \delta(a, S)$$

Finally, when used for our purposes—that is, for obtaining an upper bound for (sub-)instances S_p^{L} and S_p^{R} in Equation (1) in the context of an RFLCS instance—we can even exclude letter $l(p)$ (the letter of path p) from the sum. This results in:

$$\mathrm{UB}_1^{\mathrm{RFLCS}}(S, p) = \sum_{a \in \Sigma \setminus \{l(p)\}} \delta(a, S)\,.$$

Wang et al. (2011) proposed another upper bound function for the classical LCS problem, henceforth labelled $\mathrm{UB}_2()$, which is based on dynamic programming (DP). This function is defined as follows:

$$\mathrm{UB}_2(S) = \min_{i=1,\ldots,m-1} \mathrm{LCS}(s_i, s_{i+1})\,,$$

where $\mathrm{LCS}(s_i, s_{i+1})$ refers to the length of the longest common subsequence of input strings s_i and s_{i+1}. Using the DP recursion of Wang et al. (2010) we can obtain this bound in $O(m)$ time by using an appropriate preprocessing data structure known as the *scoring matrix* Wang et al. (2012); Inenaga and Hyyrö (2018). In particular, note that in the context of the RFLCS and LAPCS problems, the preprocessing is done in $O(n^2)$ time.

In summary, for the conflict graph reduction in the context of the RFLCS problem, UB() is defined as $\min\{\mathrm{UB}_1^{\mathrm{RFLCS}}(), \mathrm{UB}_2()\}$; and in the context of the LAPCS problem, UB() is defined as $\min\{\mathrm{UB}_1^{\mathrm{LCS}}(), \mathrm{UB}_2()\}$.

4 Experimental evaluation

The aim of the computational experiments is to compare two strategies to solve LCS problems: (1) their direct solution using a specialized state-of-the-art algorithm, and (2) their transformation to the MIS, respectively the MC, problems and the subsequent solution by Cplex[2] (in case of the MIS problem) or by different MC solvers. In the case of the transformation to an MC problem, we make use of the following solvers:

- Lmc. This exact MC solver was introduced by Jiang et al. (2016); Li et al. (2017). It is currently one of the best exact solvers available for the MC problem. It combines an aggressive preprocessing of the graph with a MaxSAT solver Li and Manya (2009) in a branch-and-bound scheme.

[2]IBM ILOG CPLEX is an optimization software package that includes state-of-the-art exact techniques for solving integer linear programming models, among others. It is available for free for academic purposes. For more information, we refer the interested reader to `http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`. In this work we made use of version 12.7.

Instance	First String			Second string		
	RNA	n	n_{arcs}	RNA	n	n_{arcs}
Real_1	*Allochromatium vinosum*	369	119	*Haemophilus influenza*	377	124
Real_2	*Bacteroides thetaiotaomicron*	361	121	*Porphyromonas gingivalis*	398	131
Real_3	*Halococcus morrhuae*	475	154	*Haloferax volcanii*	433	142
Real_4	*Klebsiella pneumoniae*	383	127	*Escherichia coli*	377	124
Real_5	*Methanococcus jannaschii*	252	75	*Archaeoglobus fulgidus*	229	67
Real_6	*Methanosarcina barkeri*	371	115	*Pyrococcus abyssi*	330	100
Real_7	*Mycoplasma genitalium*	384	119	*Mycoplasma pneumoniae*	369	112
Real_8	*Saccharomyces kluveri*	336	90	*Schizosaccharomyces octosporus*	281	71
Real_9	*Serratia marcescens*	378	125	*Shewanella putrefaciens*	354	115
Real_10	*Streptomyces bikiniensis*	398	135	*Streptomyces lividans*	405	138

Table 1: Characteristics of real instances from set LAPCS-REAL. All 20 arc-annotated RNA sequences were taken from the RNase P Database Brown (1999).

- LSCC-BMS. This is one of the best-performing heuristic algorithms for the MC problem. Wang et al. (2016) introduced this local-search-based algorithm, whose main strengths are a configuration checking procedure that reduces the probability of cycling during local search, and a low-complexity vertex swap neighbourhood which is fast even on massive graphs[3].

Note that both CPLEX and LSCC-BMS were executed on a cluster of 12-core Intel Xeon 5670 CPUs at 2.9GHz and at least 40GB of RAM. LMC was executed on a cluster with 8-core Intel Xeon E5-2680 CPUs at 2.4GHz and with 128 GB of memory. In both cases, the memory consumption of each process was limited to 16 GB.

RFLCS benchmark instances Two sets of problem instances can be found in the related literature. The first set, henceforth denoted by RFLCS-SET1, consists of 30 randomly generated problem instances for each combination of the input sequence length $n \in \{32, 64, 128, 256, 512, 1024, 2048, 4096\}$ and the alphabet size $|\Sigma| \in \{\frac{n}{8}, \frac{n}{4}, \frac{3n}{8}, \frac{n}{2}, \frac{5n}{8}, \frac{3n}{4}, \frac{7n}{8}\}$. This results in a total of 1680 instances. The second set, henceforth dented by RFLCS-SET2, consists of 30 randomly generated instances for each combination of the alphabet size $|\Sigma| \in \{4, 8, 16, 32, 64, 128, 256, 512\}$ and the maximal repetition of each letter, reps $\in \{3, 4, 5, 6, 7, 8\}$. In total, set RFLCS-SET2 contains 1440 instances.

LAPCS benchmark instances The recent literature on the LAPCS problem considers both artificial instances (benchmark set LAPCS-ARTI) and real RNA instances (benchmark set LAPCS-REAL). Each artificial instance consists of two randomly generated RNA strings of length $n \in \{100, 200, \ldots, 900, 1000\}$. Moreover, each input string has $n_{arcs} \in \left\{\frac{n}{10}, \frac{n}{5}, \frac{n}{2}\right\}$ randomly generated unique arc annotations. Set LAPCS-ARTI consists of 30 instances for each combination of n and n_{arcs}, which makes a total of 900 problem instances. Set LAPCS-REAL consists of 10 problem instances that are composed of arc-annotated RNA sequences downloaded from the RNase P Database Brown (1999). Note that the alphabet size in all cases is equal to four. Table 1 summarises the characteristics of these instances.

Due to the fact that the amount of reduction of the conflict graphs from Section 3 depends on the quality of the used primal bound per instance, we used the currently best-known solution values from the literature for all considered instances. In the case of the RFLCS problem, these values were taken from Blum and Blesa (2018), and in the case of the LAPCS problem from Blum and Blesa (2018).

4.1 Results without conflict graph reduction

All three methods—CPLEX, LMC, and LSCC-BMS—were applied with a computation time limit of 3600 seconds (1 hour) and a memory limit of 16GB per run to all RFLCS and LAPCS problem instances. The results are presented in numerical form in Tables 2 and 3 concerning the RFLCS problem, and in Tables 4 and 5 concerning the LAPCS problem. The first two columns in Tables 2–4 indicate the problem instance characteristics, while the third column provides the currently best known results from the literature. Remember, in this context,

[3]We downloaded the code of LSCC-BMS from `http://ai.nenu.edu.cn/wangyy/Yiyuandata/LocalSearchforMWCP.htm` on April 29, 2019.

Table 2: Experimental results for RFLCS instances RFLCS-SET1.

