Constrained-path labellings on graphs of bounded clique-width

Bruno Courcelle*
LaBRI, Bordeaux 1 University and CNRS
courcell@labri.fr

Andrew Twigg†
Computing Laboratory
University of Oxford
andy.twigg@comlab.ox.ac.uk

February 11, 2009

Abstract

Given a graph G we consider the problem of preprocessing it so that given two vertices x, y and a set X of vertices, we can efficiently report the shortest path (or just its length) between x, y that avoids X. We attach labels to vertices in such a way that this length can be determined from the labels of x, y and the vertices X. For a graph with n vertices of tree-width or clique-width k, we construct labels of size $O(k^2 \log^2 n)$. The constructions extend to directed graphs. The problem is motivated by routing in networks in case of failures or of routing policies which forbid certain paths.

Keywords: Algorithms, labelling schemes, compact routing, clique-width.

1 Introduction

A *labelling scheme* for a property $P(x_1, ..., x_k)$ of vertices of a graph G assigns a label $L(x)$ to each vertex x in such a way that for any vertices $x_1, ..., x_k$, the validity of $P(x_1, ..., x_k)$ can be checked from their labels. An example of a property we can handle is $P(x_1, x_2, x_3)$ stating that every path between x_1 and

*Address: Institut Universitaire de France and Bordeaux University, LaBRI, 351 Cours de la Liberation, F-33405 Talence, France, Supported by the GRAAL project of "Agence Nationale pour la Recherche".

†This work was done under the support of US Army Research Laboratory and UK Ministry of Defence grant W911NF-06-3-0001, and while at the Computer Laboratory, University Of Cambridge

x_2 goes through x_3. More generally, labelling schemes can be used in this way to compute functions. An example can be the function $F(x_1, x_2, x_3)$ giving the length of a shortest path between x_1 and x_2 that goes through x_3 or a special value, say -1, if there exists no such path.

More precisely, a labelling scheme for a property $P(x_1, ..., x_k)$ or a function $F(x_1, ..., x_k)$ on graphs of a class $\mathcal{C}$ consists of two algorithms, $\mathcal{A}$ and $\mathcal{B}$. Algorithm $\mathcal{A}$ takes as input a graph G in $\mathcal{C}$ and computes a label $L_G(x)$ for each vertex x of G (labels are bit sequences). This label encodes, among other information, the name or the index of x hence determines it in a unique way. Algorithm $\mathcal{B}$ takes a k-tuple t of bit sequences as input and reports, either that t is not $(L_G(x_1), ..., L_G(x_k))$ for any graph G in $\mathcal{C}$ and any vertices $x_1, ..., x_k$ of such a graph or determines the validity of $P(x_1, ..., x_k)$ or the value of $F(x_1, ..., x_k)$ in some graph G belonging to $\mathcal{C}$, for vertices $x_1, ..., x_k$ of this graph such that $t = (L_G(x_1), ..., L_G(x_k))$. This algorithm has no other knowlege about G than the tuple t. The truth value of $P(x_1, ..., x_k)$ or the value of $F(x_1, ..., x_k)$ must be the same for all $G, x_1, ..., x_k$ with same tuple $(L_G(x_1), ..., L_G(x_k))$. Labels are intended to be as short as possible, say of size $O(\log^k n)$ (sizes are measured in bits) for fixed k, where n is the number of vertices, and the term *compact* reflects this. For checking adjacency in a rooted tree T, it suffices to take for $L_T(x)$ the pair consisting of the identifiers of x and of its father. For a tree with n vertices numbered from 0 to $n-1$ in binary notation (and $n \geqslant 2$), the size of $L_T(x)$, i.e., its length is $2.\lceil \log n \rceil$.

Motivation from distributed computing.

In a communication network handled formally as a graph G, nodes must act according to their local knowledge only. This knowledge can be updated by message passing. Due to space constraints on the local memory of nodes, and on the sizes of messages, a distributed task like routing cannot be performed on the basis of an encoding of the whole graph G in each node or in each message, but it must rather manipulate *compact representations* of some relevant aspects of G. Gavoille and Peleg [10] survey many distributed problems that can be solved by the use of labels attached to vertices.

However, such labels should be usable even when the network has node or link crashes. This case may be handled by labelling schemes that can check properties $P(x_1, ..., x_k)$ or compute functions $F(x_1, ..., x_k)$ in the given graph from which a set of vertices X and a set of edges F have been deleted (*i.e.*, represent parts of the network that are dead or forbidden). The set X is given to Algorithm $\mathcal{B}$ by the set of its labels, and the set F by the labels of the end vertices of its edges.

Such sets X and F may arise from *failures* in the network represented by the considered graph, or from so-called *network routing policies*: each node can assign costs to paths in a way independent of its neighbours' assignments, so that the shortest path is not necessarily the most desirable one. Indeed, a policy may specify that node x wishes to send information to y by using paths avoiding a certain set $X \subset V$, while some other node z may wishes to use paths avoiding another set $Y \subset V$. Sets X, Y and F are likely to change frequently, so that we

want to avoid recomputation whenever this happens. In addition to the labels, each node can store some local information that can be updated by the reception at all surviving nodes of the list of (short) labels of all defected nodes and links, so that the surviving nodes can update their local routing tables efficiently.

One can also obtain answers to queries by using vertex labels hopefully in a quicker way than by running an algorithm on the considered (presumably large) graph, because algorithm $\mathcal{A}$ has made in advance a certain amount of computation that is useful in all cases.

The first concern in this approach is to minimize the sizes of labels. A second rank one is the time and space taken by the "decoding" algorithm, called $\mathcal{B}$ above. Last comes the time and space taken by the "labelling" algorithm $\mathcal{A}$, because it is used only once for each graph, whereas $\mathcal{B}$ is likely to be used many times. Because of the intended use in routing protocols where headers of messages must not be too large, minimizing the sizes of labels is the main objective.

For the problem of exact distance labelling in absence of failures or forbidden parts, a number of results are known including optimal label sizes for various families of n-vertex graphs. For general graphs, there is a lower bound of $\Omega(n)$ on label size, and this is achieved by a scheme in [11]. For trees, there is an optimal distance labelling scheme using $\Theta(\log^2 n)$ bits per label, and, for graphs of treewidth at most k, an optimal scheme using labels of size $O(k \log^2 n)$ (see [11] and the references cited therein).

Contribution of this article

Our aim is to construct *constrained distance labelling schemes*, that is, to define a label $L(v)$ for each vertex v in such a way that for every set of vertices Z, we can compute from $(L(z))_{z \in Z}$ the distance of a shortest path between x and y that avoids a given set X of *forbidden vertices* and a given set F of *broken edges* whenever x, y, X and the set $ends(F)$ of ends of the edges of F are contained in Z. A *constrained connectivity labelling scheme* is similar but can only report whether there exists a path between x and y that avoids X and F as above, without giving its minimal length. More general types of constraints on paths will be considered in future articles. Hence, the term *constrained-path labelling* of the title refers to a promising research topic and not to a single technical notion.

For graphs having a certain "tree structure", technically those having *tree-width* or *clique-width* at most k, we show how to construct a constrained distance labelling scheme using labels of size at most $O(k^2 \log^2 n)$ bits. As by-product, we obtain a constrained connectivity labelling scheme with labels of size $O(k^2 \log n)$. This should be contrasted with the optimal bounds of $O(k \log^2 n)$ and $O(k \log n)$ bits for distance and connectivity labellings in graphs of treewidth k graphs mentioned above [11]; hence by paying a factor $O(k)$ one can route on *arbitrary subgraphs* of the given graph.

Such labelling schemes, for the case where only vertices are deleted, can actually be obtained from a general construction by Courcelle and Vanicat [8]

that we now recall. For every graph property expressible in *monadic second-order logic* and for every integer k they construct labelling schemes for graphs of tree-width and of clique-width at most k. Labels have sizes $f(k).\log n$ but the functions f derived from the general construction are extremely large. This construction extends to optimization functions definable in monadic second-order logic (like distance), and the sizes of labels are $g(k).\log^2 n$. In both cases the properties to check and the functions to compute can take set arguments. (Since, for each graph, each vertex label identifies a single vertex, a set of vertices can be defined without ambiguity by a set of labels.)

In the present case, the graph property $P(x, y, X)$ stating that there exists a path between x and y with no vertex in X and the function $F(x, y, X)$ that defines the length of a shortest such path are monadic second-order expressible, so the results of this article are applicable, but we obtain a better result by means of a direct construction avoiding logic.

We use however some basic notions from [8], in particular, the description of a graph by a term over a finite set of binary operation symbols and constants that is *balanced*, that is, has height $O(\log n)$, where n is the number of vertices. The construction of balanced terms describing graphs of bounded clique-width motivates the introduction of *m-clique-width*, a variant of clique-width that uses graphs with vertices labelled by sets of colours taken from a finite set.This notion may be of independent interest.

Results.

Apart from the definition of m-clique and the construction of balanced terms the contributions of this article are the following ones :

(a) We replace by k^2 the large and unspecified constants depending on k that arise from the construction of [8].

(b) We give an explicit construction avoiding logic.

(c) Our construction supports edge deletions (a nonempty set F in the above description) and edge additions whereas the construction based on [8] cannot.

What about real networks ? Table 1 in [12] shows that the networks of some important major internet providers are of small tree-width, between 10 and 20, and hence dealing with graphs of small tree-width or clique-width is somehow realistic. Furthermore, by using a different technique, we can define an $O(\log n)$-labelling scheme supporting connectivity queries in planar graphs, and in graphs defined as planar combinations of graphs of bounded clique-width (see [CGKT]). Hence some of our results extend to certain graphs that are neither planar nor of bounded clique-width.

Key ideas.

If Z is a set of vertices of a graph G, we denote by $G[Z]$ the subgraph of G induced by Z and by $G_+[Z]$ the graph $G[Z]$ to which we add weighted edges representing the following information : between any two vertices x and y, we set an edge with weight d if and only if d is the minimal length of a path between x and y with no intermediate vertex in Z, and at least one intermediate vertex not in Z. Such paths have length at least 2 and do not depend on whether

x and y are adjacent. Between two vertices, one may have one edge without weight that is an edge of the graph G and another one with value at least 2 that represents a path going outside of Z. See Figure 1 of Example 2 below.

We will construct a labelling scheme making it possible to build $G_+[Z]$ from the labels of the elements of any set of vertices Z. Algorithm $\mathcal{B}$ intended to report whether two given vertices x and y are linked by a path avoiding a set X of vertices and a set F of edges and how long must be such a path uses the following two steps (we assume of course that x and y are not in X):

(1) by using the labels of the vertices in $Z = \{x, y\} \cup X \cup ends(F)$, it constructs the graph $G_+[Z]$,

(2) the final answer is then easy to obtain from this graph by a classical shortest path algorithm applied to the graph $(G_+[Z] - F)\backslash X$.

Answer (2) may be obtained from $G_+[Z']$, where Z' is any set containing $\{x, y\} \cup X \cup ends(F)$. It follows that after having computed $G_+[Z']$ for a 'large' set Z' of 'sensible vertices' we can answer all queries such that $\{x, y\} \cup X \cup ends(F) \subseteq Z'$ without having to repeat step (1).

Our main result is the description of such a labelling scheme with labels of size $O(k^2 \log^2 n)$ where k is a bound on the *m-clique-width* of the considered graph. Since graphs with tree-width at most k have m-clique-width $O(k)$, and since graphs of clique-width at most k have m-clique-width at most k, the results can be used for graphs of tree-width or clique-width at most k.

Summary of article.

Section 2 reviews notations. Section 3 defines clique-width, m-clique-width and balanced terms. The long proof of a theorem about balanced terms for m-clique-width is done in an appendix in order not to break the main exposition. Section 4 contains the main construction. A preliminary version of this article [7] was presented at STACS 2007.

2 Definitions, notation and basic facts

2.1 Graphs

All graphs are without loops. Let $G = (V_G, E_G) = (V, E)$ be a directed or undirected graph, $X \subseteq V$ be a set of vertices, and $F \subseteq E$ be a set of edges. We denote by $G[X]$ the induced subgraph of G with vertex set X, by $G\backslash X$ the graph $G[V - X]$ and by $G - F$ the graph $(V, E - F)$. We denote by $ends(F)$ the set of end vertices of the edges in F.

An (X, F)-*constrained path* in G is a path in $(G - F)\backslash X$ i.e., a path (a directed path if G is directed) that does not use the edges of F and with no (end or intermediate) vertex in X. We call it X-*constrained* if F is empty. In both cases we deal with a *constrained path problem*. We denote by $d_G(x, y, X, F)$ the length of a shortest (X, F)-constrained path from x to y, that is directed or undirected depending on the type G.

Figure 1: A graph G and the graph $G_+[\{a,b,c,d\}]$.

If $Z \subseteq V$ we denote by $G_+[Z]$ the graph consisting of $G[Z]$ to which we add *weighted edges* as follows. If G is undirected (directed), if x and $y \neq x$ belong to Z, we define an undirected (directed) edge with weight d between x and y (from x to y) if and only if d is the minimal length of a path in G between x and y (from x to y) with no intermediate vertex in Z, and at least one intermediate vertex not in Z. We take 1 as weight of an edge of $G[Z]$.

Lemma 1 : For every graph G, if $Z \subseteq V_G, F \subseteq E_G$, $X \cup ends(F) \subseteq Z$ and $x, y \in Z - X$, the length of a shortest (X,F)-constrained path in G between x and y (or from x to y) is the minimal weight of a path between them in $(G_+[Z] - F)\backslash X$ where the weight of a path is the sum of weights of its edges. It can be determined in time $O(\mid Z - X \mid^2)$.

Example 2 : Figure 1 shows a graph G and the graph $G_+[Z]$ for $Z = \{a,b,c,d\}$. From $G_+[Z]$ one obtains that :

$d_G(a,d,X,F) = 2$ if $F = \varnothing$ and $X = \{c\}$ or, if $F = \varnothing$ and $X = \{b\}$,

$d_G(a,d,X,F) = 3$ if $X = \{c\}$ and $F = \{ab\}$,

$d_G(a,d,X,F) = \infty$ if $X = \{b,c\}$.□

The graph $G_+[Z]$ contains information about separators of G since there is no edge between x, y in $G_+[X \cup \{x,y\}]$ if and only if X separates x and y in G. Our problem comprizes that of constructing a compact distributed representation of *all* separators of all pairs of vertices. We will do more because our labelling scheme will give lengths of shortest X-constrained paths, and not only their existence.

