Journal of Discrete Algorithms, CPM 2009 Special Issue

Deconstructing Intractability—A Multivariate Complexity Analysis of Interval Constrained Coloring[☆]

Christian Komusiewicz[1], Rolf Niedermeier, Johannes Uhlmann[2]

Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{*c.komus, rolf.niedermeier, johannes.uhlmann*}*@uni-jena.de*

Abstract

The NP-hard Interval Constrained Coloring (ICC) problem appears in the interpretation of experimental data in biochemistry dealing with protein fragments. Given a set of m integer intervals in the range 1 to n and a set of m associated multisets of colors (specifying for each interval the colors to be used for its elements), one asks whether there is a "consistent" coloring for all integer points from $\{1, \ldots, n\}$ that complies with the constraints specified by the color multisets. We thoroughly analyze a known NP-hardness proof for ICC. In this way, we identify numerous parameters that naturally occur in ICC and strongly influence its practical solvability. Accordingly, we present several positive (fixed-parameter) tractability results exploiting various parameterizations. We substantiate the usefulness of this "multivariate algorithmics approach" by presenting experimental results with real-world data.

1. Introduction

Althaus et al. [2, 1] identified Interval Constrained Coloring as an important combinatorial problem in the context of automated mass spectrometry and the determination of the 3-dimensional structure of proteins. It builds the key to replace a manual interpretation of exchange data for peptic fragments with computer-assisted methods, see Althaus et al. [2] for more on the biochemical background and further motivation. The NP-complete decision problem Interval Constrained Coloring (ICC) deals with matching color multi-

[☆]A preliminary version of this paper appeared under the title "Deconstructing Intractability—A Case Study for Interval Constrained Coloring" in the proceedings of the 20th Annual Symposium on Combinatorial Pattern Matching (CPM'09), volume 5577 in LNCS, pages 207–220, Springer 2009. In particular, this revised and extended version now additionally contains experimental results based on implementations of some of our algorithms.

[1]Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.
[2]Supported by the DFG, research project PABI, NI 369/7.

sets with integer intervals and can be formalized as follows.[3] To this end, for two positive integers i, j with $i \leq j$, let $[i, j] := \{k \in \mathbb{N} \mid i \leq k \leq j\}$. In addition, for $i \geq 1$ let $[i]$ denote the interval $[1, i]$.

> **Input:** A positive integer n, a multiset of m integer intervals $\mathcal{F} = \{F_1, \ldots, F_m\}$, all within $[n]$, and a multiset of m multisets of colors $\mathcal{C} = \{C_1, \ldots, C_m\}$ over k different colors.
> **Question:** Is there a coloring $c : [n] \to [k]$ such that for each interval $F_i \in \mathcal{F}$ it holds that $C_i = c(F_i)$?

Herein, $c(F_i)$ denotes the *multiset* of colors assigned by c to the integer points in the interval F_i. Throughout this paper, we assume that the input intervals cover $[n]$, since otherwise the input instance can be decomposed into independent subinstances. Moreover, we say that a coloring $c : [n] \to [k]$ *satisfies* an input interval F_i if $C_i = c(F_i)$. Finally, a coloring satisfying all input intervals is called *proper*.

From a biochemical point of view, the intervals correspond to (typically overlapping) fragments of a protein with n residues, and the k colors correspond to k different exchange rates that need to be assigned consistently to the n residues [2, 1]. The color multisets correspond to experimentally found bulk information that needs to be matched with the residues and can be interpreted as constraints that describe a set of valid colorings of the interval $[n]$. Note that, from an applied point of view, if not all constraints (that is, intervals that completely match with a given color multiset) can be fulfilled, then it is also important to investigate the corresponding optimization problems where one wants to maximize the number of fulfilled constraints [1]. However, we mainly focus on analyzing the complexity of the decision problem. In the case of yes-instances, most of our algorithms can be easily adapted to provide a corresponding coloring.

Known results. The algorithmic study of ICC has been initiated by Althaus et al. [2, 1]. ICC has been shown to be NP-complete by a reduction from the Exact Cover problem [1]. In a more applied paper [2], besides first introducing and formalizing the problem, an algorithm based on integer linear programming and branch-and-bound was presented that enumerates all valid (fulfilling all constraints) color mappings c. In particular, it was shown that in the case of $k = 2$ colors a direct combinatorial algorithm leads to polynomial-time solvability. The computational complexity of the case $k = 3$ was left open by Althaus et al. [2]. Byrka et al. [5] filled this gap by showing the NP-completeness of ICC for $k = 3$. The corresponding reduction is from 3-Satisfiability. Moreover, concerning optimization, in a similar way they also showed that the "gap version" of this restricted case is NP-hard, also implying its APX-hardness.

[3] Compared with Althaus et al. [2, 1] we choose a somewhat different but equivalent formalization here; this problem definition turns out to be more suitable for our subsequent studies.

Successful experiments with real-world instances with $n < 60$, $m \leq 50$, $k = 3$ and randomly generated instances with $n \leq 1000$, $m = n/2$, and $k = 3$ have been performed [2]. In a more theoretical paper [1], besides the NP-completeness proof, the preceding work [2] has been continued by providing results concerning polynomial-time approximability. In particular, there is an algorithm producing a coloring where all requirements are matched within an additive error of one if the LP-relaxation of the presented integer program for ICC has a feasible solution. This algorithm is based on a sophisticated polyhedral approach combined with recent randomized rounding techniques. Finally, Canzar et al. [6] provided a new method (using linear programming and backtracking) for enumerating all exact and further approximate solutions with polynomial delay between two successive outputs and using polynomial space. They confirmed the practical use of their approach by experiments.

Our contributions. This work proposes a fresh view on ICC and the development of exact algorithms for NP-hard combinatorial problems in general. The fundamental starting point here is to deconstruct proofs of NP-hardness in order to obtain new insights into the combinatorial structure of problems. The point is to analyze how different parameters occurring in a problem contribute to its computational complexity. This is where parameterized algorithmics [8, 12, 17] comes into play. Indeed, as it turns out, ICC gives a prime example for the continuing evolution of parameterized algorithmics into multivariate algorithmics [9, 18]. For ICC, there is a big number of useful parameterizations, all naturally deduced from deconstructing the known NP-hardness proof. In this line, for instance, we can show a fixed-parameter tractability result with respect to the parameter "maximum interval length". Whereas, unless P = NP, the problem is not fixed-parameter tractable with respect to the color parameter k alone [5], it is with respect to the combined parameter (n, k); that is, there is an algorithm with time complexity $(k-1)^n \cdot \text{poly}(n, m)$. These algorithms are of practical interest when the corresponding parameter values are sufficiently small. For instance, note that all experiments of Althaus et al. [2] were performed having $k = 3$ and $n \leq 60$ for real-world instances. Indeed, in the already NP-complete case of $k = 3$ we can further improve the running time to $1.89^n \cdot \text{poly}(n, m)$. In this spirit, in Section 4 we investigate a number of "single parameterizations", and in Section 5 we consider "combined parameterizations". Moreover, whereas ICC is NP-complete for "cutwidth" three [1], we present a combinatorial polynomial-time algorithm for cutwidth two.[4] Tables 1 and 2 in Sections 4 and 5 survey the current state of the art and our new results concerning (combinatorial) algorithms that can efficiently solve ICC in case of favorable parameter constellations. Finally, in Section 6, we report positive experimental results based on implementations of some of our new algorithms. We conclude with a discussion and some open questions in Section 7.

[4] The cutwidth denotes the size of a maximum-cardinality set of pairwise overlapping intervals.

2. Parameterization and the Deconstruction of NP-Hardness

Parameterized algorithmics [8, 12, 17] or, in the context of this work more appropriately, multivariate algorithmics [9, 18], aims at a fine-grained complexity analysis of problems. The hope lies in accepting the seemingly inevitable combinatorial explosion for NP-hard problems, but to confine it to some parameter p. In this paper, p always is a positive integer or a vector of positive integers. A given parameterized problem (I, p) is *fixed-parameter tractable (FPT)* with respect to the parameter p if it can be solved within running time $f(p) \cdot \text{poly}(|I|)$ for some computable function f only depending on p.

A standard question of people unfamiliar with parameterized algorithmics is how to define respectively find "the" parameter for an NP-hard problem. There are the following (partly overlapping) "standard answers" to this question:

1. The standard parameterization classically refers to the size of the solution set of the underlying problem (whenever applicable).
2. A parameter describes a structural property of the input; for instance, the treewidth of a graph or the number of input strings.
3. A parameter may restrict the "dimensionality" of the input; for instance, in the case of problems from computational geometry.
4. Finding useful parameters to some extent is an "art" based on analyzing what typical real-world instances look like.

Perhaps the most natural and constructive answer, however, is to look at the corresponding proof(s) of NP-hardness and what "parameter assumptions" they (do not) make use of. Indeed, this is what we refer to by *deconstructing NP-hardness proofs for parameter identification*. In this work, we deconstruct Althaus et al.'s [1] NP-hardness proof for ICC and gain a rich scenario of combinatorially and practically interesting parameterizations.

Let us now take a closer look at the NP-hardness of ICC. We first have to briefly review the many-one reduction from Exact Cover due to Althaus et al. [1]: The input of Exact Cover is a set $\mathcal{S}$ of subsets of a ground set $U := \{1, 2, \ldots, u\}$ and a positive integer t, and the question is whether there are t subsets from $\mathcal{S}$ such that every element from U is contained in exactly one such subset. Althaus et al.'s polynomial-time many-one reduction (using an approach by Chang et al. [7]) from Exact Cover to ICC works as follows.

1. The number of colors k is set to $s := |\mathcal{S}|$.
2. The interval range n is set to $u \cdot s$.
3. For each element from U, there are three corresponding integer intervals. Indeed, one can speak of three types of intervals, and all intervals of one type can be placed consecutively into one interval $[n]$ without overlap.
   (a) Type 1: Intervals of the form $[(i-1)s+1, is]$ for all $1 \leq i \leq u$.
   (b) Type 2: Intervals of the form $[is-t+1, (i+1)s-t]$ for all $1 \leq i \leq u-1$.
   (c) Type 3: Intervals of the form $[is-t-f_i+1, is-t+1]$ for all $1 \leq i \leq u$, where f_i denotes the number of occurrences of u in the sets of $\mathcal{S}$.

