The k-path coloring problem in graphs with bounded treewidth: an application in integrated circuit manufacturing

Dehia Ait-Ferhat, Vincent Juliard, Gautier Stauffer, Andres Torres

Version of Record: https://www.sciencedirect.com/science/article/pii/S0167637720301140
Manuscript_125476eebc28489ee66378615e50458d

The k-path coloring problem in graphs of bounded treewidth: an application in integrated circuit manufacturing

Dehia Ait-Ferhat[a], Vincent Juliard[a], Gautier Stauffer[b,*], Juan Andres Torres[a]

[a] *Mentor Graphics Corporation, 8005 S.W. Boeckman Road, Wilsonville, OR 97070. USA.*
[b] *Center of Excellence in Supply Chain. Kedge Business School, 680 Cours de la Libération, 33405 Talence, France.*

Abstract

We investigate the k-path coloring problem, a variant of vertex coloring arising in the context of integrated circuit manufacturing. We devise a direct (i.e. not relying on Courcelle's celebrated theorem) polytime dynamic programming algorithm for graphs of bounded treewidth and we provide computational evidences, on true instances from the company Mentor Graphics, that the corresponding algorithm is suitable for practice, which is quite unusual for "bounded treewidth"-based dynamic programming approaches.

Keywords: k-path coloring, dynamic programming, bounded tree-width, Integrated circuit manufacturing

1. Introduction

Integrated circuits are made of several layers. In a nutshell, the bottom layers contains the transistors, while the other layers (called *metal layers*) are used to connect the different components to comply with the designed functionalities of the device. We usually distinguish two types of components in metal layers: *vias* and *'wires'*. The former components are used for vertical connections and the latter for horizontal ones, see Fig. 1 for an illustration.

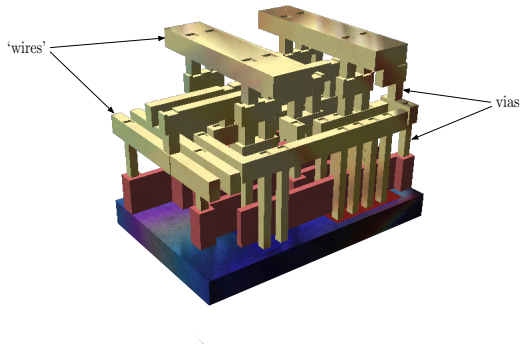

Figure 1: a 3D view of an integrated circuit (source: `https://commons.wikimedia.org/wiki/File:Silicon_chip_3d.png`)

Integrated circuit manufacturing involves the production of each layer of the circuit iteratively. The core technique used in production is *lithography*. In brief, the idea is to etch each layer by exposing a photosensitive material to a light source through a mask: this creates a kind of *mould*, which is then filled with an appropriate conductor material (the mould is later removed through some chemical process). Optical distortion might however result in the fusion of components if attention is not paid to keep the

*Corresponding author
Email addresses: `dehia.aitferhat@gmail.com` (Dehia Ait-Ferhat), `vincent_juliard@mentor.com` (Vincent Juliard), `gautier.stauffer@kedgebs.com` (Gautier Stauffer), `andres_torres@mentor.com` (Juan Andres Torres)

February 6, 2021

minimum distance between any two components above a certain threshold, called *the lithography distance*. When some components are below this distance, the production of the mould is typically decomposed into several rounds of lithography: sub-masks are defined for (feasible) subsets of components, and the mould is produced in sequence, a process called *multiple patterning*. The process requires the proper alignment of the sub-masks which is a major challenge as the number of rounds increases. Hence multiple patterning induces additional costs and time in the production process and the industry tries to keep the number of patterning steps small. The problem of finding the minimum number of patterning steps readily translates into a vertex coloring problem. Indeed, one can build a graph G whose node set is the set of components and two components are adjacent if they are at a distance less or equal to the lithography distance. This graph is sometimes called the *conflict graph*. The minimum number of rounds needed to produce the mould is the chromatic number of G.