2.2 Terms

For a finite set C of constants and a finite set F of function symbols, each given with a fixed arity, we let $T(F,C)$ be the set of finite terms over these two sets that are well-formed with respect to arities. Terms will also be handled as labelled trees in the usual way.

The size $|t|$ of a term t is the number of occurrences of symbols from $C \cup F$. Its height $ht(t)$ is 1 for a constant and $1+\max\{ht(t_1), ..., ht(t_k)\}$ for $t = f(t_1, ..., t_k)$.

We denote by $t \downarrow u$ the subterm of t starting from a *position* u, i.e. an occurrence of some symbol in $F \cup C$. Consider for an example the term $t = f(g(a,b), g(a,b))$. If u is any one of the two occurrences of g in this term, then $t \downarrow u = g(a,b)$. The same term $g(a,b)$ corresponds to two different concrete subtrees of the syntactic tree of t.

If a is a real number, we say that a term t is *a-balanced* if $ht(t) \leqslant a.\log(|t|+1)$. This definition is meaningful even if t has size 1. (All logarithms are in base 2.) We now explain how balanced terms of height $O(\log n)$ can denote n-vertex graphs.

3 Balanced terms denoting graphs, clique-width and m-clique-width.

3.1 Clique-width and m-clique-width.

Let L be a finite set of colours. A *coloured graph* is a triple $G = (V_G, E_G, \gamma_G)$ consisting of a graph (V_G, E_G) and a mapping γ_G associating with each x in V_G a colour in L. A *multicoloured graph* is a triple $G = (V_G, E_G, \delta_G)$ where δ_G associates with each x in V_G a (possibly empty) subset of L. In both cases, adjacent vertices may have some colours in common. As set L of colours, we will frequently use $[k]$ defined as $\{1, ..., k\}$.

Definition 3 : *Clique-width*

The notion of *clique-width* defined and studied in [6] is based on definitions of coloured graphs by means of the following operations : for $a \in L$ we let $c_{\mathbf{a}}$ be a constant (a nullary operation) denoting the graph with a single vertex with colour a. The unique binary operation is disjoint union, denoted by $\oplus$. It defines $G \oplus H$ as the union of G and an isomorphic copy of H disjoint with G.

The other operations are unary : $add_{a,b}$ adds undirected edges and $recol_{a \to b}$ modifies colours. For a, b in L , $a \neq b$:

$add_{a,b}(G) = G'$ where $V_{G'} = V_G, \gamma_{G'} = \gamma_G$ and $E_{G'} = E_G \cup \{\{v,w\} : \gamma_G(v) = a, \gamma_G(w) = b\}$.

$recol_{a \to b}(G) = G'$ where $V_{G'} = V_G, E_{G'} = E_G$ and $\gamma_{G'}$ is defined as follows:
$\gamma_{G'}(w) =$ `if` $\gamma_G(w) = a$ `then` b `else` $\gamma_G(w)$.

For defining directed graphs we use, instead of $add_{a,b}$, the operation $\overrightarrow{add_{a,b}}$:
$\overrightarrow{add_{a,b}}(G) = G'$ where $V_{G'} = V_G, \gamma_{G'} = \gamma_G, E_{G'} = E_G \cup \{(v,w) : \gamma_G(v) = a, \gamma_G(w) = b\}$.

We let F_L be the set of operations consisting of $\oplus, add_{a,b}, recol_{a \to b}, c_a$ for all $a, b \in L$ and $a \neq b$. If we are to define directed graphs, we replace $add_{a,b}$ by $\overrightarrow{add_{a,b}}$. We let $T(F_L)$ be the set of terms over F_L, called *clique-width terms*. Each of them denotes a coloured graph. Every coloured graph G has a corresponding

term in $T(F_L)$ for some L, and its clique-width $cwd(G)$ is the minimal cardinality of an L such that G is the value of some term in $T(F_L)$. It can be seen that $cwd(G) \leqslant |V_G|$. The clique-width of a graph G (without colours) is defined as that of G with all its vertices coloured by the same colour. The *width* of a clique-width term t is $|L|$, where L is the set of colours that occur in a constant or a unary operation of L. A cwd-term denoting G is *optimal* if its width is $cwd(G)$.

The notion of clique-width has proven useful for defining fixed-parameter tractable algorithms (see Courcelle et al. [2]) and labelling schemes (Courcelle and Vanicat [8]). The construction of compact labelling schemes needs that graphs are defined by balanced terms. It is proved in [8] that every graph of clique-width k can be defined by an $f(k)$-balanced term using $g(k)$ colours for fixed (but large and unspecified) functions f and g. The variant of clique-width defined below improves dramatically this situation, as shown by Theorem 9.

Definition 4 : *m-clique-width*

Multicoloured graphs are constructed with the following constants and binary operations, called the *m-clique-width operations*. (Slightly different operations based on colourings with several colours as defined in [6]).

For $A \subseteq L$ we let $c_{\mathbf{A}}$ be a constant denoting the graph with a single vertex and set A of colours for this vertex.

We now define binary operations. For every binary relation $R \subseteq L \times L$, for every mappings $g, h : L \longrightarrow \mathcal{P}(L)$ ($\mathcal{P}(L)$ is the powerset of L) and for multicoloured undirected graphs G and H, we define $K = G \otimes_{R,g,h} H$ if G and H are disjoint (if they are not, we replace G or H by an isomorphic copy disjoint from the other) by letting

$$
\begin{aligned}
V_K &= V_G \cup V_H \\
E_K &= E_G \cup E_H \cup \{\{v, w\} : v \in V_G, w \in V_H, R \cap (\delta_G(v) \times \delta_H(w)) \neq \varnothing\} \\
\delta_K(x) &= (g \circ \delta_G)(x) = \{a : a \in g(b), b \in \delta_G(x)\} \text{ if } x \in V_G \\
\delta_K(x) &= (h \circ \delta_H)(x) \text{ if } x \in V_H
\end{aligned}
$$

The graph $G \otimes_{R,g,h} H$ is well-defined only up to isomorphism since we may need to replace G or H by an (arbitrary) isomorphic copy disjoint from the other graph.

We denote by $\varnothing$ the empty graph. The mapping $G \longmapsto G \otimes_{\varnothing,g,\varnothing} \varnothing$ (or $G \longmapsto G \otimes_{R,g,h} \varnothing$ for any R and h) is a *recolouring* based on g. However, recolourings need not be introduced as unary operations because they can be combined with the constants and with the binary operations. One can also eliminate $\varnothing$ from terms defining graphs, although $\varnothing$ may be useful at intermediate stages in some constructions.

For defining directed graphs, instead of subsets R of $L \times L$, we take subsets R of $L \times L \times \{+, -\}$, and in the definition of $K = G \otimes_{R,g,h} H$ we only modify E_K as follows :

$$E_K = E_G \cup E_H \cup \{(v, w) : v \in V_G, w \in V_H, R \cap (\delta_G(v) \times \delta_H(w) \times \{+\}) \neq \varnothing\}$$

$\cup\{(w,v) : v \in V_G, w \in V_H, R \cap (\delta_G(v) \times \delta_H(w) \times \{-\}) \neq \varnothing\}$.

The *width* of a term is the number of colours used in this term. Every directed or undirected graph G is defined by some term, and the *m-clique-width* of G is the minimum width of an m-clique-width term that defines this graph. This number, denoted by $mcwd(G)$, is at most $cwd(G)$ as we will see. An m-clique-width term denoting G is *optimal* if its width is $mcwd(G)$. An example of m-cwd term is given in Example 8 below.

Clique-width, m-clique-width and the operations defining them can be compared as follows : the operations defining clique-width are simpler than those defining m-clique-width. The latter operations can be expressed as compositions of the former ones, and the corresponding terms are smaller : they have size $2n-1$ where n is the number of vertices. Those using the operations for clique-width have sizes $O(n)$ where the constant depends on k.

There is another difference between the two sets of operations. The operation $add_{a,b}(G)$ adds edges to the argument graph G, whereas the operation $\otimes_{R,g,h}$ adds edges between the two disjoint graphs that are its arguments (as in the operations defined by Wanke [15]).

The same classes of graphs have bounded clique-width and bounded m-clique-width (see Proposition 5) and Theorem 9 below motivates the introduction of m-clique-width.

To make notation precise, for a finite set L of colours, we let F_L be the set of all binary operations $\otimes_{R,g,h}$ with $R \subseteq L \times L$ or $R \subseteq L \times L \times \{+,-\}$, and $g, h : L \longrightarrow \mathcal{P}(L)$, and C_L be the set of constants $\{c_{\mathbf{A}} : A \subseteq L\}$. Every term t in $T(F_L, C_L)$ denotes a multicoloured graph $val(t)$ with colours in L, and every multicoloured graph G with colours in L_0 is the value of such a term t in $T(F_L, C_L)$ for large enough L containing L_0.

If G is undirected its m-clique-width is the same as that of the directed graph with directed opposite edges between any two adjacent vertices.

We do not redefine the well-known *tree-width* of a graph G (denoted by $twd(G)$) : see Bodlaender [1] for a thorough survey. We use it only in the following proposition for the purpose of comparison with m-clique-width.

Proposition 5 : For every graph G we have :

$mcwd(G) \leqslant cwd(G) \leqslant 2^{mcwd(G)+1} - 1$.

If G is undirected $mcwd(G) \leqslant twd(G) + 3$. If G directed $mcwd(G) \leqslant 2twd(G) + 4$.

Proof sketch : For proving $mcwd(G) \leqslant cwd(G) \leqslant 2^{mcwd(G)+1} - 1$, one can use constructions similar to those proving Proposition 5.4 and Theorem 5.5 in [6]. The proof that $mcwd(G) \leqslant twd(G) + 3$ for G undirected is essentially the one of Theorem 5.5 of [6]. For proving that $cwd(G) \leqslant 2^{twd(G)+1} + 1$, graphs of tree-width k are constructed with operations defining m-clique-width that use colours in $\{0, 1, ..., k, *, \$\}$, which gives $twd(G) + 3$. For directed graphs of tree-width k the set of colours is $\{0, 1, ..., k, 1', ..., k', *, \$, \$'\}$ which gives $2.twd(G)+4$. □

Remark 6 : For proving that $mcwd(G) \leqslant cwd(G)$, we transform a $cwd-$term t defining G into an equivalent $mcwd-$term t' such that $ht(t') \leqslant ht(t)$. For proving that $cwd(G) \leqslant 2^{mcwd(G)+1}-1$, we transform an $mcwd-$term t defining G of width k into an equivalent $cwd-$term t' of width at most $2^{k+1}-1$ such that $ht(t') \leqslant f(k).ht(t)$ for some fixed function f. For proving that $mcwd(G) \leqslant twd(G)+3$ for G undirected, we transform a tree-decomposition T of G of width k based on a tree of degree at most 3 into an $mcwd-$term t' of width at most $k+3$ such that $ht(t') \leqslant g(k).Diam(T)$ for some fixed function g, where $Diam(T)$ is the diameter of the tree of T.

It follows, and this is important for us, that if the given term t or tree-decomposition T is a-balanced, then the obtained term t' is b-balanced for some b.□

Our main theorem will be shown for graphs of bounded m-clique-width. By Proposition 5 and the above remark, it holds for graphs of bounded tree-width and clique-width. The construction of a labelling scheme for graphs of bounded tree-width is detailed in [7].

Definition 7 : *Specification of concrete graphs*

Terms in $T(F_L, C_L)$ define graphs up to isomorphisms. For defining concrete graphs, it will be convenient to use constants $c_{\mathbf{A}}(u)$ instead of $c_{\mathbf{A}}$. In such a constant, u is the vertex defined by this constant. We still denote by C_L the set of such constants. With this assumption, if a graph G is the value of a term t, then each vertex is defined by a constant $c_{\mathbf{A}}(u)$ having a unique occurrence in t, and since no constant denotes the empty graph, each leaf of t (considered as a tree) corresponds to such a constant and defines a vertex. We let h denote the bijection of the set of leaves of t onto the vertex set of G. For a node u of t, $val(t \downarrow u)$ is the induced subgraph of G whose vertex set is the image under h of the set of leaves of t that are below u. The colours of vertices in $val(t \downarrow u)$ may not be the same as in G, because of the recolourings that can be performed in t "above u".

Example 8. Let $L = [3]$ and t be the term

$$\otimes_D(\otimes_B[\otimes_A(c_{\mathbf{1}}(u), c_{\mathbf{1}}(v)), \otimes_A(c_{\mathbf{1}}(w), c_{\mathbf{1}}(x))], \otimes_C(c_{\mathbf{2}}(y), c_{\mathbf{12}}(z))))$$

where $A = (\{(1,1),(1,2)\}, \{1 \mapsto 2, 2 \mapsto 1\}, \{1 \mapsto 3\})$,which means :
$A = (R, g, h)$ with $R = \{(1,1),(1,2)\}$,
$g(1) = \{2\}, g(2) = \{1\}, g(3) = \varnothing$,
$h(1) = \{3\}, h(2) = h(3) = \varnothing$, and similarly :
$B = (\{(3,2),(2,1),(1,2)\}, \{2 \mapsto 1\}, \{3 \mapsto \{1,3\}\})$,
$C = (\{(2,1)\}, \{2 \mapsto 3\}, \{2 \mapsto 3\})$,
$D = (\{(1,3),(3,3)\}, \{1 \mapsto 1\}, \{3 \mapsto 2\})$.

The graph $val(t)$ defined by this term is the cycle $u_1 - v - w - x_1 - y_2 - z_2 - u_1$ with additional edges between u and y, and between x and z. The subscripts 1 and 2 indicate the colours of the vertices.□

3.2 The balancing theorem for m-cwd terms

The long proof of the following theorem is in the appendix.

Theorem 9. Every undirected (resp. directed) graph of clique-width or m-clique-width k with n vertices ($n > 1$) is the value of an m-cwd term t of width at most $2k$ (resp. $3k$) and of height at most $3.(\log(n-1)+1)$. The time taken to build t from a given term s of width k is $O(n.\log n)$, where n is the size of s.