$\|\Sigma\|$	n	Spec.	CPLEX				LMC				LSCC+BMS	
		Tech.	result	$\bar{t}$	$\bar{t}_{\text{opt}}$	#opt	result	$\bar{t}$	$\bar{t}_{\text{opt}}$	#opt	result	$\bar{t}$
$n/8$	32	4.00	**4.00**	0.09	0.09	30	**4.00**	0.00	0.01	30	**4.0**	0.01
	64	8.00	**8.00**	0.81	0.81	30	**8.00**	0.00	0.07	30	**8.0**	0.00
	128	16.00	**16.00**	8.12	8.12	30	**16.00**	0.00	49.61	30	**16.0**	0.01
	256	31.97	**31.97**	188.31	188.31	30	31.90	20.54	– –	0	**31.97**	0.09
	512	63.27	5.17	625.34	– –	0	62.50	485.59	– –	0	**63.90***	68.84
	1024	111.57	0.03	1461.74	– –	0	112.53	818.57	– –	0	**116.10***	1297.00
	2048	182.67	– –	– –	– –	0	182.40	1331.53	– –	0	181.67	1394.27
	4096	283.33	– –	– –	– –	0	281.37	1037.61	– –	0	261.37	1510.89
$n/4$	32	7.83	**7.83**	0.03	0.03	30	**7.83**	0.00	0.00	30	**7.83**	0.00
	64	14.67	**14.67**	0.29	0.29	30	**14.67**	0.00	0.01	30	**14.67**	0.00
	128	25.77	**25.93***	2.02	2.50	30	**25.93***	0.01	0.09	30	**25.93***	0.02
	256	43.70	**43.97***	30.92	51.17	30	**43.97***	0.12	0.80	30	**43.97***	0.22
	512	67.90	68.50	582.53	1622.77	27	**68.57***	75.61	185.15	30	**68.57***	7.57
	1024	103.00	0.00	240.97	– –	0	103.77	386.81	– –	0	**104.87***	877.29
	2048	154.33	0.00	1398.78	– –	0	152.87	438.52	– –	0	151.33	1485.85
	4096	226.67	– –	– –	– –	0	223.57	780.50	– –	0	207.03	1984.69
$3n/8$	32	8.77	**8.77**	0.02	0.02	30	**8.77**	0.00	0.00	30	**8.77**	0.00
	64	15.53	**15.53**	0.10	0.10	30	**15.53**	0.00	0.00	30	**15.53**	0.00
	128	24.90	**24.90**	1.75	1.79	30	**24.90**	0.00	0.03	30	**24.90**	0.01
	256	39.97	**39.97**	5.25	5.90	30	**39.97**	0.02	0.20	30	**39.97**	0.13
	512	59.77	**59.97***	106.42	133.02	30	**59.97***	0.46	1.83	30	**59.97***	1.99
	1024	90.50	90.67	2204.06	2263.32	23	**90.73***	5.71	30.67	30	**90.73***	145.24
	2048	130.57	0.00	547.50	– –	0	129.67	233.36	105.92	1	129.13	1578.88
	4096	191.37	– –	– –	– –	0	188.30	311.61	– –	0	179.73	1670.85
$n/2$	32	8.87	**8.87**	0.01	0.01	30	**8.87**	0.00	0.00	30	**8.87**	0.00
	64	14.80	**14.80**	0.06	0.06	30	**14.80**	0.00	0.00	30	**14.80**	0.00
	128	22.93	**22.93**	0.76	0.78	30	**22.93**	0.00	0.01	30	**22.93**	0.00
	256	35.10	**35.20***	2.18	2.27	30	**35.20***	0.02	0.09	30	**35.20***	0.09
	512	53.10	**53.13***	31.82	34.03	30	**53.13***	0.08	0.66	30	**53.13***	0.71
	1024	79.03	**79.13***	627.90	701.13	30	**79.13***	6.04	11.56	30	**79.13***	30.80
	2048	115.30	0.00	248.56	– –	0	115.07	432.97	598.59	19	114.87	1517.02
	4096	167.47	0.00	1295.77	– –	0	165.87	390.18	– –	0	159.37	1490.48
$5n/8$	32	8.60	**8.60**	0.01	0.01	30	**8.60**	0.00	0.00	30	**8.60**	0.00
	64	13.30	**13.30**	0.03	0.03	30	**13.30**	0.00	0.00	30	**13.30**	0.00
	128	21.20	**21.20**	0.36	0.37	30	**21.20**	0.00	0.01	30	**21.20**	0.00
	256	32.53	**32.53**	4.21	4.36	30	**32.53**	0.01	0.05	30	**32.53**	0.04
	512	47.83	**47.83**	13.06	13.15	30	**47.83**	0.04	0.33	30	**47.83**	0.28
	1024	70.03	**70.20***	208.55	215.63	30	**70.20***	1.43	4.12	30	**70.20***	8.70
	2048	103.80	48.33	2306.93	3328.76	1	**103.97***	63.19	158.21	30	103.87	936.80
	4096	150.00	0.00	878.84	– –	0	148.53	302.72	1607.66	2	145.77	1423.49
$3n/4$	32	8.17	**8.17**	0.00	0.00	30	**8.17**	0.00	0.00	30	**8.17**	0.00
	64	12.53	**12.53**	0.02	0.02	30	**12.53**	0.00	0.00	30	**12.53**	0.00
	128	19.70	**19.70**	0.17	0.18	30	**19.70**	0.00	0.00	30	**19.70**	0.00
	256	29.97	**29.97**	2.25	2.32	30	**29.97**	0.00	0.03	30	**29.97**	0.02
	512	44.53	**44.57***	4.90	4.94	30	**44.57***	0.03	0.19	30	**44.57***	0.29
	1024	65.07	**65.20***	96.77	97.46	30	**65.20***	0.75	2.11	30	**65.20***	3.39
	2048	94.53	**94.67***	1829.86	1862.21	30	**94.67***	4.57	18.69	30	94.63	638.75
	4096	136.57	0.00	500.41	– –	0	135.73	355.77	682.50	13	133.53	1617.99
$7n/8$	32	7.67	**7.67**	0.00	0.00	30	**7.67**	0.00	0.00	30	**7.67**	0.00
	64	11.57	**11.57**	0.01	0.01	30	**11.57**	0.00	0.00	30	**11.57**	0.00
	128	18.40	**18.40**	0.12	0.12	30	**18.40**	0.00	0.00	30	**18.40**	0.00
	256	27.80	**27.80**	1.21	1.22	30	**27.80**	0.00	0.02	30	**27.80**	0.01
	512	40.57	**40.60***	2.93	3.01	30	**40.60***	0.02	0.12	30	**40.60***	0.10
	1024	60.50	**60.57***	79.74	79.76	30	**60.57***	0.28	1.19	30	**60.57***	3.55
	2048	88.00	**88.00**	831.15	896.78	30	**88.00**	4.13	18.68	30	**88.00**	114.45
	4096	127.20	0.00	361.39	– –	0	126.50	212.34	478.99	17	125.47	1608.56

Table 3: Experimental results RFLCS instances RFLCS-SET2.

$\|\Sigma\|$	reps	**Spec.**	CPLEX				LMC				LSCC-BMC	
		Tech.	**result**	$\bar{t}$	$\bar{t}_{\text{opt}}$	**#opt**	**result**	$\bar{t}$	$\bar{t}_{\text{opt}}$	**#opt**	**result**	$\bar{t}$
4	3	3.47	**3.47**	0.00	0.00	30	**3.47**	0.00	0.00	30	**3.47**	0.00
	4	3.77	**3.77**	0.00	0.00	30	**3.77**	0.00	0.00	30	**3.77**	0.00
	5	3.83	**3.83**	0.00	0.00	30	**3.83**	0.00	0.00	30	**3.83**	0.00
	6	3.90	**3.90**	0.00	0.00	30	**3.90**	0.00	0.00	30	**3.90**	0.00
	7	3.97	**3.97**	0.01	0.01	30	**3.97**	0.00	0.00	30	**3.97**	0.00
	8	3.97	**3.97**	0.01	0.01	30	**3.97**	0.00	0.00	30	**3.97**	0.00
8	3	6.23	**6.23**	0.00	0.00	30	**6.23**	0.00	0.00	30	**6.23**	0.00
	4	6.87	**6.87**	0.00	0.00	30	**6.87**	0.00	0.00	30	**6.87**	0.00
	5	7.40	**7.40**	0.02	0.02	30	**7.40**	0.00	0.00	30	**7.40**	0.00
	6	7.53	**7.53**	0.02	0.02	30	**7.53**	0.00	0.00	30	**7.53**	0.00
	7	7.70	**7.70**	0.06	0.06	30	**7.70**	0.00	0.00	30	**7.70**	0.00
	8	7.77	**7.77**	0.05	0.05	30	**7.77**	0.00	0.00	30	**7.77**	0.00
16	3	9.70	**9.70**	0.01	0.01	30	**9.70**	0.00	0.00	30	**9.70**	0.00
	4	11.57	**11.57**	0.03	0.03	30	**11.57**	0.00	0.00	30	**11.57**	0.00
	5	12.93	**12.93**	0.06	0.06	30	**12.93**	0.00	0.00	30	**12.93**	0.00
	6	14.00	**14.00**	0.15	0.16	30	**14.00**	0.00	0.01	30	**14.00**	0.00
	7	14.93	**14.93**	0.30	0.30	30	**14.93**	0.00	0.02	30	**14.93**	0.02
	8	14.80	**14.80**	0.37	0.38	30	**14.80**	0.00	0.02	30	**14.80**	0.00
32	3	16.13	**16.13**	0.08	0.08	30	**16.13**	0.00	0.00	30	**16.13**	0.00
	4	19.00	**19.00**	0.27	0.27	30	**19.00**	0.00	0.01	30	**19.00**	0.00
	5	21.63	**21.63**	0.83	0.85	30	**21.63**	0.00	0.02	30	**21.63**	0.01
	6	23.73	**23.73**	1.57	1.65	30	**23.73**	0.00	0.04	30	**23.73**	0.01
	7	25.53	**25.57***	2.23	2.34	30	**25.57***	0.02	0.10	30	**25.57***	0.03
	8	27.40	**27.50***	4.59	4.71	30	**27.50***	0.06	0.23	30	**27.50***	0.07
64	3	25.43	**25.43**	0.88	0.91	30	**25.43**	0.00	0.01	30	**25.43**	0.00
	4	30.37	**30.37**	2.65	2.80	30	**30.37**	0.01	0.05	30	**30.37**	0.02
	5	34.87	**34.93***	3.57	4.66	30	**34.93***	0.02	0.13	30	**34.93***	0.07
	6	39.07	**39.13***	13.36	17.37	30	**39.13***	0.05	0.34	30	**39.13***	0.18
	7	43.50	**43.63***	28.44	55.76	30	**43.63***	0.16	0.92	30	**43.63***	0.40
	8	45.17	**45.53***	58.39	116.58	30	**45.53***	1.38	5.41	30	**45.53***	0.75
128	3	36.70	**36.77***	2.39	2.44	30	**36.77***	0.01	0.09	30	**36.77***	0.14
	4	44.90	**45.03***	12.95	15.22	30	**45.03***	0.06	0.37	30	**45.03***	0.39
	5	53.23	**53.43***	48.50	64.03	30	**53.43***	0.15	1.08	30	**53.43***	1.12
	6	61.07	**61.53***	183.29	300.56	30	**61.53***	4.55	7.76	30	**61.53***	4.42
	7	67.90	68.40	749.39	1377.40	25	**68.47***	8.25	54.19	30	**68.47***	5.13
	8	73.57	74.37	1288.16	1932.99	11	74.30	524.20	474.27	13	**74.60***	22.68
256	3	54.97	**55.03***	46.81	48.61	30	**55.03***	0.08	0.69	30	**55.03***	1.06
	4	68.70	**68.93***	247.83	268.90	30	**68.93***	0.31	2.90	30	**68.93***	8.53
	5	81.00	**81.43***	917.97	1182.86	30	**81.43***	9.65	21.74	30	**81.43***	45.01
	6	93.10	73.83	2951.48	3090.98	2	93.17	239.22	418.66	17	**93.53***	162.94
	7	103.50	0.00	308.34	– –	0	103.13	132.52	499.03	3	**104.40***	734.99
	8	113.70	0.00	501.06	– –	0	113.10	298.94	– –	0	**114.70***	1300.54
512	3	81.57	**81.63***	524.51	536.33	30	**81.63***	0.72	5.38	30	**81.63***	41.71
	4	100.83	78.63	2899.04	3142.25	3	101.10	157.68	230.29	29	**101.13***	602.19
	5	120.43	0.00	404.86	– –	0	118.70	539.36	851.21	5	119.60	1147.39
	6	137.03	0.00	681.76	– –	0	135.50	483.24	– –	0	136.00	1894.44
	7	154.57	0.00	1218.70	– –	0	152.33	784.72	– –	0	150.63	1784.08
	8	172.10	– –	– –	– –	0	169.90	698.89	– –	0	166.47	1428.87