DSA-aware Multiple Patterning (DSA-MP) is a technique that combines Directed Self-Assembly (DSA) technology with lithography in order to go beyond the resolution limit imposed by lithography alone in the fabrication of integrated circuits. Due to the nature of the process, it is particularly useful for the manufacturing of vias. DSA-MP allows to introduce within a same sub-mask some vias that are closer to the lithography distance, under certain conditions. The main idea is to intentionally fuse some vias within larger objects, called *guiding patterns*, and then to correct the corresponding design with DSA in a second step. More precisely, the mould, possibly obtained after applying several rounds of lithography, is filled with a *block copolymer* in a random state that self-organize in a structured way after a chemical reaction is triggered: if the guiding patterns are designed appropriately, and thus if the vias that are fused satisfy some specific properties, one of the two polymers assembles into a set of cylinders that can then be turned into vias. We refer the reader to [3] for more details about integrated circuit manufacturing and DSA-aware multiple patterning.

One of the core problem in DSA-MP is again the problem of minimizing the number of lithography steps as the production time and costs are still dominated by lithography. We showed in [3] that several variants are of interest in the industry. One such variant consists in fusing "small chains of vias" and it reduces to the following variant of vertex coloring: given a graph $G = (V, E)$, a subset F of edges of E and an integer $k \geq 0$, color the vertices of G with a minimum number of colors so that each color induces in G a disjoint union of paths of length less or equal to k and such that each arc in the path belongs to F. We call the problem the *k-path coloring problem*. Concretely, the node set of G is the set of vias in a layer, the edges correspond to the pairs of vias that are closer than the lithography distance, F are the pairs of vias whose distance is within a certain range specific to the DSA technology used (but smaller or equal to the lithography distance), and k is a small number. The Electronic Design Automation (EDA) industry is especially interested in the 1-path and the 2-path coloring problems as they correspond to the current technological capabilities. Regular (vertex) coloring is equivalent to 0-path coloring, which proves that (i) k-path coloring is NP-hard in general and (ii) that we can only improve upon standard multiple patterning by choosing $k \geq 1$. The problem was introduced under the same name in [4] in the special case where $F = E$ (note that some authors use the same terminology for another variant of graph coloring, see for instance [13, 17, 20]). In particular it is known that k-path L-colorability is already hard for $L = 2$ and $k = 1$ [15] and for $L = 3$ and $k = 2$ [16].

The company Mentor graphics has developed in-house heuristics to solve the problem quickly. A typical design can be made of several hundred thousands of vias per layer and Mentor Graphics' heuristics can solve (approximately) these instances in less than a second. However the company has interest in fast exact algorithms as well. Initially, their main interest for exact approaches lay in the possibility to assess the quality of their heuristics. However, because of our first encouraging results with integer programming [3], the company understood that exact approaches might actually find their way to production. While testing different integer programming models in [3], we observed that typical industrial instances exhibit some structure. We indeed noticed that most instances are extremely sparse and typically 'tree-like'. The main reason is that the design of the circuit (and the placement of vias in particular) is made with lithography constraints in mind so that distances between vias is kept as large as possible. In this project, we decided to test whether exact algorithms exploiting the 'tree-like' property could be suitable (computationally wise) for production. A well-known measure of "tree-likeness" is the notion of treewidth introduced by Robertson and Seymour [21].

Definition 1. *(Tree decomposition [21]). Let $G = (V, E)$ be a graph. A* tree decomposition *of G is a pair (X, T) where T is a tree and $X = \{X_1, \ldots, X_n\}$ is the set of nodes of T, called* bags *such that*

$\forall i \in \{1, \ldots, n\}$, $X_i \subseteq V$ *and the three following conditions are verified:*

1. $\cup_{i \in \{1,\ldots,n\}} X_i = V$.
2. $\forall (u, v) \in E, \exists X_i \in X$ *such that* $u, v \in X_i$.
3. $\forall u \in V$, *the bags containing* u *is a connected sub-tree of* T.

The *width* of a tree decomposition is the size of the largest bag minus one. The *treewidth* of a graph G is the minimum width over all possible tree decomposition of G. In particular the treewidth of a tree is 1 (each edge can be used as a bag).