Since the size of an mcwd-term t that defines a graph with n vertices is $2n-1$, this theorem defines 3-balanced terms. It proof constructs a 3-balanced m-cwd term from a cwd- or an m-cwd-term. A balanced m-cwd term can be converted into an equivalent balanced cwd-term at the cost of an exponential increase of the number of colours (see Remark 36 in Appendix). Such an increase would affect significantly the sizes of labels to be constructed. This is why we will base our constructions on m-cwd terms.

A similar result for tree-decompositions has been obtained by Bodlaender [1]. Courcelle and Kanté [5] give a general framework for establishing such results.

4 Constructions of labelling schemes

We first give constructions for undirected graphs. The extension to directed graphs is straightforward.

4.1 Adjacency labelling for m-clique-width bounded graphs

Definition 10 : An adjacency labelling.

Without loss of generality, we let $L = [k]$. We let G be a graph that is the value of a term t in $T(F_L, C_L)$ with constants of the form $c_{\mathbf{A}}(x)$ indicating the vertices they define. From now on by using Definition 7, we identify the leaves of t to the corresponding vertices of G. We assume that the n vertices are numbered from 0 to $n-1$ in an arbitrary (random) way. (Our construction will not use a particular numbering). For a vertex x of G, we let $Path(x)$ be the path $(u_m, u_{m-1}, ..., u_0)$ from leaf $x = u_m$ of t to the root u_0. We have $m+1 \leqslant ht(t)$.

We now describe an *adjacency labelling* $I(x)$, *i.e.* a labelling encoding x and intended to indicate whether vertices given by their labels are adjacent. We define

$$I(x) = (x, k, L_m, e_{m-1}, D_{m-1}, L_{m-1}, e_{m-2}, D_{m-2}, ..., e_0, D_0, L_0)$$

where : (1) $L_m = A$ and $c_{\mathbf{A}}(x)$ is the constant at leaf x in t,

(2) for each $i = 0, ..., m$, L_i is the set of colours of the vertex x in the graph $val(t \downarrow u_i)$,

(3) for each $i = 0, ..., m-1$, if $\otimes_{R,g,h}$ is the operation that occurs at node u_i, then :

(3.1) $e_i = 1$ if u_{i+1} is the left son of u_i and D_i is the set of colours j' such that $(j, j') \in R$ for some j in L_{i+1}

(3.2) $e_i = 2$ if u_{i+1} is the right son of u_i and D_i is the set of colours j' such that $(j', j) \in R$ for some j in L_{i+1}.

Let us be very concrete. A label $I(x)$ can be encoded as a word of length $\lceil \log n \rceil + \lceil \log k \rceil + (2k+1).m + 2$ over the alphabet {0,1,#}:

$$bin(x)\#bin(k)\#[L_m]e'_{m-1}[D_{m-1}][L_{m-1}]e'_{m-2}[D_{m-2}]...e'_0[D_0][L_0]$$

where $bin(x)$ denotes the binary writing of an integer x (or of the index number of a vertex x), $[L]$ is the bit sequence of length k that represents in the obvious way a subset L of $\{1, ..., k\}$, and $e'_i = e_i - 1$ for each i. The decoding algorithm must know k in order to analyse correctly the sequence $[L_m]e'_{m-1}[D_{m-1}]...e'_0[D_0][L_0]$. This word can be encoded on {0,1} by a twice longer sequence.

We obtain thus a bit sequence of length $O(k.m + \log n)$ computable from t in time $O(k^2.ht(t))$. Computing the entire labelling takes time $O(k^2.n.ht(t))$. For a balanced term t, we get labels of length $O(k.\log n)$ and the global computation time is $O(k^2.n.\log n)$.

Lemma 11. One can determine in time $O(k.ht(t) + \log n)$ whether two vertices x and y are adjacent in G from the sequences $I(x)$ and $I(y)$.

Proof. From the integers $e_{m-1}, ..., e_0, e'_{m'-1}, ..., e'_0$ in the sequences

$$I(x) = (L_m, e_{m-1}, D_{m-1}, L_{m-1}, e_{m-2}, D_{m-2}, ..., e_0, D_0, L_0)$$
$$I(y) = (L'_{m'}, e'_{m'-1}, D'_{m'-1}, L'_{m'-1}, ..., e'_0, D'_0, L'_0)$$

one can determine the position i in $Path(x)$ and $Path(y)$ of the least common ancestor u_i of x and y. Without loss of generality we assume that x is below (or equal to) the left son of u_i. Then x and y are adjacent in G if and only if $D_i \cap L'_{i+1} \neq \varnothing$. This is equivalent to $D'_i \cap L_{i+1} \neq \varnothing$.□

Proposition 12. From $(I(x))_{x \in Z}$ for a set $Z \subseteq V$, one can determine the graph $G[Z]$ in time $O(|Z|^2.(k.ht(t) + \log n))$, or $O(|Z|^2.k.\log n)$ if t is balanced, i.e. has height $O(\log n)$.

We have thus defined an *implicit representation* in the sense of Kannan et al.[14] for graphs of *mcwd* at most k, using labels of size $O(k.\log n)$, with the help of Theorem 9.

4.2 Construction of a distance labelling for constrained paths

We recall that we denote by $G_+[Z]$ the induced subgraph $G[Z]$ augmented with directed or undirected edges (according to the case) that have an integer value

at least 2 indicating the length of a shortest path between two vertices with no intermediate vertex in Z and at least one not in Z. We show how to enrich $I(x)$ in order to be able to construct $G_+[Z]$ from the labels of Z. The main construction of this article establishes the following proposition:

Proposition 13 : For every k, for every directed or undirected graph $G = (V, E)$ defined by a term t in $T(F_{[k]}, C_{[k]})$, one can build a labelling J such that, from the labels $J(x)$ of the vertices x in Z where $Z \subseteq V$, one can determine $G_+[Z]$ in time $O(k^3.|Z|^2.ht(t))$. The labels have size $O(k^2.\log^2 n)$.

Here is the idea. From $(I(x))_{x\in Z}$ for any $Z \subseteq V$, one can reconstruct $G[Z]$, but for determining the graph $G_+[Z]$ we need to know about paths going out of Z, in particular, we need to know the lengths of shortest such paths.

As in Section 4.1, we assume the graph G defined as the value of a term t in $T(F_{[k]}, C_{[k]})$. If u is a node of a path $Path(x)$ for some x in Z, and w is a son of u not on $Path(y)$ for any y in Z, then we will extract from the label of x the lengths of at most k^2 shortest paths running through the subgraph of G induced on the leaves of t below w (such leaves are not in Z). These values are precomputed and stored in a matrix of integers $MIN(w)$ inserted at the position corresponding to u in the label $J(x)$. The matrices of lengths of paths in the graphs $val(t \downarrow w)$ for all nodes w of t will be computed bottom-up by means of a rule of the form

$$MIN(u) = F(MIN(u_1), MIN(u_2), (R, g, h))$$

where u_1 and u_2 are the two sons of u and $\otimes_{R,g,h}$ is the operation at node u.

We introduce an important notion and state some lemmas.

Definition 14 *Graph representations of m-clique-width terms.*

With every term t in $T(F_{[k]}, C_{[k]})$, we associate a graph denoted by $Rep(t)$ called its *graph representation*. This graph is, roughly, the syntactic tree of t augmented with directed and undirected edges in such a way that the paths in the graph $val(t)$ can be described as paths of $Rep(t)$ obeying certains constraints on edge directions. We recall that we consider terms t defining undirected graphs $val(t)$. The extension to directed graphs will be easy.

The vertices of $Rep(t)$ are the leaves of t and the pairs (u, i) for nodes u of t and colours $i \in [k]$ that colour at least one vertex in $val(t \downarrow u)$. This graph has edges of two types: *vertical* directed and *horizontal* undirected edges.

The horizontal edges link (u_1, i) and (u_2, j) whenever u_1, u_2 are respectively the left and right sons of a node u that is an occurrence of an operation $\otimes_{R,g,h}$ such that $(i, j) \in R$.

The vertical edges are of three types :

1. $u \longrightarrow (u, i)$ for u a leaf with associated constant $c_{\mathbf{A}}(u)$ and each $i \in A$.
2. $(u_1, i) \longrightarrow (u, j)$ whenever u_1 is the left son of a node u that is an occurrence of $\otimes_{R,g,h}$ and $j \in g(i)$.

Figure 2: A graph $Rep(t)$

3. $(u_2, i) \longrightarrow (u, j)$ whenever u_2 is the right son of a node u that is an occurrence of $\otimes_{R,g,h}$ and $j \in h(i)$.

Figure 2 shows t and the graph $Rep(t)$ for the term t of Example 8. On this figure, we denote the constant $c_{\mathbf{1}}(u)$ by $1(u)$ and similarly for the others. The relation R of the operation $\otimes_A = \otimes_{R,g,h}$ contains the pair (1,2). This pair yields no horizontal edge in Figure 2 because, in the term t used as example, the arguments of $\otimes_A$ do not define vertices with the colours that would make this pair useful.

Analyzing graph representations of mcwd-terms

The graph $Rep(t)$ contains all necessary information to build the graph $val(t)$. We first examine how colours of vertices can be determined.

We will use $w \longrightarrow^* w'$ (equivalently $w' \longleftarrow^* w$) to denote a directed path from w to w'.

Lemma 15. A vertex u of G below or equal to a node w of t has colour i in $val(t \downarrow w)$ if and only if $u \longrightarrow^* (w, i)$ in $Rep(t)$.

This is clear from the definitions. We will denote by $S(w)$ the set of colours i of some vertex of $val(t \downarrow w)$, by $Rep(t)(u)$ the set of all vertices (w, i) as in Lemma 15, and by $Rep(t)(Z)$ the union of these sets for all u in Z, where Z is any subset of V.

We illustrate this lemma with help of the graph of Figure 2 defined by the term of Example 8. Vertex u initially coloured by 1 gets colour 2 by $\otimes_A$, then colour 1 by $\otimes_B$ and keeps colour 1 in $val(t)$. Vertex v initially coloured by 1 gets colour 3 by $\otimes_A$ and looses its colour after $\otimes_B$ is applied.

We have $S(b) = \{1, 3\}$ where b is the occurrence of $\otimes_B$ and $S(d) = \{1, 2\}$ where d is the root. Figure 2 shows only the colours of $S(n)$ at each node n, because a pair like $(b, 2)$ is, by Definition 14 not a vertex of $Rep(t)$.

By the definitions, for every (w, i) in $Rep(t)$ there is a path $u \longrightarrow^* (w, i)$ for some vertex u of $val(t)$. We may have $u \longrightarrow^* (w, i)$ and $u \longrightarrow^* (w, j)$ with $i \neq j$ because a vertex u may have several colours in $val(t \downarrow w)$. This is the case of vertex z in $Rep(t)$ of Figure 2. We have $z \longrightarrow (z, 1)$ and $z \longrightarrow (z, 2)$. We also have $x \longrightarrow^* (b, 1)$ and $x \longrightarrow^* (b, 3)$ where b is the occurrence of $\otimes_B$. Note that $val(t \downarrow b)$ is the path $u - v - w - x$ where u has colour 1 and x has colours 1 and 3.

Next we examine in a similar way how the adjacency of two vertrices of $val(t)$ can be determined from the graph $Rep(t)$.

Lemma 17. Two distinct vertices u, v of G are adjacent if and only if we have a mixed (directed/undirected) path $u \longrightarrow^* (w, i) - (w', j) \longleftarrow^* v$ in $Rep(t)$ for some w, w', i, j.

We call such a path an *elementary path* of $Rep(t)$. In the example of Figure 2, the adjacency of u and y in $val(t)$ is witnessed by the elementary path : $u \longrightarrow (u, 1) \longrightarrow (a, 2) \longrightarrow (b, 1) - (c, 3) \longleftarrow (y, 2) \longleftarrow y$ where a, b, c denote occurrences of $\otimes_A, \otimes_B, \otimes_C$ respectively. The adjacency of x and y is witnessed by two distinct elementary paths.

We now describe the paths of the graph $val(t)$, and more generally its walks. (A *walk* in a graph is a path where vertices may be visited several times.) A *good walk* in $Rep(t)$ is a walk that is a concatenation of elementary paths. Its *length* is the number of undirected edges it contains (hence, the number of elementary paths of which it is the concatenation).

Lemma 18 There is a walk $P = x - z_1 - ... - z_p - y$ in $G = val(t)$ if and only if there is in $Rep(t)$ a good walk of the form :

$$W = x \longrightarrow^* - \longleftarrow^* z_1 \longrightarrow^* - \longleftarrow^* ... \longrightarrow^* - \longleftarrow^* z_p \longrightarrow^* - \longleftarrow^* y$$

We say in this case that W *represents* the walk P. In the example of Figure 2, there is a path of length 3 : $u - v - w - x$ in $val(t)$ represented by the good walk :

$$u \longrightarrow (u, 1) - (v, 1) \longleftarrow v \longrightarrow (v, 1) \longrightarrow (a, 3) - (a', 2)$$
$$\longleftarrow (w, 1) \longleftarrow w \longrightarrow (w, 1) - (x, 1) \longleftarrow x$$

where a and a' are the two occurrences of $\otimes_A$.

The surgery of good walks will use the following notion. For a nonleaf node u of the term t, a *u-downwalk* in $Rep(t)$ is a walk that is formed of consecutive

steps of a good walk W and is of the form

$$(u,i) \longleftarrow^* z \longrightarrow^* - \ \longleftarrow^* \cdots - \ \longleftarrow^* z' \longrightarrow^* (u,j) \qquad (4.1)$$

where all vertices except the end vertices $(u,i),(u,j)$ are either u (if u is a leaf) or of the form w or (w,l) for w strictly below u in t. It goes through at least one leaf. We may have $z = z'$ in (4.1). Its *length* is defined as the number of undirected edges. It is *minimal* if there is no u-downwalk of smaller length with same ends. If we have $u \longrightarrow^* (w,i)$ and $u \longrightarrow^* (w,j)$ (possibly with $i = j$) then $(w,i) \longleftarrow^* u \longrightarrow^* (w,j)$ is a u-downwalk of length 0.