Table 4: Experimental results for LAPCS instances Lapcs-Arti.

n	n_{arcs}	**Spec.**	Cplex				Lmc				Lscc-Bms	
		Tech.	**result**	$\bar{t}$	$\bar{t}_{\text{opt}}$	**#opt**	**result**	$\bar{t}$	$\bar{t}_{\text{opt}}$	**#opt**	**result**	$\bar{t}$
100	10	60.17[a]	**60.20***	304.84	326.59	30	**60.20***	80.25	144.93	30	**60.20***	3.25
	20	58.13[a]	**58.20***	341.35	485.49	30	**58.20***	89.40	318.07	29	**58.20***	2.87
	50	51.87[a]	52.03	826.12	1990.85	20	52.07	147.26	1112.64	21	**52.10***	4.57
200	20	121.70[b]	– –	– –	– –	0	120.27	672.79	– –	0	121.23	977.41
	40	116.70[b]	– –	– –	– –	0	115.87	698.50	– –	0	**117.67***	1306.38
	100	104.57[a]	– –	– –	– –	0	104.30	540.59	– –	0	**106.50***	1122.56
300	30	181.30[a]	– –	– –	– –	0	178.10	781.43	– –	0	173.47	1811.81
	60	174.97[a]	– –	– –	– –	0	171.80	498.21	– –	0	169.57	1646.61
	150	157.13[a]	– –	– –	– –	0	155.93	988.86	– –	0	156.10	2010.08
400	40	242.70[b]	– –	– –	– –	0	239.53	790.89	– –	0	220.67	1772.27
	80	233.23[a]	– –	– –	– –	0	226.97	583.45	– –	0	215.10	1617.85
	200	208.77[a]	– –	– –	– –	0	205.23	931.87	– –	0	199.43	1949.73
500	50	302.27[b]	– –	– –	– –	0	295.90	746.95	– –	0	262.10	1490.81
	100	291.23[a]	– –	– –	– –	0	284.67	936.73	– –	0	256.03	1678.48
	250	259.50[a]	– –	– –	– –	0	255.83	1161.76	– –	0	240.17	1943.09
600	60	366.03[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	120	350.97[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	300	309.20[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
700	70	418.40[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	140	400.60[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	350	362.74[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
800	80	484.43[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	160	462.60[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	400	414.33[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
900	90	542.07[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	180	522.40[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	450	463.27[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
1000	100	605.10[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	200	583.30[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	500	514.80[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –

that each table row provides information for 30 problem instances of the same type. Table 5 is slightly different. The first column provides the instance name, while the second column indicates the best-known results from the literature. Moreover, each table row only covers one single problem instance. In the case of the LAPCS problem, the best-known results from the literature are additionally marked either by an *a*, indicating that an ILP-based heuristic has produced this result, or by a *b*, which indicates that the Hyb-Ea algorithm has generated this result. In Tables 2–4, the results of Cplex and Lsm are each provided in four columns. The first one (with heading **result**) contains the average solution quality obtained for the 30 problem instances. The second column (with heading $\bar{t}$) indicates the average computation time at which the best solution of a run was found, while the third column (with heading $\bar{t}_{\text{opt}}$) provides the average computation time at which optimality was proven. Finally, the fourth table column contains the number of instances that could be solved to optimality. This fourth table column is not provided in Table 5, as it only deals with one instance per table row. Furthermore, the results of Lscc-Bms are given in two columns in all cases, providing the (average) result and the (average) computation time. Note that a value in the columns with heading **result** is indicated in bold font if the value is at least as good as the best known one from the literature. Moreover, a value is marked by an asterisk in case it corresponds to a new best-known result. Finally, results of Cplex and Lmc are marked by a grey background if they correspond to provenly optimal results.

The following observations can be made in the case of the RFLCS problem:

- While both Lmc and Lscc-Bms are able to provide feasible solutions for all problem instances from both sets (Rflcs-Set1 and Rflcs-Set2), Cplex suffers from a sharp phase transition when the conflict graphs

Table 5: Experimental results for LAPCS instances Lapcs-Real.

Inst.	Spec.	Cplex			Lmc			Lscc-Bms	
Name	**Tech.**	**result**	t	t_{opt}	**result**	t	t_{opt}	**result**	t
Real_1	268^b	–	–	–	259	2691.58	–	231	3504.69
Real_2	291^b	–	–	–	283	637.94	–	216	1088.45
Real_3	294^b	–	–	–	284	104.13	–	234	1580.28
Real_4	374^b	–	–	–	**374**	34.59	–	366	2148.66
Real_5	178^b	–	–	–	**179***	6.04	–	170	2336.97
Real_6	209^b	–	–	–	206	30.64	–	197	2181.59
Real_7	330^b	–	–	–	**330**	43.61	–	251	1461.38
Real_8	177^b	–	–	–	175	3309.91	–	173	448.26
Real_9	302^b	–	–	–	**304***	44.36	–	226	49.66
Real_10	361^a	–	–	–	**361**	71.14	–	272	496.70

become too large. Observe, for example, the case ($|\Sigma| = n/8, n = 256$) in Table 2 in comparison to the next larger case ($|\Sigma| = n/8, n = 512$). While Cplex is able to solve all instances of the first case to optimality, it only provides very short solutions in the second case.

- Concerning the comparison of the two exact solvers, we can state that Lmc (the MC solver) clearly outperforms Cplex. Lmc is able to solve 1282 Rflcs-Set1 instances and 1237 Rflcs-Set2 instances to optimality, while Cplex can only solve 1221 Rflcs-Set1 instance and 1181 Rflcs-Set2 instances to optimality. Moreover, Lmc does not suffer from the above-mentioned phase transition for the remaining instances, and it requires generally less computation time. More specifically, while Lsm requires—on average—41.7 seconds for proving optimality (if possible) of Rflcs-Set1 instances, Cplex requires 187.2 seconds; respetively 34.07 and 127.14 seconds in the case of the Rflcs-Set2 instances.

- The heuristic MC solver Lscc-Bms is especially successful in those cases in which the exact techniques start to fail. See, for example, cases ($|\Sigma| = n/8, n \in \{512, 1024\}$) in Table 2 and cases ($|\Sigma| = 256, \text{reps} \in \{6, 7, 8\}$) in Table 3. Lscc-Bms can be seen as the most successful one among the techniques, providing new best-known results in 35 cases (considering both instance sets together), while Lmc provides new best-known results in 30 cases and Cplex in 24 cases.

All in all we can state that the idea of solving the RFLCS problem by means of the transformation to the MC problem is very successful, even before trying to reduce the size of the conflict graphs.