There are many combinatorial optimization problems that are hard in general but that can be solved in polynomial time on graphs of bounded treewidth (see for instance [12]). Such problems include stable set, dominating set, vertex coloring, Steiner tree, feedback vertex set, hamiltonian path, etc... Besides, Courcelle [11] has shown that a large class of problems on graphs can be solved in linear time when restricted to graphs of bounded treewidth. Courcelle's theorem essentially states that every graph property that can be formulated in *monadic second order logic* (MSOL) can be decided in linear time when restricted to graphs of bounded treewidth (monadic second order logic in graphs is a formulation of a property of a graph using logical connectors ($\wedge$, $\vee$ $\neg$, $\iff$, etc...), quantification on vertices and sets of vertices ($\forall v \in V$, $\exists v \in V$, $\forall V' \subseteq V$, $\exists V' \subseteq V$), quantification on edges and sets of edges ($\forall e \in E$, $\exists e \in E$, $\forall E' \subseteq E$, $\exists E' \subseteq E$), membership tests ($e \in F$, $v \in W$, etc...) and incidence tests (v endpoint of e, $(u, v) \in E$)). It is possible to prove that k-path L-colorability falls under Courcelle's theorem umbrella, for a fixed L, see [2]. It then follows that k-path coloring is polynomial in graphs of bounded treewidth: indeed, it is part of folklore that the chromatic number of a graph of treewidth w is at most $w + 1$ (we can 'greedily' color the vertices as there is always a vertex of degree less or equal to w, see for instance [9]); hence the k-path chromatic number is also bounded by $w + 1$; we can in particular restrict testing for k-path L-colorability to $L = 1, ..., w + 1$. Unfortunately Courcelle's theorem, albeit linear in the graph size, is considered impractical [12]: *"Courcelle's theorem and its variants should be regarded primarily as classification tools, whereas designing efficient dynamic-programming routines on tree decompositions requires 'getting your hands dirty' and constructing the algorithm explicitly"*. Such 'explicit' implementations are rare, see [5, 23, 7, 22, 19] for examples, and usually turn out to still be impractical unless instances are of small size. In this paper, we develop such an 'direct' algorithm (that is, one not relying on Courcelle's theorem) for the k-path L-coloring problem and we test the performances of the corresponding algorithm on real-world instances from Mentor Graphics arising from DSA-aware Multiple Patterning for $k = 1$ and $k = 2$ (the cases of interest in the industry) and show that, somewhat surprisingly, the corresponding approach turn out to be practical (even though a bit less performant computationally than the best integer programming approach developed in [3] as will be discussed later).

Additional definitions and notations

Given a graph $G = (V, E)$ and a subset E' of edges of E, we call a E'-neighbor of a vertex v, a vertex u such that $(u, v) \in E'$. Also a E'-path denotes a path with edges in E'. For a graph G, we sometimes denote by $V(G)$ the set of vertices of G, by $E(G)$ the set of edges of G, and, for $U \subseteq V(G)$, by $E[U]$ the set of edges of G induced by the vertices in U (that is with both extremities in U).

We call a pair (G, f), where $G = (V, E)$ is a graph and f is a positive weight function on its edge set, a *weighted graph*. We can represent a weighted graph by its *(weighted) adjacency matrix*, that is a matrix A of size $|V| \times |V|$ such that $A(u, v) = f((u, v)) > 0$ for all $u, v \in V : (u, v) \in E$ and $A(u, v) = 0$ for all $u, v \in V : (u, v) \notin E$. Note that the *support* of A, that is, the 0/1 matrix of same dimension as A whose ones indicate pairs (u, v) for which $A(u, v) > 0$, is the (standard) adjacency matrix of G. A graph G can be considered as a weighted graph with weight function $f = \mathbf{1}$.

Let P be a path of G with node set $v_0, ..., v_p$, for some integer $p \geq 0$, and edge set $\{(v_i, v_{i+1})$ for $i = 0, ..., p - 1\}$ and let $U \subseteq V$. We call v_0, v_p the extremities of P. We call a node of P *internal* if it is not an extremity. Given a path P such that $V(P) \cap U \neq \emptyset$ and a function $f : E(P) \mapsto \mathbb{R}^+ \setminus \{0\}$, we define the *trace on* U of the weighted path (P, f) as the weighted graph obtained from P by 'shrinking' the internal nodes of P not in U. More formally, if $i_1 < ... < i_l$ are the indices of the vertices of U on P, for some integer $l \geq 1$, the trace of (P, f) on U is the weighted graph (P', f'), where P' is the path with vertex set $\{v_0, v_{i_1}, ..., v_{i_l}, v_p\}$ and edge set $\{(v_0, v_{i_1}), (v_{i_l}, v_p)\} \cup \{(v_{i_j}, v_{i_{j+1}})$ for $j = 1, ..., l - 1\}$, and for any edge e of P', $f'(e)$ is the length with respect to f of the (sub)path of P in-between the two end

points of e. We define the trace on U of a union of disjoint paths as the union of the traces of each path intersecting U.