4. Every type-1 and every type-2 interval is assigned the color set $\{1, \ldots, k\}$. A type-3 interval corresponding to $i \in U$ is assigned the color set consisting of the colors associated with the subsets in $\mathcal{S}$ that contain i.

We remark that this proof of NP-hardness actually works with just using sets instead of multisets in the constructed ICC instance. After having described the construction employed in the NP-hardness proof, the deconstruction begins by making several observations about its properties:

1. The interval range n and the number m of intervals both are unbounded.
2. The number of colors k is s, hence unbounded, but all color multisets indeed are sets. That is, no interval shall be assigned the same color twice.
3. The maximum interval length is s, hence unbounded.
4. The maximum overlap between intervals is $\max\{t, s-t\}$, hence unbounded.
5. Only three different surrounding intervals $[n]$ are needed for comprising all intervals without overlap, hence the *cutwidth* of the constructed instance is bounded by three.

From the last observation we can conclude that there is no hope for fixed-parameter tractability with respect to the parameter "cutwidth" unless P=NP. Referring to the second observation, the same holds true for the parameter k (number of colors) because Byrka et al. [5] showed NP-hardness even for the case $k = 3$. On the positive side, we will show that ICC is polynomial-time solvable for cutwidth two. However, from the other three observations we directly obtain the following questions concerning a parameterized complexity analysis of ICC.

1. Is ICC fixed-parameter tractable with respect to the parameters n (interval range) or m (number of intervals)?
2. Is ICC fixed-parameter tractable with respect to the parameter "maximum interval length"?
3. Is ICC fixed-parameter tractable with respect to the parameter "maximum overlap between intervals"?

The central point underlying the above derived algorithmic questions is that whenever a quantity (that is, parameter) in an NP-hardness proof is unbounded (non-constant), then it is natural to investigate what happens if this quantity is constant or considered to be small compared to the overall input size. Clearly, one way to answer is to provide a different proof of NP-hardness where this quantity is bounded. Indeed, this now has happened with respect to the parameter k in the sense that in the NP-hardness proof due to Althaus et al. [1] k is unbounded whereas in the new NP-hardness proof due to Byrka et al. [5] we have $k = 3$. Otherwise, the main tool in answering such questions is parameterized algorithmics. Indeed, the story goes even further by also combining different parameterizations. More specifically, it is, for instance, natural to ask whether ICC is fixed-parameter tractable when parameterized by *both* cutwidth and the number k of colors (the answer is open), or whether it is fixed-parameter

tractable when parameterized by both n and k (the answer is "yes") and what the combinatorial explosion $f(n,k)$ then looks like. In this way, one ends up with an extremely diverse and fruitful ground to develop practically relevant combinatorial algorithms.

In the remainder of this paper, besides the already defined parameters n (range), m (number of intervals), and k (number of colors), we will consider the following parameters and combinations thereof:

- maximum interval length l;
- cutwidth $\mathrm{cw} := \max_{1 \leq i \leq n} |\{F \in \mathcal{F} : i \in F\}|$;
- maximum pairwise overlap between intervals $o := \max_{1 \leq i < j \leq m} |F_i \cap F_j|$;
- maximum number of different colors Δ in the color multisets.

Note that the NP-hardness result of Byrka et al. [5] for $k = 3$ also implies the NP-hardness for $\Delta = 3$ but $\Delta = 2$ is yet unclassified. In addition, one of the integer linear programs devised by Althaus et al. [2] has $O(m \cdot k)$ variables. Using Lenstra's famous result [16] on the running time of integer linear programs with a fixed number of variables then implies that ICC is fixed-parameter tractable with respect to the (combined) parameter (m,k). However, even after several improvements (for example by Frank and Tardos [13]), the combinatorial explosions in Lenstra's theorem remains huge. This fixed-parameter tractability result is thus of purely theoretical interest and more efficient combinatorial algorithms are desirable (see [14, 10, 11] for similar classification results using integer linear programs).

In what follows, we present several fixed-parameter tractability results with respect to the above parameters (Section 4) and some combinations of them (Section 5).

3. A Simple Normal Form Observation

Here, we observe that there is a "normal form" that one may assume without loss of generality for all ICC input instances. More specifically, based on simple and efficient preprocessing rules, one can perform a data reduction that yields this normal form.

Proposition 1. (Normal form for ICC)
In $O(lmn)$ time, one can transform every ICC *instance into an equivalent one such that*

1. *at every position $i \in [n]$, there is at most one interval starting at i and at most one interval ending at i, and*
2. *if the maximum interval length is l, then every position $i \in [n]$ is contained in at most l intervals.*

Proof. To achieve the claimed normal form, exhaustively perform the following two preprocessing rules.

1. If there are two intervals $F_i = [s_i, t_i]$ and $F_j = [s_j, t_j]$ with $s_i = s_j$, $t_i = t_j$, and $C_i \neq C_j$, then return "No". Otherwise, remove F_i and C_i.
2. If there are two intervals $F_i = [s_i, t_i]$ and $F_j = [s_j, t_j]$ with $s_i = s_j$ and $t_i < t_j$, then set $F_j := [t_i + 1, t_j]$ and $C_j := C_j \setminus C_i$.[5] If $|C_j| \neq |F_j|$, then return "No". The case $s_i < s_j$ and $t_i = t_j$ is handled analogously.

Obviously, the two rules directly imply normal form property 1, which again immediately implies normal form property 2. For the correctness of the first rule, observe that no coloring can simultaneously satisfy F_i and F_j. For the correctness of the second rule note that a proper coloring for the original instance is a proper coloring of the instance that results by one application of the rule, and vice versa. Hence, if the new instance obviously is a no-instance (i.e. $|C_j| \neq |F_j|$), then it is correct to reject the instance.

Next, we give an analysis of the running time. We use the following strategy. Keep an array A that holds for every position $i \in [n]$ a list with the intervals starting at i (and another list with the intervals ending at i). This array is initialized before the application of the rules in $O(nm)$ time. To decide whether a rule can be applied, iterate over the array to find a position at which two intervals start (or end). For two intervals F_i and F_j, the necessary changes can be performed in $O(n)$ time (if one implements the color multisets by an array of size $k \leq n$). Also note that we can update array A within this time bound. Hence, one application of a rule and the update of array A take $O(n)$ time. Finally, note that for every interval the first rule can be applied at most once, and the second rule at most l times. Hence, the rules can be exhaustively applied in $O(nm) + O(lmn) = O(lmn)$ time. □

Clearly, Proposition 1, property 1, implies that after preprocessing the "reduced equivalent instance" contains at most n intervals and n multicolor sets, which can be interpreted as "kernelization" with respect to the parameter n in terms of parameterized algorithmics (also see [4, 15] for surveys on kernelization).

4. Single Parameters

In Section 2, we identified various parameters as meaningful "combinatorial handles" to better assess the computational complexity of ICC. Whereas ICC is NP-complete for cutwidth cw = 3 [1], we will show that it is polynomial-time solvable for cw = 2. Obviously, the maximum length l fulfills $l \leq n$, so the fixed-parameter tractability with respect to l (as we will prove subsequently) implies the fixed-parameter tractability with respect to n. Table 1 surveys known and new results with respect to single parameters.

[5] The setminus operation here has to be adapted to multisets, that is, for example, $\{a, a, b\} \setminus \{a, b\} = \{a\}$.

Table 1: Complexity of ICC for one-dimensional parameterizations. Herein, "P" means that the problem is polynomial-time solvable, "NPc" means that the problem is NP-complete, and "?" means that the complexity is unknown. For fixed-parameter algorithms, we only give the function of the exponential term, omitting polynomial factors. The results for $k = 2$ and $\mathrm{cw} = 3$ are due to Althaus et al. [1, 2], the results for $k \geq 3$ and $\Delta \geq 3$ are due to Byrka et al. [5], the rest is new.

Parameter	k	Δ	l	cw	m	n	o
Complexity	$k = 2$: P $k \geq 3$: NPc	$\Delta = 2$: ? $\Delta \geq 3$: NPc	$l!$	$\mathrm{cw} = 2$: P $\mathrm{cw} = 3$: NPc	?	$n!$	$o = 1$: P $o \geq 2$: ?

Parameter Maximum Interval Length l. Our first algorithm exploits the parameter "maximum interval length l". The rough idea is that the coloring at position i does not affect any intervals that overlap with position $i+l$. This leads to a dynamic programming algorithm that keeps track of all possible colorings of the "last" interval (which has at most l positions).

Theorem 1. ICC *can be solved in $O(l! \cdot l \log l \cdot mn)$ time.*

Proof. We present a dynamic programming algorithm. To this end, we use the following notation. Let $\mathcal{K} = \{1, \ldots, k\}$ denote the set of all colors. For an interval $[s, t]$, a coloring c is represented by a tuple $(c_1, \ldots, c_{t-s+1}) \in \mathcal{K}^{t-s+1}$, meaning that $c(s) = c_1$, $c(s+1) = c_2$, and so on. We say that a coloring c' *satisfies* an input interval $F_i \in \mathcal{F}$ if $c'(F_i) = C_i$. For an input interval $F_i \in \mathcal{F}$, the set K_i of all satisfying colorings is given by

$$K_i := \{c' \in \mathcal{K}^{|F_i|} \mid c' \text{ satisfies } F_i\}.$$

Note that there are at most $|C_i|!$ satisfying colorings of an input interval F_i (the worst case arises when every color occurs at most once in the multiset C_i since then every permutation of the colors in C_i represents a satisfying coloring). Finally, let A denote the set of intervals completely contained in some other intervals, that is,

$$A := \{F \in \mathcal{F} \mid \exists_{F' \in \mathcal{F}} : F \subseteq F'\},$$

and $B := \mathcal{F} \setminus A$. We assume that the intervals in B are ordered in increasing order of their start points (and, hence, also in increasing order of their endpoints). Let $B = \{B_1, \ldots, B_{m'}\}$ and $B_j = [s_j, t_j]$ for all $1 \leq j \leq m'$. Note that $m' \leq n$ and that the intervals in B cover $[n]$, that is, $\bigcup_{j=1}^{m'} B_j = [n]$ (as discussed in Section 1 we assume that the input intervals cover $[n]$).