Let us now turn towards the LAPCS problem. In some aspects, the observations that can be made in the context of the artificial instances (Lapcs-Arti; Table 4) are similar to the ones made for the RFLCS problem. Cplex suffers from a sharp phase transition. In fact, it is only able to provide solutions for the case of the smallest problem instances ($n = 100$). Lmc does not suffer from this phase transition and is able to provide feasible solutions of reasonable quality until instances with input strings of length $n = 500$. Both Lmc are Cplex are able to solve 80 problem instances to optimality. And finally, the heuristic MC solver Lscc-Bms is again very successful in those cases in which Lmc and Cplex start to fail proving optimality (see the instances with $n = 200$). Concerning the results obtained for the real instances (Lapcs-Real; Table 5), we can state that Lsm is, by far, the most successful algorithm. While Cplex is not able to derive any feasible solutions and Lscc-Bms never matches the best results from the literature, Lsm matches the best results from the literature in three cases and obtains new best-known solutions in two additional cases. Nevertheless, we can state that the results—obtained before trying to reduce the size of the conflict graphs—are rather unsatisfactory in the context of the LAPCS problem. The main reason for this is the increased size of the conflict graphs in comparison to the RFLCS problem, which is due to the small alphabet size of four.

4.2 Results after conflict graph reduction

After reducing all the conflict graphs with the method described in Section 3, we first measured the amount of reduction that was achieved. This reduction is displayed for all RFLCS and LAPCS problem instances by

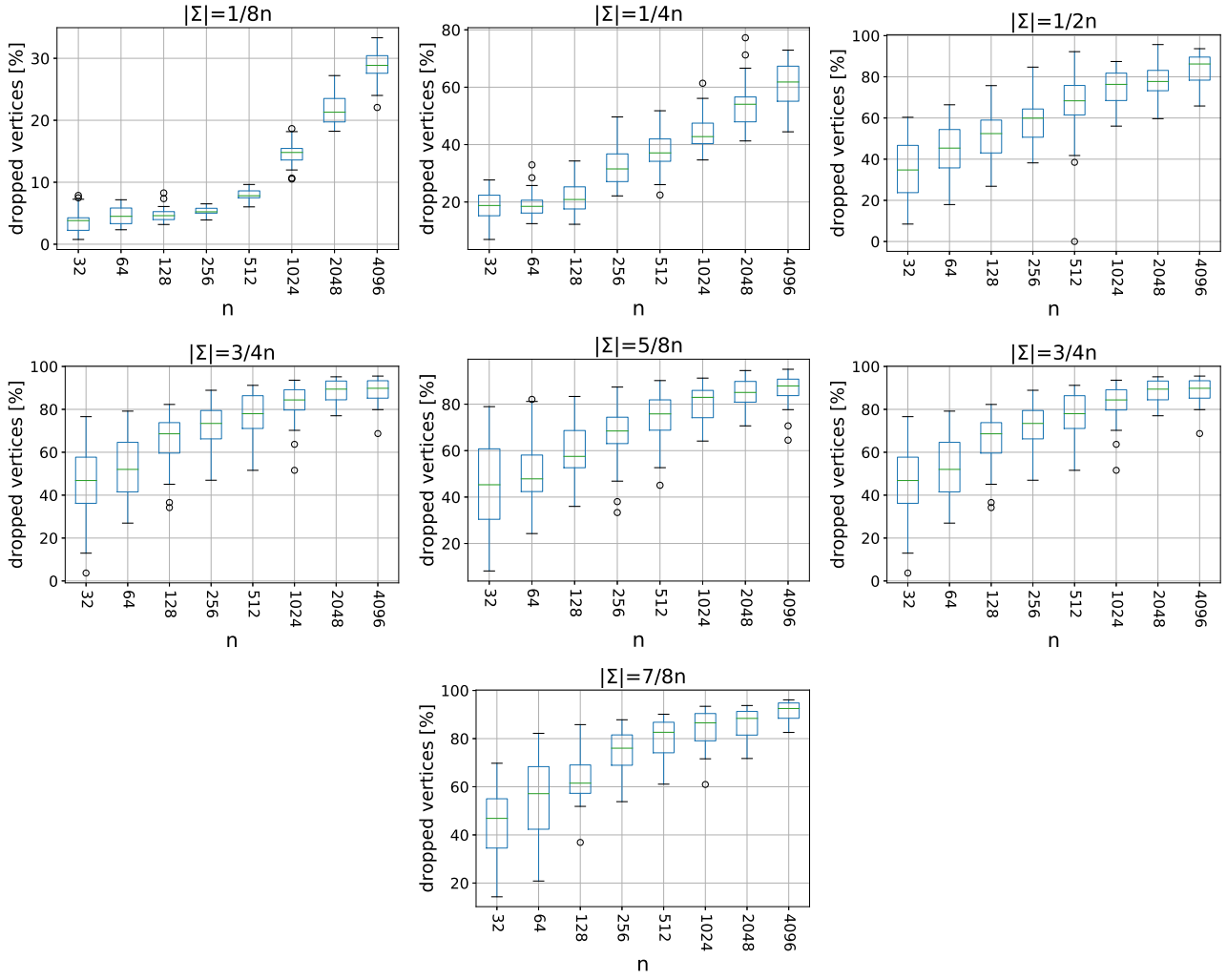

Figure 6: Graph reduction (in %) for RFLCS instances from set RFLCS-SET1.

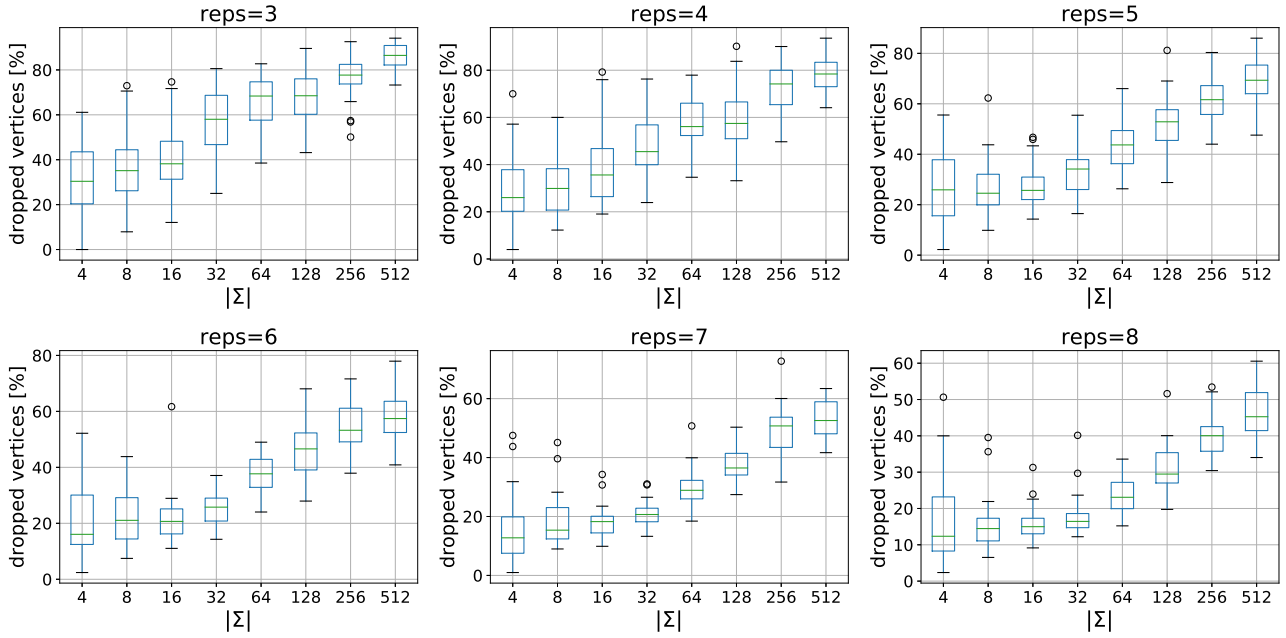

Figure 7: Graph reduction (in %) for RFLCS instances from set RFLCS-SET2.

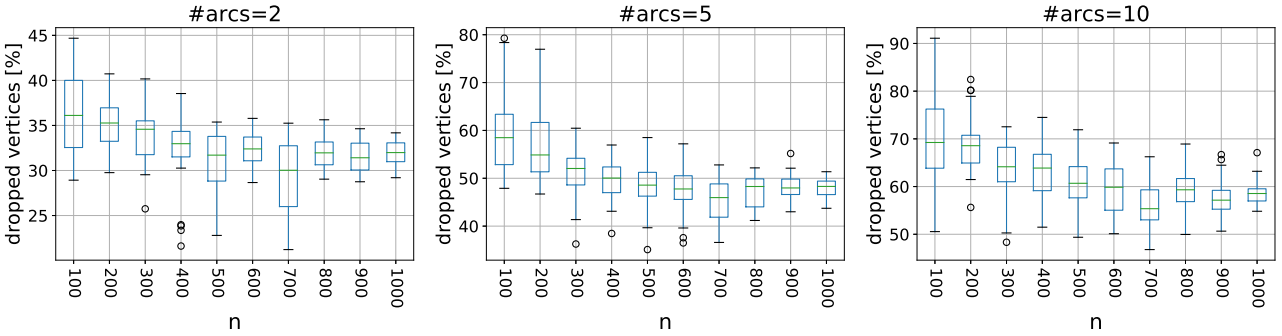

Figure 8: Graph reduction (in %) for LAPCS instances from set LAPCS-ARTI.