A *rooted tree decomposition* is a tree decomposition where a bag is chosen as a root and the edges of the tree are oriented in the direction of the root bag. In a rooted tree decomposition (X, T), we can naturally define *children* and *parent* bags and then, for a bag X_i, we denote by V_i the union of the bags in the subtree of T rooted at X_i, and by G_i the subgraph of G induced by the nodes in V_i. It is convenient to present dynamic programming algorithms on *nice tree decompositions*. We give the definition below. Note that Kloks [18] proved that a tree decomposition can be converted into a nice tree decomposition (with at most four times the number of vertices in G) in linear time, while preserving the same treewidth.

Definition 2. *(Nice tree decomposition). Let $G = (V, E)$ be a graph. A nice tree decomposition is a rooted tree decomposition of G where every bag X_i of the tree has at most two children and is one of the four following types:*

- ***Leaf bag:*** *X_i has no children and contains only one vertex $v \in V$, i.e. $|X_i| = 1$.*
- ***Introduce bag:*** *X_i has exactly one child bag noted X_j such that $X_i = X_j \cup \{v\}$ for some $v \in V$ (and $v \notin V_j$).*
- ***Forget bag:*** *X_i has exactly one child bag noted X_j such that $X_i = X_j \backslash \{v\}$ for some $v \in X_j$.*
- ***Join bag:*** *X_i has exactly two children noted X_{j_1} and X_{j_2} such that $X_i = X_{j_1} = X_{j_2}$.*

2. Dynamic programming

We now develop a dynamic programming algorithm to solve the k-path L-coloring problem on graphs of bounded treewidth. We assume that we are given a nice tree decomposition (by the result of Kloks [18] mentioned above). We start with providing the general idea of the algorithm.

Let $G = (V, E)$ be a graph, let $F \subseteq E$, and let us consider a nice tree decomposition (X, T) of G of width w. The general idea behind a dynamic programming approach to a decision problem on graph of bounded treewidth is to evaluate in each bag X_i whether there is a solution to the problem restricted to G_i, and to store enough information about the corresponding solutions, to propagate and extend the solutions from the children bag(s) to the parent node iteratively. Hence, in our setting, for each bag X_i, we would like to know if there exists a k-path L-coloring of G_i. A k-path L-coloring of G_i is obviously a k-path L-coloring of its children bag(s). Hence we can try to identify the k-path L-coloring of G_i by checking which k-path L-coloring of the children bag(s) can be extended to a k-path L-coloring of G_i. Now, because we want to design an efficient algorithm for testing k-path L-colorability of G, we cannot keep a full list of all k-path L-colorings of G_j for all bags X_j as the list could grow exponentially as we move up the tree. Because each color of a k-path L-coloring induces a disjoint union of F-paths, and because of the structure of a tree decomposition, it is enough, as we will discuss in the proof of Theorem 1, to keep the colors of the vertices in X_i and the trace on X_i of the F-paths in each color. We call such a solution a *partial solution* (this is the standard terminology in the field) as it can be extended to build a k-path L-coloring of G_i. We can encode, for each bag X_i, the (disjoint) union of the traces on X_i of the F-paths of each color as a weighted graph with at most $3w$ vertices (we call this weighted graph the *trace of the solution*): indeed each path in the trace contains at least one node from X_i by definition and thus there are at most w such (disjoint) paths and, in the worst case, each path contains exactly two additional neighbors representing the extremities outside X_i (and each path may contain only one node from X_i). Hence the number of partial solutions is bounded by $O(L^w \cdot k^{(3w)^2})$ for each bag, which is constant for k and w bounded ($O(k^{(3w)^2})$ corresponds to the number of (weighted) adjacency matrices of a weighted graph with at most 3w vertices and weights in $\{0, ..., k\}$: we could obviously use better data structures to improve upon this value). The dynamic programming algorithm and the (inductive) proof of its validity is given in Theorem 1.

Theorem 1. *There exists an algorithm that, given a graph $G = (V, E)$, a set of edges $F \subseteq E$, and a nice tree decomposition (X, T) of G of width w solves the k-path L-coloring problem in time $O(L^w \cdot k^{2(3w)^2} \cdot (3w)^2 \cdot |T|)$.*

Proof. Let $G = (V, E)$ be a graph, let $F \subseteq E$, and let us consider a nice tree decomposition (X, T) of G of width w. For conciseness, we simultaneously give the dynamic programming algorithm and the (inductive) proof that the procedure is correct. We need to distinguish according to the different types of bag.