Now, we are ready to describe the algorithm. The algorithm traverses the B_j's in increasing order of j, $1 \leq j \leq m'$. For every B_j, the algorithm maintains a table T_j with an entry for every satisfying coloring c of B_j. Informally speaking, this entry indicates whether there exists a coloring of the interval $[1, t_j]$ that agrees with c in $[s_j, t_j]$ and satisfies all intervals seen so far. More specifically, the goal of the dynamic programming procedure is to fill these tables to match the following definition. For every coloring $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$, we

have $T_j(c') = \text{true}$ if and only if there exists a coloring $c'' = (c''_1, \ldots, c''_{t_j}) \in \mathcal{K}^{t_j}$ of the interval $[t_j]$ with $(c''_{s_j}, \ldots, c''_{t_j}) = c'$ such that c'' satisfies each interval $F \in \mathcal{F}$ with $F \subseteq [t_j]$. Obviously, if the algorithm correctly computes the tables according to this definition, then $T_{m'}$ contains a true entry if and only if the input is a yes-instance.

Table T_1 is computed as follows. For every $c' \in K_1$, set $T_1(c') := \text{true}$ if and only if c' satisfies every interval $[s,t] \in A$ with $[s,t] \subseteq [s_1, t_1]$.

For $j \geq 2$, table T_j is computed based on T_{j-1}, as described next. We say that a coloring $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$ for B_j is consistent with a coloring $c'' = (c''_1, \ldots, c''_{|B_{j-1}|}) \in K_{j-1}$ for B_{j-1} if c' and c'' agree in $B_{j-1} \cap B_j$, that is, $(c''_{s_j - s_{j-1}+1}, \ldots, c''_{|B_{j-1}|}) = (c'_1, \ldots, c_{t_{j-1}-s_j+1})$. We write $c'|c''$ to denote that c' is consistent with c''. To compute the entries of T_j, proceed as follows. For j from 2 to m' and for every $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$, set

$$T_j(c') = \text{true} \iff \begin{array}{l} c' \text{ satisfies all } F \in A \text{ with } F \subseteq B_j \text{ and} \\ \exists c'' \in K_{j-1},\ c'|c'' : T_{j-1}(c'') = \text{true}. \end{array} \tag{1}$$

Finally, the algorithm returns "Yes" if T'_m contains a true entry and "No", otherwise.

For the correctness of the algorithm, we show by induction that for every j, $1 \leq j \leq m'$, table T_j meets above definition, that is, table entry $T_j(c')$ is true if and only there exists a coloring of $[t_j]$ with "suffix" c' satisfying all intervals $F \in \mathcal{F}$ with $F \subseteq [t_j]$. Clearly, T_1 is computed in accordance with this definition, yielding the induction base.

For the induction step, we show that Recursion (1) computes T_j according to the above definition assuming that T_{j-1} has been correctly computed (induction hypothesis). That is, we show that there is a coloring of $[t_j]$ with suffix c' satisfying all intervals $F \in \mathcal{F}$ with $F \subseteq [t_j]$ if and only if c' satisfies all $F \in A$ with $F \subseteq B_j$ and there is a $c'' \in K_{j-1}$ with $c'|c''$ such that $T_{j-1}(c'') = \text{true}$. The "$\Rightarrow$"-direction is obvious. For the "$\Leftarrow$"-direction, observe the following. The algorithm combines a coloring of $[t_{j-1}]$ satisfying all $F \in \mathcal{F}$ with $F \subseteq [t_{j-1}]$ with a coloring c' of $[s_j, t_j]$ that is consistent with c'' and satisfies all $F \in \mathcal{F}$ with $F \subseteq [s_j, t_j]$. This yields a coloring for $[t_j]$ that satisfying all $F \in \mathcal{F}$ with $F \subseteq [t_j]$; all $F \in \mathcal{F}$ with $F \subseteq [t_{j-1}]$ are clearly also satisfied by this coloring. Moreover, all other $F \in \mathcal{F}$ with $F \subseteq [t_j]$ are satisfied since for every input interval $[s,t] \in \mathcal{F}$ with $t_{j-1} < t \leq t_j$ it holds that $[s,t] \subseteq [s_j, t_j]$.

As to the running time, there are at most $|B_j|!$ satisfying colorings of B_j; at most one for every permutation of the associated color multiset. Hence, one has to consider at most $l!$ colorings for every B_j. For every $j = 1, \ldots, m' - 1$, the algorithm proceeds as follows.

When building table T_j, the algorithm computes an auxiliary table Q_j containing one entry for all $c' \in K_j$ with the same length-$(t_j - s_{j+1} + 1)$ suffix, indicating whether T_j contains a true entry for one of these colorings. Table Q_j can for example be realized by a dictionary for which the addition and the lookup of a key requires $O(\log(s))$ comparisons, where s is the size of the dictionary. Then,

to check whether $\exists c'' \in K_{j-1}$, $c'|c'' : T_{j-1}(c'') = \text{true}$ for a $c' = (c'_1, \ldots, c'_{|B_j|}) \in K_j$, the algorithm can check whether $Q_{j-1}(c'_1, \ldots, c'_{t_j - s_{j-1}+1}) = \text{true}$ in $O(l \log l)$ time (note that the size of Q_j does not exceed $l!$). Hence, for every position $1 \leq j \leq m'$, it needs at most $O(l! \cdot (l \log l + lm))$ time, where the factor lm is due to checking whether c' satisfies all $F \in \mathcal{F}$ with $F \subseteq [s_j, t_j]$. In summary, since $m' \leq n$ the total running time is $O(l! \cdot l \log lmn)$. □

Parameter Cutwidth cw. Here, we show that ICC is solvable in $O(n^2)$ time for cutwidth $\text{cw} = 2$. This contrasts the case $\text{cw} = 3$ shown to be NP-complete [1]. Our algorithm is based on four data reduction rules that are executable in polynomial time. The application of these rules either leads to a much simplified instance that can be colored without violating any interval constraints or shows that the instance is a no-instance.

In the following, we say that a reduction rule is *correct* if the instance after applying this rule has a proper coloring if and only if the original instance has a proper coloring. Some of the subsequent data reduction rules are based on identifying positions for which we can decide which color they will have in a proper coloring. In this context, we use the following notation. If we can decide that a position i is colored by color c_x in a proper coloring, we write $c(i) = c_x$, meaning that we simplify the instance as follows; for all $F_j = [s, t]$ with $s \leq i \leq t$, we set $C_j := C_j \backslash \{c_x\}$ and $t := t-1$. For all $F_j = [s, t]$ with $i < s$, we set $s := s-1$ and $t := t - 1$. "Empty" intervals F_j with $C_j = \emptyset$ are removed from the input. Finally, we call an instance reduced with respect to one or more data reduction rules if none of these rules applies.

We start with a basically straightforward data reduction rule.

Reduction Rule 1. *For any two intervals F_i and F_j and their corresponding color multisets C_i and C_j,*

- *if $|F_i \cap F_j| = |C_i \cap C_j|$, then set $c(F_i \cap F_j) = C_i \cap C_j$;*
- *if $|F_i \cap F_j| > |C_i \cap C_j|$, then return "No".*

Rule 1 is obviously correct: if two intervals "share" more positions than color elements, then there is no coloring that satisfies both intervals, and if the number of shared positions is equal to the number of shared color elements, then one has to color the overlapping intervals exactly with the corresponding colors. Moreover, since the cutwidth is two there is no further interval containing a position in $F_i \cap F_j$. Hence, we can color the positions of $F_i \cap F_j$ in arbitrary order with the colors in $C_i \cap C_j$.

After exhaustive application of Rule 1 we can assume that no interval is completely contained in any other interval.

In the following, assume that the intervals are ordered with respect to their startpoints, that is, for $F_i = [s_i, t_i]$ and $F_j = [s_j, t_j]$ with $i < j$ we have $s_i < s_j$. Consider a position i, $1 \leq i < n$. If there is no input interval $[s_j, t_j]$ with $s_j \leq i$ and $t_j > i$, then we can color $[i]$ independently from $[i + 1, n]$. Together with Rule 1 and the fact that $\text{cw} = 2$, we can thus assume that all intervals

except for F_1 and F_m overlap with exactly two other intervals. Hence, we can partition each interval F_j, $1 < j < m$, into at most three subintervals: the first subinterval overlaps with F_{j-1}, the second (possibly empty) subinterval does not overlap with any other interval, and the third subinterval overlaps with F_{j+1}. The following notation describes this structural property. For an interval F_j, $1 < j < m$, define

$$F_j^1 := F_j \cap F_{j-1},\ F_j^3 := F_j \cap F_{j+1}, \text{ and } F_j^2 := F_j \setminus (F_j^1 \cup F_j^3).$$

For a coloring c' of $[n]$ and j, $1 < j < m$, let $C_j^1 := c'(F_j^1)$. Define C_j^2 and C_j^3 accordingly. For F_1, define $F_1^3 := F_2^1$ and $F_1^2 := F_1 \setminus F_1^3$; for F_m, define $F_m^1 := F_m \cap F_{m-1}$ and $F_m^2 := F_m \setminus F_m^1$; C_1^3, C_1^2, C_m^1, and C_m^2 are defined analogously. Whether a coloring violates an interval F_j only depends on the sets C_j^1, C_j^2, and C_j^3. Hence, when we know that a color c_x must belong to some C_j^l, $1 \leq l \leq 3$, then we can color an arbitrary $i \in F_j^l$ with c_x. Finally, for a color multiset C and a color c_x let $\mathrm{occ}(c_x, C)$ denote the multiplicity of c_x in C.

The next rule reduces intervals F_j that have no "private" middle interval F_j^2 but more elements of some color c_x than the previous interval.

Reduction Rule 2. *For any interval F_j, if $F_j^2 = \emptyset$ and there is a color c_x such that $\mathrm{occ}(c_x, C_{j-1}) < \mathrm{occ}(c_x, C_j)$, then for some arbitrary $i \in F_j^3$ set $c(i) = c_x$.*

The rule is correct because in a proper coloring at most $\mathrm{occ}(c_x, C_{j-1})$ many positions in F_j^1 (which is the intersection of F_j and F_{j-1}) can be colored with c_x. Hence, in order to satisfy constraint C_j, all other occurrences of c_x must be at positions in F_j^3. Again, since the cutwidth is two, we can choose an arbitrary position of F_j^3. After the exhaustive application of Rule 2, for every interval F_j with $F_j^2 = \emptyset$, we have $C_{j-1} \supseteq C_j$. Next, we reduce triples of intervals F_{j-1}, F_j, F_{j+1} that have identical color multisets in case $F_j^2 = \emptyset$.