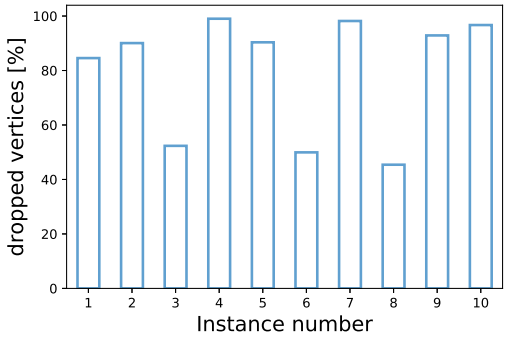

Figure 9: Graph reduction (in %) for LAPCS instances from set LAPCS-REAL.

means of boxplots in Figures 6–9. More specifically, the boxplots show the percentage reduction concerning the number of vertices of the original conflict graphs. If the reduction for an instance is at 60%, for example, this means that the reduction technique was able to remove 60% of the vertices of the original conflict graph. In the context of the RFLCS instances, we can state that the percentage reduction tends to grow with a growing string length and a growing alphabet size. Note that for long strings on large alphabets we were able to achieve reduction percentages of more than 90%. Concerning the LAPCS problem, it can be observed that the reduction percentages grow with an increasing number of arc annotations. However, they slightly increase with a growing input string length. This is due to the small alphabet size of four. Finally, it is worth mentioning that in the case of the real problem instances (set LAPCS-ARTI; Figure 9) we were able to achieve very high reduction percentages, sometimes well over 90%. This indicates the difference in structure between artificial and real problem instances.

The numerical results obtained by the three considered techniques after conflict graph reduction are provided in Tables 8–11 that can be found in Appendix A. The structure of these tables is the same as the one of Tables 2–5 which was described at the beginning of Section 4.1. However, in order to relate the two sets of results, the values in the columns with heading **result** are marked in a different way. More specifically, values marked by a preceding =-symbol are equal to the values obtained by the same technique before graph reduction. Furthermore, values marked in italic font and by a preceding −-symbol are worse than the values obtained by the same technique before graph reduction, and values marked in bold font and by a preceding +-symbol are better than the corresponding values before conflict graph reduction.

In order to relate the performance of a technique before graph reduction with its performance after graph reduction, we also computed a set of measures that are provided in Table 6 for CPLEX and LMC, and in Table 7 for LSCC-BMS. The measures regarding the exact techniques (see Table 6) are as follows.

1. **Measure E-M1** refers to those instances that were solved to optimality, both concerning the original conflict graph and the reduced conflict graph. In particular, it provides the average time saved for finding the best solution of a run (in seconds) after reducing the respective graph.

2. **Measure E-M2** is very similar, just that it refers to the average time saving for proving optimality.

Table 6: Differences in performance of the exact methods (CPLEX and LMC) summarized for the four different data sets. The five measures (E-M1–E-M5) are described in the text.

Data set	CPLEX					LMC				
	E-M1	**E-M2**	**E-M3**	**E-M4**	**E-M5**	**E-M1**	**E-M2**	**E-M3**	**E-M4**	**E-M5**
RFLCS-SET1	144.29	172.77	257	61.29	60	9.33	32.99	84	0.24	0
RFLCS-SET2	78.90	105.19	133	82.34	30	5.58	14.90	10	0.20	0
LAPCS-ARTI	330.89	526.45	30	-0.39	120	29.70	20.02	0	-0.0004	150
LAPCS-REAL	– –	– –	7	– –	9	– –	– –	5	0.0	0

Table 7: Differences in performance of the heuristic method (LSCC-BMS) summarized for the four different data sets. The four measures (H-M1–H-M4) are described in the text.

Data set	LSCC-BMS			
	H-M1	**H-M2**	**H-M3**	**H-M4**
RFLCS-SET1	77.34	296	27	0
RFLCS-SET2	39.41	114	9	0
LAPCS-ARTI	176.21	280	33	30
LAPCS-REAL	– –	9	1	0

3. **Measure E-M3** indicates the number of instances additionally solved to optimality after graph reduction.

4. **Measure E-M4** indicates the average improvement in solution quality (in percent) for all those instances for which feasible solutions can be found both before and after graph reduction, but for which optimality cannot be proven.

5. Finally, **measure E-M5** reports on the number of instances for which a feasible (and possibly optimal) solution can be found after graph reduction, and for which no feasible solution could be found before graph reduction.

In the context of the heuristic MC solver LSCC-BMS (see Table 7), measures H-M1–H-M4 can be described as follows.

1. In all those cases in which the same result is obtained by LSCC-BMS before and after conflict graph reduction, **measure H-M1** refers to the average time saving per instance (in seconds) for achieving this result.

2. **Measure H-M2** indicates the number of instances for which the result of LSCC-BMS improves after graph reduction.

3. **Measure H-M3** refers to the number of cases in which the result gets worse.

4. Finally, **measure H-M4** counts the number of instances for which LSCC-BMS can find a feasible solution after graph reduction, while before graph reduction LSCC-BMS was not able to find any feasible solution.

Remarks concerning the results for the RFLCS problem:

- The great beneficiary of the applied conflict graph reduction is CPLEX. CPLEX is now able to solve 1478 RFLCS-SET1 instances (out of 1680) and 1314 RFLCS-SET2 instances (out of 1440) to optimality, while LMC now solves 1366 RFLCS-SET1 instances and 1247 RFLCS-SET2 instances to optimality. Nevertheless, CPLEX still suffers from a sharp phase transition which, due to the graph reduction, has been moved to larger problem instances. Also the time savings achieved for finding the best solutions of a run and for proving optimality are much higher in the case of CPLEX when compared to those of LMC (see Table 6).

- The heuristic MC solver LSCC-BMS is also able to profit from the graph reduction. It provides an improved result for 296 RFLCS-SET1 instances and for 114 RFLCS-SET2 instances, while worse results are only produced in 27, respectively 9, cases. Moreover, in those cases in which LSCC-BMS obtains the same

result before and after graph reduction, the average time saving per instance is approx. 77 seconds for the RFLCS-SET1 instances, and approx. 39 seconds for the RFLCS-SET2 instances.

Finally, after studying the results obtained for the LAPCS instances, the following observations can be made:

- Concerning the set of artificial problem instances (LAPCS-ARTI), it can be observed that all three techniques are now able to provide solutions for some of the larger instances. CPLEX, for example, can now provide solutions for the instances with $n = 200$ and for the case ($n = 300, n_{\text{arcs}} = 30$), for which no result was obtained before conflict graph reduction. However, while LSCC-BMS is able to improve its results for many instances (see cases $n \in \{200, \ldots, 500\}$), LSM is again not able to take much profit from the graph reduction. In fact, the results of LSM after graph reduction are sometimes even worse than before; see case $n = 200$ and $n_{\text{arcs}} \in \{20, 40\}$, for example. On average, LSM is not able to improve its results for those instances for which a feasible solution was obtained before and after conflict graph reduction, but for which optimality could not be proven; see measure E-H4 in Table 6. CPLEX is now able to solve 110 problem instances to optimality, while LSM can solve 80 problem instances, the same ones that it was able to solve before conflict graph reduction. Again, LSCC-BMS performs best when the performance of CPLEX and LSM starts to decline (see the cases with $n = 200$).
- Finally, the results—in particular those of CPLEX—for the real-life instances of set LAPCS-REAL are quite pleasing. CPLEX is able to solve seven out of 10 instances to optimality. In three of these cases, the best-known result from the literature is improved. LSM, on the other side, obtains exactly the same results as before conflict graph reduction, with the difference that optimality can be proven now for five out of the 10 problem instances. LSCC-BMS is again able to take profit from the graph reduction, improving its results in 9 out of 10 cases.

5 Conclusions and Future Work

In this work we presented a way to transform longest common subsequence problems into maximum clique problems. Moreover, we presented a technique for the reduction of the resulting graphs, based on high-quality primal bounds. The benefits of this approach were experimentally studied in the context of two longest common subsequence variants: (1) the repetition-free longest common subsequence (RFLCS) problem and (2) the longest arc-preserving common subsequence (LAPCS) problem. Both problem variants are shown to be $\mathcal{NP}$-hard even for two input strings. We compared the application of CPLEX for solving the maximum independent set problem, which is the complimentary problem of the maximum clique problem, with the application of recent heuristic and exact maximum clique solvers. The three approaches were applied both before and after graph reduction. The best results were obtained after graph reduction, even though the impact of graph reduction was very different for the three solvers. Summarizing, we were able to solve 2613 of the 3120 RFLCS instances to optimality. Moreover, 110 out of 900 artificially created LAPCS problem instances were solved to optimality. In the context of the LAPCS problem, it was especially pleasing to see seven out of 10 real-life instances solved to optimality for the first time.

Concerning future work, we plan to study further techniques for graph reduction in order to be able to apply the utilized solvers to even larger problem instances, in particular regarding the LAPCS problem. Moreover, we plan to study additional variants of the longest common subsequence problem.