- **Leaf bag:** In a leaf bag X_i, we have $|X_i| = \{v\}$ for some $v \in V$ and we can enumerate all k-path L-coloring of G_i by enumerating all possible colorings of v. The partial solutions coincide with the solutions for G_i so the trace is trivial in this case: it consists in the graph with vertex set $\{v\}$ and edge set $\{\}$, and any weight function as the set of edges is empty.

- **Introduce bag:** In an introduce bag X_i, we add a vertex v from a child bag X_j i.e., $X_i = X_j \cup \{v\}$ for some $v \in V$ (and $v \notin V_j$). In order to check whether a k-path L-coloring S_j of G_j can be extended to a k-path L-coloring of G_i, we only need to check which coloring $c(v)$ of v is compatible with S_j. Adding v to a color set adds edges in the subgraph of G induced by the nodes of color $c(v)$. We want the resulting graph to be a union of disjoint F-paths of G_i of length at most k. Let $G_i^{c(v)}$ (resp. $G_j^{c(v)}$) be the subgraph of G_i (resp. G_j) induced by the node of color $c(v)$. Because of the structure of a tree decomposition, v is only adjacent to vertices of X_i in G_i. It follows that in order to check whether S_j can be extended, it is enough to check whether the set of edges $\mathcal{E} \subseteq E$ incident to both v and a node of color $c(v)$ in X_i are all in F and that adding v and $\mathcal{E}$ (with weight one) to the trace (H, f) of $G_j^{c(v)}$ on X_j yields a weighted graph (H', f') whose support H' is a union of disjoint paths of *weight* at most k with respect to f' (note that adding v might merge two previously disjoint paths). This can be checked in linear time by checking that: (i) v has no more than two neighbors of color $c(v)$ in X_i (and that they are linked by an edge of F), (ii) those neighbors are the extremities of different paths in the trace graph (H, f), and (iii) the weight of the F-path containing v is no more than k (for this we need to know the weight of the path(s) starting at the neighbors of v in (H, f), which can be easily computed by traversing the edges of the corresponding path(s) in (H, f)). In case of a positive result, substituting the trace (H', f') by (H, f) actually yields the trace on X_i of the extension of S_j to G_i.

- **Forget bag:** In a forget bag X_i, we delete a vertex v from a child bag X_j i.e., $X_i = X_j \setminus \{v\}$ for some $v \in V$. In such a node of the tree decomposition, any partial solution P_j for X_j yields a partial solution P_i for X_i. Indeed, any k-path L-coloring for G_j that would be consistent with P_j would again be a k-path L-coloring of G_i as G_i and G_j coincide. The traces of the solution on X_i and X_j differ though, but we can easily recover the trace on X_i from the trace on X_j. Indeed, we can update the coloring and the trace as follows: the coloring is kept identical but it is restricted to the nodes of X_i and the new trace is simply the trace on X_i of the trace in P_j (that is we simply 'shrink' v).

- **Join bag:** In a join bag X_i, we want to check, for every pair (P_{j_1}, P_{j_2}) of partial solutions for X_{j_1} and X_{j_2}, whether the union of any possible extensions of P_{j_1}, P_{j_2} to solutions of G_{j_1} and G_{j_2} yield a solution for G_i. More precisely, let P_{j_1} and P_{j_2} be two partial solutions for X_{j_1} and X_{j_2} respectively and S_{j_1}, S_{j_2} any two extensions. We first need to check that common vertices in G_{j_1} and G_{j_2} are colored in the same way in both S_{j_1}, S_{j_2}. Because common vertices are vertices in X_i (by the property of tree decompositions), it is enough to check that the partial solutions agree on the colors of the vertices in X_i. Then we need to check that the union of the F-paths induced by a same color c in S_{j_1} and S_{j_2} is a (disjoint) union of F-paths of length at most k (the union of the F-paths is actually the graph induced by color c in G_i as the only edges in common are edges in $E[X_i]$ - which are in both G_{j_1} and G_{j_2}). Now it is enough to look at the multi-union of the traces associated with color c in P_{j_1} and P_{j_2} respectively (formally, the multi-union is defined as follows: we take the union of the two corresponding traces (union of the vertices and edges, with same weight), and we allow parallel edges, except for 'original edges' in $E[X_i]$ - that is, edges that are in both trace graphs and of weight 1: edges of the traces of weight 2 or more are not edges from $E[X_i]$). Indeed we simply need to check that the multi-union of the traces induces a disjoint union of paths of *weight* at most k. Note that we do not consider parallel edges for 'original edges' in $E[X_i]$ in the definition of the multi-union as, in contrast with the other edges in the trace (which cannot appear as 'subpaths' in both solutions S_{j_1} and S_{j_2} because they include a vertex in $V_{j_1} \setminus X_i$ or