Reduction Rule 3. *For intervals F_{j-1}, F_j, and F_{j+1} such that $C_{j-1} = C_j = C_{j+1}$ and $F_j^2 = \emptyset$, remove F_{j-1} and F_j from the input and for all intervals $F_{j+l} =: [s, t]$ with $l \geq 1$ set $F_{j+l} = [s', t']$, where $s' := s - |F_j|$ and $t' := t - |F_j|$.*

Lemma 1. *Rule 3 is correct.*

Proof. Let I be an instance to which Rule 3 is applied, and let I' be the resulting instance. We show that I is a yes-instance if and only if I' is a yes-instance. Let I be a yes-instance, let c' be a proper coloring of I, and for each input interval $F_l \in \mathcal{F}$ let C_l^1, C_l^2, C_l^3 be the color multisets according to c'. Since $C_{j-1} = C_j = C_{j+1}$, we have $C_{j-1}^1 \subseteq C_{j+1}^1$ and $C_{j+1}^3 \subseteq C_{j-1}^3$. In I', F_{j+1} overlaps with F_{j-2} and F_{j+2}. Coloring F_{j+1}^1 with the colors of C_{j-1}^1 and F_{j+1}^2 with the colors of $C_j \setminus (C_{j-1}^1 \cup C_{j+1}^3)$ yields a proper coloring for I' since C_{j+1} is not violated and for the other intervals the coloring has not changed. Hence, if I is a yes-instance, then I' is a yes-instance. The other direction can be shown analogously. □

The following is our final data reduction rule.

Reduction Rule 4. *Let I be an instance that is reduced with respect to Rules 1, 2, and 3, and let F_j be the first interval of I such that there is a color c_x with* $\mathrm{occ}(c_x, C_j) > \mathrm{occ}(c_x, C_{j+1})$. *Then,*

- *if $j = 1$, then set $c(i) = c_x$ for some arbitrary $i \in F_1^2$;*
- *if $j > 1$ and $c_x \notin C_{j-1}$, then set $c(i) = c_x$ for some arbitrary $i \in F_j^2$ in case $F_j^2 \neq \emptyset$ and otherwise return "No";*
- *if $j > 1$ and $c_x \in C_{j-1}$, then set $c(i) = c_x$ for some arbitrary $i \in F_j^1$.*

Lemma 2. *Rule 4 is correct.*

Proof. Let I be an instance that is reduced with respect to Rules 1, 2, and 3 to which Rule 4 is applied, and let I' be the resulting instance. We show that I is a yes-instance if and only if I' is a yes-instance. We only show that if I is a yes-instance, then I' is a yes-instance, since the other direction trivially holds.

If $j = 1$, this is easy to see: since c_x occurs more often in F_1 than in F_2, one of the positions in $F_1 \setminus F_2$ must be colored with c_x.

If $j > 1$ and $c_x \notin C_{j-1}$, then it is clear that one of the positions in F_j^2 must be colored with c_x. We either perform this forced coloring or return "No" if this is not possible.

Finally, if $j > 1$ and $c_x \in C_{j-1}$, the situation is more complicated. Let c' be a proper coloring of I. If there is a position $i \in F_j^1$ such that $c'(i) = c_x$, then the claim obviously holds. Otherwise, we show that we can transform c' into an alternative coloring c'' that is proper and there is an $i \in F_j^1$ such that $c''(i) = c_x$. Whether coloring c'' is proper can be determined from the multisets C_l^1, C_l^2, and C_l^3, $1 \leq l \leq m$, defined by the coloring function c''. Hence, we describe the transformation applied to c' with respect to these multisets. Note that we do not modify the sets C_l^y, $y \in \{1, 2, 3\}$, for any $l > j$.

We face the following situation: $c_x \notin C_j^1$, but since c' is a coloring that does not violate any interval constraints and by the precondition of Rule 4, $c_x \in C_j^2$. By the choice of j in Rule 4, we have $C_1 \subseteq C_2 \subseteq \ldots \subseteq C_j$. We show that we can always find a series of "exchange operations" such that the resulting coloring is proper and $c_x \in C_j^1$. We perform a case distinction.

Case 1: $F_{j-1}^2 \neq \emptyset$. There are three subcases of this case.

Case 1.1: $c_x \in C_{j-1}^2$. In this case, we exchange c_x with some arbitrary $c_l \in C_j^1$. Furthermore, we remove c_x from C_j^2 and add c_l to C_j^2. The exchange is shown in Fig. 1a; the resulting coloring is clearly proper and $c_x \in C_j^1$.

Case 1.2: $c_x \in C_{j-1}^1$ and $F_{j-2}^2 \neq \emptyset$. Clearly, C_{j-2} must be involved in the exchange. We choose an arbitrary element $c_l \in C_{j-2}^2$. Since $C_{j-2} \subseteq C_{j-1}$, we also have $c_l \in C_{j-1} \setminus C_{j-1}^1$. We distinguish two further subcases.

Case 1.2.1: $c_l \in C_{j-1}^3$. We perform a *direct* exchange of c_l and c_x between C_{j-2}^2 and C_{j-1}^1 and also between C_j^1 and C_j^2. The exchange is shown in Fig. 1b; the

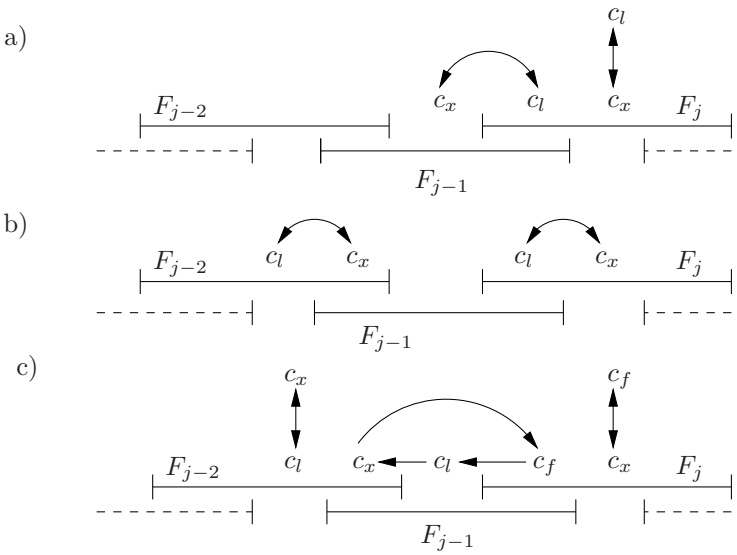

Figure 1: Exchange operations used in the proof of Lemma 2. Intervals are shown as lines. The segments of an interval F_j where no other interval starts or ends correspond to the color multisets C_j^1, C_j^3, and (if present) C_j^2. A vertical double arrow means removing the element shown at the bottom from the corresponding multiset and adding the element shown at the top of the arrow to the multiset; a double arrow between two multisets means exchanging the elements between the corresponding multisets; a simple arrow means moving an element from one multiset to another in the indicated direction.

resulting coloring is clearly proper and $c_x \in C_j^1$.

Case 1.2.2: $c_l \in C_{j-1}^2$**.** We remove c_l from C_{j-2}^2 and add c_x to C_{j-2}^2. Furthermore, we perform a *circular* exchange between C_{j-1}^1, C_{j-1}^2, and C_{j-1}^3: move c_x from C_{j-1}^1 to C_{j-1}^3, move an arbitrary element c_f from C_{j-1}^3 to C_{j-1}^2, and move c_l from C_{j-1}^2 to C_{j-1}^1. Finally, we remove c_x from C_j^2 and add c_f to C_j^2. The exchange is shown in Fig. 1c; the resulting coloring is clearly proper and $c_x \in C_j^1$.

Case 1.3: $c_x \in C_{j-1}^1$ **and** $F_{j-2}^2 = \emptyset$**.** As stated above, we have $C_{j-3} \subseteq C_{j-2}$. Furthermore, since $F_{j-2}^2 = \emptyset$ and Rule 2 does not apply, we have $C_{j-3} = C_{j-2}$. Hence, $F_{j-3}^2 \neq \emptyset$ since otherwise F_{j-3} would have been removed by Rule 3.

Case 1.3.1: $c_x \in C_{j-3}^2$**.** We pick an arbitrary element $c_l \in C_{j-3}^3$ and exchange it with $c_x \in C_{j-3}^2$. If $c_l \in C_{j-1}^3$, then we perform a direct exchange of c_x and c_l between C_{j-1}^1 and C_{j-1}^3. If $c_l \in C_{j-1}^2$, we perform a circular exchange between C_{j-1}^1, C_{j-1}^2, and C_{j-1}^3 using some arbitrary $c_f \in C_{j-1}^3$. Furthermore, we remove c_x from C_j^2 and insert c_l into C_j^2. Fig. 2a shows the more complicated case where $c_l \in C_{j-1}^2$.