References

C. S. Iliopoulos, M. Sohel Rahman, A new efficient algorithm for computing the longest common subsequence, Theory of Computing Systems 45 (2009) 355–371.

M. Castelli, S. Beretta, L. Vanneschi, A hybrid genetic algorithm for the repetition free longest common subsequence problem, Operations Research Letters 41 (2013) 644–649.

D. Gusfield, Algorithms on Strings, Trees, and Sequences, Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.

T. Smith, M. Waterman, Identification of common molecular subsequences, Journal of Molecular Biology 147 (1981) 195–197.

T. Jiang, G. Lin, B. Ma, K. Zhang, A general edit distance between RNA structures, Journal of Computational Biology 9 (2002) 371–388.

J. B. Kruskal, An overview of sequence comparison: Time warps, string edits, and macromolecules, SIAM review 25 (1983) 201–237.

P. Brisk, A. Kaplan, M. Sarrafzadeh, Area-efficient instruction set synthesis for reconfigurable system-on-chip design, in: Proceedings of DAC 2004 – The 41st Design Automation Conference, IEEE press, 2004, pp. 395–400.

J. Storer, Data Compression: Methods and Theory, Computer Science Press, MD, 1988.

A. Aho, J. Hopcroft, J. Ullman, Data structures and algorithms, Addison-Wesley, Reading, MA, 1983.

D. Maier, The complexity of some problems on subsequences and supersequences, Journal of the ACM 25 (1978) 322–336.

S. S. Adi, M. D. V. Braga, C. G. Fernandes, C. E. Ferreira, F. V. Martinez, M.-F. Sagot, M. A. Stefanes, C. Tjandraatmadja, Y. Wakabayashi, Repetition-free longest common subsquence, Discrete Applied Mathematics 158 (2010) 1315–1324.

Y.-T. Tsai, The constrained longest common subsequence problem, Information Processing Letters 88 (2003) 173–176.

Y.-C. Chen, K.-M. Chao, On the generalized constrained longest common subsequence problems, Journal of Combinatorial Optimization 21 (2011) 383–392.

P. Bonizzoni, G. Della Vedova, R. Dondi, Y. Pirola, Variants of constrained longest common subsequence, Information Processing Letters 110 (2010) 877–881.

I. M. Bomze, M. Budinich, P. M. Pardalos, M. Pelillo, The maximum clique problem, in: Handbook of combinatorial optimization, Springer, 1999, pp. 1–74.

E. K. Lee, T. Easton, K. Gupta, Novel evolutionary models and applications to sequence alignment problems, Annals of Operations Research 148 (2006) 167–187.

H. Jiang, C.-M. Li, F. Manyà, Combining efficient preprocessing and incremental MaxSAT reasoning for MaxClique in large graphs, in: Proceedings of ECAI 2016 – 22nd European Conference on Artificial Intelligence, 2016, pp. 939–947.

C.-M. Li, H. Jiang, F. Manyà, On minimization of the number of branches in branch-and-bound algorithms for the maximum clique problem, Computers & Operations Research 84 (2017) 1–15.

Y. Wang, S. Cai, M. Yin, Two efficient local search algorithms for maximum weight clique problem., in: Proceedings of AAAI 2016 – Conference on Artificial Intelligence, 2016, pp. 805–811.

C. Blum, P. Festa, Metaheuristics for String Problems in Bio-informatics, John Wiley & Sons, 2016.

E. K. Lee, K. Gupta, Algorithms for genomic analysis, in: C. A. Floudas, P. M. Pardalos (Eds.), Encyclopedia of Optimization, Springer US, 2009, pp. 33–54.

P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE transactions on Systems Science and Cybernetics 4 (1968) 100–107.

Q. Wang, M. Pan, Y. Shang, D. Korkin, A fast heuristic search algorithm for finding the longest common subsequence of multiple strings, in: M. Fox, D. Poole (Eds.), Proceedings of AAAI 2010 – Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI press, 2010, pp. 1287–1292.

M. Djukanovic, G. R. Raidl, C. Blum, Finding Longest Common Subsequences: New Hybrid A* Search Results, Technical Report AC-TR-19-008, Algorithms and Complexity Group, TU Wien, 2019. URL: http://www.ac.tuwien.ac.at/files/tr/ac-tr-19-008.pdf.

M. Djukanovic, G. Raidl, C. Blum, Exact and heuristic approaches for the longest common palindromic subsequence problem, in: Proceedings of LION12 – 12th International Conference on Learning and Intelligent Optimization, Lecture Notes in Computer Science, Springer, 2018, pp. 199–214. In press.

S. G. Vadlamudi, P. Gaurav, S. Aine, P. P. Chakrabarti, Anytime column search, in: Proceedings of AI2012 – 25th Australasian Joint Conference on Artificial Intelligence, Springer, 2012, pp. 254–265.

C. Blum, M. J. Blesa, A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem, Journal of Heuristics 24 (2018) 551–579.

P. A. Evans, Finding common subsequences with arcs and pseudoknots, in: M. Crochemore, M. Paterson (Eds.), Proceedings of CPM 1999 – 10th Annual Symposium on Combinatorial Pattern Matching, volume 1645 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 1999a, pp. 270–280.

P. A. Evans, Algorithms and Complexity for Annotated Sequence Analysis, Ph.D. thesis, University of Victoria, 1999b.

C. Blum, M. J. Blesa, Hybrid techniques based on solving reduced problem instances for a longest common subsequence problem, Applied Soft Computing 62 (2018) 15–28.

J. W. Brown, The ribonuclease P database, Nucleic Acids Research 27 (1999) 314–314.

S. R. Chowdhury, M. M. Hasan, S. Iqbal, M. S. Rahman, Computing a longest common palindromic subsequence, Fundamenta Informaticae 129 (2014) 329–340.

M. M. Hasan, A. S. M. S. Islam, M. S. Rahman, A. Sen, Palindromic subsequence automata and longest common palindromic subsequence, Mathematics in Computer Science 11 (2017) 219–232.

S. Inenaga, H. Hyyrö, A hardness result and new algorithm for the longest common palindromic subsequence problem, Information Processing Letters 129 (2018) 11–15.

A. Abboud, A. Backurs, V. V. Williams, Tight hardness results for lcs and other sequence similarity measures, in: Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on, IEEE, 2015, pp. 59–78.

C. Calabro, R. Impagliazzo, R. Paturi, The complexity of satisfiability of small depth circuits, in: Parameterized and Exact Computation, Springer, 2009, pp. 75–85.

C. Blum, M. J. Blesa, M. López-Ibáñez, Beam search for the longest common subsequence problem, Computers & Operations Research 36 (2009) 3178–3186.

C. B. Fraser, Subsequences and supersequences of strings, Ph.D. thesis, University of Glasgow, 1995.

Q. Wang, D. Korkin, Y. Shang, A fast multiple longest common subsequence (MLCS) algorithm, IEEE Transactions on Knowledge and Data Engineering 23 (2011) 321–334.

Q. Wang, M. Pan, Y. Shang, D. Korkin, A fast heuristic search algorithm for finding the longest common subsequence of multiple strings, in: Proceedings of AAAI 2010 – Conference on Artificial Intelligence, 2010, pp. 1287–1292.

L. Wang, S.-Y. Wang, Y. Xu, An effective hybrid EDA-based algorithm for solving multidimensional knapsack problem, Expert Systems with Applications 39 (2012) 5593–5599.

C. M. Li, F. Manya, Maxsat, hard and soft constraints., in: Handbook of satisfiability, volume 185, 2009, pp. 613–631.

C. Blum, M. J. Blesa, A comprehensive comparison of metaheuristics for the repetition-free longest common subsequence problem, Journal of Heuristics 24 (2018) 551–579.

A Tables showing the results after graph reduction

Table 8: Results obtained after graph reduction (RFLCS instances of RFLCS-SET1).