$V_{j_2} \setminus X_i$ that was shrunk (and thus cannot appear in the other solution because of the properties of tree decompositions)), they correspond to edges that could (and will) be in the F-paths associated with color c in both S_{j_1} and S_{j_2}. In case of a positive outcome, the trace of the solution for the join bag X_i is simply the union of the two traces (as defined above).

The discussion above shows that we do not miss any partial solution as we move up the tree (and that each partial solution we build is actually valid). Hence there exist a k-path coloring of G if and only if there exists a partial solution in the root node of the tree decomposition when applying the procedure above.

The computational time at each node of the nice tree decomposition is dominated by the join bag case. For each coloring of the vertices of X_i we then need to check whether the union of the support graph of the trace of each possible partial coloring for X_{j_1} and X_{j_2} are compatible: in the worst case we need to check L^w coloring, and $k^{(3w)^2} \times k^{(3w)^2}$ pairs of traces, and checking that the union of the traces in each color still yields a union of disjoint paths of length at most k can be done in time $O((3w)^2)$, by first building the weighted adjacency matrix of the union of the trace (and checking that it does not contain multi-edges) and by then adapting the depth first search algorithm, since each weighted adjacency matrix has size at most $(3w)^2$. The overall complexity is thus bounded by $O(L^w \cdot k^{2(3w)^2} \cdot (3w)^2 \cdot |T|)$.

□

3. Numerical Experiments

In [6], Arnborg et al have proved that deciding if the treewidth of a graph G is at most w, where $w \geq 0$, is NP-complete. However, there are good heuristics to determine a tree decomposition of a given graph G with a width 'close to' the treewidth. Different heuristics are presented and compared in [10]. Moreover, Bodlaender [8] showed that for every fixed value of w, there is a linear-time algorithm that finds a tree decomposition of width w (if it exists).

We propose to solve DSA-MP on real instances arising from integrated circuit manufacturing by first using a heuristic to get a 'small' tree-decomposition of the graph, and by then using the dynamic programming algorithm described in the previous section to solve the problem (actually, we tailored the algorithms to the case where $k = 1$ and $k = 2$ to make them slightly simpler to implement, see [2] for the details about the corresponding implementations). We used D-FLAT to implement the corresponding algorithm [1]. D-FLAT is a software system that allows for rapid prototyping of dynamic programming algorithms for graphs of 'small' treewidth. It has the advantage to implement different state-of-the-art heuristics to find a close-to-optimal nice tree decomposition and offers a generic langage (Answer Set Programming) to describe how to extend solutions for each type of nodes of the nice tree decomposition. We ran D-FLAT iteratively to solve the k-path L-colorability problem starting from $L = 2$ and increasing L until a solution was found. All tests were done on a machine equiped with an Intel(R) Xeon(R) CPU E5-2640 2.60 GHz and a memory of 529GB. As already observed, the typical industrial designs are made of several hundred thousands of vias, but because the placement of the vias is made as to anticipate as much as possible conflicts that may arise from lithography, the 'conflict graph' is usually extremely sparse and contains only hundreds or thousands of different connected components. Since the optimization of each connected component can be parallelized, we focus attention on the computational time for each connected component individually.

We report computational experiments on 23 connected components of increasing size arising from true industrial instances in Table 1. We can see that the linear time complexity is confirmed experimentally. All instances could be solved in less than 30 seconds.

The 23 industrial instances used in this study share similar properties with some pseudo-industrial instances generated in [3] (instances where the resolution limit is set to 31nm). We thus compared the dynamic programming approach developed in this paper to the best integer programming model developped in [3] on the same set of instances (and on the same machine) and the results are reported in Table 2.