Case 1.3.2: $c_x \in C_{j-3}^1$**.** Clearly, any change must also involve F_{j-4}. Furthermore, we can have a long "chain" of alternating intervals F_{j-2i} and F_{j-2i-1}, $1 \leq i < j/2$, such that $F_{j-2i}^2 = \emptyset$ and $F_{j-2i-1}^2 \neq \emptyset$. Since the instance is reduced

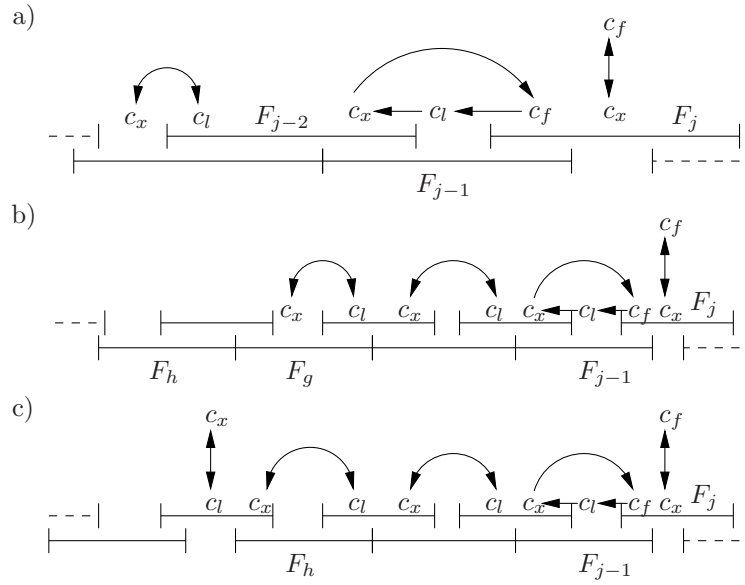

Figure 2: Exchange operations used in Case 1.3 of the proof of Lemma 2. Intervals are shown as lines. Two intervals $[s,t]$ and $[s',t']$ are drawn adjacent if $t = s' - 1$ or vice versa. The segments of an interval F_j where no other interval starts or ends correspond to the color multisets C_j^1, C_j^3, and (if present) C_j^2. A vertical double arrow means removing the element shown at the bottom from the corresponding multiset and adding the element shown at the top of the arrow to the multiset; a double arrow between two multisets means exchanging the elements between the corresponding multisets; a simple arrow means moving an element from one multiset to another in the indicated direction.

with respect to Rules 2 and 3 we have $C_{j-2i-1} = C_{j-2i} \subseteq C_{j-2i+1}$. Let F_h be the first (with lowest index) interval of the chain, that is, $F_h^2 \neq \emptyset$, $F_{h+1}^2 = \emptyset$, and either $F_h = F_1$ or $F_{h-1}^2 \neq \emptyset$. There is either some rightmost (with highest index) interval F_g such that $c_x \in C_g^2$, or for all intervals F_i of the chain we have $c_x \notin C_i^2$. We show that in the first case, for all intervals F_{g+2i}, $g+2i \leq j-3$, we have $c_x \in C_{g+2i}^1$, and in the second case for all F_{h+2i}, $h+2i \leq j-3$, we have $c_x \in C_{h+2i}^1$. This can be seen as follows. We have $c_x \in C_{j-3}^1$ and thus $c_x \in C_{j-5} \setminus C_{j-4}^1$, since $C_{j-4} = C_{j-5}$. We either have $c_x \in C_{j-5}^2$ (implying $F_g = F_{j-5}$) or $c_x \in C_{j-5}^1$ (in which case we can apply the same arguments showing that either $F_g = F_{j-7}$ or $c_x \in C_{j-7}^1$ and so on). We now sketch the exchange operations that we perform.

First, consider the case that there is some F_g with $c_x \in C_g^2$. We perform an exchange similar to the one shown in Fig. 2a. That is, we exchange $c_x \in C_g^2$ and some $c_l \in C_g^3$. Then we remove c_x from C_{g+2}^1, and add it to C_{g+2}^3. We also need to add c_l to C_{g+2}^1, which is possible since $c_l \in C_{g+2} \setminus C_{g+2}^1$. Depending on whether $c_l \in C_{g+2}^2$ or $c_l \in C_{g+2}^3$, we perform a circular or a direct exchange. These exchange operations are carried on (for F_{g+2i} for increasing $i \geq 1$) until we have reached F_j, that is, we move c_x from C_{g+2i}^1 to C_{g+2i}^3 and some c_y

(depending on the previous exchange) from $C_{g+2i} \setminus C^1_{g+2i}$ to C^1_{g+2i}. An example of this exchange is shown in Fig. 2b. Note that this also includes the case where $F_h = F_1$.

Second, we consider the case where for all intervals F_i of the chain we have $c_x \notin C^2_i$. We start the exchange operation at the first (with lowest index) interval F_h of the chain, that is, the first interval F_h of the chain such that $F^2_h \neq \emptyset$ and $F^2_{h-1} \neq \emptyset$. Note that, since we have already considered the case $F_1 = F_h$, such an interval must exist. Then we perform the exchange operations as sketched in Fig. 2c. That is, we remove an arbitrary element c_l from C^2_{h-1} and add c_x to C^2_{h-1}. Then we perform either a circular or a direct exchange of c_l and c_x in F_h, which is possible since $c_l \in C_h \setminus C^1_h$. We continue with these circular or direct exchanges for F_{h+2i} for increasing $i \geq 1$ until we have reached F_{j-1}. Finally, we remove c_x from C_{j-2} and add c_f to C_{j-2}.

Case 2: $F^2_{j-1} = \emptyset$**.** As shown in Case 1.3, we can assume that $F^2_{j-2} \neq \emptyset$ since otherwise Rule 3 would apply. Hence, there is some $c_l \in C^2_{j-2}$. Since $C_{j-2} = C_{j-1}$ and $C^2_{j-1} = \emptyset$, we also have $c_l \in C^3_{j-1}$. Hence, this case is similar to Case 1.2.1 and we can perform an exchange as shown in Fig. 1b.

In all cases, we construct an alternative coloring c'' that is proper and there is an $i \in F^1_j$, such that $c''(i) = c_x$. This means that we can assume that if I is a yes-instance, then there is some $i \in F^1_j$ such that $c'(i) = c_x$. In summary, this shows that I is a yes-instance if and only if I' is a yes-instance. □

With these four reduction rules at hand, we can describe a simple quadratic-time algorithm (for constant number k of colors) for ICC with cutwidth two.

Theorem 2. ICC *can be solved in* $O(kn^2)$ *time when the input has cutwidth two.*

Proof. The algorithm starts with exhaustively applying Rules 1 to 4. Note that before applying Rule 4 we always have to check whether Rule 1, Rule 2 , or Rule 3 can be applied, because it is only correct to apply Rule 4 when the instance is reduced with respect to the other rules. The rules either return "No" or we obtain an instance that is reduced with respect to all reduction rules. In such an instance we have $C_1 \subseteq C_2 \subseteq \ldots \subseteq C_m$. Otherwise, Rule 4 would apply, because there would be some $F_i \in \mathcal{F}$ and a color c_x such that $\mathrm{occ}(c_x, C_i) > \mathrm{occ}(c_x, C_{i+1})$. This instance can be easily colored as follows. For the first interval F_1, we choose an arbitrary coloring that does not violate C_1. Since $C_1 \subseteq C_2$, this coloring also does not violate C_2. Then we remove the colored parts from the input, adjust the color multisets and intervals accordingly, and choose an arbitrary coloring that does not violate C_2. Clearly, this does not violate C_3, since $C_2 \subseteq C_3$. After this, we again reduce the colored parts and continue with coloring F_3. This is repeated until all positions are colored and produces a coloring that does not violate any interval constraints. This proves the correctness of the algorithm.

For the running time of the algorithm consider the following. First, since the input has cutwidth two, the number m of intervals is $O(n)$. For each reduction rule, checking whether it can be applied and the application itself can be

Table 2: Complexity of ICC for combined parameters. We only give the function of the exponential term, omitting polynomial factors. Herein, $(k, *)$ and $(k, *, *)$ refer to combined parameters that feature k and one or two additional parameters, $(l, *)$ refers to combined parameters that feature l and one additional parameter. The result for parameter (k, m) is due to Althaus et al. [2], the rest is new.

Parameter	Running times
$(k, *)$	k^l, $(k-1)^n$, $f(k, m)$ (ILP)
$(k, *, *)$	$l^{\mathrm{cw}\cdot(k-1)}$, $n^{\mathrm{cw}\cdot(k-1)}$
$(l, *)$	Δ^l, $(\mathrm{cw}+1)^l$

performed in $O(kn)$ time. Furthermore, the application of any of the reduction rules removes at least one position from the interval $[n]$. Overall, the rules can thus be applied at most n times. Together with the $O(n)$ steps that are clearly sufficient for coloring any instance that is reduced with respect to the reduction rules, this leads to a total running time of $O(kn^2)$. □

Using the previous algorithm, we also obtain polynomial-time solvability in case the maximum overlap o between intervals is at most one. This follows from the observation that after achieving the normal form of the instance (see Proposition 1), each instance with overlap at most one also has cutwidth at most two, which can be seen as follows. Suppose an instance that has the normal form has overlap one and cutwidth at least three. Then there must be a position i such that at least three intervals F, F', and F'' overlap at i. By Proposition 1, at most one of these three intervals, say F, starts at i. This, however, means that F' and F'' have overlap at least two.

Corollary 1. ICC *can be solved in* $O(n^2)$ *time when the input has overlap one.*

5. Combined Parameters

In the following, as already indicated in Section 2, we turn to the study of some relevant pairs of single parameters which form a "combined parameter". Table 2 summarizes our current knowledge about combined parameterizations of ICC—there are many questions left open.

First, we present a dynamic programming strategy for solving ICC in $O(k^l \cdot (k+l)mn)$ time. This algorithm uses similar ideas as the algorithm presented in the proof Theorem 1. Note that, for a small number k of colors this algorithm is more efficient than the algorithm presented in the proof of Theorem 1.

Theorem 3. ICC *can be solved in* $O(k^l \cdot (k+l)mn)$ *time.*

Proof. We present a dynamic programming algorithm. The basic idea of the algorithm is to maintain for every length-l subinterval of $[n]$ a table with an entry for every possible coloring of that interval indicating whether this coloring can

be extended to a proper coloring of $[n]$. These tables are built by a "left to right" dynamic programming procedure. The details follow.

For every $1 \leq i \leq n-l+1$, the algorithm maintains a table T_i with an entry for every possible coloring of the interval $I_i := [i, i+l-1]$. Note that there are k^l possibilities to color a size-l interval with k colors. In the following, let $\mathcal{K} := \{1, 2, \ldots, k\}$ be the set of all colors. A coloring of a length-l interval $[i, i+l-1]$ is represented by a vector $c' = (c'_1, \ldots, c'_l) \in \mathcal{K}^l$, meaning that $c'(i) = c'_1$, $c'(i+1) = c'_2$, and so on. Recall that a coloring c' *satisfies* (the constraint of) an input interval $F_j \in \mathcal{F}$ if $C_j = c'(F_j)$. Finally, for $1 \leq i \leq n$ we say that a coloring of $[i]$ is *proper*, if it satisfies all input intervals contained in $[i]$.

The goal of the dynamic programming procedure is to fill the tables T_i in accordance with the following definition. For every $1 \leq i \leq n-l+1$ and for every coloring $c' \in \mathcal{K}^l$, we have $T_i(c') = \text{true}$ if and only if there exists a coloring $c'' = (c''_1, \ldots, c''_{i+l-1}) \in \mathcal{K}^{i+l-1}$ of the interval $[i+l-1]$ with $(c''_i, \ldots, c''_{i+l-1}) = c'$ satisfying each interval $F \in \mathcal{F}$ with $F \subseteq [i+l-1]$. That is, $T_i(c') = \text{true}$ if and only if there exists a proper coloring c'' of the interval $[i+l-1]$ that is an extension of c'.