$\|\Sigma\|$	n	Spec.	CPLEX				LMC				LSCC+BMS	
		Tech.	result	$\bar{t}$	$\bar{t}_{\text{opt}}$	#opt	result	$\bar{t}$	$\bar{t}_{\text{opt}}$	#opt	result	$\bar{t}$
$n/8$	32	4.00	=4.00	0.07	0.07	30	=4.00	0.00	0.03	30	=4.0	0.00
	64	8.00	=8.00	0.52	0.52	30	=8.00	0.00	0.11	30	=8.0	0.00
	128	16.00	=16.00	6.10	6.10	30	=16.00	0.00	76.21	30	=16.0	0.00
	256	31.97	=31.97	202.49	202.49	30	=31.90	18.82	– –	0	=31.97	0.03
	512	63.27	+**30.97**	1889.32	– –	0	*−62.40*	347.07	– –	0	=63.90	69.94
	1024	111.57	=0.03	1062.21	– –	0	=112.53	923.56	– –	0	+**116.30***	1099.26
	2048	182.67	– –	– –	– –	0	*−182.37*	1415.51	– –	0	+**182.40**	1443.72
	4096	283.33	– –	– –	– –	0	=281.37	1114.40	– –	0	+**263.83**	2079.95
$n/4$	32	7.83	=7.83	0.01	0.01	30	=7.83	0.00	0.02	30	=7.83	0.00
	64	14.67	=14.67	0.08	0.08	30	=14.67	0.00	0.03	30	=14.67	0.00
	128	25.77	=25.93	1.78	2.34	30	=25.93	0.01	0.12	30	=25.93	0.01
	256	43.70	=43.97	9.20	19.41	30	=43.97	0.13	0.67	30	=43.97	0.12
	512	67.90	+**68.57**	233.30	489.73	30	=68.57	95.78	104.92	29	=68.57	3.20
	1024	103.00	+**104.53**	1169.51	1969.63	14	*−103.73*	487.95	– –	0	+**105.00***	482.73
	2048	154.33	+**117.47**	1986.41	1001.01	3	*−152.73*	513.24	– –	0	+**153.93**	1229.79
	4096	226.67	+**23.17**	648.79	– –	0	+**224.63**	573.22	– –	0	+**215.00**	1510.06
$3n/8$	32	8.77	=8.77	0.00	0.00	30	=8.77	0.00	0.02	30	=8.77	0.00
	64	15.53	=15.53	0.03	0.03	30	=15.53	0.00	0.02	30	=15.53	0.00
	128	24.90	=24.90	0.33	0.34	30	=24.90	0.00	0.04	30	=24.90	0.01
	256	39.97	=39.97	0.81	0.98	30	=39.97	0.02	0.08	30	=39.97	0.06
	512	59.77	=59.97	9.15	14.02	30	=59.97	0.14	0.58	30	=59.97	0.39
	1024	90.50	+**90.73**	30.62	49.38	30	=90.73	2.93	19.30	30	=90.73	5.81
	2048	130.57	+**131.07***	323.31	369.50	28	+**130.57**	117.06	410.36	13	+**131.00**	726.79
	4096	191.37	+**186.27**	739.49	653.72	23	+**190.90**	399.25	– –	0	+**189.07**	1441.64
$n/2$	32	8.87	=8.87	0.00	0.00	30	=8.87	0.00	0.02	30	=8.87	0.00
	64	14.80	=14.80	0.01	0.01	30	=14.80	0.00	0.02	30	=14.80	0.00
	128	22.93	=22.93	0.05	0.06	30	=22.93	0.00	0.03	30	=22.93	0.00
	256	35.10	=35.20	0.30	0.34	30	=35.20	0.01	0.04	30	=35.20	0.03
	512	53.10	=53.13	1.41	1.78	30	=53.13	0.03	0.15	30	=53.13	0.16
	1024	79.03	=79.13	4.46	5.16	30	=79.13	0.24	0.60	30	=79.13	3.37
	2048	115.30	+**115.70***	26.72	29.61	30	+**115.67**	116.09	350.85	28	+**115.67**	174.36
	4096	167.47	+**167.97***	114.74	158.87	30	+**167.60**	36.16	208.53	18	+**167.43**	664.34
$5n/8$	32	8.60	=8.60	0.00	0.00	30	=8.60	0.00	0.02	30	=8.60	0.00
	64	13.30	=13.30	0.00	0.00	30	=13.30	0.00	0.02	30	=13.30	0.00
	128	21.20	=21.20	0.01	0.01	30	=21.20	0.00	0.02	30	=21.20	0.00
	256	32.53	=32.53	0.08	0.09	30	=32.53	0.00	0.03	30	=32.53	0.00
	512	47.83	=47.83	0.65	0.65	30	=47.83	0.02	0.06	30	=47.83	0.01
	1024	70.03	=70.20	1.29	1.32	30	=70.20	0.04	0.17	30	=70.20	0.25
	2048	103.80	+**103.97**	4.28	4.32	30	=103.97	0.49	3.18	30	+**103.97**	2.77
	4096	150.00	+**150.57***	50.27	50.36	30	+**150.43**	281.64	219.92	24	+**150.40**	365.03
$3n/4$	32	8.17	=8.17	0.00	0.00	30	=8.17	0.00	0.02	30	=8.17	0.00
	64	12.53	=12.53	0.00	0.00	30	=12.53	0.00	0.02	30	=12.53	0.00
	128	19.70	=19.70	0.00	0.00	30	=19.70	0.00	0.02	30	=19.70	0.00
	256	29.97	=29.97	0.02	0.02	30	=29.97	0.00	0.02	30	=29.97	0.00
	512	44.53	=44.57	0.17	0.19	30	=44.57	0.00	0.04	30	=44.57	0.02
	1024	65.07	=65.20	0.52	0.52	30	=65.20	0.02	0.09	30	=65.20	0.07
	2048	94.53	=94.67	0.87	0.90	30	=94.67	0.04	0.16	30	+**94.67**	1.29
	4096	136.57	+**136.77***	11.63	11.80	30	+**136.47**	171.71	3.79	26	+**136.73**	26.59
$7n/8$	32	7.67	=7.67	0.00	0.00	30	=7.67	0.00	0.02	30	=7.67	0.00
	64	11.57	=11.57	0.00	0.00	30	=11.57	0.00	0.02	30	=11.57	0.00
	128	18.40	=18.40	0.01	0.01	30	=18.40	0.00	0.02	30	=18.40	0.00
	256	27.80	=27.80	0.01	0.01	30	=27.80	0.00	0.02	30	=27.80	0.00
	512	40.57	=40.60	0.06	0.06	30	=40.60	0.00	0.02	30	=40.60	0.00
	1024	60.50	=60.57	0.34	0.35	30	=60.57	0.01	0.04	30	=60.57	0.09
	2048	88.00	=88.00	2.56	2.59	30	=88.00	0.05	8.89	30	=88.00	0.33
	4096	127.20	+**127.37***	2.93	2.97	30	+**127.37***	0.15	1.50	30	+**127.37***	2.17

Table 9: Results obtained after graph reduction (RFLCS instances of RFLCS-SET2).