Lemma 19 : A u-downwalk as defined by (4.1) represents a walk from z to z' in the graph $val(t \downarrow u)$, that may be empty, i.e. reduced to z. Conversely, every (possibly empty) walk in $val(t \downarrow u)$ from z to z' such that z has colour i and z' has colour j (in $val(t \downarrow u)$) is represented by a u -downwalk of the form (4.1) from (u,i) to (u,j).

This is clear from the definitions. If in a good walk we replace a downwalk from (u,i) to (u,j) by a shorter one also from (u,i) to (u,j) (shorter w.r.t. the particular notion of length defined above) we obtain a shorter good walk. Again with the example of Figure 2 and the same designation of occurrences of function symbols, we have the following b-downwalk of length 3 :

$(b,1) \longleftarrow (a,2) \longleftarrow (u,1) \longleftarrow u \longrightarrow (u,1) - (v,1) \longleftarrow v \longrightarrow (v,1) \longrightarrow (a,3) -$
$(a',2) \longleftarrow (w,1) \longleftarrow w \longrightarrow (w,1) - (x,1) \longleftarrow x \longrightarrow (x,1) \longrightarrow (a,3) \longrightarrow (b,3).$

Is it not minimal because the following b-downwalk is shorter :

$$(b,1) \longleftarrow (a,3) \longleftarrow (x,1) \longleftarrow x \longrightarrow (x,1) \longrightarrow (a,3) \longrightarrow (b,3).$$

Definitions 20 : *The matrices MIN(u) of shortest paths ; truncated elementary paths.*

For vertices (u,i) and (u,j) of $Rep(t)$, we let $Min(u,i,j)$ be the length of a minimal u-downwalk from (u,i) to (u,j), or ∞ if no such downwalk exists. Clearly $Min(u,i,i) = 0$ ((u,i) is a vertex of $Rep(t)$, so Lemma 15 applies), and we may have $Min(u,i,j) = 0$ for $i \neq j$. We have $Min(u,i,j) = Min(u,j,i)$. Note that $Min(u,i,j) = Min(u,j,\ell) = 0$ does not imply $Min(u,i,\ell) = 0$, and similarly we may have $Min(u,i,\ell) = \infty$ whereas $Min(u,i,j) < \infty, Min(u,j,\ell) < \infty$.

In our current example $Min(b,1,3) = 0$ and $Min(d,1,2) = 1$, where d is the root.

We define $MIN(u)$ as the symmetric $S(u) \times S(u)$ matrix of all such integers $Min(u,i,j)$ ("integer" means here nonnegative integer or ∞). We recall that $S(u)$ is the set of colours p such that (u,p) is a vertex of $Rep(t)$. We will see later how these matrices can be computed.

Yet another technical notion. A *truncated elementary path* in $Rep(t)$ is obtained from an elementary path by removing intial and/or final vertical edges.

It is a path of the form

$$q \longrightarrow^* (w,j) - (w',j') \longleftarrow^* q' \qquad (4.2)$$

where q is a leaf or a pair (u,i) and q' is a leaf or a pair (v,i'), and w and w' are the two distinct sons of some w".

Computation of $G_+[Z]$ from information attached to the vertices of Z.

Our objective is to determine $G_+[Z]$ from information attached to the paths in t between the root and the leaves belonging to Z, like in the adjacency labelling $I(x)$ of Section (4.1).

Notation : We let $a(Z)$ denote the set of vertices (u,i) in $V_{Rep(t)}$ for nodes u on paths $Path(x)$ from the root to each x in Z. Hence $a(Z) \supseteq Rep(t)(Z)$.

We let $n(Z)$ denote the set of vertices in $V_{Rep(t)} - a(Z)$ of the form (w,i) for some w that is a son of some u on a path $Path(x)$ for x in Z.

Clearly, if $(w,i) \in n(Z)$ and w' is strictly below w in t, then $(w',j) \notin n(Z) \cup a(Z)$ for any j.

Example 8 (continued) : In our example of Figure 2, if we take $Z = \{u,y\}$, then

$Rep(t)(Z) = \{u,(u,1),(a,2),(b,1),(d,1),y,(y,2),(c,3),(d,2)\}$,

$a(Z) = Rep(t)(Z) \cup \{(a,3),(b,3)\}$,

$n(Z) = \{(v,1),(a',2),(a',3),(z,1),(z,2)\}$.

And we have also :

$a(\{u\}) = \{u,(u,1),(a,2),(a,3),(b,1),(b,3),(d,1),(d,2)\}$

$n(\{u\}) = \{(v,1),(a',2),(a',3),(c,3)\}$.□

Since the vertices of $Z \cup n(Z) \cup a(Z)$ are on, or are "close to the paths in t " between the root and the leaves in Z, the edges between them, i.e. those of the induced subgraph $Rep(t)[Z \cup a(Z) \cup n(Z)]$ of $Rep(t)$ can be determined from $(I(z))_{z \in Z}$, and so can be $G[Z]$.

Definition 21: *Z-external paths.*

A *Z-external* path in G is a path of the form :

$$P = x - v_1 - v_2 - ... - v_m - y,$$

with $x,y \in Z, v_1,v_2,...,v_m \notin Z, m \geqslant 1$.

Let W be a good walk representing P (by Lemma 18). We consider it as a sequence of edges that we factorize into $W = H_1W_1H_2W_2...W_pH_{p+1}$where $H_1,H_2,...,H_{p+1}$ are truncated elementary paths with edges in $Rep(t)[Z \cup a(Z) \cup n(Z)]$ and $W_1,W_2,...,W_p$ are downwalks. Each of these downwalks has ends $(w,i),(w,j)$ in $n(Z)$, no intermediate vertex in $Z \cup a(Z) \cup n(Z)$ and no edge in $Rep(t)[Z \cup a(Z) \cup n(Z)]$.

Example 8 (continued) : We take again the example of Figure 2, with $Z = \{u,y\}$ and $P = u - v - w - x - y$ with representing good walk :

$W = u \longrightarrow (u,1) - (v,1) \longleftarrow v \longrightarrow (v,1) \longrightarrow (a,3) - (a',2)$

$\longleftarrow (w,1) \longleftarrow w \longrightarrow (w,1) - (x,1) \longleftarrow x \longrightarrow (x,1) \longrightarrow (a',3)$
$\longrightarrow (b,3) - (c,3) \longleftarrow (y,2) \longleftarrow y$

It can be factorized into $H_1W_1H_2W_2H_3$ where :

$W_1 = (v,1) \longleftarrow v \longrightarrow (v,1)$ and

$W_2 = (a',2) \longleftarrow (w,1) \longleftarrow w \longrightarrow (w,1) - (x,1) \longleftarrow x \longrightarrow (x,1) \longrightarrow (a',3)$

$H_1 = u \longrightarrow (u,1) - (v,1)$

$H_2 = (v,1) \longrightarrow (a,3) - (a',2)$

$H_3 = (a',3) \longrightarrow (b,3) - (c,3) \longleftarrow (y,2) \longleftarrow y.\Box$

In a factorization $W = H_1W_1H_2W_2...W_pH_{p+1}$ a downwalk W_ℓ with ends $(w,i),(w,j)$ represents a walk of $G[V-Z]$. A truncated elementary path H_ℓ represents a family of edges, created in the same way by an operation $\otimes_{R,g,h}$ with occurrence w" (cf. (4.2) in Definition 20). One can replace in W a downwalk W_ℓ by another one with same ends. One still gets a good walk W' of $Rep(t)$ that represents a walk in G from x to y. It follows that if P is a shortest Z-external path between x and y all downwalks of the factorization of W representing it are minimal.

Conversely, one can determine the lengths of the shortest Z-external paths between x and y by examining all truncated elementary paths in $Rep(t)[Z \cup a(Z) \cup n(Z)]$ and by using the values $Min(w,i,j)$ for all $(w,i),(w,j)$ in $n(Z)$. These remarks prove the following lemma.

Lemma 22 : One can determine $G_+[Z]$ from the graph $Rep(t)[Z \cup a(Z) \cup n(Z)]$ and the values $Min(w,i,j)$ for all $(w,i),(w,j)$ in $n(Z)$.

Lemma 26 will establish the time complexity of this computation.

Example 8 (continued) Figure 3 shows $Rep_+(t)[Z \cup a(Z) \cup n(Z)]$ where $Z = \{u,y\}$, using the term t as in Figure 2. It shows also that $Min(z,1,2) = 0$ and that $Min(a',2,3) = 1$.

Definition 23 : *The labelling* $J(x)$.

We are now ready to define the label $J(x)$ for each vertex x of G. We recall that $S(u)$ is the set of colours p such that (u,p) is a vertex of $Rep(t)$ and that $MIN(u)$ be the symmetric $S(u) \times S(u)$ matrix of integers $Min(u,i,j)$. This matrix can be stored using space $O(k^2.\log n)$ since n bounds the lengths of shortest u-downwalks in $Rep(t)$. (This is so because a factor of the form $z \longrightarrow^* - \longleftarrow^* ... \longrightarrow^* - \longleftarrow^* z$ in a u-downwalk can be deleted and one obtains a shorter u-downwalk with same ends.)

For a leaf x of t, i.e., a vertex of G, the path $Path(x)$ is of the form $(u_m, u_{m-1}, ..., u_0)$, with u_0 the root and $u_m = x$, we recall from Section (4.1) that

$$I(x) = (x, k, L_m, e_{m-1}, D_{m-1}, L_{m-1}, e_{m-2}, D_{m-2}, ..., e_0, D_0, L_0).$$

We let then

$$J(x) = (x, k, L_m, e_{m-1}, D_{m-1}, L_{m-1}, M_{m-1}, f_{m-1}, e_{m-2}, D_{m-2}, ..., e_0, D_0, L_0, M_0, f_0)$$

where f_i is the binary function symbol (some $\otimes_{R,g,h}$) occurring at node u_i,

Figure 3: The graph $Rep(t)(\{u, y\})$ with some edges valued by lengths of shortest special $\{u, y\}$-walks.

$M_i = MIN(RightSon(u_i))$ if $e_i = 1$ and $M_i = MIN(LeftSon(u_i))$ if $e_i = 2$ for each $i = 0, ..., m-1$.□

In $J(x)$ we can delete $D_{m-1}, L_{m-1}, D_{m-2}, ..., D_0, L_0$ because the corresponding data can be computed from $e_{m-1}, M_{m-1}, f_{m-1}, e_{m-2}, M_{m-2}, \ f_{m-2}, ..., e_0$, M_0, f_0. Even if we keep this redundant data in $J(x)$ we have the following fact (we omit a detailed description of the encoding of $J(x)$ as a bit sequence) :

Lemma 24 : Each label $J(x)$ has size $O(k^2.ht(t).\log n)$.

These cosntructions and lemmas are easily adapted to directed graphs. In the graph $Rep(t)$, the horizontal edges are directed, and the integers $Min(u, i, j)$ represent directed paths in $val(t \downarrow u)$. Of course, we need not have $Min(u, i, j) = Min(u, j, i)$.

4.3 The labelling algorithm

We now examine the time taken to construct the labels $J(x)$ for all vertices x, hence by Algorithm $\mathcal{A}$ of the labelling scheme.

Lemma 25 : Assuming t being given, the computation of all labels $J(x)$ for all vertices x takes time $O(k^3.|t|)$.

Proof. From a term t as in Definitions 14,20,21 and 23, the construction of $Rep(t)$ is straightforward and takes time $O(k^2.|t|)$. The construction of labels extends what was done in (4.1) for $I(x)$: we only need to know the matrices $MIN(u)$ for all nodes u of t. This can be done by means of a bottom-up traversal of t. We now describe this computation.

Case 1 : If u is a leaf, hence an occurrence of a constant $c_{\mathbf{A}}(u)$ for a set of colours A, then $S(u) = A$, the set of all colours of the unique vertex u of the graph defined by $c_{\mathbf{A}}(u)$. We have $Min(u, i, j) = 0$ for all i, j in $S(u)$.

Case 2 : Otherwise u is an occurrence of some operation $\otimes_{R,g,h}$ with sons u_1 and u_2. We will compute $MIN(u)$ from $MIN(u_1)$ and $MIN(u_2)$.

First we note that $S(u) = g(S(u_1)) \cup h(S(u_2))$.

Let W be a u-downwalk from (u, i) to (u, j). It can be factorized as $W = e_1W_1e_2W_2...W_pe_{p+1}$ where $e_1, e_2, ..., e_{p+1}$ are edges and $W_1, W_2, ..., W_p$ are u_1- and u_2-downwalks of the following forms :

e_1 is a vertical edge $(u, i) \longleftarrow (u_{\alpha_1}, i_{\alpha_1})$

W_1 is a downwalk from $(u_{\alpha_1}, i_{\alpha_1})$ to $(u_{\alpha_1}, j_{\alpha_1})$,

e_2 is a horizontal edge $(u_{\alpha_1}, j_{\alpha_1}) - (u_{\alpha_2}, i_{\alpha_2})$

W_2 is a downwalk from $(u_{\alpha_2}, i_{\alpha_2})$ to $(u_{\alpha_2}, j_{\alpha_2})$,

....

W_p is a downwalk from $(u_{\alpha_p}, i_{\alpha_p})$ to $(u_{\alpha_p}, j_{\alpha_p})$,

e_{p+1} is a vertical edge $(u_{\alpha_p}, j_{\alpha_p}) \longrightarrow (u, j)$,

where $\alpha_1, \alpha_2, ..., \alpha_p \in \{1, 2\}$, $\alpha_{i+1} = 3 - \alpha_i$ for each $i = 1, ..., p-1$, and i_{α_k} and j_{α_k} belong to $S(u_{\alpha_k})$ for each $k = 1, ..., p$.

If W is of minimal length from (u,i) to (u,j), then so is each W_p among those from $(u_{\alpha_p}, i_{\alpha_p})$ to $(u_{\alpha_p}, j_{\alpha_p})$.There lengths are given by the matrices $MIN(u_1)$ and $MIN(u_2)$.

In a factorization $e_1W_1e_2W_2...W_pe_{p+1}$ of W, all edges $e_1, e_2, ..., e_{p+1}$ are from a set determined by the triple (R,g,h). Hence the minimal length of a downwalk W from (u,i) to (u,j) can be obtained by an all-pairs shortest path algorithm in a weighted graph with $O(k)$ vertices. (Each edge has a weigth and the length of a path is the sum of weigths of its edges.) Hence, this computation takes time $O(k^3)$ at each u. We omit routine details. Hence, for all nodes the total computation time is $O(k^3.|t|)$.