For $i = 1$ and for every $c' \in \mathcal{K}^l$, this is achieved by setting $T_1(c') := \text{true}$ if and only if c' satisfies every interval $F \in \mathcal{F}$ with $F \subseteq [l]$.

For $i > 1$, table T_i is computed based on T_{i-1} as follows. For $i = 2$ to $n-l+1$ and for every $c' = (c'_1, \ldots, c'_l) \in \mathcal{K}^l$, set

$$T_i(c') = \text{true} \iff \begin{array}{l} c' \text{ satisfies every } [s,t] \in \mathcal{F} \text{ with } [s,t] \subseteq [i, i+l-1] \text{ and} \\ \exists z \in \mathcal{K} : T_{i-1}((z, c'_1, c'_2, \ldots, c'_{l-1})) = \text{true}. \end{array} \quad (2)$$

Finally, the algorithm outputs "Yes" if there exists a coloring $c' \in \mathcal{K}^l$ with $T_{n-l+1}(c') = \text{true}$, and "No", otherwise. This completes the description of the algorithm.

The correctness of the algorithm follows by induction on i. More specifically, we show that T_i meets the above definition for every $1 \leq i \leq n-l+1$. That is, we show that $T_i(c') = \text{true}$ if and only if there is a proper coloring of the interval $[i+l-1]$ with "suffix" c'. Obviously, this holds for $i = 1$.

For the induction step we show the correctness of Recursion (2). That is, we show that there is a proper coloring of $[i+l-1]$ with suffix c' (that is, $T_i(c') = \mathit{true}$) if and only if c' satisfies all input intervals contained in $[i, i+l-1]$ and there is a proper coloring of $[i+l-2]$ with suffix $(z, c'_1, c'_2, \ldots, c'_{l-1})$ for some $z \in \mathcal{K}$ (that is, $T_{i-1}((z, c'_1, c'_2, \ldots, c'_{l-1})) = \text{true}$).

The "$\Rightarrow$"-direction is straightforward. For the "$\Leftarrow$"-direction, note the following. The existence of a $z \in \mathcal{K}$ with $T_{i-1}((z, c'_1, c'_2, \ldots, c'_{l-1})) = \text{true}$ means that there is a coloring $c'' = (c''_1, \ldots, c''_{i-2}, z, c'_1, \ldots, c'_{l-1})$ of $[i+l-2]$ satisfying all $F \in \mathcal{F}$, $F \subseteq [i+l-2]$. Thus, the coloring $c^* = (c''_1, \ldots, c''_{i-2}, z, c'_1, \ldots, c'_{l-1}, c'_l)$ clearly satisfies all input intervals $[s,t] \in \mathcal{F}$ with $t \leq i+l-2$. Moreover, c^* satisfies all input intervals $[s,t] \in \mathcal{F}$ with $t = i+l-1$ since every input interval that ends at position $i+l-1$ must be completely contained in $[i, i+l-1]$.

As to the running time, for every $i \in [n]$ and for every $c' \in \mathcal{K}^l$ the computation of $T_i(c')$ according to Recursion (2) can be performed in $O(m(k+l))$

time. This can be seen as follows. For every input interval completely contained in $[i, i+l-1]$, check whether it is satisfied by c', which is doable in $O(k+l)$ time (if we realize the multisets by size-k arrays). In addition, to check whether $\exists z \in \mathcal{K} : T_{i-1}((z, c_1, c_2, \ldots, c_{l-1})) = \text{true}$, try all k colors for z; hence, the running time for the above recursion is $O(m(k+l))$. This leads to a total running time of $O(k^l \cdot (k+l)mn)$ since for every position $1 \leq i \leq n-l+1$ we have to try all k^l possible colorings of a length-l interval. □

Next, we present an alternative solution strategy also based on dynamic programming. The running time of this algorithm can be bounded by $(\text{cw}+1)^l \cdot \text{poly}(n,m)$ or $l^{\text{cw}\cdot(k-1)} \cdot \text{poly}(n,m)$. To explain the basic idea of the algorithm, consider the following. Assume that we are given a coloring c' satisfying all intervals. Consider a position i. With respect to i coloring c' partitions a color multiset C of an interval F intersecting with i into two multisets: one containing the colors that c' uses for the positions $j \in F$ with $j \leq i$ and the other containing all other colors of the color multiset. The basic idea of the dynamic programming algorithm is to traverse the instance from "left to right" and to try for every position i all partitions of the color multisets of the intervals intersecting with i. Roughly speaking, for every such partition, the algorithm remembers whether this partition is consistent with a coloring satisfying all input interval seen so far. In the proof of the next theorem, we will show that, for a position i and a partition of the color multisets intersecting with i, this decision can be made based on the stored information for position $i-1$.

Theorem 4. ICC *can be solved in* $O((cw+1)^l \cdot l \cdot (k \cdot cw)^2 \cdot \log cw \cdot n)$ *time and* $O(l^{cw\cdot(k-1)} \cdot (k \cdot cw)^3 \log l \cdot n)$ *time, respectively.*

Proof. We present a dynamic programming algorithm that yields both claimed running times. We use the following notation. For every position i, $1 \leq i \leq n$, let $\mathcal{F}_i = \{F_{i_1}, \ldots, F_{i_{n_i}}\}$ denote the input intervals containing i. Furthermore, let $\mathcal{C}_i = \{C_{i_1}, \ldots, C_{i_{n_i}}\}$ denote the color multisets associated with the intervals in $\mathcal{F}_i$, where C_{i_j} is the color multiset associated with F_{i_j}, $1 \leq j \leq n_i$. Note that $n_i \leq \text{cw}$. Let $F_{i_j} = [s_{i_j}, t_{i_j}]$ for all $1 \leq j \leq n_i$. By $\mathcal{K} = \{1, \ldots, k\}$ we refer to the set of all colors. In addition, a tuple $(M_1, \ldots, M_q)$ of multisets is called a *chain* if there exists a permutation π of $\{1, \ldots, q\}$ such that $M_{\pi(1)} \subseteq M_{\pi(2)} \subseteq \ldots \subseteq M_{\pi(q)}$. Finally, for $1 \leq i \leq n$ we say that a coloring of $[i]$ is *proper* if it satisfies all input intervals contained in $[i]$.

For every position i, the algorithm maintains a table T_i with an entry for every possible tuple of color multisets $(A_1, \ldots, A_{n_i})$ with $A_j \subseteq C_{i_j}$ and $|A_j| = i - s_{i_j} + 1$ for all $1 \leq j \leq n_i$. Informally speaking, this entry indicates whether there exists a coloring of the interval $[i]$ that uses for every j, $1 \leq j \leq n_i$, the colors in A_j for the subinterval $[s_{i_j}, i]$ and satisfies all intervals that end before position i. More specifically, the goal of the dynamic programming procedure is to compute these tables in accordance with the following definition: $T_i(A_1, \ldots, A_{n_i}) = \text{true}$ if and only if there exists a proper coloring $c' : [i] \to \mathcal{K}$ such that $c'([s_{i_j}, i]) = A_j$ for all $1 \leq j \leq n_i$ and, for every $F_l \in \mathcal{F}$ with $F_l \subseteq [i]$, it holds that $c'(F_l) = C_l$. Such a coloring is called proper with

respect to $(A_1, \ldots, A_{n_i})$. Note that an instance is a yes-instance if and only if T_n contains an entry set to true.

For position $i = 1$, initiate the table T_i as follows. According to Proposition 1, at each position in $[n]$ there starts at most one input interval and ends at most one input interval. Hence, there is exactly one interval in $\mathcal{F}_1$. Let $\mathcal{F}_1 = \{F\}$ and let C be the color multiset associated with F. Set $T_1(\{c'\}) = \text{true}$ for every $c' \in C$.

For a position $i > 1$ compute the table T_i based on T_{i-1} as described next. By Proposition 1, for every position i, $2 \leq i \leq n$, there is at most one interval in $\mathcal{F}_{i-1} \setminus \mathcal{F}_i$ and at most one in $\mathcal{F}_i \setminus \mathcal{F}_{i-1}$. Thus, assume that $\mathcal{F}_{i-1} = \{F', F_1, \ldots, F_q\}$ and $\mathcal{F}_i = \{F_1, \ldots, F_q, F''\}$, that is, $\mathcal{F}_{i-1} \cap \mathcal{F}_i = \{F_1, \ldots, F_q\}$ (if $\mathcal{F}_{i-1} \setminus \mathcal{F}_i = \emptyset$ or $\mathcal{F}_i \setminus \mathcal{F}_{i-1} = \emptyset$, then skip F' or F'' and the respective color (sub)multisets in the following formulas). Let $F_j = [s_j, t_j]$ for all $1 \leq j \leq q$.

For every tuple $(A_1, \ldots, A_q, A'')$ that forms a chain and fulfills $A_j \subseteq C_j$ with $|A_j| = i - s_j + 1$ for $1 \leq j \leq q$ and $A'' \subseteq C''$ with $|A''| = 1$, set

$$T_i(A_1, \ldots, A_q, A'') = \text{ true } \iff \exists x \in (\bigcap_{j=1}^{q} A_j) \cap A'' : T_{i-1}(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\}) = \text{true}. \tag{3}$$

Using Recursion (3), the algorithm computes the tables T_i for increasing values of i (starting with $i = 2$). Finally, it outputs "Yes" if T_n contains a true entry and "No", otherwise. This completes the description of the algorithm.

For the correctness we show by induction on i that T_i is computed in accordance with the above definition. Clearly, this is the case for $i = 1$.

For the correctness of the case $i > 1$, first note that we only consider tuples $(A_1, \ldots, A_q, A'')$ that form chains. This is correct since for a coloring $c' : [i] \to \mathcal{K}$ the tuple $(c'([s_1, i]), \ldots, c'([s_j, i]), c'([i, i]))$ forms a chain. For the induction step we show the correctness of Recursion (3). That is, we show that there exists a proper coloring c of $[i]$ with respect to $(A_1, \ldots, A_q, A'')$ (that is, $T_i(A_1, \ldots, A_q, A'') = \text{ true}$) if and only there is a proper coloring c' of $[i-1]$ with respect to $(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\})$ for some $x \in (\bigcap_{j=1}^{q} A_j) \cap A''$.