$\|\Sigma\|$	*reps*	**Spec.**	CPLEX				LMC				LSCC-BMS	
		Tech.	**result**	$\bar{t}$	$\bar{t}_{\text{opt}}$	**#opt**	**result**	$\bar{t}$	$\bar{t}_{\text{opt}}$	**#opt**	**result**	$\bar{t}$
4	3	3.47	$^{=}$3.47	0.00	0.00	30	$^{=}$3.47	0.00	0.02	30	$^{=}$3.47	0.00
	4	3.77	$^{=}$3.77	0.00	0.00	30	$^{=}$3.77	0.00	0.02	30	$^{=}$3.77	0.00
	5	3.83	$^{=}$3.83	0.00	0.00	30	$^{=}$3.83	0.00	0.02	30	$^{=}$3.83	0.00
	6	3.90	$^{=}$3.90	0.00	0.00	30	$^{=}$3.90	0.00	0.02	30	$^{=}$3.90	0.00
	7	3.97	$^{=}$3.97	0.00	0.00	30	$^{=}$3.97	0.00	0.02	30	$^{=}$3.97	0.00
	8	3.97	$^{=}$3.97	0.01	0.01	30	$^{=}$3.97	0.00	0.02	30	$^{=}$3.97	0.00
8	3	6.23	$^{=}$6.23	0.00	0.00	30	$^{=}$6.23	0.00	0.02	30	$^{=}$6.23	0.00
	4	6.87	$^{=}$6.87	0.00	0.00	30	$^{=}$6.87	0.00	0.02	30	$^{=}$6.87	0.00
	5	7.40	$^{=}$7.40	0.00	0.00	30	$^{=}$7.40	0.00	0.02	30	$^{=}$7.40	0.00
	6	7.53	$^{=}$7.53	0.01	0.01	30	$^{=}$7.53	0.00	0.02	30	$^{=}$7.53	0.00
	7	7.70	$^{=}$7.70	0.02	0.02	30	$^{=}$7.70	0.00	0.02	30	$^{=}$7.70	0.00
	8	7.77	$^{=}$7.77	0.02	0.02	30	$^{=}$7.77	0.00	0.02	30	$^{=}$7.77	0.00
16	3	9.70	$^{=}$9.70	0.00	0.00	30	$^{=}$9.70	0.00	0.02	30	$^{=}$9.70	0.00
	4	11.57	$^{=}$11.57	0.01	0.01	30	$^{=}$11.57	0.00	0.02	30	$^{=}$11.57	0.00
	5	12.93	$^{=}$12.93	0.02	0.02	30	$^{=}$12.93	0.00	0.02	30	$^{=}$12.93	0.00
	6	14.00	$^{=}$14.00	0.05	0.05	30	$^{=}$14.00	0.00	0.02	30	$^{=}$14.00	0.00
	7	14.93	$^{=}$14.93	0.10	0.10	30	$^{=}$14.93	0.00	0.04	30	$^{=}$14.93	0.00
	8	14.80	$^{=}$14.80	0.17	0.17	30	$^{=}$14.80	0.00	0.05	30	$^{=}$14.80	0.01
32	3	16.13	$^{=}$16.13	0.01	0.01	30	$^{=}$16.13	0.00	0.02	30	$^{=}$16.13	0.00
	4	19.00	$^{=}$19.00	0.02	0.03	30	$^{=}$19.00	0.00	0.02	30	$^{=}$19.00	0.00
	5	21.63	$^{=}$21.63	0.28	0.30	30	$^{=}$21.63	0.00	0.03	30	$^{=}$21.63	0.01
	6	23.73	$^{=}$23.73	0.62	0.70	30	$^{=}$23.73	0.00	0.06	30	$^{=}$23.73	0.01
	7	25.53	$^{=}$25.57	1.79	1.93	30	$^{=}$25.57	0.02	0.13	30	$^{=}$25.57	0.02
	8	27.40	$^{=}$27.50	2.49	2.67	30	$^{=}$27.50	0.07	0.29	30	$^{=}$27.50	0.05
64	3	25.43	$^{=}$25.43	0.02	0.02	30	$^{=}$25.43	0.00	0.02	30	$^{=}$25.43	0.00
	4	30.37	$^{=}$30.37	0.24	0.26	30	$^{=}$30.37	0.00	0.03	30	$^{=}$30.37	0.01
	5	34.87	$^{=}$34.93	1.46	2.21	30	$^{=}$34.93	0.01	0.09	30	$^{=}$34.93	0.04
	6	39.07	$^{=}$39.13	5.46	8.21	30	$^{=}$39.13	0.04	0.25	30	$^{=}$39.13	0.11
	7	43.50	$^{=}$43.63	11.06	24.01	30	$^{=}$43.63	0.16	0.79	30	$^{=}$43.63	0.18
	8	45.17	$^{=}$45.53	35.48	84.08	30	$^{=}$45.53	1.69	6.06	30	$^{=}$45.53	0.44
128	3	36.70	$^{=}$36.77	0.18	0.18	30	$^{=}$36.77	0.00	0.03	30	$^{=}$36.77	0.02
	4	44.90	$^{=}$45.03	1.99	2.67	30	$^{=}$45.03	0.02	0.11	30	$^{=}$45.03	0.15
	5	53.23	$^{=}$53.43	7.90	10.76	30	$^{=}$53.43	0.12	0.42	30	$^{=}$53.43	0.30
	6	61.07	$^{=}$61.53	28.13	46.99	30	$^{=}$61.53	4.27	6.76	30	$^{=}$61.53	0.98
	7	67.90	$^{+}$**68.47**	125.08	421.63	30	$^{=}$68.47	9.47	57.64	30	$^{=}$68.47	1.98
	8	73.57	$^{+}$**74.50**	554.65	1321.01	18	$^{=}$74.30	608.14	544.04	13	$^{=}$74.60	10.65
256	3	54.97	$^{=}$55.03	0.74	0.79	30	$^{=}$55.03	0.02	0.06	30	$^{=}$55.03	0.04
	4	68.70	$^{=}$68.93	5.61	6.63	30	$^{=}$68.93	0.10	1.64	30	$^{=}$68.93	0.79
	5	81.00	$^{=}$81.43	28.37	40.16	30	$^{=}$81.43	3.51	7.68	30	$^{=}$81.43	3.13
	6	93.10	$^{+}$**93.60***	227.62	476.17	30	$^{+}$**93.17**	124.54	309.18	17	$^{+}$**93.60***	47.19
	7	103.50	$^{+}$**104.27**	667.00	1338.10	24	$^{-}$*103.13*	156.41	176.14	3	$^{+}$**104.47***	262.17
	8	113.70	$^{+}$**112.07**	2277.45	1367.53	1	$^{=}$113.10	344.01	– –	0	$^{+}$**115.00***	1009.44
512	3	81.57	$^{=}$81.63	0.52	0.54	30	$^{=}$81.63	0.02	0.35	30	$^{=}$81.63	0.12
	4	100.83	$^{+}$**101.13**	10.22	10.99	30	$^{+}$**101.10**	17.70	23.64	30	$^{=}$101.13	4.44
	5	120.43	$^{+}$**121.03***	147.62	209.48	30	$^{+}$**120.03**	249.84	866.25	14	$^{+}$**121.03***	226.89
	6	137.03	$^{+}$**136.97**	1335.77	771.58	11	$^{+}$**136.47**	499.84	47.78	1	$^{+}$**137.80***	1064.07
	7	154.57	$^{+}$**111.40**	2118.90	– –	0	$^{-}$*152.27*	823.67	– –	0	$^{+}$**153.33**	1619.19
	8	172.10	$^{+}$**13.97**	577.60	– –	0	$^{-}$*169.73*	620.92	– –	0	$^{+}$**168.70**	1557.06

Table 10: Results for LAPCS instances of set Lapcs-Arti after graph reduction.

n	n_{arcs}	Spec.	Cplex				Lmc				Lscc-Bms	
		Tech.	result	$\bar{t}$	$\bar{t}_{\text{opt}}$	#opt	result	$\bar{t}$	$\bar{t}_{\text{opt}}$	#opt	result	$\bar{t}$
	10	60.17[a]	=60.20	6.57	8.56	30	=60.20	38.48	85.52	30	=60.20	0.46
100	20	58.13[a]	=58.20	12.46	22.41	30	=58.17	15.96	361.00	29	=58.20	0.80
	50	51.87[a]	=52.03	637.48	1150.86	24	=52.07	145.51	1105.66	21	=52.10	2.48
	20	121.70[b]	+**122.60**	956.11	1072.96	22	−*120.20*	635.33	– –	0	+**122.63***	671.81
200	40	116.70[b]	+**111.80**	2475.25	1972.97	4	−*115.80*	627.34	– –	0	+**118.40***	748.94
	100	104.57[a]	+**0.07**	302.02	– –	0	=104.30	539.94	– –	0	+**106.87***	1048.04
	30	181.30[a]	+**22.10**	659.71	– –	0	+**178.23**	643.25	– –	0	+**178.63**	1495.86
300	60	174.97[a]	– –	– –	– –	0	=171.80	507.46	– –	0	+**172.20**	1338.14
	150	157.13[a]	– –	– –	– –	0	+**155.97**	991.12	– –	0	+**156.97**	1410.26
	40	242.70[b]	– –	– –	– –	0	+**239.73**	650.89	– –	0	+**230.77**	1616.86
400	80	233.23[a]	– –	– –	– –	0	=226.97	578.27	– –	0	+**221.80**	1657.25
	200	208.77[a]	– –	– –	– –	0	−*205.17*	705.84	– –	0	+**202.27**	2035.29
	50	302.27[b]	– –	– –	– –	0	−*295.83*	847.81	– –	0	+**278.47**	1819.17
500	100	291.23[a]	– –	– –	– –	0	+**284.70**	891.25	– –	0	+**266.70**	1470.81
	250	259.50[a]	– –	– –	– –	0	−*255.80*	1116.97	– –	0	+**242.57**	1686.56
	60	366.03[b]	– –	– –	– –	0	+**356.83**	773.68	– –	0	+**323.13**	1598.87
600	120	350.97[a]	– –	– –	– –	0	+**341.40**	987.06	– –	0	– –	– –
	300	309.20[a]	– –	– –	– –	0	+**306.63**	906.20	– –	0	– –	– –
	70	418.40[b]	– –	– –	– –	0	+**386.93**	708.66	– –	0	– –	– –
700	140	400.60[a]	– –	– –	– –	0	+**40.23**	169.30	– –	0	– –	– –
	350	362.74[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	80	484.43[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
800	160	462.60[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	400	414.33[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	90	542.07[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
900	180	522.40[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	450	463.27[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	100	605.10[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
1000	200	583.30[a]	– –	– –	– –	0	– –	– –	– –	0	– –	– –
	500	514.80[b]	– –	– –	– –	0	– –	– –	– –	0	– –	– –

Table 11: Results for LAPCS instances of set Lapcs-Real after graph reduction.

Inst.	Spec.	Cplex			Lmc			Lscc-Bms	
Name	Tech.	result	t	t_{opt}	result	t	t_{opt}	result	t
Real_1	268[b]	+**273***	339.70	339.76	=259	2489.55	– –	+**272**	847.60
Real_2	291[b]	+**291**	21.84	21.85	=283	416.57	– –	+**291**	903.86
Real_3	294[b]	– –	– –	– –	=284	49.69	– –	+**263**	1360.83
Real_4	374[b]	+**374**	0.02	0.02	=374	0.01	0.07	+**374**	0.00
Real_5	178[b]	+**179**	4.62	4.63	=179	1.38	351.62	+**179**	11.32
Real_6	209[b]	+**0**	2376.75	– –	=206	15.19	– –	+**204**	2441.29
Real_7	330[b]	+**330**	0.05	0.05	=330	0.73	1.12	+**330**	758.07
Real_8	177[b]	+**1**	1281.87	– –	=175	2760.20	– –	−*172*	3025.38
Real_9	302[b]	+**304**	1.47	1.81	=304	16.41	54.18	+**304**	498.12
Real_10	361[a]	+**361**	0.23	0.23	=361	1.49	10.50	+**361**	74.89