The case of directed graphs is very similar. □

Example 8 continued : Figure 4 illustrates the computation at a node u of $MIN(u)$ from $MIN(u_1)$ and $MIN(u_2)$ in a case where $S(u) = S(u_2) = \{1,2,3,4\}$ and $S(u_1) = \{1,2,3\}$. The operation at u is $\otimes_{R,g,h}$ where :

$R = \{(1,4),(2,2),(3,1)\}$,

$g(1) = \{1\}, g(2) = \{2,3\}, g(3) = \{4\}$ and

$h(1) = \{3\}, h(2) = \{4\}, h(3) = \varnothing, h(4) = \{4\}$.

We assume that, for some integers $a, b, ...$ we have :

$Min(u_1,1,2) = a$, $Min(u_1,2,3) = b$ and $Min(u_1,1,3) = \infty$

(the absence of an edge between 1 and 3 below $S(u_1)$ indicates this last value). We also have :

$Min(u_2,1,2) = c$, $Min(u_2,2,3) = d$, $Min(u_2,3,4) = e$,

$Min(u_2,2,4) = f, Min(u_2,1,4) = g, Min(u_2,1,3) = \infty$.

In Figure 4 the arrows to the left from 1 to 1, from 2 to 2 and 3, and from 3 to 4 represent the mapping g (they are vertical edges of $Rep(t)$), and those to the right represent h.

The thick horizontal edges between $S(u_1)$ and $S(u_2)$ represent the pairs in R. The curves marked $a, b, c, ..., f, g$ represent entries of the matrices $MIN(u_1)$ and $MIN(u_2)$ assumed to be known and from which one wants to compute $MIN(u)$. By looking at all possible paths one gets :

$Min(u,1,2) = \min\{a, 2+f, 2+b+g\}$,

$Min(u,1,3) = \min\{a, 2+f, 1+g\}$,

$Min(u,1,4) = 1$,

$Min(u,2,3) = 0$,

$Min(u,2,4) = \min\{1, b\}$,

and $Min(u,3,4) = \min\{1, b, c, g\}$.

For instance, depending on the relative values of a, f and g, a shortest path in $val(t \downarrow u)$ from some vertex coloured 1 to some vertex coloured 3 may be one in $val(t \downarrow u_1)$ from a vertex coloured 1 to a vertex coloured 2 or a path going through $val(t \downarrow u_2)$ via a vertex coloured 4 and a vertex coloured 2 (with a portion of length f), or via a vertex coloured 4 and a vertex coloured 1 (with a portion of length g).□

Figure 4: Computation of $MIN(u)$ from $MIN(u_1)$ and $MIN(u_2)$

4.4 The decoding algorithm

Lemma 26 : One can determine $G_+[Z]$ from the graph $Rep(t)[Z \cup a(Z) \cup n(Z)]$ and the matrices $MIN(w)$ for w such that $(w,i),(w,j)$ belong to $n(Z)$ in time $O(k^3 + k. \mid Z \mid + \mid Z \mid^2)$

Proof. The algorithm is similar to that of Lemma 25 that computes the matrices $MIN(u)$. We assume that $Z \cup a(Z) \cup n(Z)$ is known. We construct the tree t' defined as the union of the paths $Path(x)$ for x that belongs to Z or is a son of a node on $Path(z)$ for some $z \in Z$. The computation will be bottom-up on the tree t' (this tree is a subgraph of t but not a term because some subterms are missing).

The following proof deals with undirected graphs.

For each node u of t' we define two tables $T1(u)$ and $T2(u)$ with values in $\mathbb{N} \cup \{\infty\}$.

Table $T1(u)$ indicates for every two vertices in $Z \cap V_{val(t \downarrow u)}$ the length of a shortest path in $val(t \downarrow u)$ between them.

Table $T2(u)$ indicates for every vertex x in $Z \cap V_{val(t \downarrow u)}$ and every colour i in $S(u)$ the length of a shortest path in $val(t \downarrow u)$ between x and some vertex having colour i (among possibly other colours). In both cases ∞ means "no path". These two tables are empty if u is not on $Path(z)$ for any $z \in Z$.

Case 1 : If u is a leaf all entries of these matrices will be 0.

Case 2 : Otherwise u is an occurrence of $\otimes_{R,g,h}$ with sons u_1 and u_2. We will compute $T1(u)$ from $T1(u_1), T2(u_1), T1(u_2)$ and $T2(u_2)$, and we will compute $T2(u)$ from $T2(u_1)$ and $T2(u_2)$.

We consider the computation of $T1(u)$. For any two vertices x, y in $Z \cap V_{val(t\downarrow u)}$ we do as follows.

Subcase 2.1 : x is in $Z \cap V_{val(t\downarrow u_1)}$ and y is in $Z \cap V_{val(t\downarrow u_2)}$.

A shortest path between x and y in $val(t \downarrow u)$ is represented by a walk W in $Rep(t)$ that can be decomposed as $W = W_1 e_1 W_2 e_2 ... W_p e_p W_{p+1}$ where

W_1 represents in $Rep(t)$ a shortest path in $val(t \downarrow u_1)$ between x and some vertex coloured by i_1, e_1 is a horizontal edge $(u_1, i_1) - (u_2, j_2)$,

W_2 is a downwalk from (u_2, j_2) to (u_2, i_2),

e_2 is $(u_2, i_2) - (u_1, j_3)$,

W_3 is a downwalk from (u_1, j_3) to (u_1, i_3),

....

W_p is a downwalk from $(u_{\alpha_p}, i_{\alpha_p})$ to $(u_{\alpha_p}, j_{\alpha_p})$,

e_p is $(u_1, i_p) - (u_2, j_p)$,

W_{p+1} represents in $Rep(t)$ a shortest path in $val(t \downarrow u_2)$ between some vertex coloured by j_p and y.

In this description $i_1, j_1, ..., i_p, j_p$ are colours in the sets $S(u_1)$ (for p odd) or $S(u_2)$ for p even and p is odd. The lengths of W_1 and W_{p+1} are obtained from $T2(u_1)$ and $T2(u_2)$; those of $W_2, ..., W_p$ are obtained from $MIN(u_1)$ and $MIN(u_2)$. The edges $e_1, e_2, ..., e_p$ are obtained from the relation R.

Subcase 2.2: x and y are in $Z \cap V_{val(t\downarrow u_1)}$.

The length of a shortest path in $val(t \downarrow u_1)$ is obtained from $T1(u_1)$. This length must be compared with those of paths that go through vertices of $val(t \downarrow u_2)$. For knowing their lengths, we use a similar decomposition as in Subcase 2.1 with p even and at least 2. The shortest length of such a path is obtained from $T2(u_1)$, $MIN(u_1)$ and $MIN(u_2)$. We obtain the desired value by taking a minimum over several possible values. See Figure 5 below.

The cases where x is in $Z \cap V_{val(t\downarrow u_2)}$ and y is in $Z \cap V_{val(t\downarrow u_1)}$, and where x and y are in $Z \cap V_{val(t\downarrow u_2)}$ are of course similar.

The time taken to determine $T1(u)$ and $T2(u)$, is $O(k^3 + k. \mid Z \mid + \mid Z \mid^2)$. The final results are in $T1(root_t)$.

For directed graphs, we need $T1(u)$ which will not be symmetric and two tables $T2^+(u)$ and $T2^-(u)$. Table $T2^+(u)$ (resp. $T2^-(u)$) indicates for every vertex x in $Z \cap V_{val(t\downarrow u)}$ and every colour i in $S(u)$ the length of a shortest directed path in $val(t \downarrow u)$ from x to some vertex coloured by i (resp. from some vertex coloured by i to x). The computation time is the same.□

Example 8 continued : Figure 5 extends the example of Figure 4. It indicates the following (finite) values of the tables for $u_1$and u_2 (absent edges stand for entries equal to ∞):

$T1(u_1)[x, y] = \delta$,

$T2(u_1)[x, 1] = \alpha, T2(u_1)[x, 2] = \gamma, T2(u_1)[y, 1] = \varepsilon, T2(u_1)[y, 2] = \beta$,

$T2(u_2)[z, 1] = \mu, T2(u_2)[z, 2] = \nu$.

From these definitions and those of $MIN(u_2)$ that are visible in Figure 5,

Figure 5: Computation of lengths of shortest Z-external paths for obtaining $G_+[Z]$

we have:

$\alpha \leqslant \delta + \varepsilon, \varepsilon \leqslant \delta + \alpha,$

$\beta \leqslant \delta + \gamma, \gamma \leqslant \delta + \beta,$

$a \leqslant \alpha + \gamma, a \leqslant \beta + \varepsilon.$

We also have that $Min(u, 1, 3) \leqslant \varepsilon + T2(u_1)[y, 3]$, hence $T2(u_1)[y, 3] = \infty$ because ε is assumed to be finite.

Analyzing the possible paths gives the following:

$T1(u)[x, y] = \min\{\delta, \alpha + \varepsilon + 2, \beta + \gamma + 2, \alpha + \beta + f + 2, \gamma + \varepsilon + f + 2\}$

$T2(u)[x, 1] = \alpha$

$T2(u)[x, 2] = T2(u)[x, 3] = \gamma$

$T2(u)[x, 4] = \min\{\alpha + 1, \gamma + 1\}$

$T2(u)[y, 1] = \varepsilon$

$T2(u)[y, 2] = T2(u)[y, 3] = \beta$

$T2(u)[y, 4] = \min\{\beta + 1, \varepsilon + 1\}$

and so on.

We can now prove the crucial Proposition 13.

Proof of Proposition 13. Let G be an undirected graph defined by a term t. Assume that J has been constructed by Lemma 25. Lemmas 22 and 26 show how to construct $G_+[Z]$. This gives a global time bound of $O(|Z|^2.k^3.ht(t))$.

We now review the modifications to be done for handling directed graphs. In the construction of the graph representation $Rep(t)$, an horizontal edge is

directed $(u_1, i) \longrightarrow (u_2, j)$ if u_1, u_2 are respectively the left and the right son of a node u that is an occurrence of an operation $\otimes_{R,g,h}$ such that $(i, j, +) \in R$. It is directed in the other direction if $(i, j, -) \in R$.

Lemma 17 is modified as follows: for distinct vertices u, v we have $u \longrightarrow v$ in G if and only if there is an elementary path of the form $u \longrightarrow^* (w, i) \longrightarrow (w', j) \longleftarrow^* v$ in $Rep(t)$ for some w, w', i, j. The modified versions of Lemmas 17,18,19,25 and 26 follow easily. For this, we use *directed u-downwalks* from (u, i) to (u, j), of the form : $(u, i) \longleftarrow^* z \longrightarrow^* \cdots \longleftarrow^* z' \longrightarrow^* (u, j)$, where all horizontal edges are from left to right, all other conditions being as for u-downwalks. We let $Min(u, i, j)$ be the smallest length of a directed u-downwalk from (u, i) to (u, j), or ∞ if no such downwalk exists.

The matrices $MIN(u)$ are no longer symmetric. The graph $G_+[Z]$ is built with integer valued *directed edges*. This completes the proof of Proposition 13.□

Theorem 9 and Proposition 13 yield the following main theorem.

Theorem 27. For a directed or undirected graph G of m-clique-width at most k on n vertices, one can assign to vertices labels $J(x)$ of size $O(k^2.\log^2 n)$ such that from the family $(J(x))_{x\in Z}$ for any set $Z \subseteq V$, one can determine the graph $G_+[Z]$ in time $O(|Z|^2.k^3.\log n)$. The length of a shortest (X, F)-constrained path such that $X \cup ends(F) \subseteq Z$ can be determined in time $O(|Z - X|^2)$ from $G_+[Z]$ and in time $O(|Z|^2.k^3.\log n)$ from $(J(x))_{x\in Z}$.

Proof. This follows from Proposition 13 and Lemma 1 by using Dijkstra's shortest path algorithm.□

Example 8 continued : From Figure 3 one can see that u and y are adjacent but that, if the edge $u - y$ is broken, they are at distance 4. Figure 6 shows that vertices v and x are at distance 2 in $val(t)$, but if the vertex w is forbidden they are at distance 3, by a path going through y. The two edges with their endpoints in the ellipse are no longer usable if w is forbidden.

Handling edge additions.

If for fixed Z , in addition to (X, F) such that $X \cup ends(F) \subseteq Z$, we are given a set H of pairs of vertices of Z representing new directed or undirected edges, one may ask about the length of a shortest path from x to y (belonging to $Z - X$) in $((G - F) \cup H)\backslash X$. We call this *handling constrained paths with edge additions*.

Corollary 28 : With the labelling of Theorem 27, one can handle constrained paths with edge additions, with same time for answering queries.

For the simpler problem of checking connectivity (possibly with edge additions) we have a more compact labelling scheme.

Corollary 29: For a directed or undirected graph G of m-clique-width at most k on n vertices, one can assign to vertices labels $C(x)$ of size $O(k^2.\log n)$ such that from the family $(C(x))_{x\in Z}$ for any set $Z \subseteq V$, one can determine the (X, F)-constrained connectivity (or the (X, F)-constrained directed connectivity in case of a directed graph) possibly with edge additions.

Figure 6: The graph $Rep(t)_+(\{v, w, x, y\})$.

Proof. In the matrices $MIN(u)$ we only have to store values "∞" or "not ∞". This gives two results: directed connectivity for directed graphs, i.e., the existence of a directed (X, F)-constrained path from x to y, and connectivity for undirected graphs, i.e., the existence of an (X, F)-constrained path between x to y.$\square$

4.5 Overview of algorithms

Let us review the steps needed to apply these results. We first consider Algorithm $\mathcal{A}$ that constructs labels.

Step 1: First, we need a tree-decomposition or an mcwd- term of the given graph, of width at most k. This can be done in linear time for obtaining a tree-decomposition (for details, see [1]). The problem of determining the m-clique-width of a graph and the corresponding optimal term is likely to be NP-hard because the corresponding one for clique-width is NP-complete [9]. The cubic algorithm given by Oum [13] that constructs non-optimal clique-width terms for undirected graphs can be used.

The obtained tree-decomposition or clique-width term can easily be transformed into an mcwd- term of width k or $k+3$ or $2k+4$, depending on the case, by Proposition 5.