For the "$\Rightarrow$-direction" note that, if there is a proper coloring c of $[i]$ with respect to $(A_1, A_2, \ldots, A_q, A'')$, then c restricted to $[i-1]$ clearly is a proper coloring of $[i-1]$ with respect to $(C', A_1 \setminus \{c'(i)\}, \ldots, A_q \setminus \{c'(i)\})$.

For the "$\Leftarrow$-direction", note that if there is an $x \in (\bigcap_{j=1}^{q} A_j) \cap A''$ and a proper coloring c' of $[i-1]$ with respect to $(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\})$, then the extension c of c' with $c(j) := c'(j)$ for $1 \leq j < i$ and $c(i) := x$ is a proper coloring of $[i]$ with respect to $(A_1, \ldots, A_q, A'')$.

Next, we show that the running time of the algorithm can be bounded by $O((\mathrm{cw}+1)^l \cdot l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw} \cdot n)$. To this end, note that for every position there are at most $(\mathrm{cw}+1)^l$ tuples of color multisets $(A_1, \ldots, A_{n_i})$ with $A_j \subseteq C_{i_j}$ and $|A_j| = i - s_{i_j} + 1$, $1 \leq j \leq n_i$, that form a chain. This can be seen as follows. Let F_z denote the interval in $\mathcal{F}_i$ with the smallest starting point. Note that a

tuple of color multisets $(A_1, \ldots, A_{n_i})$ that forms a chain corresponds to a partition of C_z into $(n_i + 1)$ subsets. Since $n_i \leq \mathrm{cw}$ and for every color in C_z there are at most $(\mathrm{cw} + 1)$ choices, there are at most $(\mathrm{cw} + 1)^l$ such partitions. Next, we show that in $O(l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw})$ time one can determine whether there exists an $x \in (\bigcap_{j=1}^{q} A_j) \cap A''$ such that $T_{i-1}(C', A_1 \setminus \{x\}, \ldots, A_q \setminus \{x\}) = \text{true}$. To this end, we implement the dynamic programming tables T_i by dictionaries for which the addition and the lookup of a key requires $O(\log(s))$ comparisons, where s is the size of the dictionary. Such a dictionary can, for example, be realized by a balanced binary search tree. Then, for computing an entry of T_i using Recursion (3), one first determines the colors in $(\bigcap_{j=1}^{q} A_j) \cap A''$, which is doable in $O(k \cdot \mathrm{cw})$ time if the multisets are realized by size-k arrays. Then, the lookup in T_{j-1} needs at most $O(l \cdot \log(\mathrm{cw}))$ comparisons. To compare two tuples one can iterate over the two tuples in parallel until finding a first pair of multisets that are different and return the result of the comparison between these multisets. Analogously, two multisets can be compared by comparing the occurrence numbers of the colors. In total, one comparison of two tuples of color multisets takes $O(\mathrm{cw} \cdot k)$ time. Hence, for a given tuple of color multisets the computations can be done in $O(l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw})$ time. This leads to a total running time of $O((\mathrm{cw} + 1)^l \cdot l \cdot (k \cdot \mathrm{cw})^2 \cdot \log \mathrm{cw} \cdot n)$.

Finally, to prove the second running time claimed in Theorem 4, we perform an alternative analysis of the running time of the above algorithm. To this end, we need the following observation. For a multiset M that contains k different colors and for an integer $q \geq 1$, there are at most $(q+1)^{k-1}$ size-q submultisets of M; first, note that for every color there are $q + 1$ choices for the number of occurrences of this color in the subset (between 0 and q times). Second, note that choosing the occurrence numbers of the first $k-1$ colors in a size-q subset (there are at most $(q+1)^{k-1}$ choices) determines the occurrence number of the kth color. With this observation, it is not hard to verify that for each position one has to consider at most $(l^{(k-1)})^{\mathrm{cw}} = l^{\mathrm{cw} \cdot (k-1)}$ tuples of multisets. Thus, the dynamic programming tables are of this size and with the same analysis as above the total running time can be bounded by $O(l^{\mathrm{cw} \cdot (k-1)} \cdot (k \cdot \mathrm{cw})^3 \log l \cdot n)$. □

Trivially, one can solve ICC in $k^n \cdot \mathrm{poly}(n, m)$ time by trying all k colors for all n positions. Subsequently, we show that we can improve on this running time bound by exploiting the fact that for two colors the problem is polynomial-time solvable [2] (whereas it is NP-complete for three colors [5]). The idea is to "guess" only $k - 2$ colors and the positions that have one of the two remaining colors. For these positions, we then use the polynomial-time algorithm for ICC with two colors, giving the following result.

Proposition 2. ICC *can be solved in* $O((k-1)^n \cdot g(n, m))$ *time, where* $g(n, m)$ *is the time needed to solve* ICC *for* $k = 2$.

Proof. For each position $1 \leq i \leq n$, we branch into $k - 1$ cases. The first case corresponds to i being assigned color 1 *or* 2. The other $k - 2$ cases each

correspond to i being assigned one of the other $k-2$ colors. When there is no more position that we can branch on, we first check whether any of the interval constraints has been violated so far. That is, we check whether there is an interval F with associated color multiset C and a color $x \in \{3, \ldots, k\}$ such that the occurrence number of x in C is different from the number of positions in F that are colored by x. If this is the case, then this branch does not lead to a proper coloring. Otherwise, the positions with fixed colors, that is, the positions that have been assigned a color from 3 to k, are removed from $[n]$ and the intervals with their color constraints are updated accordingly (that is, we ignore the colors $3, \ldots, k$ in the color constraints). For the remaining positions, we can only assign colors 1 or 2. This problem can be solved in polynomial time [2].

Overall, we branch into $(k-1)^n$ possibilities and for each of them the problem can be solved in polynomial time. □

For the practically relevant [2] NP-complete [5] case where $k = 3$, we can achieve a further speed-up by the following simple observation: At least one of the colors appears at most on $n/3$ positions.

Proposition 3. *For $k = 3$,* ICC *can be solved in $O(1.89^n \cdot g(n, m))$ time, where $g(n, m)$ is the time needed to solve* ICC *for $k = 2$.*

Proof. For each of the three colors, we solve the problem of finding a coloring in which this particular color is assigned to at most $n/3$ positions. We try all possibilities of selecting the positions, and since at most $n/3$ positions have to be selected, the number of these possibilities is $\sum_{0 \le i \le n/3} \binom{n}{i}$. Using Stirling's approximation of factorials, we obtain an upper bound of $O(1.89^n)$ for this number. For each of these possibilities, we then solve ICC for the remaining two colors in polynomial time [2]. □

Beigel and Eppstein [3] gave a thorough study of exact exponential-time algorithms for the NP-complete 3-Coloring problem. It is tempting to investigate whether some of their tricks can be applied to ICC with three colors; in particular, a simple randomized strategy presented by Beigel and Eppstein might be promising. This is left as a challenge for future work.

6. Implementations and Experiments

We performed computational experiments on peptide fragment data that were also used by Althaus et al. [2] to find out whether our theoretical algorithms are valuable in practice. We considered only the non-trivial instances with more than one fragment. Our aim was mainly to answer the following three questions: First, what do the parameters derived from the NP-hardness reduction look like in real-world data? Second, how do the presented algorithms behave on real-world data? Third, what can we conclude from these experiments; for example, can we find new promising parameterizations either from the data itself or from the behavior of our algorithms?

Table 3: Parameters and running times for peptide fragment data from Canzar et al. [6]. Algorithm 1 is the $k^l \cdot \text{poly}(n, m)$ algorithm, Algorithm 2 stores all equivalent colorings, Algorithm 3 stores only those colorings that differ in the segments overlapping with the last l positions. The shown parameters are range n, number of intervals m, maximum interval length l, and cutwidth cw; running times are given in milliseconds. In the column for Algorithm 1 an entry "—" means that the the respective instance could not be solved because the space consumption exceeded 2GB. In the columns for Algorithms 2 and 3 an entry "—" means that the respective instance could not be solved within a time limit of one hour.

Instance	n	m	l	cw	Algorithm 1	Algorithm 2	Algorithm 3
Cabin	78	34	74	21	—	—	—
CytoCA	27	6	16	5	11,511,587	1225	1642
CytoCB	26	6	26	4	—	1,690	1,649
CytoCC	15	5	14	4	88,684	427	528
FKBP-both-A	49	24	25	19	—	13,499	17,163
FKBP-both-B	11	4	7	4	403	50	46
FKBP-both-C	25	26	25	23	—	—	—
FKBP-both-D	4	2	3	2	3	33	31
FKBP-ilp-A	35	12	21	8	—	11,074	14,074
FKBP-ilp-B	16	5	9	3	1,838	237	219
FKBP-ilp-C	36	14	23	8	—	966,583	26,758
FKBP-mem-A	35	12	21	8	—	5,494	7,127
FKBP-mem-B	16	5	9	3	1,857	133	122
FKBP-mem-C	36	14	23	8	—	3,241,131	53,662
FKBP-xiii-A	22	16	21	16	—	1,453	2,224
FKBP-xiii-B	10	4	10	4	235	106	97
FKBP-xiii-C	11	4	7	4	440	48	45
FKBP-xiii-D	25	22	25	19	—	8,334	15,626
HorseHeart-A	17	10	17	7	1,047,098	2,831	4,197
HorseHeart-B	12	4	12	4	879	58	55
HorseHeart-C	22	8	11	7	32,750	880	1,669
HorseHeart-D	37	17	23	10	—	—	6,443
HorseHeart-F	21	6	19	6	—	760	760

All experiments were run on an AMD Athlon 64 3700+ machine with 2.2 GHz, 1 M L2 cache, and 3 GB main memory running under the Debian GNU/Linux 4.0 operating system with Java version 1.5.0_14; the Java VM was invoked with 2 GB heap size.[6]

Aspects of the Data and Choice of Algorithms. First, we examined the data with respect to the parameters we identified from the NP-hardness proof in Section 2. The most obvious observation is that $k = 3$ in all instances; the other parameter values are shown in Table 3. Unfortunately, the difference between