It appears that the best integer programming formulations from [3] outperform the dynamic programming approach on these instances. However, from an application point of view, the computation times of the dynamic programming approach are very satisfactory (both on the true industrial instances and on the pseudo-industrial ones). Indeed the manufacturing process is defined before the launch of

Instance name	$\|V\|$	$\|E\|$	$\|F\|$	$\omega(G)$	$\Delta(G)$	DFLAT_TW(G)	χ^1_{path}	cpu time (sec)	χ^2_{path}	cpu time (sec)
industrial_1	54	56	52	3	4	2	2	0.35	2	0.92
industrial_2	61	86	85	3	5	2	2	0.75	2	1.62
industrial_3	89	112	107	3	5	2	2	0.51	2	1.5
industrial_4	90	113	109	3	5	2	2	0.98	2	1.21
industrial_5	96	111	105	3	4	2	2	0.55	2	1.73
industrial_6	98	126	120	3	5	3	2	0.84	2	3.45
industrial_7	102	144	141	3	5	2	2	0.94	2	1.47
industrial_8	111	140	137	3	5	2	2	1.45	2	2.21
industrial_9	114	131	125	3	4	2	2	0.79	2	2.21
industrial_10	116	155	151	3	5	3	2	1.15	2	2.5
industrial_11	119	142	136	3	4	3	2	1.41	2	4.02
industrial_12	128	159	149	3	5	2	2	1.17	2	2.83
industrial_13	137	167	160	3	5	3	2	1.22	2	3.05
industrial_14	159	196	188	3	5	2	2	1.42	2	3.53
industrial_15	173	224	216	3	5	3	2	1.83	2	3.44
industrial_16	382	396	339	3	4	2	2	2.57	2	4.95
industrial_17	969	1001	900	3	3	2	2	7.26	2	12.52
industrial_18	993	1009	927	3	4	2	2	6.19	2	12.63
industrial_19	997	1047	906	3	4	2	3	8.86	2	12.82
industrial_20	998	1024	924	3	4	2	3	8.65	2	12.95
industrial_21	1900	1937	1804	3	4	2	3	21.09	2	26.87
industrial_22	1912	1960	1809	3	4	2	3	18.22	2	26
industrial_23	1937	1996	1812	3	4	2	2	6.29	2	27.02

Table 1: Industrial instances characteristics and results: $\omega(G)$ is the size of the maximum clique in G, $\Delta(G)$ is the maximum degree of G, DFLAT_TW(G) is the width of the tree decomposition returned by D-FLAT heuristics, and χ^k_{path} is the k-path chromatic number.

Instance	$\|V\|$	$\|E\|$	$\|F\|$	$\omega(G)$	$\Delta(G)$	DFLAT_TW(G)	χ^1_{path}	DFLAT 1 time (sec)	IP 1 time (sec)	χ^2_{path}	DFLAT 2 time (sec)	IP 2 time (sec)
clip1_31	191	242	242	3	5	3	2	2,71	0,8	2	465,74	1,93
clip2_31	139	188	188	3	5	3	3	21,15	1,54	2	64,89	2,11
clip3_31	98	117	108	3	4	2	2	0,8	0,11	2	1,69	0,34
clip4_31	120	147	139	3	4	3	2	0,87	0,49	2	9,45	0,46
clip5_31	170	213	213	3	5	3	3	5,45	0,99	2	9,57	1,52
clip6_31	178	229	229	3	5	3	2	2,1	0,68	2	38,45	1,89
clip7_31	203	256	223	3	5	3	3	8,83	0,94	2	92,05	0,73
clip8_31	122	162	160	3	5	3	2	0,83	0,45	2	2,89	1,35
clip9_31	152	193	193	3	4	3	2	2,8	0,28	2	31,13	1,96
clip10_31	139	175	175	4	4	3	2	0,91	0,51	2	2,3	0,64

Table 2: Pseudo-industrial instances characteristics and results: $\omega(G)$ is the size of the maximum clique in G, $\Delta(G)$ is the maximum degree of G, DFLAT_TW(G) is the width of the tree decomposition returned by D-FLAT heuristics, DFLAT k time (resp. IP k time) represents the time used by DFLAT (resp. IP) to solve the k-path coloring problem, and χ^k_{path} is the k-path chromatic number.

the production and obtaining optimal solutions in a few minutes is perfectly fine. Besides, the dynamic programming approach offers the additional benefit of not requiring the use of expensive licenses (the license fees for advanced programming software such as CPLEX or Gurobi, can reach prohibitive amounts when the algorithm must run in parallel on thousands of instances and this has not been considered a viable alternative by the company). The algorithm is now implemented in the company's toolbox. They run the corresponding algorithms both to compare and improve their heuristics and also to find the optimal solution in cases where the algorithm ends within a certain time (beyond the limit, they use their heuristics).