Step 2 : The term must be turned into a 3-balanced one of width $O(k)$ in time $O(n.\log n)$, denoted by t.

Step 3 : We then construct the graph $Rep(t)$, which can be done in time $O(k^2.\mid t \mid) = O(k^2.n)$.

Step 4 : The next step consists in computing the matrices $MIN(u)$. This can be done bottom-up in the term t by Lemma 25. Since at each step we need time $O(k^3)$, we need in total time $O(k^3.n)$.

Step 5 : The final step consisting in building the labels $J(x)$ for all vertices takes time $O(k^3.ht(t)) = O(k^3.\log n + k^2.n.\log n)$.

The decoding algorithm $\mathcal{B}$ uses two steps :

Step 1 : Construction of $G_+[Z]$ in time $O(k^3 + k.\mid Z \mid + \mid Z \mid^2)$.

Step 2 : Answers to queries in time $O(|Z - X|^2)$. This time does not depend on k.

These constructions apply to directed graphs in a straightforward manner except the cubic algorithm of [13] which is designed for undirected graphs. However there exists a bijective encoding of directed graphs G as bipartite undirected graphs $B(G)$ (defined in [3]). There exist strictly increasing functions f and g such that for every directed graph G :

$$f(cwd(B(G))) \leqslant cwd(G) \leqslant g(cwd(B(G)).$$

Hence the cubic algorithm for undirected graphs can be used for directed graphs.

Another extension can be done for shortest constrained paths in directed graphs with edges having nonnegative integer lengths. However, all edges created by a single pair of colours (a, b) in R in an operation $\otimes_{R,g,h}$ must have the same length. Otherwise the notion of graph representation of a term, where one horizontal edge represents all edges created from one pair (a, b) by an operation $\otimes_{R,g,h}$ cannot be used. The above methods and results are easy to adapt to this extension.

4.6 A compact routing scheme

We now describe how to use a modification of the labelling J to build a compact routing scheme.

The construction of J is based on matrices that give for each position u in a term t the length of a shortest u-downwalk in $Rep(t)$ from (u, i) to (u, j). Storing the sequence of vertices of the corresponding path in $G = val(t)$ uses space $O(n \log n)$ instead of $O(\log n)$ for each entry (assuming there are n vertices numbered from 1 to n, so that a path of length p uses space $(p+1).\lceil \log n \rceil$). The corresponding labels $J'(x)$ have size $O(k^2.n.\log^2 n)$ for each x. This labelling yields for every pair of vertices x, y a shortest (X, F)-constrained path and not only its length. This path is obtained by piecing together some edges (the edges e_i of the proof of Lemma 26) and some of the paths stored in the labels $J'(z)$ for vertices z in the set $\{x, y\} \cup X \cup ends(F)$.

We now give a more economical construction. For having a *compact routing scheme* (as opposed to a distance labelling scheme), it suffices to be able to construct the path in an incremental manner, by finding the next hop at each node, then forwarding the relevant data to that node. Here is a construction permitting this.

We only store at each entry (i, i) of the matrix $MIN(u)$ one vertex x such that $(u, i) \in Rep(t)(x)$ instead of the value $0 = Min(u, i, i)$. If $Min(u, i, j) = 0$, we store the same vertex at entries (i, i) and (j, j). This uses slightly more space than for $MIN(u)$ as in Section 4.2 but still $O(k^2.\log n)$ and the corresponding labels $J''(x)$ have size $O(k^2.\log^2 n)$ for all x.

From the labels $J''(x)$ for all x in Z, one can build an edge-labelled graph $G_{++}[Z]$ defined by adding new labels to those of $G_+[Z]$. For each edge between x and y in $G_+[Z]$ having weight 2, we add to the label of this edge the intermediate vertex z of one of the shortest paths it represents. For each edge between x and y in $G_+[Z]$ having weight at least 3, we add two intermediate vertices z and z' of one of the shortest paths it represents, such that z is next to x and z' is next to y. It is easy to adapt the algorithm of Lemma 26 so as to construct $G_{++}[Z]$ from $(J''(x))_{x \in Z}$. And from $G_{++}[Z]$ for $Z \supseteq \{x, y\} \cup X \cup ends(F)$, one can determine, not only the length of shortest (X, F)-constrained path from x to y , but the vertex z following x on such a path. This vertex may not be in Z. If it is not, and to continue the construction one must compute $G_{++}[Z \cup \{z\}]$ in order to determine the vertex following z is a shortest path from x to y.

These definitions and observations yield the following result.

Theorem 30. Let G be a graph of m-clique-width at most k, each vertex u of which has an associated set of vertices $Forb(u)$ of size at most r. This graph has a compact forbidden-set routing scheme using routing tables of size $O(r.k^2.\log^2 n)$ and message headers of size $O(r.k^2.\log^2 n)$ that permits to route on shortest paths in $G \setminus Forb(x)$ from every vertex x.

Proof. Let $G = (V, E)$ be given by a balanced mcwd- term of width at most $2k$ (or $3k$ if G is directed), let $J''(x)$ be constructed as explained above. Let a set $Forb(u) \subseteq V$ such that $|Forb(u)| \leqslant r$ be defined for each vertex u. We store at each u the labels $J"(w)$ for all w in $Forb(u)$. The algorithm intended to route a message from x to y in $G \setminus Forb(x)$ works as follows.

By using $J"(x)$, $J"(y)$ and $\{J"(w) : w \in Forb(x)\}$, it determines a vertex z_1 on some shortest path $x - z_1 - ... - y$ in $G \setminus Forb(x)$. The message together with $J"(y)$ and the set $\{J"(w) : w \in Forb(x)\}$ is sent to z_1. Then some z_2 on a shortest path $z_1 - z_2 - ... - y$ in $G \setminus Forb(x)$ is determined by using $J"(z_1)$, $J"(y)$ and the set $\{J"(w) : w \in Forb(x)\}$. The message with the same set of labels is sent to z_2, and the procedure is repeated until y is reached. Since exact distances are computed and since the set of forbidden vertices is not changed on the way, the message gets closer to y at each step and reaches it. □

This method constructs a path from x to y in $G \setminus Forb(x)$ that may go through some z and then through some u such that $u \in Forb(z)$. If at each step we increase the set of forbidden vertices in order to prevent such a situation, we may build a path that is not shortest or we may find no path at all whereas there exists one.

5 Open problems

A major open problem is to get good bounds for constrained distance labelling on planar graphs. Since they have balanced separators of size $\Theta(\sqrt{n})$ they have tree-width $O(\sqrt{n})$, hence our results can be applied with $k = O(\sqrt{n})$. This gives a labelling scheme with labels of size $O(n \log^2 n)$, but we think that it is possible to do much better.

Recently, Courcelle et al. [4] presented an $O(\log n)$-bit labelling scheme for constrained *connectivity* in planar graphs, but the problem of constrained distance labelling is still open.

Acknowledgements : We thank Cyril Gavoille and Mamadou Kanté for helpful discussions and the referees for their comments which triggered many improvements.

References

[1] H. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 226–234, New York, NY, USA, 1993. ACM Press.

[2] B. Courcelle, J.A. Makowsky, and U. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Systems*, 33:125–150, 2000.

[3] B. Courcelle. The monadic second-order logic of graphs xv : On a conjecture by d. seese. In *Journal of Applied Logic 4*, pages 79–114, 2006.

[4] B. Courcelle, C. Gavoille, M. Kanté, and A. Twigg. Connectivity check in 3-connected planar graphs with obstacles. In *Electronic Notes in Discrete Mathematics*, volum 31 (2008), 151–155.

[5] B. Courcelle and M. Kanté. Graph operations characterizing rank-width and balanced graph expressions. In D. Kratsch A. Brandstädt and H. Müller, editors, *Proccedings of the 33rd International Workshop on Graphs (WG07)*, volume 4769 of *LNCS*, pages 66–75. Springer, June 2007.

[6] B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101(1-3):77–114, 2000.

[7] B. Courcelle and A. Twigg. Compact forbidden-set routing. In W. Thomas and P. Weil, editors, *STACS*, volume 4393 of *Lecture Notes in Computer Science*, pages 37–48. Springer, 2007.

[8] B. Courcelle and R. Vanicat. Query efficient implementation of graphs of bounded clique-width. *Discrete Applied Mathematics*, 131(1):129–150, 2003.

[9] M. Fellows, F. Rosamond, U. Rotics, and S. Szeider. Clique-width minimization is NP-hard. In *STOC 2006: Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, New York, NY, USA, 2006, pages 354–362. ACM Press.

[10] C. Gavoille and D. Peleg. Compact and localized distributed data structures. *Distrib. Comput.*, 16(2-3):111–120, 2003.

[11] C. Gavoille, D. Peleg, S. Perennes, and R. Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85–112, 2004.

[12] A. Gupta, A. Kumar, and M. Thorup. Tree based mpls routing. In *SPAA '03: Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, pages 193–199, New York, NY, USA, 2003. ACM Press.

[13] S. il Oum. Approximating rank-width and clique-width quickly. In Dieter Kratsch, editor, *Proc. 31st International Workshop on Graphs (WG 2005)*, volume 3787 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2005.

[14] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. Discret. Math.*, 5(4):596–603, 1992.

[15] E. Wanke. k-nlc graphs and polynomial algorithms. *Discrete Applied Mathematics*, 54(2-3):251–266, 1994.

6 Appendix : Balanced terms defining graphs

We prove Theorem 9 that we restate for the reader's convenience.

Theorem 9. Every undirected (resp. directed) graph of clique-width or m-clique-width k with n vertices ($n > 1$) is the value of an m-cwd term t of width at most $2k$ (resp. $3k$) and of height at most $3.(\log(n-1)+1)$. The time taken to build t from a given term s of width k is $O(n.\log n)$, where n is the size of s.

A similar result for graphs of bounded tree-width has been proved by Bodlaender [1] : every graph with n vertices of tree-width k has a tree-decomposition of width $3k+2$ with underlying tree of height at most $2\log_{5/4}(2n)$.

We first prove that every m-cwd term can be transformed into a balanced term using new binary operations that express substitutions of terms for variables with unique occurrences. These new operations are not among the operations defining m-clique-width but they can be simulated by m-clique-width operations using more labels than those of the original term.

6.1 Contexts and special terms

Let F is a set of function symbols, and C be a set of constants. A *context* is a term in $T(F, C \cup \{u\})$ having a single occurrence of the variable u (nullary symbol). We denote by $Ctxt(F,C)$ the set of contexts, and by **Id** (for identity) the particular context u. We define two binary operations $\circ$ and $\bullet$ on terms and contexts for which we use infix notation :

$s \circ s' = s[s'/u]$, belongs to $Ctxt(F,C)$ for s, s' in $Ctxt(F,C)$,

$s \bullet t = s[t/u]$, belongs to $T(F,C)$ for s in $Ctxt(F,C)$, t in $T(F,C)$, where $s[w/u]$ denotes the substitution in s of a term or context w for u at its unique occurrence.

Clearly $s \circ \mathbf{Id} = \mathbf{Id} \circ s = s$; $\mathbf{Id} \bullet t = t$.The operation $\circ$ is associative and we have $s \bullet (s' \bullet t) = (s \circ s') \bullet t$.

We will consider terms in $T(F \cup \{\circ, \bullet\}, C \cup \{\mathbf{Id}\})$ that evaluate to terms or contexts according to the above definitions. The evaluation of these terms consists in eliminating $\circ$ and $\bullet$ by performing the substitutions they stand for.

Example 31 :

The term $f(\mathbf{Id}, b) \circ (g(a, \mathbf{Id}) \circ f(\mathbf{Id}, c))$ evaluates to the context $f(g(a, f(u, c)), b)$. The term $f(\mathbf{Id}, b) \circ (g(a, \mathbf{Id}) \bullet f(d, c))$ evaluates to the term $f(g(a, f(d, c)), b)$.

From now on we only consider sets F of binary operation symbols.

Definition 32 : *The sets of special terms and contexts* $SPE_t(F, C)$ *and* $SPE_c(F, C)$.

We let S_c and S_t be the least subsets of $T(F \cup \{\circ, \bullet\}, C \cup \{\mathbf{Id}\})$ such that:

$S_t = S_c \bullet S_t \ \cup \ f(S_t, S_t) \ \cup \ b \cup ...$

$S_c = S_c \circ S_c \ \cup f(S_t, S_c)... \ \cup \ f(S_c, S_t) \ ... \cup \ f(S_t, \mathbf{Id}) \ ... \cup f(\mathbf{Id}, S_t)...$

where the unions extend to all f in F and b in C.

We denote these sets by $SPE_t(F, C)$ and $SPE_c(F, C)$ if we need to specify F and C. Note that $\mathbf{Id} \notin S_t \cup S_c$.

Every term t in $SPE_t(F, C)$ evaluates into a term $Eval(t)$ in $T(F, C)$ and every term c in $SPE_c(F, C)$ evaluates into a context $Eval(c)$ in $Ctxt(F, C) - \{\mathbf{Id}\}$. The evaluation rules are as follows:

$$\begin{aligned} Eval(c \circ c') &= Eval(c)[Eval(c')/u], \\ Eval(c \bullet t) &= Eval(c)[Eval(t)/u], \\ Eval(\mathbf{Id}) &= u, \\ Eval(b) &= b, \end{aligned}$$

and

$$Eval(f(w, w')) = f(Eval(w), Eval(w'))$$

for terms or contexts c, c', t, w, w', functions f and constants b.

For a term t in $SPE_t(F, C) \cup SPE_c(F, C)$ we denote by $|t|_{FC}$ the number of occurrences of symbols from $F \cup C$, by $|t|_0$ the number of occurrences of $\circ$ and $\bullet$, and, by $|t|_{\mathbf{Id}}$ the number of occurrences of $\mathbf{Id}$. Since F is a set of binary function symbols, each such term has an odd size $|t|$ defined as $|t|_{FC} + |t|_0 + |t|_{\mathbf{Id}}$ and it is clear from the recursive equations defining special terms that $|t|_{\mathbf{Id}} = |t|_0$ if $t \in SPE_t(F, C)$ and $|c|_{\mathbf{Id}} = |c|_0 + 1$ if $c \in SPE_c(F, C)$.

The following proposition shows how a term or a context can be split into two or three terms or contexts of less than half size.