[6]The Java program is free software and available from `http://theinf1.informatik.uni-jena.de/icc`

range n and maximum interval length l is not that big in many instances. This is usually due to one or two intervals that are very long in comparison to the other intervals. Furthermore, the cutwidth cw is usually much larger than k and only in trivial instances less than 3. Hence, we have not implemented our algorithm for cw $= 2$ since it seems unattractive for the *available* real-world data. We decided to implement the dynamic programming algorithm with running time $k^l \cdot \text{poly}(n, m)$ (called Algorithm 1 in the following) presented in the proof of Theorem 3 because it was conceptually the easiest and because, with $k = 3$, its running time of $3^l \cdot \text{poly}(n, m)$ is much better than the running times of $l! \cdot \text{poly}(n, m)$ and $(\text{cw} + 1)^l \cdot \text{poly}(n, m)$ of the algorithms from Theorems 1 and 4. Moreover, it is easy to extend this algorithm to solve the error minimization variant of ICC introduced by Althaus et al. [2]. Note, however, that the algorithm from Theorem 4 has an alternative running time bound which, for $k = 3$, is $l^{2\text{cw}} \cdot \text{poly}(n, m)$. Since in the data often cw $\ll l$ it seems worthwhile to consider this algorithm, even though, compared to the algorithm from Theorem 3, it uses three parameters (l, k, and cw) instead of two (k and l). We thus implemented two algorithms that can be seen as variants of the algorithm from Theorem 4. The main idea of these two algorithms can be described as follows. We use dynamic programming. Both algorithms process in "left" to "right" order. The positions for which values are stored in the dynamic programming table are the start- and endpoints of the input intervals. For such a position i we store a description of each proper coloring of range $[i]$. Next, we describe this description in more detail. A segment of $[n]$ is a subinterval $[s, t] \subseteq [n]$ such that each position of $[s, t]$ is contained in exactly the same set of input intervals and $[s, t]$ is maximal with respect to this property. A description of a coloring for $[i]$ contains for each segment $[s, t]$ of $[i]$ and for each color c the number of positions of $[s, t]$ that are colored with c. When creating the table entries for position i, the algorithms try all possible combinations of extending a proper coloring stored for $[j]$, $j < i$, with colorings of the segment $[j + 1, i]$, where j is the last position of the preceding segment. Basically, the algorithm as described so far (Algorithm 2) is an enumeration of all proper colorings. We have also implemented an adaption of this algorithm that, as long as there are two colorings that differ in general, but not in the segments that overlap with the last l positions, removes one of these colorings from the dynamic programming table (Algorithm 3). This latter algorithm can be shown to have a running time of $l^{(k-1)\text{cw}} \cdot \text{poly}(n, m)$, as it stores at most as many combinations as the algorithm from Theorem 4. Finally, we implemented both algorithms to not only store proper colorings, but also colorings whose total sum of errors is below an error threshold ϵ, where the sum of errors is defined as by Althaus et al. [2]. This way an optimal solution can be found by incrementally increasing the error threshold until one coloring that has an error below the considered threshold is found.

Evaluation. Algorithm 1 can compute the minimum error for all given instances with $l \leq 17$ (11 of 23 instances in total). However, one can observe an explosion in the running time for growing values of l. This is expected since for every

length-l subinterval this algorithm enumerates all 3^l colorings. Note that for instances with $l > 17$, the space consumption of our implementation is too large and the algorithm terminates immediately.

In terms of running time, for most instances there is no big difference between Algorithms 2 and 3 and both algorithms outperform Algorithm 1. Algorithms 2 and 3 also have a moderate space consumption for all instances. Moreover, Algorithms 2 and 3 can solve all but two instances within one hour. For many instances, however, the performance is much better. For example, 14 out of 23 instances could be solved in less than four seconds. Interestingly, the fact that Algorithm 3 stores only feasible colorings if they differ in the last l positions and that Algorithm 2 stores all feasible colorings does not result in better running times of Algorithm 2 for most instances. Recall that Algorithms 2 and 3 check for an increasing error value ϵ whether the input instance admits a coloring with error ϵ. The case $\epsilon = 0$ corresponds to the decision version as introduced in Section 1. For the case that $\epsilon = 0$ Algorithms 2 and 3 can solve all instances in less than one second (not shown here).

So far our algorithms are not competitive with the state-of-the-art algorithms for ICC such as the ILP based approach by Althaus et al. [2] that solves every instance in less than 10 seconds or a polynomial-delay algorithm by Canzar et al. [6] that solves every instance in less than 57 seconds. In particular, Algorithm 1 is rather slow and has a high space consumption. Algorithms 2 and 3 perform much better but are still slower than the polynomial-delay algorithm due to Canzar et al. [6]. However, there is one instance (FKBP-mem-A) where Algorithm 2 (running time 5.5 sec) is competitive with the polynomial-delay algorithm by Canzar et al. [6] (running time 7.81 sec) and one instance (FKBP-mem-C) where Algorithm 3 (running time 53.66 sec) is competitive with the polynomial-delay algorithm [6] (running time 56.31 sec).

In summary, for favorable parameter constellations our algorithms solve ICC within seconds. However, for some instances (such as FKBP-both-C) none of the considered parameters are sufficiently small. Accordingly, none of the implemented algorithms solve this instance within acceptable running time.

Conclusions from the Experiments. The experiments show that the approach by deconstructing an NP-hardness proof can lead to algorithms that are capable of solving real-world instances. However, it is also obvious that the theoretical algorithms if implemented straightforwardly, such as Algorithm 1, may be inefficient. It also becomes clear that the parameters should not only be derived from deconstructing intractability but also from examining the structure of the data. For example, cw is often much smaller than l which might be a reason for the relatively good performance of Algorithms 2 and 3 in comparison with Algorithm 1. Also note that often $l \approx n$, but the number of long intervals is usually very small. Hence, it is intriguing to consider the parameter "number of long intervals" in combination with other parameters.

7. Conclusion

Through deconstructing intractability and using methods of parameterized algorithmics, we started a multivariate complexity analysis of ICC. Refer to Tables 1 and 2 in Sections 4 and 5 for some overview and several challenges for future research. To name a concrete one here, we emphasize our specific interest in the parameter m (number of intervals). Beyond that, there remain many further tasks: For instance, also combinations of three or more parameters may be relevant. Besides that, already for combinations of two single parameters there are several qualitatively different fixed-parameter tractability results one can strive for and which typically are independent from each other. For instance, for a combined parameter (p_1, p_2) the incomparable combinatorial explosions $p_1^{p_2}$ and $p_2^{p_1}$ can both be useful for solving specific real-world instances. In addition, although polynomial-time executable data reduction rules played a significant role in this work, we achieved no nontrivial problem kernelization results (see [4, 15] for general outlines on this topic) for fixed-parameter tractable problem variants. Finally, in our theoretical algorithms, we focused attention on the decision version and corresponding exact solutions; the investigations should be extended to the optimization variants. Summarizing, the theoretical research challenges offered by ICC and, more generally, the multivariate algorithmics approach [9, 18], seem to be (almost) inexhaustible. Finally, our experimental work indicates the practical potential of a multivariate approach to the design of combinatorial algorithms for NP-hard problems such as ICC.

Acknowledgment. We thank Michael R. Fellows for early discussions about the deconstructive approach to NP-hard problems and Nadja Betzler for pointing us to the ICC problem. We thank the anonymous referees of *CPM 2009* and *Journal of Discrete Algorithms* for comments that have improved the presentation of this work. We are grateful to Ernst Althaus for providing us with experimental real-world data. Finally, we are indebted to Sven Thiel for his great job concerning implementation and experimentation.

References

[1] E. Althaus, S. Canzar, K. Elbassioni, A. Karrenbauer, and J. Mestre. Approximating the interval constrained coloring problem. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory (SWAT '08)*, volume 5124 of *LNCS*, pages 210–221. Springer, 2008.

[2] E. Althaus, S. Canzar, M. R. Emmett, A. Karrenbauer, A. G. Marshall, A. Meyer-Baese, and H. Zhang. Computing H/D-exchange speeds of single residues from data of peptic fragments. In *Proceedings of the 23rd ACM Symposium on Applied Computing (SAC '08)*, pages 1273–1277. ACM, 2008.

[3] R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms*, 54(2):168–204, 2005.

[4] H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proceedings of the 4th International Workshop on Parameterized and Exact Computation (IWPEC'09)*, volume 5917 of *Lecture Notes in Computer Science*, pages 17–37. Springer, 2009.

[5] J. Byrka, A. Karrenbauer, and L. Sanità. The interval constrained 3-coloring problem. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium (LATIN'10)*, volume 6034 of *LNCS*, pages 591–602. Springer, 2010.

[6] S. Canzar, K. Elbassioni, and J. Mestre. A polynomial delay algorithm for enumerating approximate solutions to the interval constrained coloring problem. In *Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10)*, pages 23–33. SIAM, 2010.

[7] J. Chang, T. Erlebach, R. Gailis, and S. Khuller. Broadcast scheduling: Algorithms and complexity. In *Proceedings of the 19th ACM-SIAM Symposium on Discrete Algorithms (SODA '08)*, pages 473–482. ACM-SIAM, 2008.

[8] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[9] M. R. Fellows. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In *Proceedings of the 20th International Workshop on Combinatorial Algorithms (IWOCA'09)*, volume 5874 of *LNCS*, pages 2–10. Springer, 2009.

[10] M. R. Fellows, D. Lokshtanov, N. Misra, F. A. Rosamond, and S. Saurabh. Graph layout problems parameterized by vertex cover. In *Proceedings of the 19th International Symposium on Algorithms and Computation (ISAAC '08)*, volume 5369 of *LNCS*, pages 294–305. Springer, 2008.

[11] J. Fiala, P. A. Golovach, and J. Kratochvíl. Parameterized complexity of coloring problems: Treewidth versus vertex cover. In *Proceedings of the 6th Annual Conference on Theory and Applications of Models of Computation (TAMC'09)*, volume 5532 of *LNCS*, pages 221–230. Springer, 2009.

[12] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[13] A. Frank and É. Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.

[14] J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for Closest String and related problems. *Algorithmica*, 37(1):25–42, 2003.

[15] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[16] H. W. Lenstra. Integer programming with a fixed number of variables. *Mathematics of Operations*, 8:538–548, 1983.

[17] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

[18] R. Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS'10)*, volume 5 of *LIPIcs*, pages 17–32. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2010.