4. Acknowledgment

This project has been partly supported by the Association Nationale de la Recherche et de la Technologie (Convention CIFRE 2015/0553). The authors would like to thank two anonymous referees for helping in improving the exposition of the paper, in particular, the proof of Theorem 1.

References

References

[1] Abseher M., Bliem, M., Charwat G., Dusberger F., Hecher M., Woltran S.(2014). The D-FLAT System for Dynamic Programming on Tree Decompositions. Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, 558–572, 2014, Springer.

[2] Ait-Ferhat, D. (2018). Design of exact solutions for the manufacturing of 'vias' using DSA technology. Doctoral Dissertation. Grenoble University.

[3] Ait-Ferhat, D., Juliard, V., Stauffer, G., and Torres, J.A. (2018). Combining lithography and Directed Self Assembly for the manufacturing of vias: Connections to graph coloring problems, Integer Programming formulations, and numerical experiments. arXiv:1902.04145.

[4] Akiyama, J., Era, H., Gervacio, S. V., and Watanabe, M. (1989). Path chromatic numbers of graphs. Journal of Graph Theory, 13(5), 571-573.

[5] Arnborg, S. (1985.) Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. BIT, 25, pp. 2 – 23.

[6] Arnborg S., Corneil D.G., and Proskurowski A. (1987). Complexity of Finding Embeddings in a k-Tree. SIAM Journal on Algebraic Discrete Methods 8(2), 277-284

[7] Arnborg, S. and Proskurowski, A. (1989). Linear time algorithms for NP-hard problems restricted to partial k-trees. Discret. Appl. Math., 23, pp. 11-24.

[8] Bodlaender, H.L. (1996). A linear time algorithm for finding tree-decompositions of small treewidth. SIAM J. Comput., 25, 1305 -1317.

[9] Bodlaender, H.L., and Fomin F.V. (2005). Equitable colorings of bounded treewidth graphs. Theoretical Computer Science 349(1), 22-30.

[10] Bodlaender H.L., Koster A.M.C.A (2010). Treewidth computations I. Upper bounds, Information and Computation, 208(3), 259-275.

[11] Courcelle B.(1990). The Monadic Second-Order Logic of Graphs. I. Recognizable Sets of Finite Graphs. Information and Computation, 85(1):12–75.

[12] Cygan M., Fomin F.V., Kowalik L., Lokshtanov D., Marx D., Pilipczuk M., Pilipczuk M., and Saurabh S. (2015). Parametrized Algorithms. Springer.

[13] Frick, M., and Bullock, F. (2001). Detour chromatic numbers. Discussiones Mathematicae Graph Theory, 21(2), 283-291.

[14] Havet, F., Kang, R. J., and Sereni, J. S. (2009). Improper coloring of unit disk graphs. Networks: An International Journal, 54(3), 150-164.

[15] Jinjiang, Y. (1995). Computational complexity of (2, 2) path chromatic number problem. Applied Mathematics-A Journal of Chinese Universities, 10(1), 83-88.

[16] Jiaiiang, Y., and Yixun, L. (1992). Some results about path chromatic numbers of graphs J. Zhengz. hou L m, 21(4), 1-8.

[17] Johns, G., and Saba, F. (1989). On the Path-Chromatic Number of a Graph. Annals of the New York Academy of Sciences, 576(1), 275-280.

[18] T. Kloks (1994). Treewidth, computations and approximations. Lecture Notes in Computer Science, 842, 1994.

[19] Koster A. M. C. A., van Hoesel S.P. M., and Kolen A.W. J. (2002). Solving Partial Constraint Satisfaction Problems with Tree Decomposition. Networks 40(3), 170-180.

[20] Mynhardt, C. M., and Broere, I. (1985, September). Generalized colorings of graphs. In Graph theory with applications to algorithms and computer science (pp. 583-594). John Wiley and Sons, Inc.

[21] Robertson N., Seymour P.D. (1986). Graph minors. II. Algorithmic aspects of treewidth. Journal of Algorithms 7(3), 309-322,

[22] Telle, J.A. and Proskurowski, A. (1997). Algorithms for vertex partitioning problems on partial k-trees. SIAM J. Discret. Math., 10, pp. 529 – 550.

[23] Wimer, T.V., Hedetniemi, S.T. and Laskar, R. (1985). A methodology for constructing linear graph algorithms. Congr. Numer., 50, pp. 43–60.