Proposition 33 [Lemmas 1,2 in [8]]:

1. Every term $t \in T(F, C)$ of size $n = 2p + 1, \ p \geqslant 1$ can be written $t = c_1 \bullet f(t_1, t_2)$ where $c_1 \in Ctxt(F, C)$, $t_1, t_2 \in T(F, C)$, $|c_1| \leqslant p$, c_1 is of maximal size with this property, and then $|t_i| \leqslant p + 1$ for each $i = 1, 2$.

2. Every context $c \in Ctxt(F, C)$ of size $n = 2p + 1, \ p \geqslant 1$ can be written $c = c_1 \circ f(c_2, t_1)$ or $c = c_1 \circ f(t_1, c_2)$ for $c_1, c_2 \in Ctxt(F, C)$, $t_1 \in T(F, C)$ and $|c_1| \leqslant p$, c_1 is of maximal size with this property, and then $|c_2| \leqslant p + 1, |t_1| \leqslant 2p - 1$.

Remark : Let us look at some particular cases. Let $t = f(s_1, s_2)$ with $p = (|t| - 1)/2 = (|s_1| + |s_2|)/2$.

If $|s_1| - 2 \leqslant |s_2| \leqslant |s_1| + 2$ then in Case 1 of Proposition 33, we must take $c_1 = u$.

If $|s_1| = |s_2|+2$ then the "larger context" $c'_1 = f(u, s_2)$ has size $2+|s_2| \geqslant p+1$ since $p + 1 = (|s_1| + |s_2|)/2 + 1 = |s_2| + 2$, hence c_1 is maximal of size $\leqslant p$. If $|s_1| = |s_2|$ or if $|s_2| = |s_1| + 2$ the same argument works.

Similarly if $c = f(c', s_2)$ and $|c'| \leqslant |s_2| + 2$ (in particular if $c' = u$) we must take $c_1 = u$ to satisfy (2). Taking the "larger context" $c'_1 = f(u, s_2)$ would necessitate $|c'_1| \leqslant p$ that is $2 + |s_2| \leqslant (|c'| + |s_2|)/2$, i.e., $|c'| \geqslant |s_2| + 4$. □

A more careful proof than the one of [8] gives the following result.

Proposition 34 : For every term t in $T(F, C) - C$ one can construct a term t^b in $SPE_t(F, C)$ such that $|t^b|_{FC} = |t|_{FC} = |t|$, $Eval(t^b) = t$, $ht(t^b) \leqslant 3.\log(|t|-1)$ and $|t^b| \leqslant 2.|t|-1$. This term can be constructed in time $O(n.\log n)$ where $n = |t|$.

Proof. We will use an induction to construct t^b for each $t \in T(F, C)$ and to construct also a context c^b in $SPE_c(F, C)$ such that $Eval(c^b) = c$ for each $c \in Ctxt(F, C)$. The construction will ensure the following properties (note that $|c| = |c|_{FC} + 1$) :

$|c^b|_{FC} = |c|_{FC}$, $ht(c^b) \leqslant 3.\log(|c| - 1) + 2$ and $|c^b| \leqslant 2.|c| - 1$.

Case 1: Let $t \in T(F, C)$ have size $|t| = 2p + 1$.

Subcase 1.1 : If $|t| = 3$ we let $t^b = t$, then $ht(t^b) = 2 < 3.\log(|t| - 1)$ and $|t^b| = |t| \leqslant 2.|t| - 1$.

Subcase 1.2: If $|t| = 2p + 1 > 3$. We use Proposition 33 and write $t = c_1 \bullet f(t_1, t_2)$.

Subcase 1.2.1: $c_1 = u$. This means that $||t_1| - |t_2|| \leqslant 2$: assume on the contrary that $|t_1| \geqslant |t_2| + 4$, then $|f(u, t_2)| = 2 + |t_2| \leqslant p = (|t_1| + |t_2|)/2$ and c_1 is not of maximal size such that case 1 of Proposition 33 holds, because it can be replaced by a "larger context", e.g., $f(u, t_2)$. In this case we let $t^b = f(t^b_1, t^b_2)$. We have $|t_1| = 2p_1 + 1$, $|t_2| = 2p_2 + 1$, $|p_1 - p_2| \leqslant 1$. We first assume $t_1, t_2 \notin C$. By inductive hypothesis :

$ht(t^b_i) \leqslant 3.\log(2p_i) = 3.\log(p_i) + 3$.

We note that $|t| - 1 = 2.p_1 + 2.p_2 + 2$. Since $|p_1 - p_2| \leqslant 1$ we have $2.p_i \leqslant (|t| - 1)/2$ for each $i = 1, 2$. Hence:

$$\begin{aligned} 1 + ht(t^b_i) &\leqslant 1 + 3.\log(2p_i) \\ &\leqslant 1 + 3.\log((|t| - 1)/2) \\ &= -2 + 3.\log(|t| - 1) \\ &< 3.\log(|t| - 1) \end{aligned}$$

and $ht(t^b) = \max\{1 + ht(t^b_1), 1 + ht(t^b_2)\} < 3.\log(|t| - 1)$.

The size of t^b is $|t^b| = |t^b_1| + |t^b_2| + 1 \leqslant 2.(|t_1| + |t_2|) - 1 < 2.|t| - 1$ by using induction.

If $t_1 \in T(F, C)$, then $|t_1| = 1$ which implies $|t_2| = 3$, $|t| = 5$ and $t^b = t$,

$ht(t) = 3 \leqslant 3\log(4) = 6$. The case $t_2 \in T(F,C)$ is similar.

Subcase 1.2.2: $c_1 \neq u$. We let $t^b = c_1^b \bullet f(t_1^b, t_2^b)$. We have $|c_1| \leqslant p$, $|t_i| \leqslant p+1$. We must prove that $1 + ht(c_1^b) \leqslant 3.\log(2p)$ and that

$2 + ht(t_i^b) \leqslant 3.\log(2p)$ for $i = 1, 2$.

By induction : $ht(c_1^b) \leqslant 3.\log(p-1) + 2 \leqslant 3.\log(p) + 2$.

Hence : $1 + ht(c_1^b) \leqslant 3 + 3.\log(p) = 3.\log(2p) = 3.\log(|t| - 1)$.

We have also : $2 + ht(t_i^b) \leqslant 2 + 3.\log(p) < 3.\log(2p)$.

This proves the desired assertion.

Case 2: We now consider the case of $c \in Ctxt(F,C)$ of size $n = 2p+1$.

Subcase 2.1: If $n = 3$ then $c^b = c$ and the result holds as in Subcase 1.1.

Subcase 2.2: We consider the case $|c| = 2p+1 > 3$. We write $c = c_1 \circ f(c_2, t_1)$ or $c = c_1 \circ f(t_1, c_2)$ with c_1 of maximal size with $|c_1| \leqslant p$. We only consider the first case.

Subcase 2.2.1: $c_1 = u$. This means that $c = f(c_2, t_1)$, $|c_2| \leqslant |t_1| + 2$, because if $|c_2| \geqslant |t_1| + 4$ then $c_1 = u$ could be replaced by a "larger context", e.g., $f(u, t_1)$.

We take $c^b = f(c_2^b, t_1^b)$. We have $|c_2| = 2p_2 + 1$, $|t_1| = 2p_1 + 1$, $p_2 \leqslant p_1 + 1$. The proof is similar to that of Subcase (1.2.1). We must prove that :

$1 + ht(t_1^b) \leqslant 3\log(2p) + 2$ and $1 + ht(c_2^b) \leqslant 3\log(2p) + 2$.

We have

$1 + ht(t_1^b) \leqslant 1 + 3\log(|t_1| - 1) \leqslant 1 + 3\log(|c| - 1) < 3\log(|c| - 1) + 2$.

We also have $1 + ht(c_2^b) \leqslant 1 + 3\log(2p_2) + 2$. We have

$4.p_2 \leqslant 2.p_2 + 2.(p_1 + 1) = |c| - 1$.

Hence :

$1 + ht(c_2^b) \leqslant 3 + 3\log((|c| - 1)/2) = 3\log(|c| - 1) < 3\log(|c| - 1) + 2$.

We also have $|c^b| = |t_1^b| + |c_2^b| + 1 \leqslant 2.(|t_1| + |c_2|) - 1 < 2.|c| - 1$ by using induction, as in Subcase 1.2.2.

Subcase 2.2.2: $c_1 \neq u$. Then we let $c^b = c_1^b \circ f(c_2^b, t_1^b)$. We have $|c_1| \leqslant p$, $|c_2| \leqslant p+1$ and $|t_1| \leqslant 2p - 1$. We must prove :

$1 + ht(c_1^b) \leqslant 3\log(2p) + 2$ (7.1)

$2 + ht(c_2^b) \leqslant 3\log(2p) + 2$ (7.2)

$2 + ht(t_1^b) \leqslant 3\log(2p) + 2$ (7.3).

For (7.1) we have using induction

$1 + ht(c_1^b) \leqslant 1 + 3\log(p-1) + 2 < 3 + 3\log(p) = 3\log(2p)$.

For (7.2) we have using induction

$2 + ht(c_2^b) \leqslant 2 + 3\log(p) + 2 = 3\log(2p) + 1$.

For (7.3) we have

$2 + ht(t_1^b) \leqslant 2 + 3\log(2p-2) < 3\log(2p) + 2$.

We have $|c^b| = |t_1^b| + |c_1^b| + |c_2^b| + 2 \leqslant 2.(|t_1| + |c_1| + |c_2|) - 1 < 2.|c| - 1$ by using induction.

In all these cases and subcases, we get $|t^b|_{FC} = |t|_{FC}$ and $|c^b|_{FC} = |c|_{FC}$ by induction. □

The decomposition of Proposition 33 can be found in time $O(|t|)$. For a term t the recursive decomposition procedure (cf. Subcase 1.2) is called for c_1, t_1, t_2 each of size at most $(|t|+1)/2$. For a context t the decomposition procedure (cf. Subcase 2) is called for c_1, c_2 , each of size at most $(|t|+1)/2$ and t_2 of size at most $|t|$. The decomposition of t_2 calls the procedure for at most three terms and contexts of size at most $(|t|+1)/2$. Hence, the procedure applied to a context t uses recursive calls to at most five terms and contexts of size at most $(|t|+1)/2$. So the total time is $O(|t|.\log(|t|)$. □

Example 35 : A term of the form $f(a, f(a, f(a, ..., f(a, b)))...)))$ with 2^n occurrences of f, hence of height 2^n+1 and size $2^{n+1}+1$ is $Eval(t)$ for a term t in S_t of height $n+2$ and size $2^{n+2}-1$. For $n=3$, we obtain the following term

$t = [(f(a, \mathbf{Id}) \circ f(a, \mathbf{Id})) \circ (f(a, \mathbf{Id}) \circ f(a, \mathbf{Id}))]\bullet$
$[(f(a, \mathbf{Id}) \circ f(a, \mathbf{Id})) \bullet (f(a, \mathbf{Id}) \bullet f(a, b))]$.

6.2 Balanced m-cwd terms

We now prove Theorem 9 by using Propositions 33 and 34 for m-cwd terms. We will write F_k and C_k instead of $F_{[k]}$ and $C_{[k]}$. The operations and constants of these sets transform and define multicoloured graphs with colours in $[k]$. We first consider the case of undirected graphs.

The idea of the proof is as follows. Let G be a graph defined by a term in $T(F_k, C_k)$; by Proposition 34 there exists a 3-balanced special term evaluating to G but this term uses the operations $\circ$ and $\bullet$ which are *not* allowed in the definition of m-clique-width. We will eliminate them at the cost of using k more colours : a special term $s \bullet t$ will be rewritten into an equivalent term $G_s \otimes_{R,g,h} t$. That is, the action of s on arbitrary terms t is simulated by a graph G_s "representing s" and an appropriate operation $\otimes_{R,g,h}$ depending on s.

The transformation of s into G_s is *compositional*, that is: $G_{s \circ s'} = G_s \otimes_{R,g,h} G_{s'}$ for some $\otimes_{R,g,h}$ depending on s and s'. We can thus eliminate $\circ$ and $\bullet$ without increasing the size and height of the given term. The graphs G_s have colours in the set $[k] \cup [k]'$ where $[k]' := \{i' \mid i \in [k]\}$. We can of course replace i' by $k+i$ so that we obtain terms in $T(F_{2k}, C_{2k})$.

Proof. The sets of special terms S_t and contexts S_c are here $SPE_t(F_k, C_k)$ and $SPE_c(F_k, C_k)$. For a context defined by a special term s, we denote by $\widetilde{s}$ the associated unary graph operation that transforms H into $G = \widetilde{s}(H)$ (that is, $G = val(s \bullet t)$ if H is the value of a term t). Our aim is to express G as $G_s \otimes_{R,g,h} H$ where G_s is a multicoloured graph over the set of colours $L := \{1, ..., k, 1', ..., k'\}$ and R, g, h are chosen adequately. When we write $G = \widetilde{s}(H)$ we will assume, without loss of generality that the vertices of H are vertices of G, because when one builds a graph $K = M \otimes_{R,g,h} N$ from graphs M and N, one can choose to keep any of M or N untouched and to make a disjoint copy of the other. Hence,

in the construction of $G = \widetilde{s}(H)$ the graph H need not be copied. However, $\widetilde{s}$ may add new vertices to H.

We let $\mathbf{I}$ be the graph $c_{\mathbf{1}}(w_1) \oplus ... \oplus c_{\mathbf{k}}(w_k)$ with vertex set $\{w_1, ..., w_k\}$ and we will use the graph $\widetilde{s}(\mathbf{I})$; this graph contains $\widetilde{s}(\varnothing)$ as an induced subgraph. The following fact is clear from the definitions.

Claim 1. A vertex y of $\widetilde{s}(\varnothing)$ is linked to a vertex x of H in $\widetilde{s}(H)$ if and only if y is linked to w_i in $\widetilde{s}(\mathbf{I})$ for some i in $\delta_H(x)$ (i.e., i is one of the colours of x in H). □

The graph $G = \widetilde{s}(H)$ is obtained from H as follows:

1. Add to H the vertices and edges of $\widetilde{s}(\varnothing)$, all created by the operations and constants of s independently of H.

2. Add edges between the vertices x of H and these new vertices y, on the basis of $\delta_H(x)$. These edges are defined by the binary operations of s.

3. Recolour the vertices of H according to the operations of s. This recolouring is defined by the mapping $h_s : [k] \longrightarrow \mathcal{P}([k])$ such that $h_s(i)$ is the set of colours of w_i in $\widetilde{s}(\mathbf{I})$.

The graph G_s representing the context s is defined as the graph $\widetilde{s}(\varnothing)$ modified as follows : we add to the list of colours of each vertex y the colours i' such that $y - w_i$ in $\widetilde{s}(\mathbf{I})$. (Recall that $u - v$ indicates that u and v are adjacent vertices.) Hence G_s has multiple colours in L. In the particular case where $s = \mathbf{Id}$, this gives $G_s = \varnothing$, $h_s(i) = \{i\}$ for every i in $[k]$.The following is clear from this construction and Claim 1.

Claim 2. For every graph H multicoloured in $[k]$, we have $\widetilde{s}(H) = G_s \otimes_{R,g,h_s} H$ where $R = \{(i', i) \mid i \in [k]\}$ and $g(i) = \{i\}$,$g(i') = \varnothing$ for i in $[k]$. □

The relevant information associated with a context s is the pair (G_s, h_s). Contexts are defined inductively; we now show that this information is also computable inductively.

Claim 3. For any two contexts s, s', we have

1. $h_{s \circ s'} = h_s \circ h_{s'}$ and;

2. $G_{s \circ s'} = G_s \otimes_{R,g,h} G_{s'}$, where $R = \{(i', i) \mid i \in [k]\}$ and $g(i) = \{i\}$, $g(i') = \{j' \mid i \in h_{s'}(j)\}$, $h(i) = h_s(i)$, $h(i') = \{i'\}$, for all i in $[k]$.

Proof. The first condition is clear from the definitions. We now prove the second condition. The particular cases where s or s' is $\mathbf{Id}$ can be checked directly.

Otherwise, by using Claim 2 we have $\widetilde{s \circ s'}(\varnothing) = \widetilde{s}(\widetilde{s'}(\varnothing)) = G_s \otimes_{R,m,h_s} \widetilde{s'}(\varnothing)$, $m(i) = \{i\}$,$m(i') = \varnothing$ for i in $[k]$ and $G_{s \circ s'}$ is obtained by adding new colours to this graph.

We first compare the vertices and edges of $G_s \otimes_{R,m,h_s} \widetilde{s'}(\varnothing)$ and $G_s \otimes_{R,g,h} G_{s'}$. From the definitions, the vertices are the same.

Let $x - y$ be an edge of $G_s \otimes_{R,m,h_s} \widetilde{s'}(\varnothing)$. If x and y are both in G_s or both in $\widetilde{s'}(\varnothing)$ then $x - y$ is also an edge of $G_s \otimes_{R,g,h} G_{s'}$ (because $G_{s'}$ is $\widetilde{s'}(\varnothing)$ with

some new colours added).

If x is in G_s and y is in $\widetilde{s'}(\varnothing)$ then x has colour i' and y has colour i for some i, but y has also colour i in $G_{s'}$ hence $x-y$ is also an edge in $G_s \otimes_{R,g,h} G_{s'}$.The argument is the same in the other direction. Hence $G_s \otimes_{R,m,h_s} \widetilde{s'}(\varnothing)$ and $G_s \otimes_{R,g,h} G_{s'}$ have the same vertices and edges. We now compare the colours of a vertex x in these two graphs.

Case 1 : x is in G_s .

Its colours belonging to $[k]$ are as in $\widetilde{s}(\varnothing)$ and are not modified either by m or by g. They are the same in the two graphs we compare.

The vertex x has colour j' in $G_{s\circ s'}$ if and only if $x-w_j$ in $\widetilde{s \circ s'}(\mathbf{I}) = \widetilde{s}(\widetilde{s'}(\mathbf{I})$ if and only if x has colour i' in G_s for some i in $h_{s'}(j)$. Hence, the set of colours j' of x in $G_{s\circ s'}$ is the union of the sets $g(i')$ for i' colour of x in G_s.

From the definitions of the mappings m and g (note how g depends on s'), we get that x has the same colours in $G_{s\circ s'}$ and in $G_s \otimes_{R,g,h} G_{s'}$.

Case 2 : x is in $G_{s'}$.

The vertex x has colour i in $G_{s\circ s'}$ if and only if $i \in h_s(j)$ and x has colour j in $\widetilde{s'}(\varnothing)$ equivalently colour j in $G_{s'}$. Since $h(j) = h_s(j)$, the vertex x has the same colours belonging to $[k]$ in $G_{s\circ s'}$ and in $G_s \otimes_{R,g,h} G_{s'}$.

The vertex x has colour i' in $G_{s\circ s'}$ if and only if $x-w_i$ is an edge of $\widetilde{s}(\widetilde{s'}(\mathbf{I}))$ if and only if $x-w_i$ is an edge of $\widetilde{s'}(\mathbf{I})$ if and only if x has colour i' in $G_{s'}$ if and only if it has colour i' in $G_s \otimes_{R,g,h} G_{s'}$ because $h(i') = \{i'\}$ for all i in $[k]$.

This completes the proof of Claim 3. □

Next we consider the basic contexts $f(t, \mathbf{Id})$ or $f(\mathbf{Id}, t)$, but we need only consider the first case because $f(\mathbf{Id}, t) = f'(t, \mathbf{Id})$, for some f'.

Claim 4. For $s = t \otimes_{T,m,p} \mathbf{Id}$, we have $h_s = p$ and $G_s = val(t) \otimes_{\varnothing,\overline{m},\varnothing} \varnothing$, where $\overline{m}(i) = m(i) \cup \{j' \mid (i,j) \in T\}$, and $\overline{m}(i') = \varnothing$, for i in $[k]$.

Proof. Easy verification from the definitions. We have specified $\overline{m}(i') = \varnothing$, for i in $[k]$ but actually, no vertex of $val(t)$ has a colour i', hence we could take any set for $\overline{m}(i')$. □

Next we consider the contexts of the form $f(t, c)$ or $f(c, t)$, but again we need only consider the first case.

Claim 5. For $s = t \otimes_{T,m,p} c$, we have $h_s = p \circ h_c$, $G_s = val(t) \otimes_{T,g\circ\overline{m},h} G_c$, where $\overline{m}$ is as in Claim 4, $g(i) = \{i\}$, $g(i') = \{j' \mid i \in h_c(j)\}$, $h(i) = p(i)$, $h(i') = \{i'\}$ for all i in $[k]$.

Proof. We observe that s is equivalent to $d \circ c$ where $d = t \otimes_{T,m,p} \mathbf{Id}$ hence, we can use Claims 3 and 4. These two claims give $G_s = G_d \otimes_{R,g,h} G_c$ where $R = \{(i', i) \mid i \in [k]\}$ and $g(i) = \{i\}$, $g(i') = \{j' \mid i \in h_c(j)\}$, $h(i) = h_d(i) = p(i)$, $h(i') = \{i'\}$, for all i in $[k]$, and $G_d = val(t) \otimes_{\varnothing,\overline{m},\varnothing} \varnothing$, where $\overline{m}(i) = m(i) \cup \{j' \mid (i,j) \in T\}$, and $\overline{m}(i') = \varnothing$, for i in $[k]$.

But for all graphs H and K we have that

$$(H \otimes_{\varnothing,\overline{m},\varnothing} \varnothing) \otimes_{R,g,h} K = H \otimes_{T,g\circ\overline{m},h} K \qquad (7.4)$$

Hence we have the desired equality by taking $H = val(t)$ and $K = G_c$. □

These claims yield the following one:

Claim 6. Let us fix k. Every term t in $SPE_t(F_k, C_k)$ can be transformed in linear time into a term $\widehat{t}$ in $T(F_{2k}, C_{2k})$ that defines the same graph as $Eval(t)$ and has no larger height and no larger size than t.

Proof. The proof is an induction on the structure of t. This induction constructs for every term in $SPE_c(F_k, C_k)$ a term $\widehat{c}$ in $T(F_{2k}, C_{2k})$ that defines the representing graph G_c and the mapping $h_c : [k] \longrightarrow \mathcal{P}([k])$. The term $\widehat{c}$ has no larger height and size than c.

1. Definition of $\widehat{t}$ in $T(F_{2k}, C_{2k})$ for t in $SPE_t(F_k, C_k)$.

If $t = b \in C_k$ then $\widehat{t} = b$.

If $t = f(t_1, t_2)$ then $\widehat{t} = f(\widehat{t_1}, \widehat{t_2})$.

If $t = c \bullet t_1$, then $\widehat{t} = \widehat{c} \otimes_{R,g,h_c} \widehat{t_1}$, where R and g are defined in Claim 2.

2. Definition of $\widehat{c}$ in $T(F_{2k}, C_{2k})$ and of h_c for c in $SPE_c(F_k, C_k)$.

If $c = s \circ s'$ we let $h_{s\circ s'} = h_s \circ h_{s'}$ and $\widehat{c} = \widehat{s} \otimes_{R,g,h} \widehat{s'}$, where R, g, h are as in Claim 3.

If $c = t \otimes_{T,m,p} \mathbf{Id}$, let $h_c = p$ and $\widehat{c} = \widehat{t} \otimes_{\varnothing,\overline{m},\varnothing} \varnothing$, where $\overline{m}$ is as in Claim 4.

If $c = t \otimes_{R,g,h} c$, we let $h_s = p \circ h_c$, $G_s = val(t) \otimes_{T,g\circ\overline{m},h} G_c$ where $\overline{m}, g, h$ is as in Claim 5.

The size and height of $\widehat{t}$ are exactly those of t, and the same for c. However, one can still decrease them : the constant $\varnothing$ can be eliminated by using equalities like (7.4) used in Claim 5. The size and height can only decrease.

This transformation of terms can be done in linear time. □

End of proof of Theorem 9. Let G be a graph with n vertices, $n > 1$, given by an m-cwd term of width k in $T(F_k, C_k)$. This term has size $2n - 1$. It can be transformed into a special term of height at most $3.\log(2n - 2)$. This term can be transformed into an equivalent one in $T(F_{2k}, C_{2k})$ of height at most $3.\log(2n - 2) = 3.(\log(n - 1) + 1)$. This completes the proof for the case of undirected graphs.

For directed graphs, instead of one auxilliary set $[k]'$ we will use two : $[k]'$ and $[k]" := \{i" \mid i \in [k]\}$. The proof is similar, we only indicate the modifications to be done. For constructing G_s, we modify the graph $\widetilde{s}(\varnothing)$ as follows : we add to the list of colours of each vertex y the colours i' such that $y \longrightarrow w_i$ and the colours $i"$ such that $y \longleftarrow w_i$ in $\widetilde{s}(\mathbf{I})$. (We write $u \longrightarrow v$ to indicate an edge from u to v.) Then Claim 2 holds with $R := \{(i', i, +) \mid i \in [k]\} \cup \{(i", i, -) \mid i \in [k]\}$ and $g(i) = \{i\}$,$g(i') = g(i") = \varnothing$ for i in $[k]$.

For Claim 3 we let R be the same and $g(i) = \{i\}$, $g(i') = \{j' \mid i \in h_{s'}(j)\}$,$g(i") = \{j" \mid i \in h_{s'}(j)\}$, $h(i) = h_s(i)$, $h(i') = \{i'\}$, $h(i") = \{i"\}$ for all i in $[k]$.

For Claim 4 we let $\overline{m}(i) = m(i) \cup \{j' \mid (i,j,+) \in T\} \cup \{j" \mid (i,j,-) \in T\}$, and $\overline{m}(i') = \overline{m}(i") = \varnothing$, for i in $[k]$.

For Claim 5 we let $\overline{m}$ be as above and $g(i) = \{i\}$, $g(i') = \{j' \mid i \in h_c(j)\}$, $g(i") = \{j" \mid i \in h_c(j)\}$, $h(i) = p(i)$, $h(i') = \{i'\}, h(i") = \{i"\}$ for all i in $[k]$.

Finally for Claim 6 we transform a term t in $SPE_t(F_k, C_k)$ into a term $\widehat{t}$ in $T(F_{3k}, C_{3k})$ where i' is encoded into $i+k$ and $i"$ into $i+2k$, which explains the use of (F_{3k}, C_{3k}) in place of (F_{2k}, C_{2k}).

By Proposition 5, one can apply this result to graphs given by clique-width terms of width at most k and produce balanced m-clique-width terms denoting them of width $2k$ or $3k$. □

Remark 36 : For a graph given by a cwd-term of width k, if one insists on obtaining a clique-width term, one can get one of width $k.2^k$ for undirected graphs (and $k.2^{2k}$ for directed graphs) with height $a.\log n$ where a varies with k and is not bounded by a constant.

We think that $k.2^k$ cannot be replaced by a polynomial in k, however, we have no proof.

The following example indicates why the construction of a clique-width term needs exponential number of colours.

Example 37: Let $I_1, ..., I_h$ enumerate the nonempty subsets of $[k]$. Let S be the edgeless graph with vertices $v_1, ..., v_h$ all coloured by $k+1$. For each $[k]-$coloured graph H, we let $f(H)$ be the graph $H \oplus S$ augmented with undirected edges linking v_j to the vertices of H with a colour belonging to I_j for all $j = 1, ..., h$. It is easy to write a context s with cwd-operations using only colors in $[k+2]$ and such that $\widetilde{s}(H) = f(H)$ for every $[k]-$coloured graph H. For some colouring S' of S with colors in a set L, and some fixed composition ADD of recolouring and edge addition operations (among cwd-operations), we can have $f(H) = ADD(H \oplus S')$ for every $[k]-$coloured graph H, but this implies that L has at least $h = 2^k - 1$ colours. This is so because no two elements of S are linked in $f(H)$ to the same vertices of H in the case where each element of $[k]$ colours some vertex of H.

The result in terms of m-clique-width avoids the exponential jump on the number of colours and the constant a is the same for all k.

However, in view of a concrete implementation, a binary operation in F_k for defining graphs of m-clique-width at most k may need $3k^2$ bits to be encoded, whereas, if we use clique-width operations, one bit is enough for disjoint union and $2\lceil \log k \rceil$ bits make it possible to encode unary operation symbols that use k colours. It remains open to design efficient data structures for both types of operations.